Swift and Trustworthy Large-Scale GPU Simulation with Fine-Grained Error Modeling and Hierarchical Clustering

Euijun Chung
euijun@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Seonjin Na
seonjin.na@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Sung Ha Kang
kang@math.gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Hyesoon Kim
hyesoon@cc.gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Abstract

Kernel-level sampling is an effective technique for running large-scale GPU workloads on cycle-level simulators by selecting a representative subset of kernels, thereby significantly reducing simulation complexity and runtime. However, in large-scale GPU workloads, kernels often exhibit heterogeneous runtime behaviors where some identical kernels show fluctuating performance, while others display multiple performance saturation points. We find that the kernel execution time distribution serves as a powerful signature for addressing this complexity. By carefully analyzing execution time distributions, we show that heterogeneous kernels can be effectively classified and sampled, significantly reducing errors in sampled simulations.

In this paper, we propose STEM+ROOT, a fine-grained kernel-level sampling methodology that enables trustworthy sampled simulation by achieving minimal sampling error. STEM leverages the distribution of kernel execution times as a signature and applies statistical techniques to determine optimal sample sizes with tight error bounds. ROOT is a novel hierarchical clustering framework built on top of STEM that ensures the sampled kernels faithfully represent the entire workload in terms of both execution time and a wide range of microarchitectural metrics. STEM achieves high scalability for large-scale GPU workloads by significantly reducing offline profiling overhead for collecting kernel execution times. When evaluated on the latest GPU benchmark suite, our proposed methodology reduces sampling error by 27.6-81.9× and achieves 7-600× faster kernel profiling compared to existing approaches while achieving comparable performance.

CCS Concepts

• **Computing methodologies** → **Modeling and simulation**; • **Computer systems organization** → **Distributed architectures**; *Heterogeneous (hybrid) systems*; • **Mathematics of computing** → *Probabilistic representations*.

Keywords

GPU, Workload sampling, Simulation methodology

MICRO 2025, Seoul, South Korea

ACM ISBN 978-X-XXXX-XXXX-X/2025/XX
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Cycle-level simulations play a critical role in computer architecture research, enabling detailed evaluation of microarchitectural changes, design space exploration, power and energy estimation, and more [7, 18, 22]. While widely adopted tools such as AccelSim [15], MGPUSim [37], and MacSim [16] support cycle-level GPU simulation, the growing computational demands of modern machine learning (ML) workloads have made it increasingly challenging to simulate them effectively [25]. Our observations show that even a 1-second large language model (LLM) inference workload can require several days of simulation, as the simulator must update the microarchitectural state of the GPU at every cycle. Without effective optimization techniques, this challenge limits the practicality of simulation-based performance modeling and exacerbates the gap between workloads used in real GPU deployments and those used in simulators.

Workload sampling is a widely adopted technique for accelerating cycle-level simulations by reducing the workload size while preserving its unique runtime characteristics. This approach has been extensively studied in both CPU [3, 9, 33, 43] and GPU [2, 10, 21, 24] domains. The core idea is to divide the workload into multiple intervals and extract a *signature* from each interval that captures its microarchitectural runtime behavior. A subset of representative intervals is then selected for simulation, and the results are extrapolated to estimate the performance of the full workload. While selecting less representative intervals can compromise accuracy, it enables substantial reductions in simulation time compared to running the full workload.

Kernel-level sampling is widely used in GPU simulation to improve scalability, but the sampling error can be significant if the kernel signatures used to select representative kernels fail to capture their runtime behavior. Previous approaches have relied on various instruction-level and control-flow-related metrics as kernel signatures [2, 10, 21, 24]. Although these metrics capture the program characteristics related to control flow graphs or static code features, we observe that they often fail to capture input-dependent

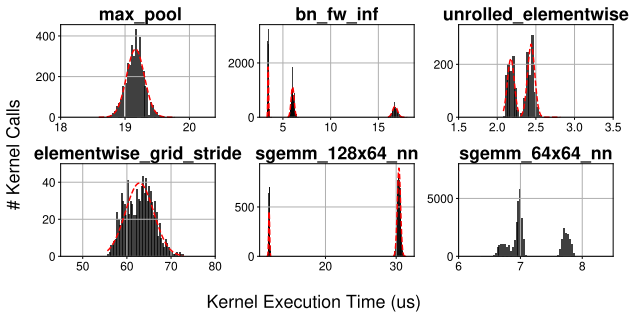

Figure 1: Execution time histograms of repeated GPU kernel calls from ML workloads (CASIO benchmark suite). Kernel names shown above each plot. Runtime heterogeneity observed in the execution times of repeatedly executed kernels.

characteristics at runtime, which are often a more significant source of variation in modern GPU applications. For example, even the same GPU kernel, such as `gemm` can be invoked repeatedly in a fixed compute graph but show significantly varying performance due to microarchitectural effects. Although kernel code and control flow remain constant, factors such as input sparsity, tensor layout, memory alignment, and cache locality can greatly affect execution efficiency. This highlights the need for a new kernel signature that can differentiate these performance-influencing factors.

We observe that ML workloads [6] on GPUs involve a substantial number of repeated kernel invocations [27, 30] due to extensive batching and layer-level iterations. However, the same kernel often exhibits heterogeneous runtime behaviors across invocations, which poses a significant challenge for kernel-level sampling since the samples need to capture all the different ways the kernel behaves during the workload. Surprisingly, we discovered that the distributions of kernel execution times offer powerful insights in categorizing these differences, enabling more accurate sampling strategies when used. For instance, if the execution time distribution of a kernel exhibits distinct peaks, this indicates the presence of multiple performance saturations–each peak reflecting the kernel's operation in a different context within a workload. In such cases, separate samples must be taken from each peak to accurately capture these distinct behaviors. Conversely, when the same kernel exhibits a large standard deviation in execution time, it indicates significant runtime variability–often caused by its memory-bound nature and fluctuating memory latencies. In these scenarios, a larger number of samples is required to fully capture the performance variability, ensuring statistically confident simulation results.

Based on these observations, we propose STEM+ROOT: a kernel-sampling technique that leverages the execution time distribution to extract microarchitectural insights from kernels and perform fine-grained sampling. Our approach uses hierarchical clustering and statistical error modeling to fully capture the runtime heterogeneity shown in GPU kernel execution profiles, thereby accurately characterizing the overall behavior of the workload for sampled simulation. STEM employs statistical error modeling to characterize each kernel's runtime behavior and determine the optimal sample size that balances accuracy and efficiency. ROOT refines the process by employing fine-grained hierarchical clustering to determine the optimal number of kernel clusters, selecting representative kernels that mirror the full workload's behavior in terms

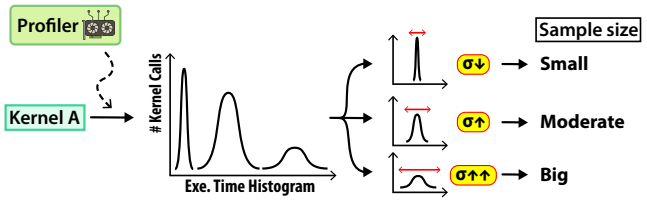

Figure 2: Motivation on execution-time-based kernel sampling. Kernels show wide variability and/or multiple peaks, requiring both fine-grained clustering and sampling for accurate sampled simulation.

of both execution time and microarchitectural metrics. Together, this approach delivers significant simulation acceleration while ensuring that the sampling error is minimized. Furthermore, because execution time data can be gathered with lightweight profiler [1, 29] and a near-linear algorithm, STEM scales effectively to large-scale workloads with millions of kernel invocations. Our evaluation on GPGPU benchmarks [4], ML benchmarks [6] and large-scale LLM/ML workloads [12] demonstrates that STEM+ROOT significantly reduces sampling error compared to prior methods, while achieving comparable speedups with substantially lower profiling overhead.

2 Observation and Motivation

Our work leverages the execution time distribution of GPU kernels to enable accurate kernel-level sampling. This section describes our observations and motivations on how execution time distributions provide valuable insights for kernel-level sampling and why this approach can be broadly generalized across various applications.

2.1 Heterogeneous runtime behavior of repeated GPU kernels

Observation 1: In large-scale GPU workloads with a massive number of repeated kernel invocations, identical GPU kernels often exhibit substantial variation in execution time across invocations. Figure 1 shows execution time histograms of several kernels sampled from ML workloads in CASIO suite [6]. Some histograms display widely or narrowly spread distributions, while others exhibit multiple distinct peaks. This variability is especially common in modern GPU workloads compiled from high-level frameworks such as PyTorch [30], where compute graphs translate into numerous kernel launches from a relatively small set of kernel types. Even for kernels like `sgemm` or `winograd`, launched with identical code and consistent parameters (e.g., grid size, block size, and instruction count), runtime behavior can vary greatly depending on the context in which the kernel is used and the specific input data it processes.

This runtime heterogeneity arises because the same kernel (e.g., `sgemm`) is invoked repeatedly in different contexts as compute graphs are translated into microarchitecture-specific GPU kernels. Although the kernel logic and launch configurations remain unchanged, each invocation often operates on different types of data (e.g., activations, weights, or biases in neural networks) residing in various memory regions such as global memory, shared memory, or L1/L2 cache. These differences in input characteristics and memory locality lead to diverse execution behaviors. Moreover, as shown

in the `sgemm_128x64` histogram in Figure 1, the presence of multiple narrow, distinct performance peaks suggests that the kernel is used in at least two different contexts within the workload. In such cases, these distinct behaviors should be treated separately during sampled simulation to ensure accuracy. However, relying solely on code-level analysis makes it difficult to capture this heterogeneity as such methods struggle to account for dynamic factors like memory access patterns, input data characteristics, and runtime dependencies that vary based on the kernel's execution context.

