GPU Occupancy Prediction of Deep Learning Models Using Graph Neural Network

Hengquan Mei[1*], Huaizhi Qu[1*], Jingwei Sun[1†], Yanjie Gao[2], Haoxiang Lin[2], Guangzhong Sun[1]
[1]*University of Science and Technology of China, China*
[2]*Microsoft Research, China*
{hengquanmei, qhz991029}@mail.ustc.edu.cn, {sunjw, gzsun}@ustc.edu.cn, {yanjga, haoxlin}@microsoft.com

***Abstract*—Over the past few years, deep learning has been rapidly adopted in many fields. Among the various hardware accelerators specifically for deep learning computation, graphics processing units (GPUs) are mainly used. GPU occupancy—the average ratio of active warps to maximum supported warps on all streaming multiprocessors—is an essential indicator of how well GPUs are utilized. Predicting the GPU occupancy of deep learning models is critical for boosting both job runtime performance and platform resource efficiency. However, GPU occupancy prediction is challenging due to the complex factors hidden in framework runtimes and diverse architectures and hyperparameters of models. In this paper, we propose DNN-occu to predict the GPU occupancy of deep learning models. Our key observation is that models can be represented as directed acyclic computation graphs. DNN-occu extracts a set of occupancy-related features from the computational semantics of the graph nodes and edges. It also employs a novel graph neural network for better feature encoding and prediction generalization. The experiments on various configurations of real-world deep learning models show that DNN-occu achieves high accuracy for occupancy prediction (with an overall error of 9.271%) and has a strong generalization ability for unseen models. In addition, we apply DNN-occu in a trace-driven simulation of deep learning workload scheduling and achieve up to a 31.45% increase in overall GPU utilization and a 19.71% reduction in makespan.**

***Index Terms*—deep learning, GPU occupancy, graph neural network, performance prediction**

I. Introduction

Over the past few years, deep learning (DL) has been rapidly adopted in various fields, including computer vision, natural language processing, recommender systems, gaming, etc. The growth of data volume and model complexity has led to increasing demands for deep learning accelerators. Graphics processing units (GPUs) have emerged as the most prevalent accelerators for deep learning computation. Prominent cloud platforms, such as Microsoft Azure Machine Learning [1] and Amazon SageMaker [2], provide GPU resources for deploying and executing DL workloads as a part of their Infrastructure as a Service (IaaS) [3]–[5].

To understand resource efficiency, it is crucial to monitor and forecast the hardware utilization of GPUs [6]–[9]. For resource providers, understanding the variation trend in GPU utilization allows them to make appropriate scheduling strategies to deploy co-located workloads efficiently on proper hardware instances [7], [10]. For users, since GPU rental can be costly, estimating the GPU utilization of their DL models can aid in selecting a cost-efficient GPU resource. When users employ automated machine learning (AutoML) tools, it is also beneficial to take GPU utilization into account for better hyperparameter tuning and neural architecture search [11]–[14]. For developers, extracting the GPU utilization patterns of different DL models can help design resource-efficient DL systems [15], [16].

In order to portray the patterns of GPU utilization of DL models, employing a quantitative utilization model for predictions proves to be a more time-efficient and resource-saving approach compared to actual execution and profiling [6], [7], especially when a variety of DL model architectures and GPU types need to be evaluated [17]. However, GPU utilization modeling faces two primary challenges. The first challenge involves the readily available GPU utilization metric, NVIDIA Management Library (NVML) utilization, typically acquired through the *nvidia-smi* command. This metric tends to overestimate actual GPU utilization and lacks generalizability to unseen DL model architectures. NVML utilization is defined as the "percent of time over the past sample period during which one or more kernels was executing on the GPU" [18] and does not accurately represent the finer-grained runtime status within GPUs. For instance, a GPU kernel (i.e., a compiled routine designed to execute on GPUs) with a continuous single-thread run will result in 100% NVML utilization, although a large portion of hardware resources are actually idle. The second challenge is caused by the vast architecture and hyperparameter configuration space of DL models, complicating the design of an accurate ML-based prediction model from limited samples. It is difficult to generalize to unseen DL models, even if they use the same operators as seen models [13], [19], [20]. For example, despite both BERT [21] and GPT [22] being transformer-based, using the training data from a BERT model to accurately predict the performance of a GPT model is still challenging.

In this paper, we propose DNN-occu, a tool that predicts the *GPU occupancy* of DL models before job execution using Graph Neural Network (GNN). This tool provides insightful guidance for subsequent scheduling or decision-making. DNN-occu addresses the above-mentioned two challenges as follows. Firstly, instead of NVML utilization, DNN-occu predicts the GPU occupancy of DL models, which is a fine-

*Hengquan Mei and Huaizhi Qu contributed equally to this work.
†Corresponding author.

grained GPU utilization metric. The definition of streaming multiprocessor occupancy is "the ratio of active warps on a streaming multiprocessor (SM) to the maximum number of active warps supported by the SM." [23] The average occupancy across all SMs represents the overall occupancy of the entire GPU. Such a metric is much more precise in portraying the portion of occupied GPU computation units than NVML utilization. Because obtaining exact occupancy needs costly runtime profiling, a quick prediction of GPU occupancy is necessary for guiding better scheduling. Secondly, since the architectures of DL models evolve quickly, it is impractical to collect all DL models' runtime profiling data exhaustively. Therefore, DNN-occu leverages a representative set of DL models to build a prediction model and extrapolates the GPU occupancy of unseen DL model architectures. To achieve this goal, DNN-occu adopts a unified representation for all DL models, regarding each of them as a directed acyclic graph. Each node in the graph represents a computational operator (e.g., 2D convolution), and an edge denotes the data flow between two nodes. Building upon this data representation and feature engineering, we design a novel attention-based GNN architecture with a transformer encoder to fit the mapping between DL model architectures, the order of kernel execution, runtime factors, and the associated GPU occupancy.

In addition, we conduct a trace-driven simulation of co-location DL workload scheduling to demonstrate the application of the predicted GPU occupancy from DNN-occu. Although co-location could improve GPU utilization, it also introduces performance interference. Based on the predicted GPU occupancy of unseen DL models from DNN-occu, our scheduler determines an appropriate combination of co-location jobs to reduce interference. Our approach avoids profiling kernel patterns, modifying the underlying DL framework, and tracing job execution online (which requires GPU isolation at the scheduler runtime). All of these tasks are expensive and time-consuming. Experimental findings demonstrate that GPU occupancy outperforms NVML utilization as a more effective guiding metric for downstream co-location tasks [6], [7]. Furthermore, our approach facilitates hyperparameter selection and improves the performance of DL models.

We list the main contributions of this paper as follows:

- We propose DNN-occu, a tool designed to predict the GPU occupancy of DL models accurately during inference. DNN-occu effectively learns the representation of different DL model architectures and demonstrates a promising generalization ability to unseen models.
- We design a set of node and edge features to represent GPU runtime factors. We also propose an effective GNN-based model to map these factors to the corresponding GPU occupancy. The prediction targets cover a broad range of DL models, including CNN-based, RNN-based, and Transformer-based models. DNN-occu provides fine-grained forecasts of hardware resource usage by DL models.
- We experimentally demonstrate that, when integrated into a simulated DL job co-location scheduler, the prediction engine DNN-occu reduces the performance interference caused by GPU over-allocation. This leads to up to a 31.45% increase in overall GPU utilization and a 19.71% reduction in makespan compared to existing approaches.

