PrecisionProbe: Non-intrusive Performance Analysis Tool for Deep Learning Recommendation Models

Weiyu Peng, Jinghao Wang, Tianyu Wo, Renyu Yang
School of Software, Beihang University
{pengwy, wang_jinghao, woty, renyuyang}@buaa.edu.cn

***Abstract*—Deep learning recommendation models (DLRM) exploit user behaviors such as clicks, browse footprints, preferences, etc. for improved personalized experiences. However, in the face of the exponential growth of user data, such models require increasing GPU resources that are unaffordable and insufficient in a computing cluster. To improve GPU utilization and facilitate the advances of GPU scheduling algorithms, we present PrecisionProbe, a non-intrusive monitoring and analysis tool that can run upon Kubernetes and conduct sophisticated analytics of GPU resource utilization without altering the existing training code. PrecisionProbe captures fine-grained GPU metrics at the level of individual model layers and allows for a precise understanding of resource consumption patterns by exploring such detailed metrics. The mechanism is crucial for devising effective GPU scheduling algorithms, particularly tailored for DLRM training jobs dependent upon consumption patterns. Experimental results show that the recommendation models, as opposed to CV and NLP models, utilize less FP32 processing but have higher memory interaction frequencies. These findings indicate the unique resource needs of recommendation systems and necessitate the need of performance analytic using PrecisionProbe.**

***Index Terms*—Deep Recommendation Training, Kubernetes, Performance Analysis, Cloud Computing**

I. Introduction

Deep learning recommendation models (DLRMs) are the key driving force behind many application domains including e-commerce, online search, video, and advertising, etc. These systems can help to alleviate information overload, aid users in finding relevant information, and boost traffic and revenue for service providers. Personalized recommendations offer valuable insights into user behavior, and thus can enable precise decision-making and strategy optimization.

Despite the abundance of CPU and memory resources, GPU accelerators remain the critical bottleneck of training deep learning models [1]. The increasing data volumes and complexity of deep recommendation models are facing non-negligible challenges of resource management in a GPU cluster. In reality, These models often involve complex datasets and large embedding layers[2], and hence require substantial memory and hinder data processing. Improving cluster efficiency and accessibility is imperative for making effective resource management and job scheduling.

In this paper, we introduce PrecisionProbe, a non-intrusive software tool for comprehensive performance analysis of DLRMs to navigate the complexity of model training and resource management. At the core of PrecisionProbe is capturing and exploring fine-grained GPU metrics at the model layer level, without altering the underlying deep learning framework. Based on the performance tracing and profiling, we present an in-depth analysis of GPU resource consumption patterns across different layers within deep recommendation models. Doing so can dig out the GPU resource requirements of large embedding layers and unleash the potential of pattern-aware GPU scheduling. We conduct an experimental study to compare deep recommendation models with other models from other domains, such as computer vision (CV) and natural language processing (NLP). The results reveal the detailed memory and bandwidth-intensive characteristics of DLRMs and identify potential bottlenecks of training DLRMs.

II. Background and Motivation

The increasing volume of data and continuous advancements in deep learning models have introduced significant challenges in the allocation of computational resources. As a result, the efficient management and scheduling of these computational tasks have become crucial. A recent survey on deep learning workload scheduling [1] highlights several key objectives for scheduling algorithms, with the analysis of resource consumption being particularly critical. Therefore, there is an urgent need for a fine-grained analysis of GPU resource utilization.

GPU performance analysis has been widely studied in various computing domains [3], [4], categorized into micro-level and macro-level analysis, each offering different insights into GPU performance. Micro-analysis tools are utilized to dissect GPU performance at the hardware level during program execution. These tools often operate within the context of program execution and are based on sampling techniques. Additionally, there are methodologies designed to provide a top-down analysis of performance for GPU-accelerated jobs [5]. While these approaches facilitate a granular examination of GPU resource consumption patterns, they typically necessitate intrusive code modifications. Moreover, the analysis can be quite complex, with a primary focus on hardware-level details.