2.2 Extracting kernel's runtime diversity with exe. time distributions

Observation 2: Kernel execution time distribution is a powerful signature that can effectively reveal such heterogeneous runtime characteristics of kernels. The execution time reflects its usage context within a given workload, enabling us to differentiate identical kernels operating under different conditions. For example, as shown in Figure 1, the three clearly separated peaks in the histogram suggest that the same `bn_fw_inf` kernel is used in three different runtime contexts through tens of thousands of repeated kernel calls. This indicates that the kernel shows different execution time, likely due to different input or usage patterns within its workload context. By applying clustering methods to group kernels in each peak, we can easily classify the kernel's usage and take separate samples from each peak for accurate kernel-level sampling.

Moreover, statistical measures such as standard deviation offer valuable insights into kernel performance variability. For example, kernels with wide execution time distributions (e.g. `max_pool` in Figure 1) exhibit significant runtime jitter. This fluctuation is often due to the kernel's memory-bound nature and its sensitivity to microarchitectural factors. In such cases, a larger sample size is required to capture the full range of variability and ensure statistical confidence in sampled simulation results. In contrast, kernels like `sgemm_128x64_nn`, which show narrow peaks, suggest more stable performance. For such kernels, fewer samples per peak are sufficient to maintain high accuracy for sampled simulation. Note that maxpool and GEMM operations are generally known to be memory-bound and compute-bound in convolutional neural network (CNN) workloads, respectively.

Figure 2 summarizes our key insight on how we can leverage the execution time distributions to perform fine-grained clustering and sampling. As execution time profiles can simultaneously exhibit multiple performance peaks and large standard deviations at the same time, a solution that can address both dimensions is necessary. This motivates the design of STEM+ROOT, where ROOT clusters kernels based on execution profiles, and STEM dynamically adjusts sample sizes according to observed runtime variability.

2.3 Using execution time as a kernel signature for robust and accurate sampling

We claim that kernel execution time distributions and their derived statistical measures are robust signatures for kernel sampling, even across different GPU microarchitectures. While execution time is inherently hardware-dependent, the distribution of execution times yields meaningful, relatively hardware-agnostic insights. Statistical features such as standard deviation, coefficient of variation (CoV), and the number of peaks (as shown in Figure 1) capture important behavioral properties: such as memory or compute intensity, workload phase behavior, or input-dependent memory access patterns that are not tied to the absolute timing values.

Our work leverages these distribution-based features to guide sampling decisions, allowing it to remain effective even when the hardware changes. Instead of comparing kernels solely based on their static information, our algorithm *adaptively increases the sample size* for kernels that are more sensitive to microarchitectural changes (e.g., memory-bound kernels) based on their runtime statistics, making it more likely to capture diverse runtime behaviors even on new hardware or a system. We evaluate this claim in Sec. 4.5 by simulating design space exploration across architectures, and we observe consistently low error rates, often outperforming prior approaches that use microarchitecture-independent signatures. In Sec. 5.1, while we discuss the potential limitations of using execution times for kernel sampling, we argue that exploiting these statistics offers high achievements in robustness of our approach.

The rapid evolution of GPU architectures makes execution time a more practical sampling metric than unstable low-level parameters. In contrast to stable CPU ISAs, proprietary GPU ISAs change significantly with each generation, rendering kernel signatures based on instruction or basic-block counts unreliable. For example, features introduced in NVIDIA's Volta architecture, like Tensor Cores (for FP8/BF16) [23] and warp-level primitives (`vote`, `shfl`) [26, 32], can cause the same high-level source code to compile into drastically different machine code with new performance behaviors. Furthermore, recent studies show that GPUs even differ in how they inject register dependency chains, further undermining architectural consistency [11]. Consequently, execution time provides a more versatile and dependable signature for kernels.

3 STEM and ROOT methodology

STEM+ROOT is our kernel-level sampling solution that directly leverages execution time data to perform fine-grained clustering and sampling. Figure 3 illustrates an overview of our methodology. First, kernel calls are grouped by their names, as most large-scale GPU workloads typically consist of numerous repetitive invocations of the same kernel types. ROOT performs fine-grained hierarchical clustering to classify kernels with different runtime behaviors, as observed in Figure 1. STEM then applies precise statistical error modeling to determine the optimal number of samples from each cluster. This ensures minimal sampling error while achieving substantial simulation speedups. A key takeaway of our approach is that it can provide theoretical error bounds, offering both transparency and predictable accuracy for sampled simulation results.

Since STEM and ROOT are tightly integrated, we begin by describing the core methodology of STEM and subsequently illustrate how ROOT leverages it to perform fine-grained clustering in the following subsections.

3.1 Kernel-level sampling for GPU workloads

A GPU workload consists of many kernel invocations, often with repeated executions of the same kernels. Running a *full simulation* (simulating every kernel invocation in the workload without sampling) can be prohibitively time-consuming. To address this, kernel-level sampling selects a subset of kernel calls to simulate,

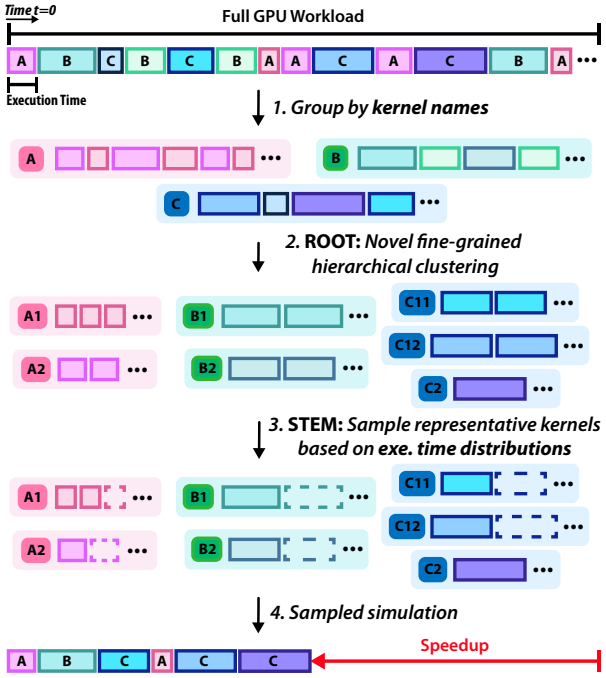

Figure 3: Overview of the proposed methodology. ROOT applies hierarchical clustering on same kernels by execution time to differentiate kernels with similar runtime behaviors. STEM selects adaptive number of representative samples from each cluster to enable fast and accurate simulation.

resulting in a so-called "sampled" simulation. The goal of sampled simulation is to carefully select kernels that capture the runtime characteristics of most workload phases, while minimizing the length of the sampled simulation to reduce overall time.

After the sampled simulation, the *total execution time* (execution time of the full simulation) can be estimated using a weighted sum. Specifically, the total time is computed as the sum of the execution times of the sampled kernels, each multiplied by a weight that corresponds to the number of corresponding kernel invocations in the full workload that the sample represents.

Let t^* denote the ground-truth total execution time of the full simulation, which is the value we aim to estimate. Let t_{total} represent the estimated execution time obtained from the sampled simulation, computed as a weighted sum over the sampled kernels. We define the **sampling error** e between the estimated and ground-truth total execution times as follows:

$$e \equiv \left| \frac{t_{\text{total}} - t^*}{t^*} \right| \times 100(\%). \tag{1}$$

We use this sampling error to quantify the accuracy of the sampled simulation relative to the full simulation.

3.2 STEM: Statistical Error Modeling for GPU simulation

STEM is our statistical error model that leverages the Central Limit Theorem (CLT) and KKT-solver to obtain the optimal sample sizes for a set of kernels. The summary of STEM is shown in the upper part of Figure 4. STEM demonstrates that leveraging execution time distribution for workload sampling provides significant advantages in terms of accuracy (Sec. 4.2), theoretical error bounds (Sec. 3.4), and low profiling overhead (Sec. 4.7).

We begin with the simplest case, where C is a set of invocations of the same kernel. The objective is to select a subset of these invocations for sampled simulation, but the key question is how many samples are required. Specifically, we aim to determine the minimal **sample size** m that ensures the sampling error remains within a bound of ϵ. The error bound ϵ is a tunable parameter that works as a desired upper bound on the theoretical sampling error, and can be set to values such as 1% or 5%. According to the Central Limit Theorem (CLT), the sample mean $\bar{X}$ will *always* follow a normal distribution, *regardless of the original distribution of execution times in* C. This powerful result holds under two key assumptions: (1) the sample size is sufficiently large (rule of thumb is $m \geq 30$), and (2) the samples are independent and identically distributed (i.i.d.) [38]. Fortunately, both conditions are satisfied in large-scale GPU workloads. The massive degree of kernel invocations ensures the first condition in most workloads, and the use of random sampling with replacement satisfies the i.i.d. assumption. As a result, the sample mean $\bar{X}$ of kernel execution times follows a normal distribution: $\bar{X} \sim \mathcal{N}(\mu, \sigma^2/m)$, where μ and σ^2 denote the true mean and variance of kernel execution times in C [17].

The objective of kernel-level sampling is to estimate the total execution time of GPU kernels using only a subset of sampled executions [20]. Our estimation for the total execution time is $t_{\text{total}} = |C| \cdot \bar{X}$, and its true value is $t^* = |C| \cdot \mu$. The sampling error, which is the error between t_{total} and t^* is as follows when the given confidence level is $1 - \alpha$:

$$e = \left| \frac{t_{\text{total}} - t^*}{t^*} \right| = \left| \frac{|C|\bar{X} - |C|\mu}{|C|\mu} \right| = \left| \frac{\mu \pm \frac{z_{1-\frac{\alpha}{2}}\sigma}{\sqrt{m}} - \mu}{\mu} \right| = \frac{z_{1-\frac{\alpha}{2}}\sigma}{\mu\sqrt{m}}. \tag{2}$$

$z_{1-\alpha/2}$ is a standard score when the confidence interval is $1 - \alpha$, and this value becomes 1.96 on 95% confidence level.