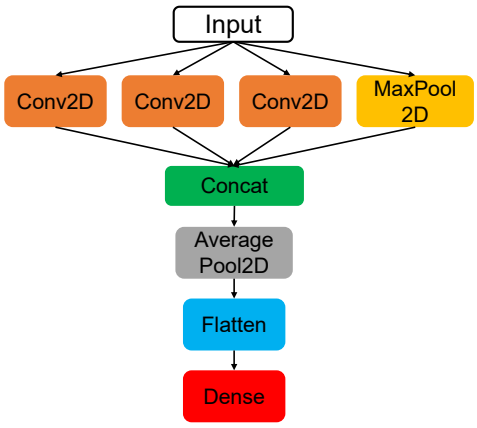


Fig. 1. Computation graph of an example DL model.

II. Background

A. DL Models and Computation Graphs

Fig. 1 shows the computation graph of an example DL model in popular deep learning frameworks, such as PyTorch [24] and TensorFlow [25]. A DL model is represented by a directed acyclic graph, also known as a computation graph. A node in the graph represents a tensor computation operator (e.g., `Conv2d`), and an edge denotes the data flow between two nodes. Each operator invokes specific APIs from GPU-accelerated libraries (e.g., NVIDIA cuDNN [26]), and the library executes a sequence of kernels on GPUs to perform the computation.

B. GPU Occupancy

The occupancy of a streaming multiprocessor is "the ratio of active warps on a streaming multiprocessor (SM) to the maximum number of active warps supported by the SM." [23] GPU occupancy is the average occupancy across all SMs. It gives a fine-grained and low-level measurement of GPU utilization. In practice, GPU occupancy is sampled by hardware performance counters on each warp scheduler of SMs. GPU occupancy is available in Nsight Compute [27] by the *ncu* command. However, collecting such a metric is both time-consuming and resource-consuming. Therefore, it motivates us to design a model for efficiently predicting the GPU occupancy of DL models rather than relying on costly profiling.

NVIDIA Management Library (NVML) utilization is another metric for GPU utilization, which is often easily obtained. It is defined as the "percent of time over the past sample period during which one or more kernels was executing on the GPU." [18] Users acquire this metric from the *nvidia-smi* (NVIDIA System Management Interface) command, which invokes the backend NVML API. Such a high-level utilization

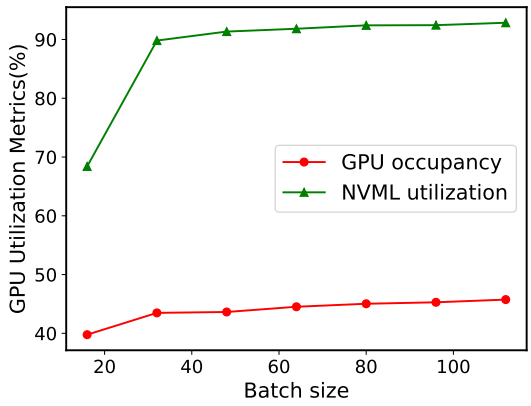


Fig. 2. Comparison of GPU occupancy and NVML utilization.

metric is pretty coarse and cannot measure how many SMs are actually in use.

Fig. 2 shows the difference between the two metrics of GPU occupancy and NVML utilization when training ResNet-50 [28] on the CIFAR10 dataset [29] and an NVIDIA A100. The y-axis value of each point in Fig. 2 is the average metric value weighted by the kernels' duration. With the increase in the batch size, NVML utilization reaches around 90%, while GPU occupancy is about 45%. In general, NVML utilization provides a relaxed upper bound of GPU utilization; therefore, GPU occupancy is much more precise in measuring the actual usage of hardware resources, revealing more potential for performance improvement.

C. Graph Neural Networks

The graph is a commonly used data structure for representing elements and their dependencies. The family of graph neural networks (GNNs) has demonstrated effectiveness in various applications and domains [30]–[32], prompting the emergence of numerous methods aimed at accelerating GNN training [33]. The main objective of GNNs is to learn to structure the information representation as a graph. The learning process of GNNs consists of propagating information between the nodes and edges of the graph through multiple message-passing mechanisms, followed by aggregation. In each message-passing iteration, the representation of the nodes and edges are updated. The final learned embeddings are used for downstream tasks such as regression, classification, and link prediction.

III. DNN-occu Design

A. Problem Formulation of GPU Occupancy Prediction

Given a DL model, including model architecture $\mathcal{M}$, model configuration $\mathcal{C}_m$ (e.g., the batch size, input channel size, hidden size, etc.), and runtime configuration $\mathcal{C}_r$ (e.g., GPU floating-point operations per second (FLOPS), memory bandwidth of the device, etc.), DNN-occu is to find a function

$$f^* : (\mathcal{M}, (\mathcal{M} \,|\, \mathcal{C}_m) \times \mathcal{C}_r) \,|\, w^* \rightarrow occu$$

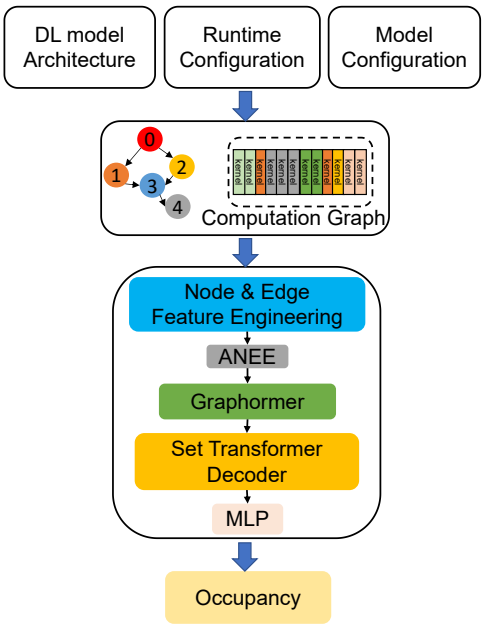


Fig. 3. Workflow of DNN-occu.

that predicts the GPU occupancy $occu$ for $\mathcal{M}$ during inference iterations of the input DL model, achieving a minimum empirical loss L between the predicted $occu$ and the measured occupancy $\hat{occu}$. The parameters of f^* are solved by:

$$w^* = \arg\min_w L(\hat{occu}, f((\mathcal{M}, (\mathcal{M} \,|\, \mathcal{C}_m) \times \mathcal{C}_r) \,|\, w)).$$

For a model $\mathcal{M}$ consisting of n kernels at runtime, DNN-occu provides a general form of occupancy predictions:

$$\text{occupancy} = \text{aggr}\left\{occu_1, occu_2, ..., occu_n\right\},$$

where $occu_i$ denotes the GPU occupancy of the i-th kernel. The aggregation function $\text{aggr}(\cdot)$ can be max, min, mean, etc.

In this paper, we choose *mean* as a representative aggregation function for studying. Average GPU occupancy usually reveals the resource usage of DL workloads throughout the life cycle and is an essential indicator to provide more insights for co-location DL workload scheduling [7], [34].

B. DNN-occu Overview

The overall workflow of DNN-occu is shown in Fig. 3. As introduced in Section II-A, a DL model is represented by a computation graph and is later transformed into a sequence of kernels executed on GPUs. Based on this fact, we consider the prediction of GPU occupancy as a mapping from a computation graph to an occupancy value. The workflow of DNN-occu mainly consists of four stages: (1) DNN-occu takes the DL model architecture, runtime configuration, and model configuration as input. We use Nsight Compute [27] to analyze the kernel and corresponding GPU occupancy, and we use ONNX [35] to obtain an intermediate representation of the input model. (2) We calculate the arithmetic average occupancy of the DL model according to the kernel. The

TABLE I
NODE AND EDGE FEATURES.

	Name	Description
Node Features	Operator Type	Type of the operator (e.g., `Conv2d`, `MaxPool2d`)
	Hyper-parameter	Type and value of each hyper-parameter of the operator (e.g., kernel size and channel size)
	Temporary Tensor Size	Sizes of temporary variables used by the operator
	Input/Output Tensor Size	Sizes of inputs, outputs tensor shape of the operator
	Operator FLOPs	Number of floating-point operations of the operator
	GPU FLOPS	GPU peak floating-point operations per second
	GPU Memory Capacity	Memory capacity of GPU
	SMs	Number of streaming multiprocessors
Edge Features	Edge Type	Type of the edge (e.g., Forward or Backward)
	Edge Tensor Size	Size of the delivered tensor
	Edge Bandwidth	Bandwidth for processing the delivered tensor

fused data contains complete dependency relations among occupancy data and the computation graph so that the subsequent supervised learning can be smoothly conducted. (3) DNN-occu adopts an effective GNN-based encoder to aggregate and update the features of the graph nodes over several rounds. Then the aggregated results are fed to an MLP [36]. (4) DNN-occu outputs the final prediction of GPU occupancy based on the MLP predictor.

C. Feature Engineering

For designing features, we carefully consider the underlying implementation of DL frameworks and ensure the features have precise computational semantics and a high correlation to GPU occupancy. Table I lists the adopted features for graph nodes and edges. These features are concatenated into a numerical feature vector for each node and edge. Categorical features, such as the operator type, are converted into numerical features by one-hot encoding.

The tensor size and number of floating-point operations (FLOPs) are calculated from the hyperparameter values of the DL model. We take the 2D convolution operator (denoted by `Conv2d`) as an example to illustrate how to compute them under forward propagation for inference. `Conv2d` receives an input feature mapping $N \times C \times H \times W$ (i.e., a batch of N input feature maps with the height and width $H \times W$ and C channels) and a set of convolutional filters $K \times C \times R \times S$ (K filters with the kernel size $R \times S$ and C channels). `Conv2d` generates an output feature map of size $N \times K \times P \times Q$ (with a batch of N output feature maps with the height and width $P \times Q$ and K channels). P and Q can be inferred from the input feature maps and filter shapes as well as from the additional stride and padding parameters [26]. Thus, the FLOPs of the `Conv2d` is calculated by $2 \times K \times C \times R \times S \times N \times P \times Q$.

Similarly, for operators in RNN-based models, we calculate the FLOPs based on their input and output tensor sizes. For Transformer-based models such as BERT, since they contain a large number of attention modules that are essentially generalized matrix multiplication (GEMM) operations, we calculate the FLOPs of GEMM as their features.

D. GNN-based Encoder

Transformer-based models have demonstrated excellent learning capabilities so far and are now widely adopted in computer vision and natural language processing. Models with transformer architecture [31], [37], [38] have further exhibited exceptional performance in a wide range of tasks on the graph. Motivated by these advances, our study aims to leverage the strong learning abilities to develop a model with solid generalization capabilities, allowing it to perform well on unseen input models. Consequently, we propose our model design, consisting of the ANEE (attention-based node-edge encoder) layer [13], Graphormer layer [31], and Set Transformer Decoder layer [32], which is shown in the middle part of Fig. 3.

The ANEE layer utilizes attention to encode the feature of input graphs efficiently. Suppose g is a computation graph, we denote a node as u and an edge as $l = (u_s, u_d)$. We further use h_u^i and e_l^i to represent the computed feature vectors of node u and edge l in the i-th round. First, the ANEE layer comprising parameter matrices W_u, W_e, and W_m computes the intermediate result of h_u^i (denoted by $\bar{h}_u^i$) as follows [13]:

$$\bar{h}_u^i = \text{LeakyReLU}\left(W_u \times h_u^{i-1}\right).$$

Next, the ANEE layer updates e_l^i of edge $l = (s, d)$ as follows:

$$e_l^i = \sigma\left(a^T \times \left(\bar{h}_s^i \,\|\, \bar{h}_d^i\right) \times W_e \times e_l^{i-1}\right),$$

where $a \in \mathbb{R}^{2\times N_1}$ is a weight vector and a^T is its transpose, and $\|$ is the vector concatenation operation. Then, the layer gathers the information of node u from its neighboring nodes and associated edges as follows:

$$f\left(u', l'\right) = \text{Softmax}\left(W_m \times e_{l'}^i\right) \times \bar{h}_{u'}^i,$$
$$h_u^i = \text{LeakyReLU}\left(\sum_{l'=(u',u)} f\left(u', l'\right)\right),$$

where u' is a neighbor node of u.

The encoded features are then fed forward to Graphormer layers. The Graphormer layer can be characterized by the layer normalization (LN), multi-head self-attention (MHA), and feedforward network (FFN) as follows:

$$\bar{h}_u^i = \text{MHA}\left(\text{LN}\left(h_u^{i-1}\right)\right) + h_u^{i-1},$$
$$h_u^i = \text{FFN}\left(\text{LN}\left(\bar{h}_u^i\right)\right) + \bar{h}_u^i.$$

To introduce Set Transformer Decoder, we first define Multihead Attention Block (MAB) on the input $X, Y \in \mathbb{R}^{n\times d}$ with respect to the previous denotation as follows:

$$\text{MAB}(X, Y) = \text{LN}(\bar{H}^i + \text{FFN}(\bar{H}^i)),$$

TABLE II
STATISTICS OF THE DATASETS AND MODELS.

○ CNN-based, △ RNN-based, □ Transformer-based	
Model Name	Variants
○ ConvNext	ConvNext-B[1]
○ ResNet	ResNet-18, ResNet-34, ResNet-50
○ VGG	VGG-11, VGG-13, VGG-16
○ AlexNet	Vanilla model
○ LeNet	Vanilla model
△ LSTM	Vanilla model
△ RNN	Vanilla model
□ ViT	ViT-S, ViT-T
□ Swin Transformer	Swin-S
□ MaxViT	MaxViT-T
□ BERT	DistilBERT
□ GPT-2	Vanilla model
□ CLIP	RN-50, ViT-B/32, ViT-B/16
Values of typical hyperparameters for the above models. CNN-based: batch size $\in \{4x \mid 4 \leq x \leq 32, x \in N\}$, input channel $\in [1, 10]$. RNN-based: batch size $\in \{8x \mid 16 \leq x \leq 64, x \in N\}$, sequence length $\in \{8x \mid 2 \leq x \leq 16, x \in N\}$. Transformer-based: batch size $\in \{4x \mid 4 \leq x \leq 32, x \in N\}$, input channel $\in [1, 10]$, sequence length $\in \{20, 512\}$.	

[1] B: base, S: small, T: tiny, RN50: ResNet-50

where

$$\bar{H}^i = \mathrm{LN}(X + \mathrm{MHA}(X, Y, Y)).$$

Using MAB, we define Set Attention Block as

$$\mathrm{SAB}(X) = \mathrm{MAB}(X, X).$$

Then, we define Pooling by Multihead Attention (PMA) with k learnable seed vectors $S \in \mathbb{R}^{k \times d}$ on a set of feature vectors $H^i \in \mathbb{R}^{n \times d}$ as

$$\mathrm{PMA}_k(H^i) = \mathrm{MAB}(S, \mathrm{FFN}(H^i)).$$

Finally, we define Set Transformer Decoder operating on features $H^i \in \mathbb{R}^{n \times d}$ as

$$\mathrm{Decoder}(H^i) = \mathrm{FFN}\left(\mathrm{SAB}\left(\mathrm{PMA}_k(H^i)\right)\right) \in \mathbb{R}^{k \times d}.$$

E. Loss Function

The GNN-based prediction of GPU occupancy can be formulated as a graph regression problem. We use the mean square error (MSE) as the loss function for this task:

$$L = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2,$$

where N is the number of models in the training dataset, $\hat{y_i}$ and y_i are the predicted and real GPU occupancy values of the i-th model configuration, respectively.

IV. EXPERIMENTAL SETUP

A. Datasets and Models

As shown in Table II, we select various representative DL models and use a stochastic strategy to generate model configurations based on the provided API. We create numerous data and models by refining each model architecture into several configurations by changing the values of hyperparameters such as the number of layers, batch size, and hidden size. For CNN-based models, we choose ConvNext [39], ResNet [28], VGG [40], AlexNet [41], and LeNet [42]. We consider the batch size (from 16 to 128 with step 4), input channel size (from 1 to 10 with step 1), input height (224), and input width (224) as the hyperparameters. For RNN-based models, we use LSTM [43] and RNN [36]. We use different domains of the batch size (from 128 to 512 with step 8) and sequence length (from 16 to 128 with step 8) as the hyperparameters. For Transformer-based models, we select ViT [44], Swin Transformer [45], MaxViT [46], BERT (distilbert-base-uncased-finetuned-sst-2-english) [47], GPT-2 [22], and multimodal model CLIP [48]. The hyperparameters include the batch size (from 16 to 128 with step 4), input channel size (from 1 to 10 with step 1), and sequence length (from 20 to 512). Other model configuration parameters are default. We run these models at maximum capacity on the machine until they encounter out-of-memory (OOM) errors. These models basically include typical model configurations in various real-world domains.

In total, our dataset contains 20 models with various operators (>30 types), from 13 to 2,664 nodes, and from 12 to 2,722 edges per computation graph. We employ ONNX [35] to generate DL model intermediate representation, so theoretically, DNN-occu can support more than 140 operators included in ONNX. Currently, the model configuration is deterministic, and there are no control flow operators (e.g., loops and conditional branches), so we assume the same execution flow and runtime performance across different inference iterations.

TABLE III
SYSTEM SETUP.

Specification	**System-1**	**System-2**	**System-3**
GPU Model	A100	RTX 2080Ti	P40
GPU Number	2	8	4
GPU Arch	Ampere	Turing	Tesla
GPU Memory	80 GB	11 GB	22.5 GB
CPU Core	12	28	24
CPU Model	Intel Xeon w5-2455X	Intel Xeon E5-2680 v4	Intel Xeon E5-2690 v4
CUDA	11.7	10.2	11.0
PyTorch	1.13.1	1.12.0	1.12.0
TensorFlow	1.15.0	1.15.0	1.15.0
Nsight Compute	2023.1.1	2019.5	2020.1

B. Hardware and Runtime Environment

Our experiments are conducted on three systems, and their setups are listed in Table III. Any other parameters keep the default settings.

C. Metrics

For evaluating GPU occupancy, we evaluate the prediction performance by Mean Square Error (MSE) and Mean Relative Error (MRE):

$$\mathrm{MRE} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{\hat{y}_i - y_i}{y_i} \right|, \qquad \mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2,$$

where N is the number of models in the test dataset, and $\hat{y}_i$, y_i are the predicted and real peak GPU occupancy of the i-th model, respectively. We chose these two metrics because they are widely used as standard measures of the accuracy of predictive models in classification.

D. Comparison Baselines

To compare with the GNN-based model of DNN-occu, we consider the following models as the baselines:

(1) MLP (multilayer perceptron) [36] is a foundational model in machine learning and serves as the basis for many complex neural network designs. It was employed to predict the execution time of operators [49]. Here, we adopt the same node features utilized by DNN-occu to calculate the MLP predictions.

(2) LSTM (long short-term memory) [50] is widely used in various applications, including natural language processing, time series analysis, and other applications where the input has a sequential structure.

(3) Transformer [51] is a well-known time series prediction model whose attention module can efficiently utilize the information on different time steps between series data.

(4) DNNPerf [13] is a runtime performance prediction tool for deep learning models. It adopts a GNN-based ANEE layer and systematically explores performance-related features derived from the semantics of the computation graph and hidden factors within the framework.

(5) BRP-NAS [20] adopts a graph convolutional network for predicting model latency in neural architecture search tasks. It focuses on modeling the impact from the computation graph structure while overlooking runtime factors associated with nodes and edges.

We have implemented MLP, LSTM, Transformer, DNNPerf, and BRP-NAS with PyTorch 1.12.0 [24] and DGL 1.0.1 [52]. After tuning these models, we choose the following hyperparameter values. For MLP, we use four layers whose widths are set to 80, 512, 512, and 256, respectively. For LSTM, we use two layers, each having 256 channels. For Transformer, we only use the encoder part. We use three encoder layers, and each layer consists of four attention heads and a 512-channel FFN. For DNNPerf and BRP-NAS, the learning rate and weight_decay are 0.0001, the same as other baselines.

V. Evalution

To assess the prediction accuracy of DNN-occu, we employ it to predict GPU occupancy during inference for the DL models listed in Table II. We divide 80% of the total dataset (ViT-T, LSTM, RNN, ResNet-34, ResNet-18, VGG-16, VGG-13, VGG-11, AlexNet, and LeNet) to the training dataset and allocate the remainder to the test dataset. These models are referred to as *seen* test models. Furthermore, we create a separate test dataset for ViT-S, BERT, ConvNext-B, and ResNet-50, whose models are *unseen* by DNN-occu. That is, the training dataset does not include any of their configurations.

DNN-occu uses one ANEE layer, two Graphormer layers, and two Set Transformer Decoder layers. We set the hidden dimension of all the layers to 256; the learning rate and weight decay are both set to 0.0001. We use the Adam [53] optimizer with default hyperparameters to train DNN-occu.

A. How Effective is DNN-occu?

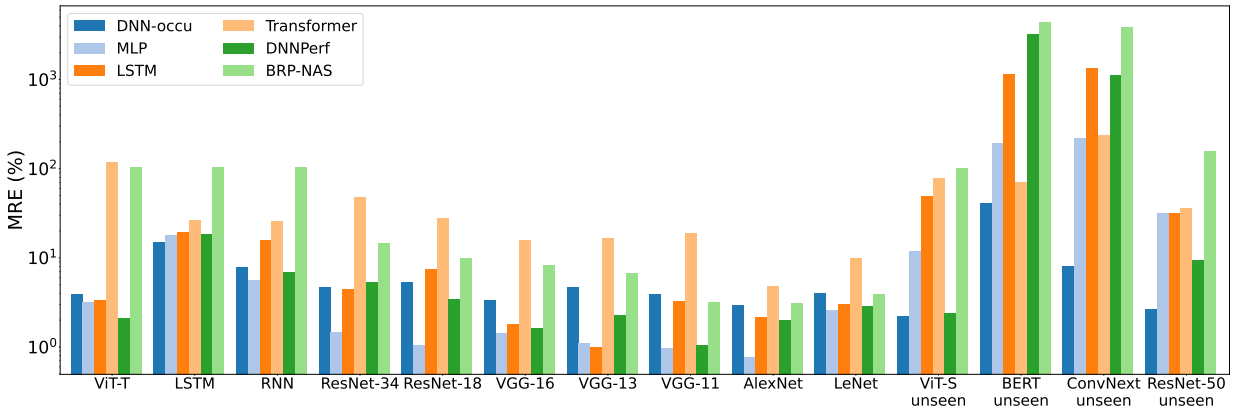
1) Results of predicting GPU occupancy for a single model: In this section, we compare DNN-occu with five baselines on the same test dataset to evaluate the effectiveness of predicting GPU occupancy for a single DL model that runs exclusively on a GPU. Fig. 4 shows the results of our experiments involving all the compared baselines for predicting GPU occupancy.

Fig. 4 (a) shows the prediction results of DNN-occu on NVIDIA A100 for various models. While MLP performs well on seen test models, its predictions on unseen test models are poor, with an MRE/MSE of 90.435%/0.721. Such poor performance is attributed to the overfitting of MLP and its lack of strong generalization ability. On the seen test models, the difference between the predictions of DNN-occu and all the baselines is insignificant. In contrast, on unseen test models, DNN-occu achieves the best MRE/MSE values of 5.496%/0.003, demonstrating a notable generalization ability.

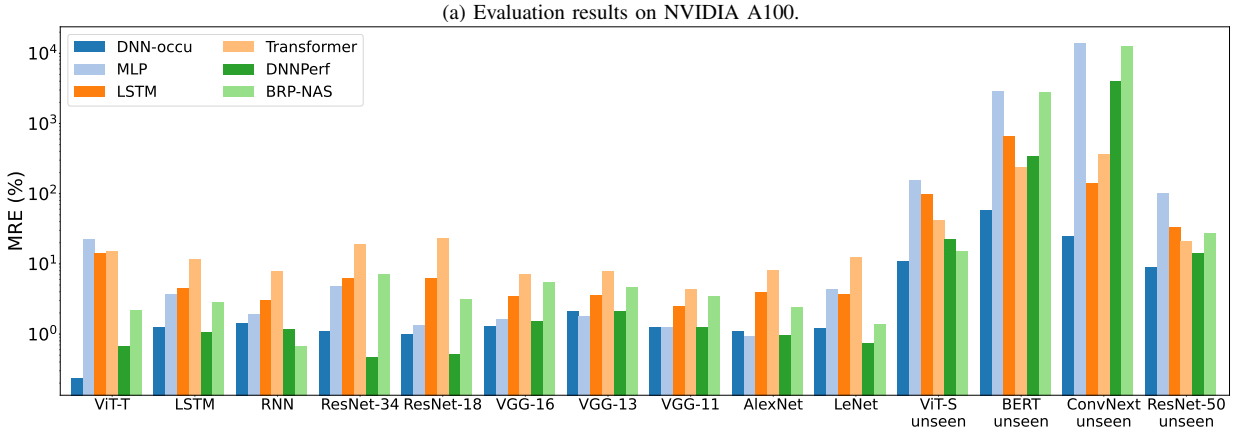
Fig. 4 (b) and Fig. 4 (c) show the prediction results of DNN-occu on different devices for the tested models. Once again, DNN-occu achieves the optimal predictions on the unseen test models, surpassing all the baselines on NVIDIA RTX 2080Ti and P40. The experimental results confirm the effectiveness of DNN-occu across various models and devices, demonstrating superior extensible-model and extensible-device generalization. The advantage of DNN-occu is its ability to learn the characteristics, trends, and development patterns of GPU occupancy changes from the DL model computation graph, model configuration, and runtime configuration. This ability enables flexible prediction of future changes in GPU occupancy.

2) Results of predicting GPU occupancy for multimodal models: Multimodal models usually consist of multiple input modalities (e.g., images, text, and speech), so their analysis and prediction are much more complex than unimodal models. The GNN, a deep learning model for graph-structured data, can process multimodal data and discover underlying relationships and patterns. We model multimodal data separately as independent graph structures and utilize DNN-occu for learning, fusion, and prediction. DNN-occu employs the neural architecture to model the consistency and correlation between these modalities and obtain better multimodal data representation and feature extraction.

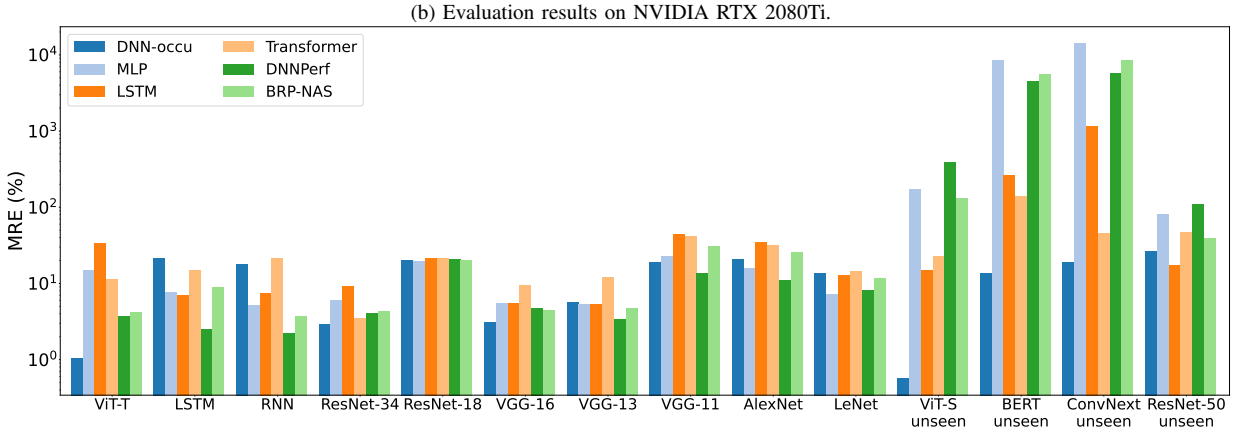
We take the multimodal model CLIP as an example and run both language and image encoders simultaneously. DNN-occu constructs multimodal graphs for learning the relationships between nodes and edges. This method can efficiently process multimodal data and effectively discover relationships between different modalities. Predicting the multimodal models GPU occupancy helps select optimal hyperparameters and their values (such as the batch size). As demonstrated in Table



(a) Evaluation results on NVIDIA A100.



(b) Evaluation results on NVIDIA RTX 2080Ti.



(c) Evaluation results on NVIDIA P40.

Fig. 4. Evaluation results of GPU occupancy prediction.

IV, when compared to other baseline methods, DNN-occu can effectively learn performance patterns, fuse computation graphs, and make predictions on multimodal models. The experimental results illustrate the advantages of GNN-based DNN-occu prediction and the key importance of GNNs in predicting the runtime performance of DL models.

B. How Robust is DNN-occu?

Our dataset contains a variety of neural architectures whose numbers of the nodes and edges are widely distributed. We split the dataset into several subsets based on the value ranges of the numbers, perform experiments on each subset, and present the prediction results in Fig. 5. Moving from left to right in the figure, the experimental devices are NVIDIA A100, RTX 2080Ti, and P40, respectively. DNN-occu achieves a satisfactory level of accuracy and outperforms all the baseline methods. The MRE values of DNN-occu for different numbers of nodes range from 2.940% to 5.014% for A100 predictions, from 0.233% to 2.095% for RTX 2080Ti predictions, and from 1.046% to 21.382% for P40 predictions, respectively. The MRE values of DNN-occu for different numbers of edges range from 2.197% to 41.298% for A100 predictions,

TABLE IV
PREDICTION OF GPU OCCUPANCY ON THE MULTIMODAL MODEL CLIP.

Model Name	Prediction of GPU occupancy on multimodal model on NVIDIA A100			
	Metric	DNN-occu	DNNPerf	BRP-NAS
RN50 (seen)	MRE (%)	**3.686**	637.369	108.611
ViT-B/16 (seen)	MRE (%)	**1.829**	924.529	144.702
ViT-B/32 (unseen)	MRE (%)	**3.614**	937.356	134.573
Model Name	**Prediction of GPU occupancy on multimodal model on NVIDIA P40**			
	Metric	**DNN-occu**	**DNNPerf**	**BRP-NAS**
RN50 (seen)	MRE (%)	**11.724**	112.863	173.318
ViT-B/16 (seen)	MRE (%)	**3.395**	115.226	175.242
ViT-B/32 (unseen)	MRE (%)	**2.73**	115.108	175.333

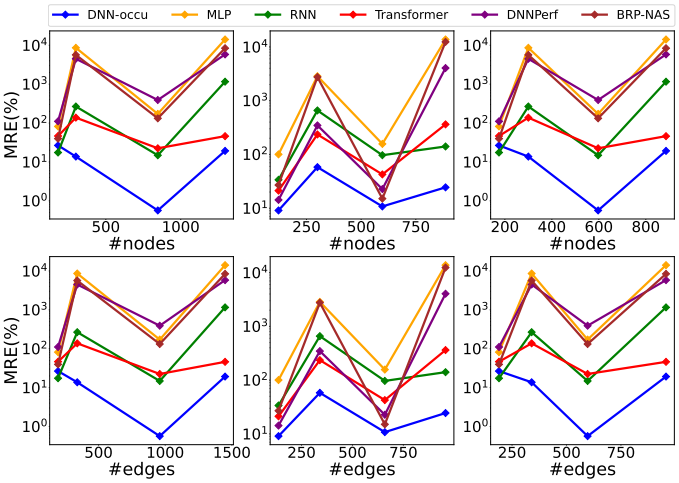


Fig. 5. Robustness evaluation across various numbers of nodes and edges. Moving from left to right, the experimental devices are NVIDIA A100, RTX 2080Ti, and P40, respectively.

from 9.090% to 58.080% for RTX 2080Ti predictions, and from 0.567% to 26.652% for P40 predictions, respectively. The experimental results demonstrate that DNN-occu exhibits satisfactory robustness across various neural architectures.

C. How Generalizable is DNN-occu?

Generalization is a significant concern in the field of deep learning. For a DL model, we expect it to perform well not only on known (seen) datasets but also on unknown (unseen) datasets, i.e., with remarkable generalization ability. Therefore, we evaluate the generalization ability of DNN-occu through extensive experiments.

We utilize ViT-T with the attention operator as a training dataset to assess the generalization of DNN-occu for Swin Transformer, MaxViT, ViT-S, BERT, and GPT-2. As shown in Table V, DNN-occu achieves an MRE of 9.829% for Swin Transformer, 8.075% for MaxViT, 5.239% for ViT-S, 8.470% for BERT, and 185.817% for GPT-2 on NVIDIA A100, surpassing the optimal MRE values of the baselines. Furthermore, on NVIDIA RTX 2080Ti and P40, the predictions of DNN-occu also outperform all the baselines. The experimental results demonstrate that DNN-occu exhibits a strong generalization ability to unseen models.

TABLE V
GENERALIZATION ON TRANSFORMER-BASED MODELS.

Model Name	Generalization on NVIDIA A100			
	Metric	DNN-occu	DNNPerf	BRP-NAS
Swin Transformer	MRE (%)	**9.829**	735.787	235.174
MaxViT	MRE (%)	**8.075**	175636.913	68579.639
ViT-S	MRE (%)	**5.239**	163.795	118.171
BERT	MRE (%)	**8.470**	1025.604	581.469
GPT-2	MRE (%)	**185.817**	901.541	421.053
Model Name	**Generalization on NVIDIA RTX 2080Ti**			
	Metric	**DNN-occu**	**DNNPerf**	**BRP-NAS**
Swin Transformer	MRE (%)	**8.032**	115.171	129.084
MaxViT	MRE (%)	**10.445**	742606.716	19475.475
ViT-S	MRE (%)	**7.459**	104.621	107.621
BERT	MRE (%)	**9.071**	760.725	470.512
GPT-2	MRE (%)	**36.220**	103.071	216.373
Model Name	**Generalization on NVIDIA P40**			
	Metric	**DNN-occu**	**DNNPerf**	**BRP-NAS**
Swin Transformer	MRE (%)	**8.883**	363.801	142.618
MaxViT	MRE (%)	**9.750**	49309.784	37711.313
ViT-S	MRE (%)	**0.624**	126.345	108.217
BERT	MRE (%)	**7.126**	764.709	531.211
GPT-2	MRE (%)	**69.318**	473.288	246.846

VI. APPLICATION CASES OF DNN-OCCU

In this section, we investigate the benefits of DNN-occu in supporting downstream DL workload scheduling. As case studies, we use two representative tasks: hyperparameter optimization and schedule guidance. In addition, DNN-occu can be adopted in other applications, such as power management and GPU kernel scheduling, which are of interest to our future work.

A. Hyperparameter Optimization

The general utilization of GPUs in deep learning systems tends to be relatively modest. According to a recent production study [54], the target system's average GPU utilization was about 52%, implying that nearly half of the GPUs remained idle. By analyzing and forecasting the GPU occupancy in deep learning models, DNN-occu helps tune critical parameters capable of affecting GPU occupancy and enhancing the program's overall GPU utilization.

Fig. 6 illustrates the GPU occupancy and NVML utilization under different batch sizes on NVIDIA A100. By utilizing DNN-occu, GPU occupancy can be accurately predicted across various hyperparameters and their values before job execution. Therefore, DNN-occu helps identify the values of hyperparameters that achieve maximum GPU occupancy without time-consuming runtime profiling, thereby accelerating hyperparameter optimization and boosting developer productivity. We notice that GPU occupancy is always lower than NVML utilization, confirming that GPU occupancy serves as a tighter upper bound for GPU utilization. This also indicates that, as the batch size increases, other bottlenecks emerge that hinder further improvement of GPU occupancy.

B. Schedule Guidance

Co-location execution means that applications with complementary resource requirements can be placed on the same GPUs, resulting in improved aggregated throughput. Due to

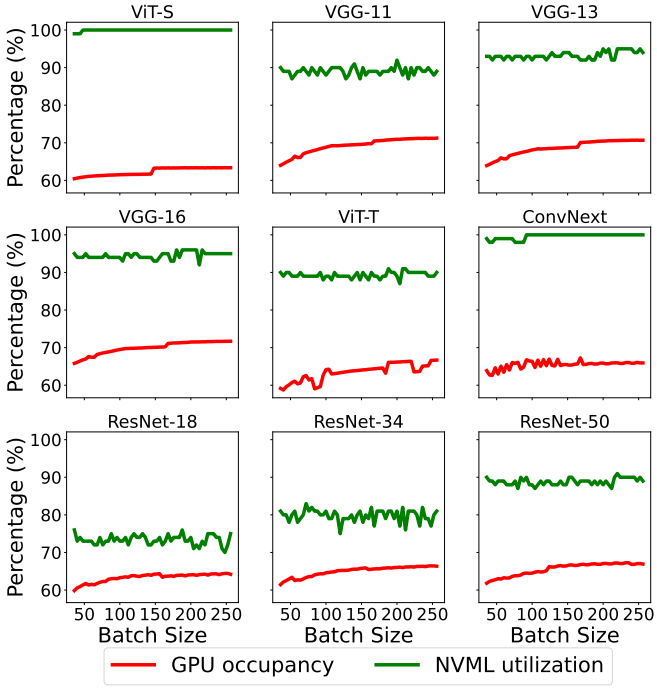


Fig. 6. Impact of the batch size on GPU occupancy and NVML utilization.

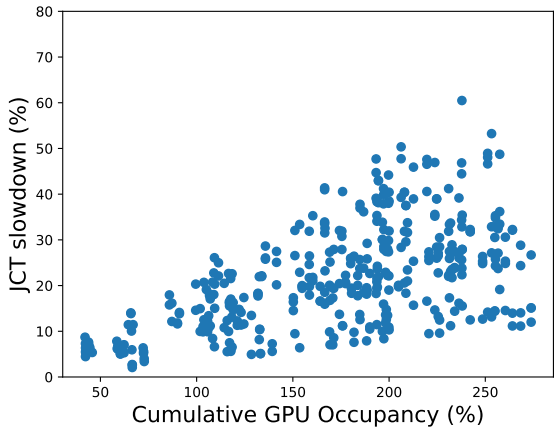


Fig. 7. Correlation of JCT slowdown and GPU occupancy

the contention of shared resources (e.g., cache capacity and global memory bandwidth), co-location may impact the *job completion time* (JCT) of applications when compared to applications running separately [7], [34]. To improve the results of these interference studies, we conducted a preliminary JCT interference analysis. This analysis involved 200 randomly selected combinations of co-location DL model executions from Table II, each of which was run 100 times. Fig. 7 illustrates that the rise in JCT is positively correlated with cumulative GPU occupancy, leading to a JCT rise ranging from 10% to 60% for each DL model.

Therefore, a safe co-location policy is necessary to achieve better SM allocation and avoid overprovision while still ensuring QoS and fair sharing. GPU occupancy reflects the actual SM usage at a fine-grained level and is an essential performance indicator for a secure co-location policy. Fig. 7 indicates that the JCT slowdown starts to rise dramatically, especially when the cumulative occupancy exceeds 100%. Therefore, an appropriate packing policy for multiple jobs should ensure the cumulative occupancy is at most 100% [7].

TABLE VI
ANALYSIS OF THE PACKAGING STRATEGY OF DL SCHEDULER

Pack Strategy	Makespan(s)		NVML utilization (%)	
	Avg	Gain	Avg	Gain
occu-packing	**106.87**	**19.71%**	**61.23**	**31.45%**
nvml-util-packing	126.58	4.90%	46.48	-0.21%
slot-packing (disabling co-location)	133.10	N/A[1]	46.58	N/A

[1] As the baseline, it has no gain and is represented as N/A.

We integrate DNN-occu into a DL scheduler deployed on a machine with four NVIDIA P40 GPUs and one Intel Xeon E5-2690 v4 processor. The scheduler uses a GPU-occupancy-based bin-packing algorithm (with co-location ensuring GPU occupancy does not exceed 100%; denoted by occu-packing) to maximize GPU utilization, while minimizing GPU over-allocation and the need for job resubmission caused by out-of-memory failures. First, before submitting DL workloads, the resource manager derives the corresponding computational graph from the DL model and adapts the input of DNN-occu. DNN-occu predicts the GPU occupancy of the unexecuted DL workloads. Then, the system uses the predicted value as a metric for co-location scheduling. The system combines the resource requirements of the DL workload (i.e., GPU occupancy) and the current GPU usage on the nodes, and schedules DL models to the appropriate node based on the cumulative GPU occupancy of the DL load not exceeding 100%. We compare our approach with slot-packing (disabling co-location) and NVML-utilization-based bin-packing [6], [7] (denoted by nvml-util-packing). The experiments are conducted on a mix of DL workload samples from Table II. Each experiment simulates the scheduling of different DL workloads into GPUs using the three scheduling approach above, and each approach is run 100 times with workload combinations scaled from prior work [11], [12].

Table VI summarizes that our occu-packing approach achieves an average NVML utilization of 61.23% compared to slot-packing (46.58%) and nvml-util-packing (46.48%). This is because GPU occupancy is a much more precise measurement of actual GPU usage compared to the coarse-grained NVML utilization. It provides a tighter resource upper bound for packing jobs, therefore the occu-packing strategy can make more aggressive packing with little interference. In all experimental runs, our approach achieves the lowest makespan, with a 19.71% improvement over the slot-packing approach. It also shows that the overhead of DNN-occu is tiny, and the inference time does not affect the model's ready scheduling wait at all.

VII. Related Work

GPU profiling. Regarding GPU utilization analysis, Gandiva [11] focuses on time-sharing and utilizes online profiling on isolated machines to determine proper co-location and migration strategies. Antman [15] leverages GPU SM utilization to identify jobs that might be suitable for co-location. The work of Yeung et al. [6] predicts coarse-grained GPU utilization based on FLOPs, input data size, and the number of convolutional layers. Moneo [8] intelligently collects SM utilization in real time at a finer granularity without detecting or tracking workloads. Prior work deals with coarse-grained GPU utilization, while we concentrate on GPU occupancy, which better reflects the actual usage of GPU resources.

Performance prediction for DL models. Performance modeling and prediction for DL models have recently attracted researchers' interest. Paleo [19] predicts the execution time of DL models based on FLOPs. Habitat [55] uses wave scaling (a technique for executing models on GPUs) and pre-trained multilayer perceptrons to make predictions by scaling the execution time of each operation during training iterations from one GPU to another. However, these analytical techniques require extensive handcrafted efforts and are specific to certain tasks. Some research work [13], [20], [56] proposes GNN-based timing predictors, which mainly focus on the performance impacts from operator type and computation graph structure. Compared with prior works, DNN-occu captures not only features at the operator level but also computation graph information and hidden factors within the framework. Our prediction-based learning method reduces handcrafted operations and achieves better prediction on unseen models. We apply DNN-occu to downstream scheduling tasks, demonstrating a practical application of performance prediction.

Workload scheduling. There are some work considering DL workload scheduling on GPU clusters. Horus [7] proposes a prediction-based interference-aware mechanism to determine good decisions for DL job placement. Instead of predicting GPU utilization directly, Themis [34] predicts the slowdown of each job, given k jobs sharing the GPU execution. The prediction model is a simple MLP, but the experiments are all based on GPGPU-Sim [57], which also requires a minor modification of the GPU structure to obtain feature counts. This means that the work cannot be implemented on a real GPU. Abacus [58] considers a more fine-grained scheduling approach, treating DL jobs as a series of operators and predicting the latency of a group of operators while running DL jobs simultaneously. The prediction model is also a simple MLP. This approach lacks extrapolation capability for unseen models and handles only a limited number of models.

VIII. Conclusion

In this paper, we propose DNN-occu, a tool for predicting runtime GPU occupancy of deep learning models. We design a GNN-based prediction model that utilizes elaborate features derived from the computation graph semantics within profiling statistics and runtime libraries. Our extensive experiments demonstrate that DNN-occu accurately predicts the GPU occupancy of deep learning models during inference. DNN-occu is practical, robust, and generalized across various hyperparameter values and neural architectures. We apply DNN-occu to hyperparameter optimization and employ it to guide co-location packing, effectively reducing the makespan of deep learning workloads and improving the overall GPU utilization of systems.

IX. Acknowledgment

This work is funded by Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS).

References

[1] Microsoft, "Microsoft azure machine learning," https://azure.microsoft.com/en-us/services/machine-learning-service, 2023.

[2] Amazon, "Amazon sagemaker," https://aws.amazon.com/sagemaker, 2023.

[3] Microsoft, "Azure vm sizes - gpu - azure virtual machines," https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-gpu, 2023.

[4] Google, "Cloud gpus — google cloud," https://cloud.google.com/gpu, 2023.

[5] Amazon, "Amazon ec2 p3 ideal for machine learning and hpc - aws," https://aws.amazon.com/ec2/instance-types/p3/, 2023.

[6] G. Yeung, D. Borowiec, A. Friday, R. Harper, and P. Garraghan, "Towards GPU utilization prediction for cloud deep learning," in *12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, July 13-14, 2020*, A. Phanishayee and R. Stutsman, Eds. USENIX Association, 2020.

[7] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-aware and prediction-based scheduling in deep learning systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 1, pp. 88–100, 2022.

[8] Y. Jiang, Y. Xiong, L. Qu, C. Luo, C. Tian, P. Cheng, and Y. Xiong, "Moneo: Monitoring fine-grained metrics nonintrusively in AI infrastructure," *ACM SIGOPS Oper. Syst. Rev.*, vol. 56, no. 1, pp. 18–25, 2022.

[9] W. Chen, C. Lu, K. Ye, Y. Wang, and C. Xu, "RPTCN: resource prediction for high-dynamic workloads in clouds based on deep learning," in *IEEE International Conference on Cluster Computing, CLUSTER 2021, Portland, OR, USA, September 7-10, 2021*. IEEE, 2021, pp. 59–69.

[10] S. Kim and Y. Kim, "Co-scheml: Interference-aware container co-scheduling scheme using machine learning application profiles for GPU clusters," in *IEEE International Conference on Cluster Computing, CLUSTER 2020, Kobe, Japan, September 14-17, 2020*. IEEE, 2020, pp. 104–108.

[11] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, A. C. Arpaci-Dusseau and G. Voelker, Eds. USENIX Association, 2018, pp. 595–610.

[12] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. H. Liu, and C. Guo, "Tiresias: A GPU cluster manager for distributed deep learning," in *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, J. R. Lorch and M. Yu, Eds. USENIX Association, 2019, pp. 485–500.

[13] Y. Gao, X. Gu, H. Zhang, H. Lin, and M. Yang, "Runtime performance prediction for deep learning models with graph neural network," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2023, pp. 368–380.

[14] Y. Gao, Y. Zhu, H. Zhang, H. Lin, and M. Yang, "Resource-guided configuration space reduction for deep learning models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 175–187.

[15] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "Antman: Dynamic scaling on GPU clusters for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 2020, pp. 533–548.

[16] Y. Wang, G. Wei, and D. Brooks, "A systematic methodology for analysis of deep learning hardware and software platforms," in *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, Eds. mlsys.org, 2020.

[17] Q. Yao, M. Wang, H. J. Escalante, I. Guyon, Y. Hu, Y. Li, W. Tu, Q. Yang, and Y. Yu, "Taking human out of learning applications: A survey on automated machine learning," *CoRR*, vol. abs/1810.13306, 2018.

[18] NVIDIA, "nvmlutilization," https://docs.nvidia.com/deploy/nvml-api/structnvmlUtilization__t.html#structnvmlUtilization__t, 2022.

[19] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[20] L. Dudziak, T. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "BRP-NAS: prediction-based NAS using gcns," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[21] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186.

[22] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[23] NVIDIA, "Achieved occupancy," https://docs.nvidia.com/gameworks/content/developertools/desktop/analysis/report/cudaexperiments/kernellevel/achievedoccupancy.htm, 2015.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035.

[25] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, K. Keeton and T. Roscoe, Eds. USENIX Association, 2016, pp. 265–283.

[26] NVIDIA, "Nvidia cuda® deep neural network library," https://developer.nvidia.com/rdp/cudnn-archive, 2023.

[27] NVIDIA, "Nvdia nsight compute," https://docs.nvidia.com/nsight-compute/NsightCompute/index.html, 2023.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.

[29] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[30] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[31] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[32] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 3744–3753.

[33] Q. Sun, Y. Liu, H. Yang, R. Zhang, M. Dun, M. Li, X. Liu, W. Xiao, Y. Li, Z. Luan *et al.*, "Cognn: efficient scheduling for concurrent gnn training on gpus," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15.

[34] W. Zhao, Q. Chen, H. Lin, J. Zhang, J. Leng, C. Li, W. Zheng, L. Li, and M. Guo, "Themis: Predicting and reining in application-level slowdown on spatial multitasking gpus," in *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*. IEEE, 2019, pp. 653–663.

[35] O. R. developers, "Onnx," https://github.com/onnx/onnx, 2022.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[37] D. Q. Nguyen, T. D. Nguyen, and D. Phung, "Universal Self-Attention Network for Graph Classification," *arXiv preprint arXiv:1909.11855*, 2019.

[38] J. Zhang, H. Zhang, C. Xia, and L. Sun, "Graph-bert: Only attention is needed for learning graph representations," *arXiv preprint arXiv:2001.05140*, 2020.

[39] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1106–1114.

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[43] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 802–810.

[44] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[45] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 9992–10 002.

[46] Z. Tu, H. Talebi, H. Zhang, F. Yang, P. Milanfar, A. C. Bovik, and Y. Li, "Maxvit: Multi-axis vision transformer," in *Computer Vision - ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXIV*, ser. Lecture Notes in Computer Science, S. Avidan, G. J. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., vol. 13684. Springer, 2022, pp. 459–479.

[47] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019.

[48] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[49] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*, N. Abe, H. Liu, C. Pu, X. Hu, N. K. Ahmed,

M. Qiao, Y. Song, D. Kossmann, B. Liu, K. Lee, J. Tang, J. He, and J. S. Saltz, Eds. IEEE, 2018, pp. 3873–3882.

[50] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008.

[52] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, and Z. Zhang, "Deep graph library: Towards efficient and scalable deep learning on graphs," *CoRR*, vol. abs/1909.01315, 2019.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[54] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, D. Malkhi and D. Tsafrir, Eds. USENIX Association, 2019, pp. 947–960.

[55] G. X. Yu, Y. Gao, P. Golikov, and G. Pekhimenko, "Habitat: A runtime-based computational performance predictor for deep neural network training," in *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, I. Calciu and G. Kuenning, Eds. USENIX Association, 2021, pp. 503–521.

[56] K. P. Selvam and M. Brorsson, "DIPPM: a deep learning inference performance predictive model using graph neural networks," *CoRR*, vol. abs/2303.11733, 2023.

[57] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2009, April 26-28, 2009, Boston, Massachusetts, USA, Proceedings*. IEEE Computer Society, 2009, pp. 163–174.

[58] W. Cui, H. Zhao, Q. Chen, N. Zheng, J. Leng, J. Zhao, Z. Song, T. Ma, Y. Yang, C. Li, and M. Guo, "Enable simultaneous dnn services based on deterministic operator overlap and precise latency prediction," in *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.