# BALTO: FAST TENSOR PROGRAM OPTIMIZATION WITH BIASED-DIVERSITY-BASED ACTIVE LEARNING

**Anonymous authors**
Paper under double-blind review

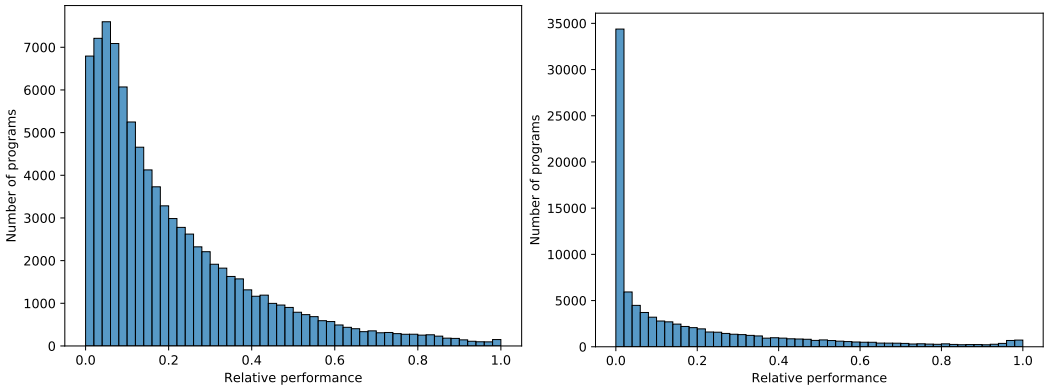
## ABSTRACT

Tensor program optimization (TPO) based on pre-trained models can effectively reduce the computing time of deep neural networks. However, training of such models is prohibitively expensive, which highly depends on a large-scale dataset and thus requires tremendous time-consuming performance measurements (more than 1 million) on target platforms. In this paper, we propose BALTO, a fast TPO approach with biased-diversity-based active learning, aiming at reducing much lower training costs under similar optimization accuracy. The key insight is that random sampling of existing approaches suffers from a heavy redundancy of low-performance programs, which incurs tremendous duplicated time-consuming measurements. Inspired by this, BALTO removes such redundancy by introducing active learning (AL) to TPO for a much lower training cost. However, applying AL with a brute-force way in BALTO can lead to an overestimation problem. To address this, we further propose a biased-diversity-based diversity scheme specially designed for BALTO. We compare BALTO against TenSet on 6 typical hardware platforms over 2 learning models. Experimental results show that, on average, BALTO only requires 5% of the total performance measurements of TenSet to achieve the same or higher model accuracy. Moreover, the optimized tensor programs even outperform that of TenSet by 1.06% due to higher model accuracy.

## 1 INTRODUCTION

Tensor program optimization (TPO) based on pre-trained machine learning (ML) models (Adams et al., 2019; Haj-Ali et al., 2020; Anderson et al., 2020; Zheng et al., 2021), can effectively reduce the computing time of neural networks. The general process of ML-based tensor program optimization mainly consists of three stages. First, a large number of random tensor programs are generated and their corresponding performance is then measured on target hardware platforms. Second, a cost model for predicting program performance is pre-trained based on the dataset constructed from the generated programs and their measured performance. After that, due to high accuracy and fast inference speed, such pre-trained cost model can be leveraged to effectively guide the TPO (Chen et al., 2018; Zheng et al., 2021). Therefore, the pre-trained model is the core of existing TPO approaches.

However, training the pre-trained models is prohibitively expensive. The main reason is that these models highly depend on a large-scale training dataset. Unfortunately, collecting such a dataset involves a large number of performance measurements on diverse range of hardware platforms, suffering from excessively long execution time. For example, for each single hardware platform, around 8 million different tensor programs are required to be measured (Adams et al., 2019; Zheng et al., 2021). Even on a high-end GPU like V100, such measurements still can consume about 4000 GPU hours. This burden could be much worse in those capability-limited systems, e.g., mobile devices such as NVIDIA Jetson TX2, most of which often require more than 5000 GPU hours to conduct the measurements. More importantly, a much larger dataset is required in real-world optimization tasks, so as to achieve better model generalization on different tensor programs. Consequently, as the size of the dataset increases, the number of the measurements can be significantly increased correspondingly. This can lead to great consumption of time and energy, and thus hinders wide deployment of ML-based TPO in industrial practice.




(a) Performance distribution on Platinum 8272CL. (b) Performance distribution on T4.

Figure 1: The ratio of high-performance programs is much smaller than low-performance programs.