Therefore, the sample size m ensuring error smaller than the bound ($e \leq \epsilon$) can be obtained as follows, with a ceiling function for ensuring the m is an integer.

$$m = \left\lceil \left(\frac{z_{1-\alpha/2}}{\epsilon} \frac{\sigma}{\mu} \right)^2 \right\rceil \tag{3}$$

This equation represents the statistical error model for the simplest scenario where a single set of kernels is considered. A similar analysis can be found in prior statistical sampling studies [5, 43].

The beauty of STEM lies in its versatility, as it can be applied to *any set of kernels with arbitrary distributions* if the μ and the σ of the kernels are known. For example, no matter a kernel exhibits a narrow or broad execution time histogram, the same STEM equation (Eq. (3)) can be used to determine the optimal sample size m.

Equation (3) intuitively demonstrates that by leveraging the mean (μ) and standard deviation (σ) of kernel execution times, one can derive an optimal sample size. Specifically, kernels with *wide execution time distributions* will have *high* σ/μ *values* and this will hint STEM to determine *larger sample sizes*. This increased sample size lets the sampled simulation to capture most of the diverse runtime behaviors even if the different microarchitecture is used during the sampled simulation, thereby enhancing both the accuracy and effectiveness of kernel-level sampling.

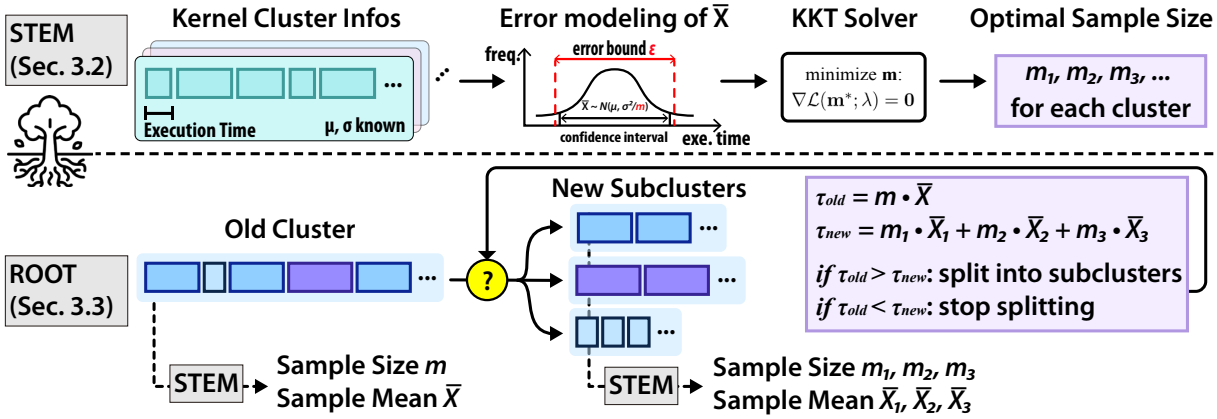

Figure 4: Overview of STEM (top) and ROOT (bottom). STEM estimates simulation error from kernel execution time distributions (using the sample mean, $\bar{X}$) and employs a KKT solver to minimize the sample size per cluster while meeting error bounds. ROOT leverages STEM to determine whether splitting kernel clusters will lead to additional simulation time savings.

We use the value σ/μ obtained from kernel-level profilers [1, 29], as a proxy for the true value, which is otherwise unobtainable without full simulation. This value, the CoV (coefficient of variation), represents the relative width of the execution time distribution. Although the exact values of σ and μ may vary across different simulation settings, the CoV (defined as their ratio) effectively reflects a kernel's inherent runtime behavior, such as whether it is memory-bound or highly sensitive to hardware changes.

3.3 Optimizing STEM for multiple clusters

In real-world GPU workloads, multiple kernel clusters appear due to the 1) usage of different kernels and 2) cases where the same kernel often showing multiple peaks in its execution time distribution. As a result, sampling typically requires selecting representatives from several clusters at once. We now consider optimizing STEM for such multi-cluster cases. While one could apply Eq. (3) independently to each cluster, this approach imposes strict error bounds on every cluster, often resulting in a larger total sample size than necessary. To address this, we introduce an additional optimization that jointly considers all clusters. This allows STEM+ROOT to reduce the required sample size by 2–3× on average, enabling faster simulations without compromising accuracy.

Let a set of kernel clusters as $\{C_0, C_1, ..., C_{k-1}\}$ and denote $N_i = |C_i|$ for convenience. Assume we sample $m_0, m_1, ..., m_{k-1}$ number of kernels from each corresponding cluster. Then, for any i in the range, C_i's estimated execution time, t_i, can be obtained as $t_i = N_i\bar{X}_i$ where $\bar{X}_i \sim \mathcal{N}(\mu_i, \sigma_i^2/m_i)$ by the CLT. Using the linear combination rule of normal random variables [35], the estimation for the total execution time t can be expressed as

$$t = \sum_{i=0}^{k-1} t_i = \sum_i N_i\bar{X}_i \sim \mathcal{N}(\sum_i N_i\mu_i, \sum_i N_i^2\frac{\sigma_i^2}{m_i}) = \mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2), \quad (4)$$

where $\tilde{\mu} \equiv \sum_i N_i\mu_i$ and $\tilde{\sigma} \equiv \sum_i N_i^2\sigma_i^2/m_i$ are used for brevity.

By using the same error equation as Eq. (2), the error bound inequality $e \le \epsilon$ becomes

$$\sum_i N_i^2\frac{\sigma_i^2}{m_i} \le \left(\frac{\epsilon}{z_{1-\alpha/2}}\sum_i N_i\mu_i\right)^2. \quad (5)$$

Since the goal of STEM is to minimize simulation time, we define τ as the total execution time of the samples, a proxy to the total simulation time. The optimal τ that satisfies the error constraint $e \le \epsilon$ can be obtained by solving the following non-linear minimization problem using a KKT solver.

Problem 1.

$$\underset{m_i}{\text{minimize}} \quad \tau = \sum_i m_i\mu_i$$

$$\text{subject to} \quad \sum_i N_i^2\frac{\sigma_i^2}{m_i} \le \left(\frac{\epsilon}{z_{1-\alpha/2}}\sum_i N_i\mu_i\right)^2$$

$$\text{and} \quad m_i > 0 \ \text{ for } \ \forall i \in \{0, ..., k-1\}.$$

Solution. Let $a_i \equiv \mu_i$, $b_i \equiv N_i^2\sigma_i^2$, and $c \equiv (\epsilon\sum_i N_i\mu_i/z_{1-\alpha/2})^2$ for brevity. We apply the Karush-Kuhn-Tucker (KKT) conditions to obtain the following solution:

$$m_i = \left\lceil \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \right\rceil \text{ for } \forall i \in \{0, ..., k-1\}, \quad (6)$$

where the ceiling function ensures integer m_i values, with minor sub-optimality. The detailed solution is shown in Sec. 8.1. □

Using this KKT Solver, we determine the optimal sample sizes for a given set of clusters. We define the solution in Equation (6) as STEM, an extended version of Equation (3) optimized for multiple kernel clusters.

3.4 ROOT: Fine-grained hierarchical GPU kernel clustering

ROOT is our novel fine-grained GPU kernel sampling methodology built upon STEM. Its primary objective is to differentiate distinct execution time peaks in invocations of identical kernels as illustrated in Figure 1 and 2. For instance, consider the `sgemm_128x64_nn` kernel shown in Figure 1. If one were to directly compute the optimal sample size using STEM over the entire execution time distribution, the resulting sample size would be overly large due to the high standard deviation introduced by multiple distinct peaks. However, by partitioning the cluster such that each cluster contains only a single peak, the standard deviation within each cluster is significantly

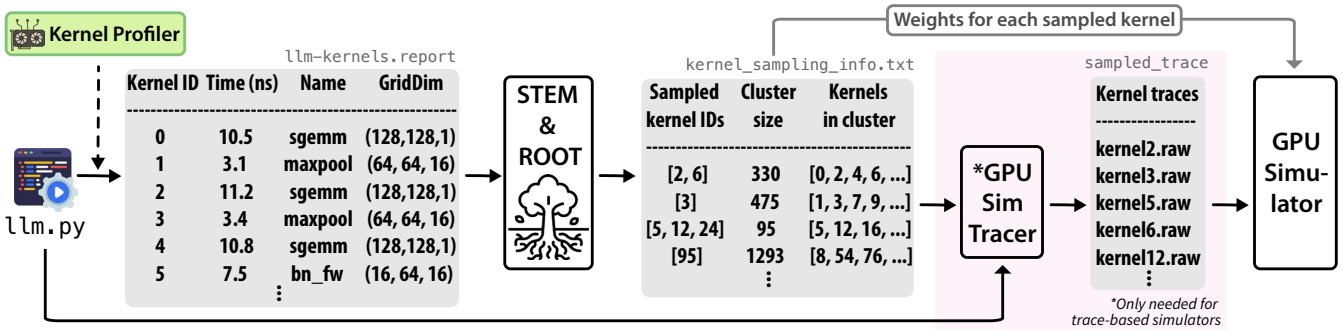

Figure 5: End-to-end pipeline of STEM's sampled simulation framework. Kernel profiler extracts execution time per kernel and STEM+ROOT creates sampling information based on it. GPU tracer and simulator use the information to run a sampled simulation.

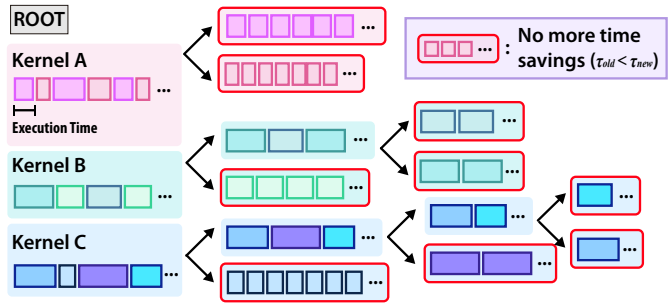

Figure 6: ROOT's recursive methodology. ROOT hierarchically splits kernel clusters until further simulation time savings are no longer possible.

reduced. Consequently, the optimal sample size decreases, leading to faster sampled simulation while maintaining high accuracy.