Macro-analysis tools, such as the roofline model [6], evaluate the theoretical performance limits of applications on a given platform, identifying performance bottlenecks. While widely used for analyzing deep learning models, the roofline model is limited to calculating performance upper bounds and does not provide detailed metrics on resource utilization during

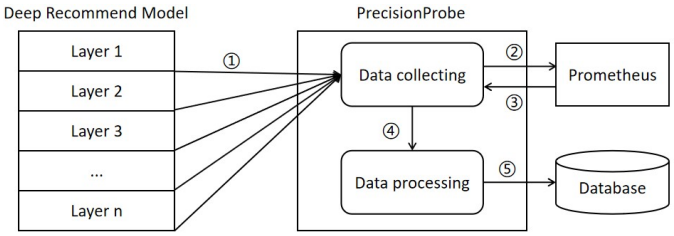


Figure 1: Overview of PrecisionProbe

training. Extensive research on GPU utilization during deep learning model training [7] has focused on models like VGG, ResNet, and LSTM on heterogeneous GPUs. However, many studies are outdated, based on older GPU generations, and few offer comprehensive analyses of GPU resource consumption patterns in deep recommendation model training.

To address these limitations, we introduce PrecisionProbe, a tool for analyzing GPU resource consumption patterns in deep recommendation models. PrecisionProbe provides precise measurements of key GPU usage metrics throughout the training process, aiding the development of GPU scheduling algorithms. We collect comprehensive, non-intrusive data on critical GPU resources, including Streaming Multiprocessors (SM) occupancy and floating-point (FP32) operations. Our experiments reveal unique GPU resource consumption patterns in deep recommendation models, facilitating the design of more effective GPU scheduling algorithms.

III. Our Approach

A. Overview

PrecisionProbe is a non-intrusive tool that captures detailed, fine-grained GPU metrics at the individual model layer level. It is designed to operate seamlessly on the Kubernetes platform, leveraging Prometheus for a comprehensive analysis of GPU resource utilization patterns. This sophisticated tool enables such analysis without the need for refactoring the underlying deep learning framework.

Figure 1 provides an overview of PrecisionProbe. Upon the completion of processing for each layer in the deep recommendation model, PyTorch Hooks registered after each layer transmit messages to PrecisionProbe. Subsequently, PrecisionProbe collects data via the HTTP API provided by Prometheus. The collected data are then processed and stored in the database.

B. Details

GPU utilization is a crucial metric that provides a snapshot of the GPU's operational status, but it can offer a misleading broad view. For instance, if one Streaming Multiprocessor (SM) within the GPU is active 100% of the time, the reported GPU utilization might incorrectly suggest full engagement, even if other SM units are idle. Relying solely on GPU utilization can thus lead to underutilization and inefficiency.

To address this, PrecisionProbe examines multiple GPU resources, selecting key metrics that together offer a more nuanced and comprehensive view of how deep learning models utilize GPU resources. Table I presents the metrics collected by PrecisionProbe from DCGM(NVIDIA Data Center GPU Manager), offering detailed insights into GPU performance during training:

Table I: Metrics collected by the system.

Metric
GPUUtil, GPUMem, DRAMActive, FP32Active, SMActive, SMOccupancy

Table II: Result Table Definition

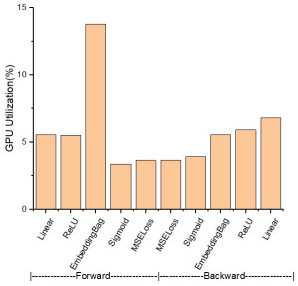
Field	**Type**	**Description**
job_id	varchar(500)	unique identifier of the job
host_name	varchar(100)	host name of the job
ip	varchar(100)	IP address of the host
forward	bool	forward inference flag
layer	varchar(100)	the name of the layer
cpu_num	float	job's requested CPU cores num
data_num	bigint	the number of collected data
gpu_util	float	average GPU Utilization
gpu_mem	float	average GPU memory usage
dram_active	float	average DRAM active rate
fp32_active	float	average FP32 active rate
sm_active	float	average SM active rate
sm_occupancy	float	average SM occupancy rate

- **GPUUtil**: Indicates GPU utilization, showing the percentage of active kernel functions.
- **GPUMem**: Represents GPU memory usage, related to the model's parameter size and data volume.
- **DRAMActive**: Reflects the ratio of cycles where the device memory interface is active.
- **FP32Active**: Shows the ratio of cycles where the FP32 pipeline is active.
- **SMActive**: Indicates the ratio of cycles where an SM has at least one active warp.
- **SMOccupancy**: Describes the ratio of warps residing on a single SM.

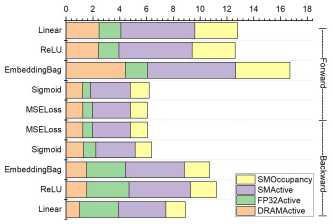
PrecisionProbe operates as a non-intrusive tool that doesn't require modifications to deep recommendation models' frameworks. It provides an HTTP interface to externally access the model's current operational status, enabling the extraction of real-time data on model execution. Key fields in the request body detail the model's execution status, including training progress, job tracking, propagation phase, layer information, and timestamps for layer execution.

To implement this, PyTorch Hooks are registered before and after targeted layers to capture performance data at critical junctures. As the model undergoes each forward and backward propagation cycle, these Hooks trigger PrecisionProbe to retrieve detailed information on GPU resource utilization, ensuring a thorough and uninterrupted assessment of GPU resources throughout the training phases.

Upon receiving requests, PrecisionProbe retrieves six key metrics from Prometheus based on timestamps. The retrieved data, along with task-specific information such as task name, Pod name, GPU UUID, epoch, batch, and layer numbers, are stored in a database. Given that PyTorch Hooks are invoked



(a) An overview of GPU utilization in each layer in DLRM.



(b) Fine-grained performance analysis for each layer in DLRM.

Figure 2: Performance analysis of DLRM.

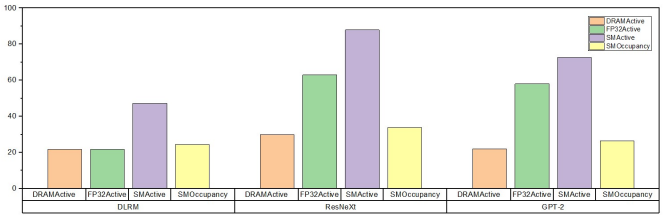


Figure 3: Comparison between models

Table III: Overview of GPU resource consumption

Model	Parameters	GPU Utilization	Memory Used
DLRM	514M	65.26%	24.87GB
ResNeXt	460M	94.92%	21.12GB
GPT-2	461M	85.8%	18.08GB

during each forward and backward propagation, data collection can be extensive. To mitigate data volume, PrecisionProbe first queries existing GPU metric results for the current layer. If the new results are similar to previous ones, they are not persisted. Only one data set per layer is stored, reducing data volume by approximately 90% and improving query efficiency. For comparative analysis, PrecisionProbe calculates and stores the average values of the metrics for each model layer in the result table (Table II). If no existing information is found for a specific layer, a new record is created and initialized. The metrics are updated incrementally to maintain accuracy and efficiency.

IV. Evaluation

A. Evaluation Setup

We deployed PrecisionProbe on a Kubernetes cluster configured with Prometheus 2.34.0, Kubernetes 1.26.9, Go 1.20, CUDA 12.2, and NVIDIA Driver 535.104.05. To analyze the GPU resource consumption of DLRM [2], we used a node equipped with an NVIDIA Tesla V100 32GB GPU and 256GB of memory. The model comprises a bottom MLP with three layers (512, 512, 64 nodes) and a top MLP with four layers (1024, 1024, 1024, 1 node), totaling 514.5M parameters. We used a dataset of 2048K samples in 1K mini-batches, with each sample having 8 categorical and 512 continuous features. Categorical features utilize an embedding table with 1M vectors of dimension 64, while continuous features form a vector of dimension 512. We compared DLRM's GPU resource consumption with ResNeXt [8] and GPT-2 [9] in the same environment.

B. Detailed Performance Analysis at Fine-Grained Level

We conducted a fine-grained performance analysis of DLRM, focusing on the Linear, ReLU, EmbeddingBag, Sigmoid, and MSELoss layers.

Figure 2a shows GPU utilization for each layer during forward and backward propagation. EmbeddingBag layers consume the most GPU resources during forward propagation due to large-scale sparse matrix computations. Sigmoid and MSELoss layers consume the least GPU resources as they do not require intensive GPU computations. Figure 2b provides detailed GPU resource utilization patterns, including DRAMActive, FP32Active, SMActive, and SMOccupancy metrics. EmbeddingBag layers show high SMActive and SMOccupancy, indicating substantial parallel computations and memory accesses. They also frequently transfer data between the GPU and DRAM, as shown by the DRAMActive metric. This is due to large-scale sparse matrix operations that require significant GPU resources.

During backward propagation, the high resource usage of EmbeddingBag layers is notable. This is likely due to gradient accumulation, which involves combining gradients of input samples and accessing embedding vectors for each sample.

C. GPU Resource Consumption Pattern

We tested ResNeXt and GPT-2 with PrecisionProbe to compare their GPU resource consumption with DLRM. Table III shows the models, their parameters, and overall GPU resource consumption. Models were adjusted to have similar parameter sizes and used GPUs with approximately 20GB of memory. DLRM has lower GPU utilization than ResNeXt and GPT-2. DLRM's large embedding layers process extensive categorical features, making it memory-capacity and bandwidth-intensive. Thus, at similar parameter levels, DLRM consumes more memory resources and relatively less GPU utilization.

Figure 3 shows the GPU resource utilization of DLRM, ResNeXt, and GPT-2. DLRM has similar DRAMActive levels as ResNeXt and GPT-2 but consumes less computational GPU resources, highlighting its memory-intensive nature. Memory operations like data transfer occupy a significant portion of DLRM's GPU usage. Insufficient memory can lead to underutilized GPU computing resources, causing an imbalance in GPU utilization.

V. Related Work

A. GPU Performance Analysis

GPU performance analysis can be categorized into microscopic, examining hardware events related to kernel invocations, and macroscopic, looking at overall system utilization.

Microscopic tools like HPCToolkit[10] analyze GPU performance by attributing metrics to calling contexts for both CPUs

and GPUs. It uses a wait-free data structure to monitor and attribute GPU performance, creating detailed call path profiles for GPU computations. [11] measures GPU kernel executions using hardware performance counters and collects call path traces. [5] extended it with the CUPTI interface for top-down performance analysis. Macroscopic tools like roofline[6] describe application performance by connecting processor performance to off-chip memory usage, measuring traffic between caches and memory, allows determination of DRAM bandwidth requirements based on operational intensity. Yang et al. proposed a new methodology to give a hierarchical roofline performance analysis for deep learning models[3]. The sparsity roofline reveals the intricate connection between the arrangement of sparse elements, precision, and the efficiency of inference[12].

However, these studies do not detail the entire process. Our PrecisionProbe provides a macroscopic view of GPU resource utilization during deep learning model training, capturing detailed metrics like SMOccupancy and FP32Active.

B. Performance Analysis of Deep Learning Models

Analyzing GPU utilization during deep learning training is crucial as models scale. [13] examined the impact of different GPU architectures on CNNs. [14] proposed GENIE to predict diverse deep learning workloads. [7] designed an interference-aware execution framework for GPU applications. [15] evaluated a deep learning application for statistical temperature downscaling. [16] proposed a framework to enhance the training speed of the Wide & Deep model.

These works focused on GPU utilization, memory usage, DRAM data transfer frequency, and network bandwidth. Similar to these studies, our PrecisionProbe focuses on GPU resource consumption patterns for deep learning recommendation models. We capture detailed GPU resource metrics such as SMOccupancy and FP32Active, tracking these metrics throughout the entire training process.

VI. Conclusion

We introduced PrecisionProbe, a non-intrusive tool for analyzing fine-grained GPU resource utilization during the training of deep learning recommendation models. The tool collects comprehensive metrics throughout each epoch, providing detailed insights into the GPU resource utilization patterns of each layer.Our testing of PrecisionProbe on the Deep Learning Recommendation Model revealed distinct consumption patterns across layers. Linear layers showed the highest GPU utilization, while MSELoss and Sigmoid layers had lower utilization. EmbeddingBag layers were particularly computation-intensive but engaged in fewer floating-point operations (FLOPs) compared to other layers.The detailed analysis of GPU resource utilization metrics provided by PrecisionProbe will aid future research and development of scheduling algorithms tailored for deep recommendation models.

Acknowledgment

This work is supported in part by National Key R&D Program of China (2022YFB4502003), by the Fundamental Research Funds for the Central Universities (501QYJC2023121001). For any correspondence, please refer to Renyu Yang (renyuyang@buaa.edu.cn).

References

[1] W. Gao, Q. Hu, Z. Ye, P. Sun, X. Wang, Y. Luo, T. Zhang, and Y. Wen, "Deep learning workload scheduling in gpu datacenters: Taxonomy, challenges and vision," *arXiv preprint arXiv:2205.11913*, 2022.

[2] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[3] C. Yang, Y. Wang, T. Kurth, S. Farrell, and S. Williams, "Hierarchical roofline performance analysis for deep learning applications," in *Intelligent Computing: Proceedings of the 2021 Computing Conference, Volume 2*. Springer, 2021, pp. 473–491.

[4] T. Miao, Q. Wu, T. Liu, P. Cui, R. Ren, Z. Li, and G. Xie, "Md-roofline: A training performance analysis model for distributed deep learning," in *2022 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2022, pp. 1–8.

[5] K. Zhou, M. W. Krentel, and J. Mellor-Crummey, "Tools for top-down performance analysis of gpu-accelerated applications," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[6] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[7] S. Kim and Y. Kim, "Interference-aware execution framework with coscheml on gpu clusters," *Cluster Computing*, vol. 26, no. 5, pp. 2577–2589, 2023.

[8] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[9] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[10] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "Hpctoolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[11] K. Zhou, L. Adhianto, J. Anderson, A. Cherian, D. Grubisic, M. Krentel, Y. Liu, X. Meng, and J. Mellor-Crummey, "Measurement and analysis of gpu-accelerated applications with hpctoolkit," *Parallel Computing*, vol. 108, p. 102837, 2021.

[12] C. Shinn, C. McCarthy, S. Muralidharan, M. Osama, and J. D. Owens, "The sparsity roofline: Understanding the hardware limits of sparse neural networks," *arXiv preprint arXiv:2310.00496*, 2023.

[13] S. Dong, X. Gong, Y. Sun, T. Baruah, and D. Kaeli, "Characterizing the microarchitectural implications of a convolutional neural network (cnn) execution on gpus," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 96–106.

[14] Z. Chen, W. Quan, M. Wen, J. Fang, J. Yu, C. Zhang, and L. Luo, "Deep learning research and development platform: Characterizing and scheduling with qos guarantees on gpu clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 34–50, 2019.

[15] S. Kum, S. Oh, J. Yeom, and J. Moon, "Optimization of edge resources for deep learning application with batch and model management," *Sensors*, vol. 22, no. 17, p. 6717, 2022.

[16] Y. Zhang, L. Chen, S. Yang, M. Yuan, H. Yi, J. Zhang, J. Wang, J. Dong, Y. Xu, Y. Song *et al.*, "Picasso: Unleashing the potential of gpu-centric training for wide-and-deep recommender systems," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 3453–3466.