To look more closely at the inside of the large-scale datasets sampled randomly by existing approaches, we conduct a deep exploration on the distribution of these datasets. We observe that random sampling adopted in existing approaches can result in imbalanced training dataset, where the high-performance programs are excessively less than that of the low-performance ones. We randomly sample 90,000 tensor programs of TenSet (as shown in Figure 1), and find that high-performance programs only account for 19% and 8% of the total dataset on Platinum 8272CL CPU and T4 GPU respectively. In contrast, the low-performance programs take the bulk of the total dataset (81% and 92% respectively). Such imbalance can further lead to a heavy redundancy of the low-performance programs that incurs in tremendous duplicated time-consuming measurements. The main reason behind the redundancy is that the importance of generated tensor programs is highly different. In fact, program optimization pays more attention to those high-performance programs. Excessively more low-performance programs cannot offer additional benefit of predicting high-performance programs, thus being a heavy redundancy.

To this end, we propose BALTO, a fast TPO approach with biased-diversity-based diversity active learning, aiming at reducing much lower training cost under similar optimization accuracy. The key insight is that random sampling of existing approaches suffers from a heavy redundancy of low-performance programs, which incurs in tremendous duplicated time-consuming measurements. Inspired by this, BALTO removes such redundancy by introducing active learning (AL) to TPO for a much lower training cost. In this way, the duplicated measurements can be significantly lowered, and thus greatly reducing the training cost of building pre-trained model. However, applying AL with a brute-force way in BALTO can lead to overestimation problem, where relative performance of the estimated accuracy is much better than that of the ground truth. To address this problem, we further propose a biased-diversity-based scheme specifically designed for BALTO, which can efficiently reduce the distribution imbalance of the sampled programs caused by overestimation.

Finally, we integrate BALTO into TenSet, and compare BALTO with state-of-the-art baseline TenSet on 6 typical hardware platforms (e.g, GPU platforms including NVIDIA T4 and NVIDIA K80, CPU platforms including Intel Platinum 8272CL, Intel E5-2673 v4, AMD EPYC 7452, and ARM Graviton2) over two learning models (e.g., XGBoost, MLP). The experimental results show that BALTO can achieve same or higher model accuracy while only requiring 5% of TenSet's total hardware measurements. Moreover, the optimized tensor programs even outperform that of TenSet by 1.06% due to higher model accuracy.

To the best of our knowledge, BALTO is the first work to reduce the training cost of TPO by introducing AL. Summarily, our key contributions are three-fold:

- We conduct a deep exploration on the distribution of large-scale datasets sampled randomly by existing approaches, and observe that the random sampling results in imbalanced dataset, and suffers from heavy redundancy in training dataset.
- We propose BALTO, a fast TPO approach with biased-diversity-based diversity active learning, aiming at reducing much lower training cost under similar optimization accuracy.

- We propose a biased-diversity-based scheme specifically designed for BALTO , so as to efficiently apply active learning in BALTO and address the overestimation problem.
- We conduct a comprehensive performance evaluation and analysis across multiple tasks for BALTO, indicating that BALTO can consistently outperform TenSet without accuracy degradation with much fewer performance measurements on six hardware platforms and achieves even better optimization results on T4 .

## 2 BACKGROUND

### 2.1 TENSOR PROGRAM AND ITS REPRESENTATION.

Tensor programs describe the computation of multi-dimensional data arrays, which can be either dense or sparse. Figure 2 shows an example tensor program for the padded 2D convolution operator with a fused ReLU layer. Concretely, the program contains four statements, including the input padding, the initialization of the convolution output, the computation of convolution, and the fused ReLU operation. Compilers usually extract loop-related features (e.g., the number of loop parallelism and the length of loop unroll) for each statement to form a vector of features $v_s$, where $s$ denotes the $s$th statement. Then the example program can be represent as $x = [v_1, v_2, v_3, v_4]$, which combines all of the statements' features.

```
parallel ax0.0@ax1.0@ax2.0@ (0,4)
 for i1 (0,57)
  for i2 (0,15)
   for i3 (0,64)
    PaddedInput = ...          → v1
 for ax3.0 (0,2)
  for ax2.1 (0,7)
   for ax3.1 (0,8)
    Conv2dOutput = 0           → v2
    for rx.0 (0,3)
     for rc.0 (0,4)
      for ry.1 (0,3)
       for rc.1 (0,16)
        for yy.3 (0,28)
         vectorize ff.3 (0,8)
          Conv2dOutput = ...   → v3
    for ax1.2 (0,28)
     vectorize ax3.2 (0,8)
      T_relu = ...             → v4
```

Figure 2: 2D convolution's tensor program.

### 2.2 TENSOR PROGRAM OPTIMIZATION.

TPO is a process of transforming an input tensor program into another tensor program with optimal or near optimal performance. Figure 3 demonstrates a workflow of a typical TPO. As shown in Figure 3, it mainly consists of three modules in a TPO, including program performance estimation, optimization space exploration, and program performance evaluation. The program performance estimation module is used for predicting the performance of input programs, which is usually composed of machine learning models or expert-designed models. The optimization space exploration module searches for high-performance program transformations in a pre-defined search space and outputs a batch of programs with high scores on predicted performance. The program performance evaluation module compiles the batch of programs and measures their execution times on the hardware platform. The measured performance can be used for further fine-tuning the models. Finally, the program with the highest measured performance is output as the optimization result.

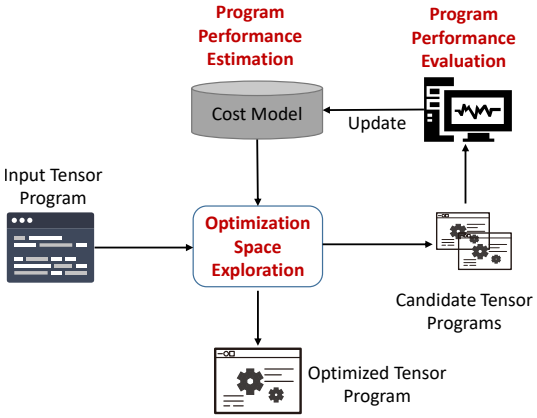



Figure 3: Workflow of TPO.

### 2.3 PRE-TRAINING OF PERFORMANCE MODEL.

Pre-training of performance model has two key stages typically: sampling stage and training stage. In sampling stage, model pre-training usually depends on a large-scale dataset based on program performance, which are generated by performance measurements on target hardware platforms. Concretely, the sampling process mainly consists of three steps. First, a set of program optimization

tasks are generated by compiler, such as conv2d operator with different workloads. Second, for each generated task, random sampling is conducted in the optimization search space to produce a set of tensor programs without performance measurement. Third, these tensor programs are delivered to the module of performance evaluation for performance measurement, and finally the measured performance and the features of tensor programs are kept into the dataset.

In training stage, generally, the performance model aims at fitting the relative performance of the programs, which can be denoted as $y = \frac{T}{T_{max}}$. Concretely, $T$ denotes the throughput of the program, and $T_{max}$ denotes the maximum throughput that can be reached by the corresponding optimization task of that program. Since the maximum throughput $T_{max}$ of the task is unknown, $T_{max}$ usually can be estimated by the maximum throughput of the sampled programs. Importantly, such estimation can be quite accurate if given a large number of samples. The commonly-used training models include XGBoost, MLP and LSTM, etc, and the goal of model training is to minimize $rmse$ or $lambdarankloss$. Specifically, the evaluation criteria of the model is greatly critical. Compared with $rmse$ or $R^2$, the Top-k performance predicted by the model on each optimization task is more related to the optimization quality of the compiler (Zheng et al., 2021). For a given set of tensor programs, the Top-k performance can be calculated as the division of the predicted Top-k programs' average performance and the optimal programs' performance.

### 2.4 Active Learning

The training of machine learning, especially of deep learning, highly depends on a large number of labeled datasets. However, the generation of these labels is prohibitively expensive. Active learning (AL) is a popular training framework, aiming at effectively reducing the requirement of the expensive labels. A commonly-used way of AL is the pool-based active learning. In this way, unlabelled samples are first selected from a pool with a fixed size, then the selected samples are delivered to be labelled, and finally these labelled samples are used for model training. Since there usually exists redundancy in dataset, a desirable acquisition method can effectively reduce the redundancy and uses as few data as possible to achieve the similar model accuracy.

## 3 Active Learning Based Model Training

### 3.1 Overview of BALTO

BALTO is a pre-trained-model-based TPO framework that mainly consists of two parts, including AL-based model training and pre-trained-model-based program optimization, shown in Figure 4. The model training part consists of 4 different modules, including initial labelled dataset building module, model training module, program selection module, and program performance evaluation module. When building the initial labelled dataset $D_l$, BALTO randomly samples programs from $N$ optimization tasks and measures their performance. Then BALTO iteratively trains the model for $T$ time steps based on active learning. At each time step, BALTO first trains the model based on $D_l$ and then selects a batch of programs for performance measurement if the total budget is not met. The measured programs are then used to update the labelled dataset for later model training. Finally, the model training part outputs a trained model for later program optimization tasks once the total measurement budget is met. The program optimization part takes the commonly used workflow as described in the background part.

### 3.2 Diversity-based AL for Program sampling

Diversity-based active learning aims at reducing the redundancy of similar samples by increasing the diversity of the selected samples. Core-set (Sener & Savarese, 2018), one of the state-of-the-art diversity-based approaches, models the selection of samples as solving the following problem:

$$\min_{S_t \subseteq D_u} \max_{x_i \in S_t} \min_{x_j \in D_l \cup S_t} d(x_i, x_j) \quad s.t. \quad |S_t| \leq B_t \tag{1}$$

where $S_t$ is the samples selected at time step $t$, $d(x, y)$ is a measure of the distance between sample $x$ and $y$, and $B_t$ is the total budget of samples at step $t$. In our case, the program selection process can be formulated as :

$$select_{diverse} = \min_{S_{t,k} \subseteq D_u} \max_{x_{t,k,i} \in S_{t,k}} \min_{x_{t,k,j} \in D_l \cup S_{t,k}} d(x_{t,k,i}, x_{t,k,j}) \quad s.t. \quad |S_{t,k}| \leq B_{t,k} \tag{2}$$

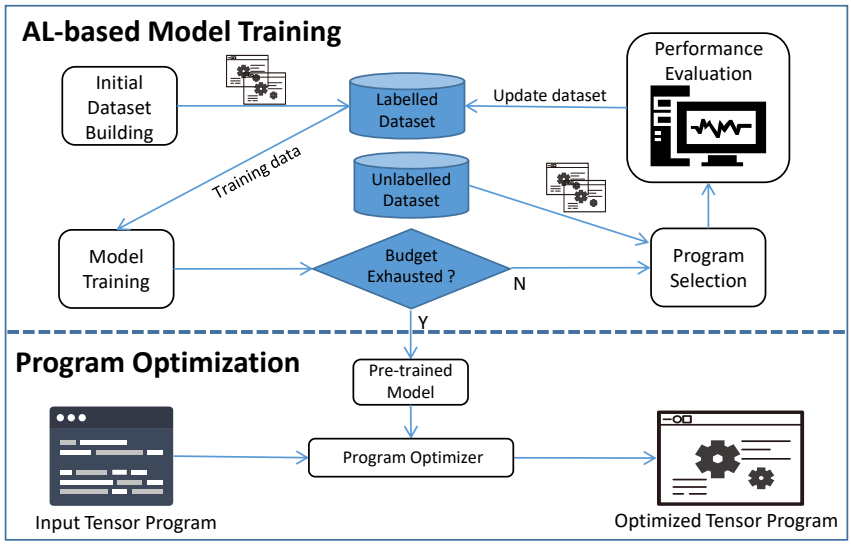



Figure 4: Overview of BALTO.

where $S_{t,k}$ is the selected programs for task $k$ at time step $t$, $x_{t,k,j}$ is the $j$th program of task $k$, and $B_{t,k}$ is the total budget allocated for task $k$ at step $t$.

**Output-diversity-based program selection** Random sampling can result in unbalanced distribution of the samples. In such an unbalanced distribution, the number of low-performance programs accounts for a large part of the total samples, which makes the model easily fit to these low-performance programs instead of the high-performance ones given a relatively small number of total samples. As a result, the prediction of the model for those high-performance programs falls into a very low accuracy, and thus cannot effectively guide the compiler to identify high-performance programs. Therefore, it is essential to balance the distribution of the samples by increasing the proportion of high-performance programs reasonably. To achieve this goal, BALTO increases the diversity of the sampling by solving Equation-2 based on an effective distance function. Here, we design this distance function for BALTO by taking the distance between the predicted performance of programs as the distance between programs, which can be formulated as $d(x_{t,k,i}, x_{t,k,j}) = |f(x_{t,k,i}) - f(x_{t,k,j})|$ where $f(x_{t,k,i})$ is the predicted performance of program $x_{t,k,i}$. The main advantage of the distance function is that it can ensure a much more balanced distribution of the sampling. This is because the diversity of the predicted performance makes the predicted performance of the sampled programs more decentralized into the range [0,1], rather than centralized around 0, thus greatly reducing the proportion of low-performance programs.

### 3.3 Biased-Diversity-based AL for Program Sampling

For a given task, the maximal program throughput $\tilde{T_m}$ can be underestimated to a large extent under a small number of performance measurements. Correspondingly, the underestimated $\tilde{T_m}$ further incurs in overestimated relative performances of most programs at the early stage of AL process. As a result, even with the diversity-based selection, such overestimation can still make the sampled programs imbalanced and mainly gathered at low-performance regions.

**Biased selection of high-performance programs.** To solve the overestimation problem, we propose a biased selection process that can effectively select the high-performance programs in case of that the sampled programs are gathered at low-performance regions. Here, we formulate the biased selection as following equation:

$$select_{biased} = \min_{S_{t,k} \subseteq D_u} \max_{x_{t,k,i} \in S_{t,k}} f(x_{t,k,i}) \min_{x_{t,k,j} \in D_l \cup S_{t,k}} d(x_{t,k,i}, x_{t,k,j}) \quad s.t. \quad |S_{t,k}| \leq B_{t,k} \quad (3)$$

where we multiply the predicted performance of program $x_{t,k,i}$ with the minimal distance between $x_{t,k,i}$ and the selected programs. The biased selection gives high weights to those programs with

high scores on predicted performance, so as to accelerate the exploration of programs in high-performance regions. BALTO conducts Equation-3 only when the measured performances are gathered in the low-performance regions and conducts Equation-2 when the distributions are well balanced. To decide how to switch between Equation-2 and Equation-3, BALTO leverages the following criterion for selection:

$$select = \begin{cases} select_{diverse} & \text{if} \quad -\mathbb{E}[\log(y_{t,k,i})] \leq C \\ select_{biased} & \text{else} \end{cases} \tag{4}$$

where $y_{t,k,i} \in [0,1]$ is the relative performance of program $x_{t,u,i}$, and$C$ is a constant threshold for choosing the selection approaches. For a binomial distribution $P(z), z \in \{0,1\}$ with $p(1) = 1$, it describes an distribution of selected programs, where all the programs are high-performance. For a given tensor program $x_{t,k,i}$, we create another binomial distribution $Q(x_{t,k,i})$ with $q(\{x_{t,k,i} \text{ is performant}\}) = y_{t,k,i}$, which means that a program with a relative performance of $y_{t,k,i}$ has a possibility of $y_{t,k,i}$ to be high-performance. BALTO leverages KL-divergence to describe the distance between the distribution $P$ and $Q$, which can be formulated as following:

$$\begin{aligned} D_{KL}(P||Q) &= \mathbb{E}[p(z)\log(p(z)) - p(z)\log(q(x_{t,k,i})] \\ &= -\mathbb{E}[p(z=1)\log(q(\{x_{t,k,i} \text{ is performant}\})] \\ &= -\mathbb{E}[\log(y_{t,k,i})] \end{aligned} \tag{5}$$

Although the biased selection scheme can alleviate the imbalance of performance distribution caused by overestimation, it scarifies the diversity of high performance programs to certain extent. In addition, the KL-divergence in Equation-5 is positively related to such diversity. The reason is that KL-divergence can effectively estimate how close the performance distribution of the currently sampled programs to $P$. As such, the lower the KL-divergence is, the closer the relative performance of the sampled programs is to 1 and the lower the diversity is. Thus, to ensure the selection of representative programs, BALTO has to switch to the output-diversity-based selection scheme, when the value of KL-divergence is below a constant threshold.

**Algorithm 1** Biased-diversity-based active learning algorithm

**Require:** $N$, $T$, $D_l$, $D_u$, $C$, $\{B_{t,k}\}_{1 \leq t \leq T, 1 \leq k \leq N}$
**Output:** $f$
1: Train model $f$ on $D_l$
2: **for** $t = 1, ..., T$ **do**
3:  **for** $k = 1, ..., N$ **do**
4:  $S_{t,k} \leftarrow \phi$, $Y_{t,k} \leftarrow \{y_{t,k,i} | x_{t,k,i} \in D_l\}$
5:  $dist_{KL} = -\frac{1}{|Y_{t,k}|} \sum_i \log(y_{t,k,i})$
6:  **while** $|S_{t,k}| < B_{t,k}$ **do**
7:  **if** $dist_{KL} \leq C$ **then**
8:  $p = \arg\max_{p_{t,k,i} \in D_u} \min_{p_{t,k,j} \in D_l \cup S_{t,k}} d(p_{t,k,i}, p_{t,k,j})$
9:  **else**
10:  $p = \arg\max_{p_{t,k,i} \in D_u} f(x_{t,k,i}) \min_{p_{t,k,j} \in D_l \cup S_{t,k}} d(p_{t,k,i}, p_{t,k,j})$
11:  $S_{t,k} \leftarrow S_{t,k} \cup \{p\}$
12:  Measure performances of programs in $S_{t,k}$
13:  $D_l \leftarrow D_l \cup S_{t,k}$, $D_u \leftarrow D_u \setminus S_{t,k}$
14:  Update model $f$ on $D_l$
15: **return** $f$

**Biased-diversity-based active learning algorithm.** Algorithm 1 shows the details of our AL algorithm which is based on a greedy solution of Equation-4. The algorithm requires 6 inputs: 1) $N$, the total number of tasks, 2) $T$,the total number of time steps for active learning, 3) $D_l$, the initial labelled dataset, 4) $D_u$, the initial unlabelled dataset, 5) $C$(e.g., $C = 1$), the threshold for the choosing of selection approaches, and 6) $\{B_{t,k}\}_{1 \leq t \leq T, 1 \leq k \leq N}$ (e.g., 0.25% of the total programs), the hardware measurement budgets for task $k$ at time step $t$. At each time step, the algorithm iterates over all of the $N$ tasks to select programs for measurement. For each task, the algorithm can select at most $B_{t,k}$ samples to form a dataset $S_{t,k}$. For a single selection of programs, the algorithm chooses the proper selection scheme according to the $D_{KL}$ caculated at line 5. The selected programs are sent to the program performance evaluation module for performance measurement. The measured

programs are inserted into the labelled dataset and removed from the unlabelled dataset. Then the cost model is updated based on the new labelled dataset. Once the total budget is reached, the algorithm outputs the trained model.

# 4 PERFORMANCE EVALUATION

In this section, we evaluate the performance of BALTO comprehensively. We integrate BALTO into TenSet, and use TenSet as the state-of-the-art baseline for comparison. The experimental methodology is four-fold. First, we show that BALTO can achieve comparable accuracy while with much less hardware measurements on 6 hardware platforms. Second, we demonstrate that the models trained with much less performance measurements have comparable optimization performance on a real-world hardware platform. Third, we perform ablation study to verify the effectiveness of the core components of BALTO. Fourth, we visualize the performance distribution of our proposed sampling strategy to illustrate that the proposed strategy helps balancing the distribution of programs.

## 4.1 COMPARISON OF BALTO AND TENSET FOR MODEL PRECISION

**Dataset.** We evaluate BALTO's effectiveness on the dataset provided by TenSet. The dataset consists of program performance measurement records from 6 different hardware platforms (e.g, GPU platforms including NVIDIA T4 and NVIDIA K80, CPU platforms including Intel Platinum 8272CL, Intel E5-2673 v4, AMD EPYC 7452, and ARM Graviton2). Each platform includes a total number of 8,596,208 tensor program measurement records that are sampled from 2307 different types of tasks. We use 10% of the records as the test dataset and the remaining 90% records as the train dataset for the baselines. We further select at most 5% of the training dataset for training BALTO and report the Top-1 and Top-5 score on the test dataset.

**Results.** We compared BALTO with TenSet on two ML models including XGBoost and MLP. All the models adopt the same hyperparamenters with TenSet. As shown in Table 1, BALTO achieves same or even higher accuracy but with $20\times$ reduction of the hardware measurements. Take T4 for example, BALTO achieves an accuracy improvement of 4.3% on XGBoost model and achieves the same accuracy on MLP model respectively. The reason that TenSet achieves a relatively lower accuracy is that TenSet trains on a randomly sampled program dataset where the ratios of low-performance programs and high-performance programs are extremely imbalanced. Differently, such imbalance can be greatly alleviated by the proposed biased-diversity-based active learning of BALTO, thus delivering much lower training cost but similar or even higher accuracy.

Table 1: Comparison between BALTO and TenSet on 6 hardware platforms with 2 learning models.

| Methods | T4 | K80 | P-8272CL | E5-2673 v4 | EPYC 7452 | Graviton2 |
|---|---|---|---|---|---|---|
| TenSet-XGBoost | 84.6(91.6) | 85.0(91.7) | 82.7(90.6) | 83.0(90.4) | 83.9(90.9) | 79.2(89.5) |
| BALTO-XGBoost | **88.9(94.3)** | **88.9(94.4)** | **87.5(93.4)** | **86.7(93.1)** | **88.0(93.7)** | **81.5(91.2)** |
| TenSet-MLP | 90.5(95.3) | 90.5(95.2) | 86.0(92.8) | 85.9(92.9) | 88.3(94.0) | 81.4(91.2) |
| BALTO-MLP | 90.6(95.1) | 90.5(95.1) | **87.4(93.9)** | **87.6(94.0)** | **89.4(94.7)** | 81.4(91.4) |

## 4.2 OPTIMIZATION PERFORMANCE ON A REAL-WORLD PLATFORM

To verify that cost models trained on much fewer programs can still effectively guide the TPO process, we evaluate BALTO on NVIDIA T4 and report the optimized execution time of 5 commonly used neural networks of TenSet and BALTO.

**Benchmarks.** We evaluate the optimization results on 5 commonly used neural networks including ResNet-50 (He et al., 2016), MobileNet-v2 (Sandler et al., 2018), ResNext-50 (Xie et al., 2017), BERT-tiny and BERT-base (Devlin et al., 2018). For the three CNN models, we set the batch size to 1 and the input shape to $224 \times 224$. For BERT models, we set the sequence length to be equal to 128. We assign at most 1000 trials of hardware measurements for each network and report the average execution time based on 5 independent experiments of program optimization.

**Results.** As shown in Table 2, BALTO achieves the same or better optimization performance compared to TenSet. Specially, when using the XGBoost cost model, BALTO achieves an average of $1.06\times$ performance improvement over TenSet while requiring only 5% of the total hardware measurements for training the model. The improvement comes from a more accurate performance model that helps the compiler to identify the high-performance programs efficiently. When using the MLP cost model, the performance of BALTO is the same as that of TenSet since the two approaches' models have the same accuracy on T4. The experiments' results show that models trained on much fewer measured programs can still guide the optimization efficiently. Thus we do not need to measure as much of programs as existing optimization frameworks do.

Table 2: Evaluations of BALTO and TenSet on 5 commonly used neural networks.

| Methods | ResNet-50 | MobileNet-v2 | ResNext-50 | BERT-tiny | BERT-base |
|---|---|---|---|---|---|
| TenSet-XGBoost | 3.81 ms | 0.61 ms | 3.37 ms | 3.34 ms | 11.34 ms |
| BALTO-XGBoost | **3.45** ms | 0.59 ms | **3.25** ms | **3.02** ms | 11.3 ms |
| TenSet-MLP | 3.62 ms | 0.59 ms | 3.30 ms | 2.9 ms | 11.38 ms |
| BALTO-MLP | 3.62 ms | 0.59 ms | 3.31 ms | 2.92 ms | 11.37 ms |

### 4.3 Ablation Study

To evaluate the effectiveness of the components in BALTO, we take the model training process on NVIDIA T4 as an example for ablation studies. Firstly, we verify the effectiveness of output-diversity-based AL by comparing it with other approaches. Then, we verify the effectiveness of biased selection by comparing it to the output-diversity-based AL approach.

**Effectiveness of output-diversity-based AL.** We compare the output-diversity-based AL with three different sampling strategies including a random sampling approach, an input-diversity-based AL approach, and an uncertainty-based AL approach. As for the input-diversity-based AL approach, we calculate the distance as the absolute error of the given two program's features. As for the uncertainty-based AL approach, we calculate the uncertainty as the variance of predictions using an ensemble of three independently trained models. As shown in Figure 5, the output-diversity performs best among the four approaches. The input-diversity-based AL approach performs even worse than the random sampling approach since the sampled programs are more likely to be low-performance programs. The uncertainty-based AL approach is more efficient than the random sampling approach, since most of the programs with high uncertainty are high-performance programs. However, the uncertainty-based approach tends to sample programs with similar performance and thus cannot outperform the output-diversity-based approach.

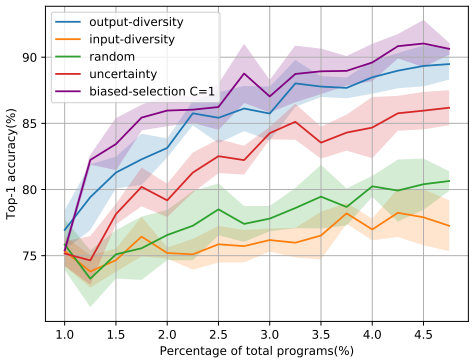



Figure 5: Comparison of different selection schemes.

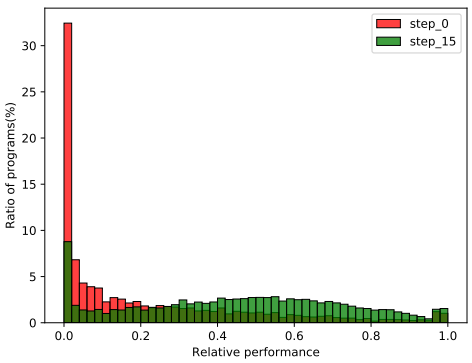



Figure 6: Visualization of performance distributions.

**Effectiveness of biased selection.** We further compare the proposed biased selection approach with the output-diversity-based approach. As shown in Figure 5, biased selection with a proper $C$ (e.g., $C = 1$) can outperform the output-diversity-based approach. At the early stage of the training, the accuracy score of the biased selection approach grows much faster than that of the output-diversity-based approach, since the bias for high-performance programs effectively reduces the distribution

imbalance caused by the overestimation problem. Meanwhile, after the early stage of the training, the estimation of the relative performance becomes more accurate as well as the distribution becomes more balanced. Therefore, the biased selection switches to the output-diversity-based selection and finally achieves a higher precision score under the same time budget.

### 4.4 VISUALIZATION

**Visualization of the performance distributions.** Figure 6 is a histogram of program performance distributions at two different time steps (e.g., the first time step and the last time step). The y-axis represents the percentage of programs and the x-axis represents the relative performance. As shown in Figure 6, the program distribution of the first time step is extremely unbalanced since the programs are sampled randomly. After 15 steps of program sampling based on active learning, the program distribution of the last time step becomes more balanced, which clearly demonstrates the effectiveness of our biased-diversity-based AL approach.

## 5 RELATED WORK

**Tensor optimization frameworks.** Commonly used TPO frameworks include online-trained-model-guided program optimization (Chen et al., 2018; Zheng et al., 2020; 2022; Ahn et al., 2020), and predefined-model-guided program optimization (Zhu et al., 2022; Dave et al., 2019; Adams et al., 2019; Haj-Ali et al., 2020; Anderson et al., 2020; Zheng et al., 2021). Predefined-model-guided program optimization can optimize the given tensor programs in a short time and its predefined model can be either designed manually by domain experts (Zhu et al., 2022; Dave et al., 2019) or pre-trained using randomly generated programs (Adams et al., 2019; Haj-Ali et al., 2020; Anderson et al., 2020; Zheng et al., 2021). The manually designed model are usually designed for specific hardware platforms and thus hard to adapt to new platforms. The pre-trained-model based approaches can effectively adapt to the new platforms but require a large number of hardware measurements of randomly sampled programs. The online-trained-model-guided approaches relies training the model with data collected online. Thus this kind of optimization approaches is time consuming for each input optimization task. Although BALTO is a fast tensor optimization approach designed for the pre-trained-model-based approaches, the optimization schemes in BALTO can also be applied to the online-trained-model-guided approaches, which is our future work.

**Active learning.** AL is widely used for reducing the expensive annotation cost for model training (Aghdam et al., 2019; Joshi et al., 2009; Culotta & McCallum, 2005). The acquisition function can reduce the redundancy in sampling by only anotating informative or representative samples and thus is the core to AL. Typically, the most frequently used acquisition strategies can be divided into uncertainty-based approaches (Yoo & Kweon, 2019; Roth & Small, 2006) and diversity-based approaches (Sener & Savarese, 2018; Hasan & Roy-Chowdhury, 2015). Uncertainty-based approaches tend to query samples that are most uncertain (Lewis & Catlett, 1994). MC Dropout (Gal & Ghahramani, 2016)adopts random dropout for obtaining CNN's prediction uncertainty. (Beluch et al., 2018) uses an ensemble of classifiers for obtaining the uncertainty. Diversity-based acquisition emphasizes on the diversity of selected samples. GSx (Yu & Kim, 2010) maximizes the diversity in the feature space, while iGS (Wu et al., 2019) maximizes the diversity in both feature space and label space. Our active learning approach is based on the Core-Set framework and explores the diversity of predicted programs' performance.

## 6 CONCLUSION

In this paper, we propose BALTO, a fast TPO approach with biased-diversity-based active learning to reduce the training cost of the pre-trained model while with the same or higher prediction accuracy. The key insight is that there exists a heavy redundancy of low-performance programs when sampling randomly with existing approaches. To remove such redundancy and for a much lower training cost, active learning is introduced in BALTO. Moreover, we further propose a biased-diversity-based scheme specially designed for BALTO to address the overestimation problem. The empirical results demonstrate that BALTO consistently shows superior training performance (i.e., $20\times$ reduction in required hardware measurements.) on a wide range of environment configurations (i.e., 6 hardware platforms and 2 learning models) and even better optimization performance on T4.

## REFERENCES


Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, et al. Learning to optimize halide with tree search and random programs. *ACM Transactions on Graphics (TOG)*, 38(4): 1–12, 2019.

Hamed Habibi Aghdam, Abel Gonzalez-Garcia, Joost van de Weijer, and Antonio M. López. Active learning for deep detection neural networks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3671–3679, 2019.

Byung Hoon Ahn, Prannoy Pilligundla, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. Chameleon: Adaptive code optimization for expedited deep neural network compilation. *arXiv preprint arXiv:2001.08743*, 2020.

Luke Anderson, Andrew Adams, Karima Ma, Tzu-Mao Li, and Jonathan Ragan-Kelley. Learning to schedule halide pipelines for the gpu. *arXiv preprint arXiv:2012.07145*, 2020.

William H. Beluch, Tim Genewein, A. Nürnberger, and Jan M. Köhler. The power of ensembles for active learning in image classification. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9368–9377, 2018.

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. *Advances in Neural Information Processing Systems*, 31, 2018.

Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI*, 2005.

Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. Dmazerunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–27, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *ArXiv*, abs/1506.02142, 2016.

Ameer Haj-Ali, Hasan Genc, Qijing Huang, William Moses, John Wawrzynek, Krste Asanović, and Ion Stoica. Protuner: tuning programs with monte carlo tree search. *arXiv preprint arXiv:2005.13685*, 2020.

Mahmudul Hasan and Amit K. Roy-Chowdhury. Context aware active learning of activity recognition models. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4543–4551, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ajay J. Joshi, Fatih Murat Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2372–2379, 2009.

David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *ICML*, 1994.

Dan Roth and Kevin Small. Margin-based active learning for structured output spaces. In *ECML*, 2006.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1aIuk-RW.

Dongrui Wu, Chin-Teng Lin, and Jian Huang. Active learning for regression using greedy sampling. *Inf. Sci.*, 474:90–105, 2019.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Donggeun Yoo and In So Kweon. Learning loss for active learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 93–102, 2019.

Hwanjo Yu and Sungchul Kim. Passive sampling for regression. *2010 IEEE International Conference on Data Mining*, pp. 1151–1156, 2010.

Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, et al. Ansor: Generating {High-Performance} tensor programs for deep learning. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, pp. 863–879, 2020.

Lianmin Zheng, Ruochen Liu, Junru Shao, Tianqi Chen, Joseph E Gonzalez, Ion Stoica, and Ameer Haj Ali. Tenset: A large-scale program performance dataset for learned tensor compilers. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Size Zheng, Renze Chen, Anjiang Wei, Yicheng Jin, Qin Han, Liqiang Lu, Bingyang Wu, Xiuhong Li, Shengen Yan, and Yun Liang. Amos: enabling automatic mapping for tensor computations on spatial accelerators with hardware abstraction. In *ISCA*, pp. 874–887, 2022.

Hongyu Zhu, Ruofan Wu, Yijia Diao, Shanbin Ke, Haoyu Li, Chen Zhang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Wei Cui, et al. {ROLLER}: Fast and efficient tensor compilation for deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 233–248, 2022.

# A Appendix

You may include other additional sections here.