A key challenge in this approach is that the number of peaks (or runtime contexts) is unknown in advance, making it difficult to apply clustering methods like k-means as they require the number of clusters as input. To overcome this, ROOT applies clustering recursively: it continues splitting clusters until further splits no longer yield meaningful simulation time savings. This hierarchical process ensures that we isolate kernels with similar execution behavior while avoiding unnecessary over-partitioning. Figure 6 shows an example of ROOT performing hierarchical kernel clustering.

The bottom part of Figure 4 illustrates the branching condition used in ROOT's recursive algorithm. Given a kernel cluster C, we apply a clustering method (e.g., k-means) to divide it into subclusters $C_0, C_1, \ldots, C_{k-1}$. ROOT then uses STEM (Eq. 6) to estimate the simulation time before and after the split and compares the results. If sampling from the new subclusters $C_0, \ldots, C_{k-1}$ reduces total simulation time compared to using the old cluster C, ROOT accepts the split. This decision is made by comparing the total simulated time before and after the split, as shown in Equations (7) and (8).

$$\tau_{old} = m\bar{X} = \lceil (z_{1-\alpha/2}\sigma/\mu\epsilon)^2 \rceil \cdot \bar{X} \tag{7}$$

$$\tau_{new} = \sum_i m_i \bar{X}_i = \sum_i \left\lceil \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \right\rceil \cdot \bar{X}_i \tag{8}$$

If $\tau_{old} > \tau_{new}$, partitioning the old cluster C into new multiple subclusters $\{C_0, \ldots, C_{k-1}\}$ will reduce the overall simulation time. We recursively apply this decision process to achieve fine-grained kernel clustering with bounded error.

The following is a proof that any union of error-bounded cluster sets also maintains error-bounded. Applying Theorem 3.1 to each cluster set ensures that the total sampling error across all clusters remains bounded by ϵ.

Theorem 3.1. *Let $S^{(0)} = \{C_0^{(0)}, C_1^{(0)}, \cdots\}$, $S^{(1)} = \{C_0^{(1)}, \cdots\}$, $\cdots$, $S^{(N-1)} = \{C_0^{(N-1)}, \cdots\}$ be N sets of kernel clusters where the corresponding sampling error of each cluster set is bounded by ϵ with sample sizes $\{m_i^{(j)}\}$ for $S^{(j)}$. Then, the same set of sample sizes gives a bounded error for the union of every cluster set $\bigcup_{j=0}^{N-1} S^{(j)}$.*

Proof. The proof is found in Appendix 8.2. □

3.5 Running the sampled simulation

Once ROOT completes clustering the kernel invocations into subclusters, sampling with sample sizes m_i determined by STEM is performed. Random sampling with replacement is used to select the samples from each subcluster. This also satisfies the i.i.d. conditions required by the CLT. The sample sizes $\{m_i\}$ for each subcluster are computed according to Equations (3) and (6), ensuring that the sampling process adheres to the desired error bound ϵ.

Figure 5 describes the end-to-end pipeline of our kernel-level sampled simulation framework and illustrates how our STEM+ROOT method integrates with existing GPU simulators. The process begins with a GPU kernel profiler [1, 29] that collects execution time information for each kernel invocation. This data is then fed into the STEM+ROOT algorithm, which selects representative kernel samples and determines how many kernel invocations each sample should represent. The generated sampling information is then passed to a GPU simulator of choice [15, 16] along with the corresponding code or trace of the workload. For trace-based simulators, traces are generated only for the sampled kernels, significantly reducing trace generation overhead. This can be viewed as embedding the sampling information into the workload code or trace, allowing it to be reused across simulators with different configurations. The simulator uses the sampling information to compute a weighted sum of the sampled execution times, enabling accurate estimation of the total simulation time.

Table 1: Comparison of previous kernel-sampling methods.

Sampling Methods	PKA [2]	Sieve [24]	Photon [21]	STEM+ROOT (ours)
Kernel signature	12 instr. level metrics	Kernel name & Num. of instrs	GPU Basic Block Vector (BBV)	Kernel name & Exe. time distribution
Clustering	k-means	Hand-tuned, based on CoV (σ/μ)	Find a kernel with similar BBV and #warps (95% threshold)	Fine-grained hierarchical (ROOT)
Kernel sample size	Single per cluster, first chronological	Single per cluster, first chronological		Adaptive sampling with statistically determined sample size (STEM)
Profiling granularity	Instr. count and statistics *per warp*	Instr. count *per warp*	Basic block count *per warp*	Execution time *per kernel*
Scalability for large-scale workloads	Very low	Low	Low	High

Table 2: Workloads used in evaluation of STEM and baselines.

Benchmark Suites	Num. of workloads	Avg. Execution time (sec)*	Avg. number of kernel calls
Rodinia (GPGPU) [4]	13	6.46	1403
CASIO (ML) [6]	11	7.26	64279
Huggingface (LLM/ML) [12]	6	1835.27	11599870

*The execution time of workloads are measured on RTX 2080 GPU.

4 Evaluation

This section presents the evaluation results for STEM+ROOT, including its accuracy, performance, error bound sensitivity, validation on microarchitectural metrics, profiling overhead, and additional analyses on simulators and various hardware.

4.1 Experiment Setup

Experiment Environment. Our experiments are on a range of GPUs, including the NVIDIA H100, H200, and RTX 2080. The RTX 2080 was used for profiling experiments, as prior sampling techniques required over a month of exclusive system use, and it was the only machine available for that duration. We used Nsight Systems for kernel-level profiling, but our method applies to any GPU system that supports similar kernel profilers, such as NVIDIA, AMD, and Intel GPUs, or TPU/NPUs with similar support [1, 13, 14, 29].

Benchmark Suites. Three benchmark suites were used to evaluate our method in terms of speedup, accuracy, and scalability. We used Rodinia GPU Benchmark Suite 3.1 [4] for small-scale GPGPU workloads (input config. from the baseline work [24]), CASIO DL Suite [6] for state-of-the-art ML applications. Although some workloads in Rodinia suite has small number of kernel calls, it is presented as a reference for irregular and diverse GPGPU/HPC workloads. For large-scale workloads, we used a set of ML/LLM workloads using models from Huggingface repository [12]. The list of models includes `Bert`, `Bloom`, `DeiT`, `Gemma`, `GPT-2`, and `ResNet-50`, and the workloads involve generating 1000+ sentences or classifying 7,000+ images. CUDA version 12.6 is used.

Table 2 shows the summary of workloads including the execution time and number of kernel calls. The three suites will demonstrate our method's effectiveness across applications from small-scale (e.g., Rodinia, input size 1MB to 1000MB) to large-scale ML models (e.g., HuggingFace, model size 25M to 2B parameters). Table 2 highlights the massive number of kernel calls in the CASIO and HuggingFace suites, where STEM+ROOT is expected to fully leverage its statistical and fine-grained capabilities.

Speedup and error of sampled simulations. We define *speedup* as the ratio of the cycle count of the full workload to that of the sampled workload. Sampling error is computed using the definition shown in Eq. (1), comparing the full workload's cycle count and the estimate obtained from the sampled workload. In cases where running the full workload on a simulator was infeasible, we used cycle counts from machine profiles to compute the speedup and sampling error of the sampled simulations. In these experiments, perfect warmup of the GPU cache and GPU microarchitectural states is assumed, as the cycle counts are measured on real hardware and direct manipulation of the hardware was infeasible. See Sec. 5.2 for further discussion of how we acknowledge this limitation and estimate its potential impact.

Baseline Methods. As summarized in Table 1, we compare STEM against three kernel sampling baselines: PKA [2], Sieve [24], and Photon [21]. We used Nsight Compute (NCU) [28] to gather the instruction-level metrics required for PKA, and NVBit (NVIDIA Binary Instrumentation Tool) [40] to collect instruction counts for Sieve. For Photon, we built a BBV profiler based on their `instr_count_bb` example to extract GPU BBVs for each kernel, enabling compatibility with both trace-based simulators [15, 16] and execution-driven simulators [37].

Instruction-level profiling per warp introduces substantial performance overhead, making methods like PKA and Sieve impractical for large workloads, expected to take months to complete profiling Huggingface workloads. Moreover, Photon requires kernel processing time that grows quadratically with the number of kernel calls, making it infeasible for such workloads with over millions of kernels. We analyze and discuss the overhead of previous sampling methods in more detail in Sec. 4.7. Therefore, we set uniform random sampling, in which each kernel is independently selected with a 0.1% probability, as a baseline for HuggingFace workloads.

Replication & Hyperparameters. We repeated every experiment 10 times and averaged the results to minimize the randomness. The harmonic mean was used for speedup [8], while the arithmetic mean was used for sampling error. We set the error bound ϵ to 0.05 and used $k = 2$ for k-means clustering used in each iterative step of ROOT. A z-score of 1.96 was used for $z_{1-\alpha/2}$ (95% confidence level). Sec. 4.4 discusses the sensitivity of error bound ϵ.

4.2 Speedup and Error validation

Table 3 provides a summary of the average speedup and sampling error achieved by the four sampling methods across the workloads. Figures 7 and 8 present the speedup and error results for each

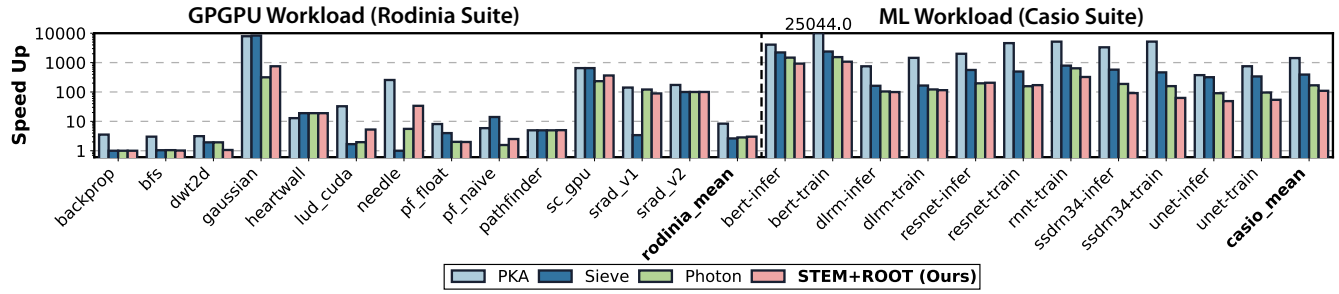

Figure 7: Speedup comparison of four kernel sampling methods on the Rodinia and CASIO benchmark suites. The speedup is presented in log-scale, the average speedup is shown on the far right.

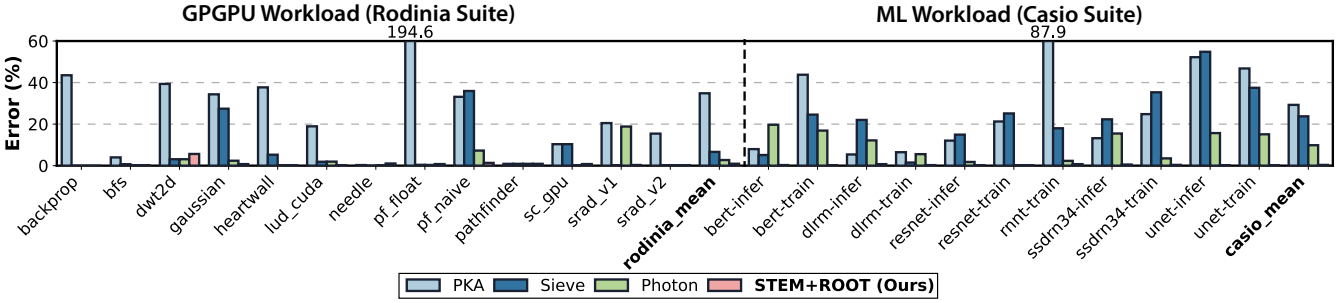

Figure 8: Sampling error comparison of four sampling methods on Rodinia and CASIO suites. STEM+ROOT shows near-zero sampling error on CASIO suite as it leverages the massive number of kernel calls and their execution time distributions.

Table 3: Average speedup (×) and error (%) of 5 kernel sampling methods on 3 GPU benchmark suites. Some values not available due to excessive overhead (details in Sec. 4.7).

Methods	Rodinia (GPGPU)		CASIO (ML)		Huggingface (LLM & ML)	
	Speedup (×)	Error (%)	Speedup (×)	Error (%)	Speedup (×)	Error (%)
Random*	7.09	26.67	984.87	28.39	1004.97	2.40
PKA	8.35	34.85	1425.01	29.26	N/A (*Profiling overhead*)**	
Sieve	2.62	6.63	391.09	23.75	N/A (*Profiling overhead*)**	
Photon	2.84	2.71	168.61	9.85	N/A (*BBV process overhead*)**	
STEM	**3.00**	**0.93**	**109.595**	**0.36**	**31719.057**	**0.57**

*Uniform random; 10% and 0.1% of kernels were sampled for Rodinia and CASIO respectively.

**Profiling and BBV processing overhead is estimated up to 78.68 days.

benchmark. Scatter plot of CASIO and Huggingface speedup/error results are shown in Figure 9, where our method is only compared with random sampling. As a full simulation was intractable for most workloads, we used profiler's cycle counts to calculate speedup and error of sampling methods.

Rodinia Suite. The speedup and error evaluation results are shown at the first column of Table 3. STEM significantly outperforms prior methods in reducing sampled simulation error. While kernel sampling methods generally yield lower speedups on the Rodinia suite, STEM achieves the best balance between speedup and error. STEM achieves speedups comparable to Sieve and Photon while reducing the error from 2–6% to below 1%.

Results on *irregular* workloads from the Rodinia suite demonstrate the robustness of each kernel sampling method. For instance, in `gaussian`, the same kernel is invoked repeatedly for Gaussian elimination, but the number of executed instructions decreases steadily, approaching zero in later iterations. In `heartwall`, while the same kernel runs multiple times, the first invocation is much shorter; subsequent invocations execute roughly 1500× more instructions. Similarly, in `pf_float/naive`, certain kernels are up to 100× longer than others. For workloads like `heartwall` and `pf_naive`, PKA and Sieve struggle to distinguish kernels with drastically different execution times. For example, sampling only the first short kernel in `heartwall` leads to a severe underestimation of total execution time, resulting in a massive 99.9% error. Likewise, in `bfs` and `gaussian`, where kernel execution times vary widely, PKA and Sieve often sample too few kernels, leading to large total time estimation errors.

For `gaussian` and `heartwall` workloads, we manually hand-tuned PKA and Sieve to randomly sample kernels instead of the first-chronological kernel, and the error dropped significantly (e.g., `heartwall`: from 99.9% to 37.69% and 5.27% for PKA and Sieve). Although these improved results are used in our evaluation table and figures, this tuning must be done per workload, highlighting a key limitation of these approaches. In contrast, STEM automatically adapts its sample size based on the runtime variability of profiled kernels. This leads to substantially lower error with only a modest increase in simulation cost, achieving a better trade-off between accuracy and speedup without hand-tuning. We discuss this trade-off further in Section 4.4.

CASIO Suite: Massive number of kernel calls involved in CASIO suite enabled STEM to fully leverage its statistical modeling capabilities. PKA and Sieve, which sample only one or a few kernels per cluster, suffer from large sampling errors. Where Photon reduces error from ~30% to 9.85% with a slight sacrifice in speedup, remaining error accounts to the BBVs that are not capable of fully

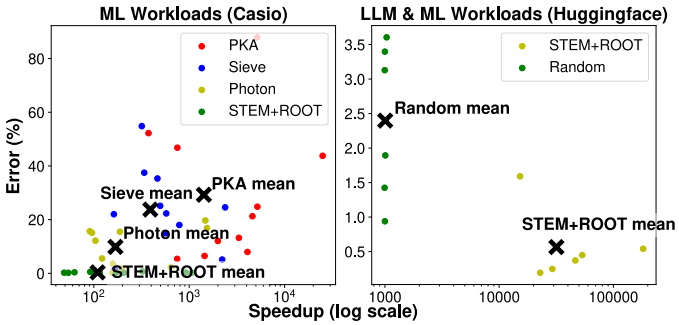

Figure 9: Scatter plot showing the speedup (log scale) and error (%) of different kernel sampling methods on CASIO suite (left) and Huggingface suite (right).

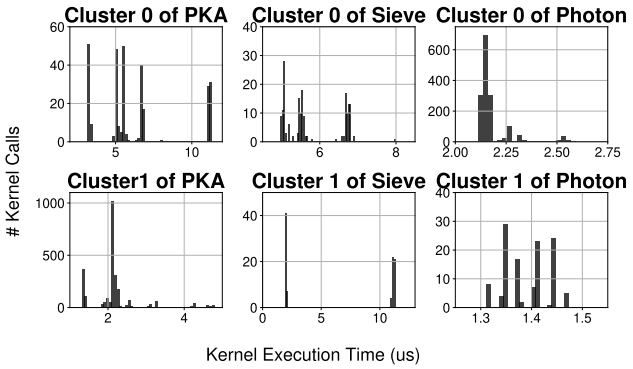

Figure 10: Distribution of execution times for kernels grouped as "identical" by previous sampling techniques, using the DLRM workload from the CASIO benchmark suite.

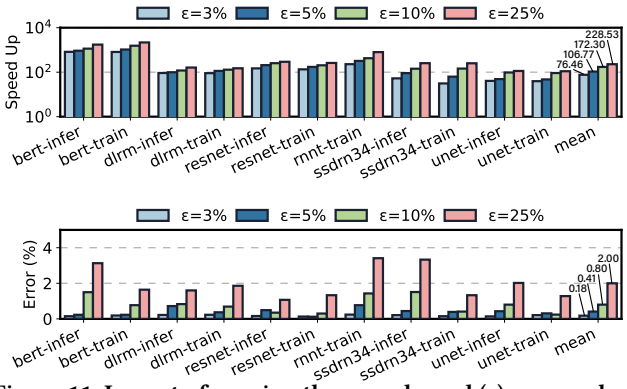

Figure 11: Impact of varying the error bound (ϵ) on speedup and sampling error for STEM. Larger ϵ values enhance speedup with increased error.

distinguishing kernels executed in different contexts or with different inputs. STEM achieves a significantly lower error of just 0.36% while maintaining a 109.60× speedup, representing a 27.6–81.89× reduction in error compared to previous methods. Furthermore, by tuning the hyperparameter (i.e., the error bound ϵ), additional speedup can be achieved. Our sensitivity analysis in Sec. 4.4 shows that we can reach a speedup of 172.30× with only 0.80% error.

Similar to Rodinia, we tuned Sieve to use random sampling for the `ssdrn34-infer` and `unet-infer/train` workloads, where first-chronological sampling resulted in large errors. For CASIO workloads, we disabled Sieve's additional clustering using KDE (kernel density estimation), as it led to oversampling and limited speedups to below 2 – 5× on each workload.

Huggingface Suite: The Huggingface workloads closely align with how recent GPUs are utilized during ML/LLM serving, exhibiting long execution times of ~30 minutes (Table 2). STEM+ROOT significantly outperformed the uniform random sampling, achieving a 31,719× speedup with an error that is 4.25× smaller. In both the CASIO and HuggingFace suites, the observed sampling error is significantly smaller than our theoretical error bound of 5%, indicating that our statistical modeling and clustering approach is highly effective for ML workloads with statistically analyzable runtime behavior, as discussed in our observations (Sec. 2.1).

4.3 Limitations of current sampling methods

While the diverse and unpredictable runtime behaviors of GPU kernels pose a significant challenge for accurate kernel sampling (as discussed in Section 2.1), prior methods that rely on instruction-level or control-flow-based features often fail to distinguish kernels with substantially different execution characteristics. Conventional kernel signatures are limited in their ability to capture the wide and sparse execution time distributions commonly observed in GPU workloads, as they overlook dynamic runtime context and the kernel's sensitivity to the underlying hardware such as memory hierarchy. In contrast, ROOT differentiates kernels that share the same code but differ in runtime behavior using fine-grained clustering. STEM then adaptively assigns sample sizes based on variability within each cluster, allocating more samples to unstable kernel clusters and improving sampled simulation accuracy.

Figure 10 illustrates this limitation of current kernel signatures: each histogram shows a group of kernels that are considered "identical" according to previous methods. For example, in cluster 0 of PKA, all kernels with execution times ranging from 2 μs to 11 μs are treated as identical kernels by PKA's kernel selection algorithm. PKA selects *one* sample from this group and assumes the rest behave identically, leading to significant errors in sampled simulation. Photon performs slightly better in distinguishing kernels, but all kernels shown in the histogram still share a single common proxy for sampled simulation, potentially missing to catch the runtime diversity.

4.4 Tradeoff between the sample size and error

As increasing the number of sampled kernels reduces simulation error but also diminishes speedup, the core challenge is to identify a sweet spot that minimizes error while maintaining high speedup. As PKA and Sieve assume sampling only one or a few kernels per cluster is sufficient, this lead to high sampling error as shown in Table 3, despite its aggressive speedup. Photon improves upon this by comparing GPU BBVs for each kernel, but it still relies on a fixed threshold of 95% and does not adjust sample sizes based on kernel behaviors. In contrast, STEM adaptively determines sample sizes based on the runtime fluctuation of kernels: for kernels exhibiting wide or multi-modal histograms, it selects multiple samples to accurately capture their heterogeneous behavior. This fine-grained approach enables STEM to achieve near-zero sampling error, while still preserving much of the simulation speedup.

Table 4: Average error (%) of sampled simulation on 11 Rodinia and 6 LLM workloads across various GPU microarchitectures using various kernel sampling methods.

μarch Changes	*PKA error (%)	*Sieve error (%)	Photon error (%)	**STEM (ours) error (%)**
Baseline	20.06	24.40	5.96	**2.03**
Cache size ×2	22.66	25.67	5.44	**1.93**
Cache size ×$\frac{1}{2}$	16.65	22.61	5.33	**1.96**
#SM ×2	17.90	28.18	6.49	**2.28**
#SM ×$\frac{1}{2}$	23.68	23.08	5.14	**2.30**

*We observe high error on Rodinia workloads, as smaller configurations are used to run full cycle-level simulation for error measurement.

This behavior is shown in Figure 9, a scatter plot with speedup on the x-axis and sampling error on the y-axis. Black × markers indicate the mean performance of each method. Our method consistently achieves near-zero error with only modest speedup loss, effectively capturing the sweet spot in the speedup–error tradeoff.

Sensitivity Analysis on the Error Bound. We evaluated how varying the error bound ϵ impacts the tradeoff between simulation speedup and sampling error using the CASIO benchmark suite. We tested ϵ values of 3%, 5%, 10%, and 25%, with a fixed 95% confidence level. As shown in Figure 11, smaller ϵ values reduce sampling error but lower speedup due to more samples, while larger values yield higher speedup at the cost of accuracy. For instance, at ϵ = 3%, STEM achieved a 0.18% mean error with 76.46× speedup, whereas ϵ = 25% gave 228.53× speedup with 2.00% error. These results show that STEM enables flexible tuning to balance accuracy and efficiency.

4.5 Validating STEM on various GPU microarchitectures

We evaluate STEM's robustness using a design space exploration (DSE) experiment on the cycle-accurate simulator MacSim [16]. The results suggest that the sampling error on new hardware remains comparable to the error on the baseline machine, even if the same sampling information extracted from execution time profile is used. We modified key microarchitectural parameters, including L1/L2 cache sizes and the number of streaming multiprocessors (SMs), to model GPUs with varying hardware configurations. We selected 11 Rodinia and 6 ML workloads from the HuggingFace suite and reduced their sizes to run full simulation within a few days on MacSim.

Table 4 reports the average error of each method across different hardware variants. The scale of error compared to Table 3 increased due to the smaller input configurations and less number of kernel calls for the Rodinia workloads. STEM consistently maintains significantly lower error than baseline methods on each hardware change. Although such hardware differences cause slight variations in error, STEM's low error across variants highlights the robustness of its execution time–based sampling strategy, supported by its statistically rigorous design. We sampled six different LLM and Rodinia workloads and compared the estimated cycle counts of each method against ground truth, as shown in Figure 12. PKA and Sieve often under- or overestimate total cycle counts depending on the workload, while STEM consistently produce accurate estimates, even under significant microarchitectural changes.

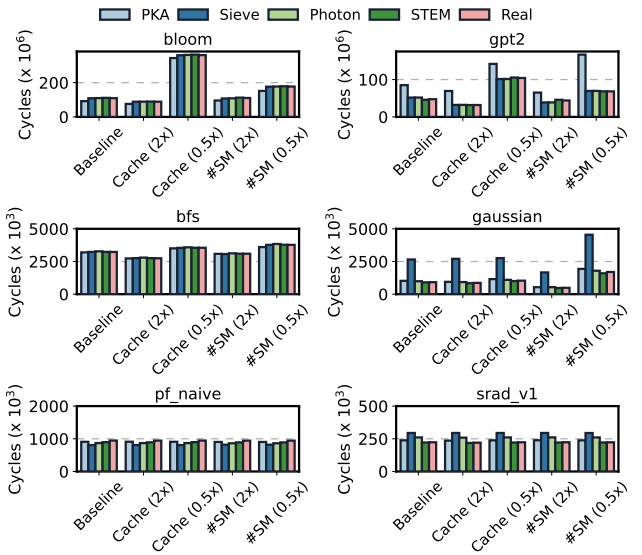

Figure 12: Cycle count comparison between sampled and full simulation across GPU microarchitecture changes using various kernel sampling methods and workloads.

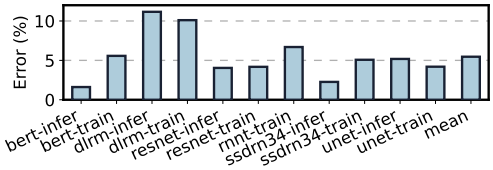

Figure 13: Sampling kernels with STEM on H200 using kernel profiles from H100 results in low sampling error.

Second, we evaluate cross-GPU portability by using sampling information obtained from the NVIDIA H100 and measuring the sampling error on the newer H200 GPU, which features increased global memory capacity and bandwidth. As shown in Figure 13, sampling decisions made on the H100 result in an average error of 5.46% when applied to the H200. The `dlrm` workload, known for its memory-intensive behavior and random access patterns due to large embedding tables, exhibits the highest error due to the hardware's significant memory subsystem upgrades.

Across both experiments, despite hardware-induced changes in absolute kernel execution time, the underlying kernel behaviors and their microarchitectural characteristics captured by STEM using the execution time distributions remain effective for identifying representative kernels–yielding sampling errors on various GPU microarchitectures.

4.6 Validation on microarchitectural metrics

We conducted a detailed microarchitectural behavior comparison to evaluate how well the sampled workload represents the full workload beyond total execution time. 13 metrics from four microarchitectural categories were collected: ① shared/global memory access patterns, ② L1/L2 cache accesses, ③ 16/32-bit floating-point operation counts, and ④ warp execution/branch efficiencies. These provide a comprehensive view of the workload's interaction with key GPU subsystems, offering insights into memory hierarchy utilization, computational precision, and execution control efficiency.

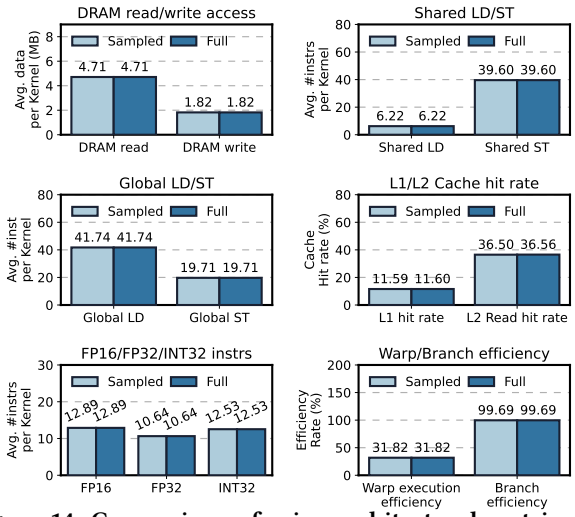

Figure 14: Comparison of microarchitectural metrics between the full workload and the sampled workload. `bert_infer` workload of CASIO benchmark suite was used.

Predictions for these metrics were computed using a weighted sum over the sampled kernels, following the same approach used to estimate total execution time in Section 3.1. Figure 14 shows near-zero differences between the sampled and full simulations across all metrics for the `bert_infer` workload in the CASIO suite. Similar trends were observed across all other CASIO workloads. The same error bound of $\epsilon = 5\%$ was used, which empirically achieved near-zero error in microarchitectural metrics without significant compromise in speedup.

These results suggest that STEM accurately captures diverse microarchitectural behaviors, despite relying primarily on execution time for sampling. This ensures that the sampled simulation reflects the runtime characteristics of the full workload–a critical requirement for GPUs, where performance is shaped by the interplay of parallelism, memory hierarchy, and control flow. For this evaluation, we assumed an optimally warmed-up cache and focused on L2 read hit rate, as GPU cache policies guarantee 100% L2 hit rate for writes.

4.7 Scalability of STEM on large workloads

STEM+ROOT is significantly more scalable than prior methods, as it relies solely on kernel-level execution time data and employs an efficient hierarchical clustering algorithm. In contrast, methods such as PKA and Sieve depend on instruction- or basic-block-level statistics collected *per warp*, incurring substantial overhead due to frequent atomic operations and heavy reliance on limited GPU hardware counters. These methods often require multiple kernel replays and experience slowdowns from contention, making them impractical for large-scale workloads. While Photon collects BBVs more efficiently than instruction counts, it still suffers from high time and space overhead when comparing BBVs for every kernel. Its comparison cost grows quadratically with the number of kernels, becoming infeasible for workloads with millions of kernel invocations. Photon's time complexity ranges from $O(NSd)$ to $O(N^2d)$, where N is the number of kernels, S is the number of samples, and d is the BBV dimensionality. As a result, Photon cannot scale effectively to workloads with millions of kernel calls. In contrast, STEM+ROOT achieves a lower complexity of $O(N \log K)$ to $O(N \log N)$ in the worst case, where K is the number of subclusters.

Table 5: Comparison of profiling overheads across benchmark suites relative to original uninstrumented wall time. Some values were not measured due to excessive overhead.

Sampling methods	Profiler used, metrics collected	Rodinia (GPGPU)	CASIO (ML)	Huggingface (LLM & ML)
PKA [2] (baseline)	NCU, collecting 12 metrics	35.57×	3704.23×	N/A
Sieve [24] (baseline)	NVBit, collecting num. of instrs	94.14×	293.58×	N/A
Photon [21] (baseline)	NVBit, collecting & processing BBVs	12.81×	38.58×	N/A
STEM (ours)	NSYS, collecting kernel exe. time	**1.54**×	**5.53**×	**1.33**×

Table 5 illustrates the trend that profiling overhead increases significantly with larger workloads. We measured the profiling overheads of our method and prior approaches using the profilers noted in our experiment setup. PKA and Sieve introduce overheads of 200×–3000× on the CASIO suite, rendering them impractical for large workloads such as those from HuggingFace. While Photon incurs less overhead for BBV collection, its high-dimensional BBV comparison algorithm introduces quadratic time complexity, making it infeasible for workloads like GPT-2, which contains over 50 million kernel invocations with 800+ BBV dimensions per kernel before the dimension reduction with PCA. STEM reduces profiling overhead by 53.07× to 669.60× on CASIO, making it practical for modern ML workloads. Unlike prior methods, whose overhead grows with kernel count, STEM's profiling remains lightweight and scales well due to fixed post-processing cost. For HuggingFace models, prior methods would require up to 78.68 days of profiling and processing per workload, assuming the same overhead ratio.

5 Discussion

5.1 Potential limitations of using execution time in kernel sampling

Leveraging kernel execution time for workload sampling provides three key advantages: accuracy through fine-grained sampling, statistical feasibility supported by a rigorous error model, and scalability due to minimal profiling overhead. While these advantages are highly effective, some potential concerns rise on our approach.

A potential concern of STEM is that it depends on hardware-dependent data for sampling. When profiling and sampling are performed on hardware A but the simulation is run on hardware B, the sampled kernels from hardware A may fail to capture the workload's runtime behavior on hardware B. For instance, a kernel that exhibits a consistent execution time on hardware A might display heterogeneous runtime behavior on a new GPU microarchitecture. In such cases, the original samples may not fully represent the runtime variability on the target hardware, potentially compromising the accuracy of sampled simulation. While STEM is not entirely immune to this issue, we claim that STEM's fine-grained kernel analysis and adaptive sampling strategy are designed to minimize such hardware-dependent errors. The core of this resilience lies in

STEM's adaptive sampling, which naturally allocates more samples to kernels that are sensitive to microarchitectural changes. Typically, kernels with highly varying runtime at every invocation, which are often those whose performance relies heavily on the memory system, are most susceptible to hardware changes. Because STEM samples these variable kernels more frequently on the source hardware, it preemptively captures a diverse range of behaviors. This inherent oversampling of sensitive kernels ensures that the approach remains robust, even when microarchitectural changes on the target hardware affect their performance. Therefore, STEM often shows much higher accuracy compared to previous works that use hardware-independent parameters, as they only take one or very less samples from each kernel or cluster. This minimizes hardware-dependent inaccuracies in applications like hardware design space exploration (DSE).

To illustrate this principle, consider two kernels: Kernel-A, which is memory-bound and exhibits high runtime variability, and Kernel-B, which is compute-bound and shows stable performance. Based on its analysis, STEM would select many representative samples from Kernel-A but only a few from Kernel-B. Now, consider a microarchitectural change on the target hardware–such as a new cache replacement policy, page management, or prefetcher behavior. This change would likely impact the performance of the memory-sensitive Kernel-A but have a minimal effect on Kernel-B. Even if the change alters the execution of Kernel-A, the impact on the overall estimated performance remains low due to the large number of samples already chosen from it. Therefore, such deviations are unlikely to significantly affect the accuracy of the simulation results.

STEM's robustness is supported by our empirical observations. As shown in Figure 14, kernels within the same cluster tend to maintain similar microarchitectural behavior, even though they were clustered solely based on execution times. This suggests that while a sampled simulation cannot be identical to a full simulation, the selected samples are likely to preserve their core microarchitectural characteristics despite hardware changes. Furthermore, our experiments confirm this resilience. In both DSE and hardware-switching scenarios (Figure 12, 13), our methodology proves reliable and demonstrates superior accuracy compared to previous sampling methods.

5.2 Limitations and Future works

Multi-GPU workloads. Extending to multi-GPU workloads is a promising direction for future work. Supporting multi-device environments with STEM requires careful handling of both synchronous and asynchronous communication kernels, as well as consideration of data/control dependencies, computation–communication overlap, and inter-device synchronization. Future extension of our work could involve using Chakra ET (execution trace), which is a standard method of representing multi-device ML workloads with a DAG (directed acyclic graph) of operations and dependencies [36]. Node and edge sampling on such DAG-style ETs would serve as a decent starting point to analyze data and control dependencies between computation and communication kernels with implicit synchronizations between devices. This would be a foundational step toward fast and accurate sampling for large-scale, multi-GPU simulators [19, 42].

Warm-up of hardware states in sampled GPU simulations. STEM+ROOT's selection algorithm for representative kernels assumes ideal warmup of cache and hardware states. However, certain microarchitectural components, such as the L2 cache, may retain state across kernel boundaries in real hardware, potentially leading to discrepancies in cache reuse during sampled simulation. Efficient and accurate warmup of architectural and microarchitectural states in sampled GPU simulations remains an open research problem that has yet to be fully addressed in the GPU domain.

Despite this limitation, we observe that in most workloads evaluated in this paper, kernel-level simulation time is sufficiently long to mitigate the impact of imperfect cache warmup. For instance, assuming an L2 cache size of 10–50 MB, a few million warp-level memory instructions are typically enough to saturate the GPU caches; a negligible fraction (less than 0.1%) of total instructions in the majority of our benchmarks. To quantify the potential effect of inter-kernel cache reuse, we performed an extreme-case experiment by flushing the L2 cache between every kernel. The results show minimal accuracy degradation: for the STEM method, error increased by only 0.70% on Rodinia and 0.07% on CASIO. For comparison, PKA exhibited 0.92%, Sieve 4.08%, and Photon 0.61% error on Rodinia. This limited impact can be attributed to the large memory footprints of the kernels and as most cache reuse occurs within kernels rather than across them.

Exploring alternative sampling granularity, such as grouping multiple consecutive kernels as the minimum unit, could help capture inter-kernel cache effects. However, this would likely introduce substantial overhead, significantly impacting the speedup benefits of sampling. Another potential solution, hardware state checkpointing, explored in CPU simulation [39], may provide accuracy but remains impractical on modern GPUs due to the significant performance and storage costs of saving large L2/L3 states (e.g., B100, MI300X) and register files. Nonetheless, lightweight warmup strategies, such as inserting warmup instructions or short warmup kernels, may offer practical benefits with minimal simulator modifications.

6 Related works

6.1 Workload sampling for CPUs

SimPoint [9] uses Basic Block Vectors (BBVs) to identify representative regions in CPU workloads. It segments execution into slices and applies K-means clustering on BBVs, enabling sampled regions to reflect full workload behavior across different architectures. SMARTS [43] and SimFlex [41] build on SimPoint by incorporating statistical techniques such as matched-pair comparison to reduce simulation points. Extensions like those by Perelman et al. [31], BarrierPoint, and LoopPoint [3, 33] adapted SimPoint for multi-threaded workloads. Due to GPUs' high thread-level parallelism (TLP) and distinct execution characteristics, new sampling techniques have been developed specifically for GPU workloads, though they share the same core goal of representative workload sampling.

6.2 Kernel-level workload sampling for GPUs

TBPoint [10] uses microarchitecture-independent metrics obtained from profiling to apply hierarchical clustering, grouping similar kernels together and then sampling the kernel closest to the center

of each group. PKA [2] extends this idea by performing k-means clustering on feature vectors from hardware-profiled data, sweeping through k=1 to 20 to find the optimal k and then sampling the first-chronological kernel from each clusters. Sieve [24], on the other hand, only uses the number of instructions as the feature vector to reduce profiling overhead. It stratifies the kernels into three groups based on the degree of instruction count variation across different invocations of the same kernel code. Sieve then samples the first-chronological one for each kernel with the most dominant CTA size. Photon [21] employs online analysis to dynamically determine at runtime whether a basic block (BB), warp, or kernel has stabilized, enabling it to skip ahead to the next simulation phase. It collects and compares GPU BBVs across all kernel invocations to enable accurate kernel-level sampling.

6.3 Other sampling methods in GPU workloads

Intra-kernel sampling is a technique for finding simulation points within a single kernel. GPGPU-MiniBench [44] performs intra-thread-block analysis, while TBPoint and PKA incorporate intra-kernel sampling to gain further speedup beyond kernel-level methods. These techniques detect stable runtime behavior and, once observed, skip remaining simulation phases. Photon also uses online analysis to assess the stability of basic blocks (BBs) and warps for effective intra-kernel sampling. Since kernel-level sampling is orthogonal to warp- or BB-level sampling [2, 21], our method can be combined with cases of few kernel calls or long-running kernels. SeyyedAghaei et al. [34] accelerate GPU simulation using small-scale models, but their approach is limited to workloads that scale linearly with the number of streaming multiprocessors (SMs), covering only a narrow class of GPU applications.

7 Conclusion

This paper introduces STEM+ROOT, an accurate, scalable, and statistically robust kernel-level sampling solution for large-scale GPU workloads. STEM and ROOT leverage key observations on the heterogeneous runtime characteristics of modern GPU kernels, particularly how their execution time distributions provide valuable insights for accurate sampling. STEM+ROOT offers a fast and reliable kernel sampling solution with high speedup and minimal error. Our evaluation demonstrates that STEM+ROOT significantly reduces sampling error on cycle-level simulations with design space exploration (DSE) experiments. Moreover, STEM exhibits excellent scalability across modern large-scale GPU applications.

8 Appendix

8.1 Solution for Problem 1.

Let $a_i \equiv \mu_i$, $b_i \equiv N_i^2\sigma_i^2$, and $c \equiv (\epsilon \sum_i N_i\mu_i/z_{1-\alpha/2})^2$ for simplicity. Then, the **Problem 1** becomes as below:

$$\begin{aligned} \underset{m_i}{\text{minimize}} \quad & \sum_i a_i m_i \\ \text{subject to} \quad & \sum_i \frac{b_i}{m_i} - c \le 0 \\ \text{and} \quad & m_i > 0 \ \text{ for } \ \forall i \in \{0...k-1\}. \end{aligned}$$

The corresponding Lagrangian function $\mathcal{L}$ can be written as follows:

$$\mathcal{L}(\mathbf{m}, \lambda) = \sum_i m_i a_i + \lambda_k \cdot (\sum_i \frac{b_i}{m_i} - c) + \sum_i \lambda_i \cdot (-m_i).$$

The solution $\mathbf{m}^*$ must satisfy the following four Karush–Kuhn–Tucker (KKT) conditions:

- Stationary Condition: $\nabla\mathcal{L}(\mathbf{m}^*; \lambda) = \mathbf{0}$ (a)
- Primal Feasibility: $\sum_i b_i/m_i^* - c \le 0$ (b)
  and $(-m_i^*) \le 0$ for $\forall i \in \{0...k-1\}$ (c)
- Dual Feasibility: $\lambda_i \ge 0$ for $\forall i \in \{0...k\}$ (d)
- Complementary Slackness: $\lambda_k \cdot (\sum_i b_i/m_i^* - c) + \sum_i \lambda_i \cdot (-m_i^*) = 0$ (e)

From (b), (c), and (d), we can see that in each term either one of λ_i or the multiplied term should be zero. Since we are assuming $m_i > 0$, $\lambda_i = 0$ for $\forall i \in \{0...k-1\}$.

Also, from (a), $a_i - \lambda_k b_i/(m_i^*)^2 - \lambda_i m_i^* = 0$.

Since $a_i \neq 0$, $\lambda_k \neq 0$ and thus the equality of (b) holds and thus $m_i^* = \sqrt{\lambda_k b_i/a_i}$ for $\forall i \in \{0...k-1\}$.

By putting this into (b), we obtain $\sum_i \sqrt{a_i b_i/\lambda_k} = c$ and thus $\lambda_k = (\sum_i \sqrt{a_i b_i}/c)^2$. Therefore, the solution to the non-linear optimization problem is:

$$m_i = \frac{\sqrt{\sum_j a_j b_j}}{c} \cdot \sqrt{\frac{b_i}{a_i}} \text{ for } \forall i \in \{0...k-1\}.$$

8.2 Proof of Theorem 3.1

Proof. By the definition of sampling errors,

$$\sum_i (N_i^{(j)})^2 \frac{(\sigma_i^{(j)})^2}{m_i^{(j)}} \le \left(\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i^{(j)}\mu_i^{(j)}\right)^2 \tag{9}$$

satisfies for arbitrary $\forall j \in \{0, ..., N-1\}$.

Since $\frac{\epsilon}{z_{1-\alpha/2}} \sum_i N_i^{(j)}\mu_i^{(j)}$ is positive for every j, we apply the following inequality: $\sum_j x_j^2 \le (\sum_j x_j)^2$ when $x_j \ge 0$ for $\forall j$.

We then sum (9) by j to get

$$\sum_{ij} (N_i^{(j)})^2 \frac{(\sigma_i^{(j)})^2}{m_i^{(j)}} \le \left(\frac{\epsilon}{z_{1-\alpha/2}}\right)^2 \sum_j \left(\sum_i N_i^{(j)}\mu_i^{(j)}\right)^2 \tag{10}$$

$$\le \left(\frac{\epsilon}{z_{1-\alpha/2}}\right)^2 \left(\sum_{ij} N_i^{(j)}\mu_i^{(j)}\right)^2. \tag{11}$$

The sum $\sum_{ij}$ in (11) is the same as summing through every cluster in the union set $\bigcup_{j=0}^{N-1}\{C_{i_j}^{(j)}\}$. By substituting

$$\tilde{\mu} = \sum_{ij} N_i^{(j)}\mu_i^{(j)} \text{ and } \tilde{\sigma}^2 = \sum_{ij} (N_i^{(j)})^2 \frac{(\sigma_i^{(j)})^2}{m_i^{(j)}},$$

we transform (11) into the following inequality

$$\left|\frac{(\tilde{\mu} + z_{1-\alpha/2}\tilde{\sigma}) - \tilde{\mu}}{\tilde{\mu}}\right| \le \epsilon, \tag{12}$$

which implies that the union of cluster sets also gives bounded sampling error under a $1-\alpha$ confidence interval.

□

References

[1] AMD. 2025. ROCProfiler documentation. https://rocm.docs.amd.com/projects/rocprofiler/en/latest/.

[2] Cesar Avalos Baddouh, Mahmoud Khairy, Roland N. Green, Mathias Payer, and Timothy G. Rogers. 2021. Principal Kernel Analysis: A Tractable Methodology to Simulate Scaled GPU Workloads. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 724–737. https://doi.org/10.1145/3466752.3480100

[3] Trevor E. Carlson, Wim Heirman, Kenzo Van Craeynest, and Lieven Eeckhout. 2014. BarrierPoint: Sampled simulation of multi-threaded applications. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2–12. https://doi.org/10.1109/ISPASS.2014.6844456

[4] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. IEEE, Piscataway, NJ, USA, 44–54.

[5] Euijun Chung, Seonjin Na, and Hyesoon Kim. 2024. Allegro: GPU Simulation Acceleration for Machine Learning Workloads. In *Machine Learning for Computer Architecture and Systems 2024*.

[6] Michael Davies, Ian McDougall, Selvaraj Anandaraj, Deep Machchhar, Rithik Jain, and Karthikeyan Sankaralingam. 2024. A Journey of a 1,000 Kernels Begins with a Single Step: A Retrospective of Deep Learning on GPUs. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 20–36. https://doi.org/10.1145/3620665.3640367

[7] Lieven Eeckhout. 2022. *Computer Architecture Performance Evaluation Methods* (1st ed.). Springer Cham.

[8] Lieven Eeckhout. 2024. RIP Geomean Speedup Use Equal-Work (Or Equal-Time) Harmonic Mean Speedup Instead. *IEEE Computer Architecture Letters* (2024).

[9] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. 2005. SimPoint 3.0: Faster and More Flexible Program Phase Analysis. *J. Instr. Level Parallelism* 7 (2005). https://api.semanticscholar.org/CorpusID:11937761

[10] Jen-Cheng Huang, Lifeng Nai, Hyesoon Kim, and Hsien-Hsin S. Lee. 2014. TBPoint: Reducing Simulation Time for Large-Scale GPGPU Kernels. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 437–446. https://doi.org/10.1109/IPDPS.2014.53

[11] Rodrigo Huerta, Mojtaba Abaie Shoushtary, José-Lorenzo Cruz, and Antonio González. 2025. Analyzing Modern NVIDIA GPU cores. *arXiv preprint arXiv:2503.20481* (2025).

[12] Huggingface. 2025. *Huggingface*. Retrieved Jan 17, 2025 from https://huggingface.co

[13] Intel. 2025. Profiling with Intel Gaudi Software. https://docs.habana.ai/en/latest/Profiling/Intel_Gaudi_Profiling/.

[14] JAX. 2025. JAX Profiling computation. https://docs.jax.dev/en/latest/profiling.html.

[15] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 473–486. https://doi.org/10.1109/ISCA45697.2020.00047

[16] Hyesoon Kim, Jaekyu Lee, Nagesh B Lakshminarayana, Jaewoong Sim, Jieun Lim, and Tri Pho. 2023. *Macsim: A CPU-GPU heterogeneous simulation framework user guide*.

[17] David M. Lane. 2025. *OnlineStatBook: Sampling Distribution of the Mean*. Retrieved Jan 17, 2025 from https://onlinestatbook.com/2/sampling_distributions/samp_dist_mean.html

[18] Jaewon Lee, Euijun Chung, Saurabh Singh, Seonjin Na, Yonghae Kim, Jaekyu Lee, and Hyesoon Kim. 2025. Let-Me-In:(Still) Employing In-pointer Bounds Metadata for Fine-grained GPU Memory Safety. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1648–1661.

[19] Ying Li, Yuhui Bao, Gongyu Wang, Xinxin Mei, Pranav Vaid, Anandaroop Ghosh, Adwait Jog, Darius Bunandar, Ajay Joshi, and Yifan Sun. 2025. TrioSim: A Lightweight Simulator for Large-Scale DNN Workloads on Multi-GPU Systems. (2025).

[20] David J. Lilja. 2000. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press.

[21] Changxi Liu, Yifan Sun, and Trevor E. Carlson. 2023. Photon: A Fine-grained Sampled Simulation Methodology for GPU Workloads. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 1227–1241. https://doi.org/10.1145/3613424.3623773

[22] Xueyang Liu, