A Additional Dataset information

License and Documentation The documentation and license of the dataset is at `https://github.com/tlc-pack/tenset`. The github repo contains instructions and code on how to download and use the dataset.

Maintenance Plan We will monitor the issues and provide necessary maintenance to ensure the access to the data.

Author Statement The authors bear all responsibility in case of violation of rights.

Additional Tables and Figures

Network architecture	Batch size set	Image size (sequence length) set
resnet-18	{1, 4, 8}	{224, 240, 256}
resnet-50	{1, 4, 8}	{224, 240, 256}
wide-resnet-50	{1, 4, 8}	{224, 240, 256}
resnext-50	{1, 4, 8}	{224, 240, 256}
mobilenet-v2	{1, 4, 8}	{224, 240, 256}
mobilenet-v3	{1, 4, 8}	{224, 240, 256}
densenet-121	{1, 2, 4}	{224, 240, 256}
inception-v3	{1, 4, 8}	{299}
resnet-3d-18	{1, 2, 4}	{112, 128, 144}
dcgan	{1, 4, 8}	{64, 80, 96}
bert-tiny	{1, 2, 4}	{64, 128, 256}
bert-base	{1, 2, 4}	{64, 128, 256}
bert-medium	{1, 2, 4}	{64, 128, 256}
bert-large	{1, 2, 4}	{64, 128, 256}

Table 6: Networks Configurations. We vary the batch size and input image size (or sequence length) of common network architectures to generate different computational graphs of neural networks. For example, the first row means that we set the input shape of resnet-18 as (batch size=1, image size=224), (batch size=1, image size=240), ..., (batch size=8, image size=240), (batch size=8, image size=256) respectively, which means 9 different configurations in total.

B Definition of Dataset-based Evaluation Metrics

For a single task, we define the following metrics. Let N denote the number of programs for this task in the dataset. Let y_i denote the measured normalized throughput of program i. The throughput is the inverse of latency. The throughputs of all programs in a task are normalized to the range of [0, 1], so the best program has a normalized throughput of 1. Let $\hat{y_i}$ denote the predicted normalized throughputs.

- **Rooted Mean Square Error (RMSE):** $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2}$.
- **Pairwise Comparison Accuracy:** $\frac{C}{N\cdot(N-1)/2}$, where C is the number of correct pairs. For each pair (i, j) where $0 \leq i < j < N$, the pair is correct if $(y_i \leq y_j \wedge \hat{y_i} \leq \hat{y_j}) \vee (y_i > y_j \wedge \hat{y_i} > \hat{y_j})$.
- R^2 **(Coefficient of Determination)**: $1 - \frac{\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{N}(y_i-\bar{y})^2}$, where $\bar{y} = \frac{\sum_{i=1}^{N} y_i}{N}$.
- **Mean Absolute Percentage Error (MAPE):** $\frac{100}{N}\sum_{i=1}^{N}|\frac{y_i-\hat{y}_i}{y_i}|$.
- **Top-k Score**: Since the ultimate goal of a cost model is to identify the best programs instead of to accurately predict the latency for each of them, we propose the top-k score metric, where k is a customized parameter (*e.g.*, 1, 5). The top-k score is

$$\frac{\text{The highest throughput if using the top-k prediction of the cost model}}{\text{The highest throughput if knowing all labels}} = \frac{\max_{i\in A} y_i}{\max_{0\leq i<N} y_i}$$

, where let A is set of indices of the k programs with the highest predicted throughputs. The higher the score, the better the model.

For a dataset with M tasks, we can average the metrics for all tasks in a weighted way. The weight of a task is the number of programs belong to the task.

If we know the M tasks come from a single network, then we can use a special definition of top-k score. This definition defines top-k score at the neural network level and is better than a simple weighted average. Let z_{ij} be the measured latency of program j in task i. Let A_i be the set of indices of the k programs with the lowest predicted latencies in task i. Let w_i be the number of occurrence of task i in the network. We can define the top-k score as

$$\frac{\text{The lowest latency if knowing all labels}}{\text{The lowest latency if using the top-k prediction of the cost model}} = \frac{\sum_i w_i \cdot \max_{0 \le j < N_i} z_{ij}}{\sum_i w_i \cdot \max_{j \in A_i} z_{ij}}$$

C Experimental Setup

In all sections, for all models, we use all the listed features in the Appendix B of [48], combined with a workload embedding generated from an unsupervised whole-graph embedding model, Local Degree Profile (LDP) [9].

In Sec. 5.1, Model #1 is a XGBoost model trained with Mean Squared Error (MSE) loss, Model #2 is a MLP model trained with Mean Squared Error (MSE) loss, Model #3 is a MLP model trained with LambaRank loss.

The hyperparameters of MLP and XGBoost models are listed in Table 7 and Table 8. For LSTM models, we replace the sum aggregation in MLP with a LSTM layer and keep the same hyperparameters.

For all search-related experiments, we use Azure D32s_v4 for Intel Platinum 8272CL, Azure F16s for Intel E5-2673 and AWS p2.8xlarge for NVIDIA K80. The training of a model takes about several GPU hours. We use TVM v0.8dev, LLVM 9 and CUDA 10 for compilation. We use XGBoost 1.2 and PyTorch 1.7 for training models.

Parameter	Value
in_dim	324
hidden_dim	256
out_dim	1
learning rate	7e4
num_layers (encoder + decoder)	5

Table 7: Hyperparameters of MLP Models

Parameter	Value
max_depth	6
gamma	0.003
min_child_weight	2
eta	0.2

Table 8: Hyperparameters of XGBoost Models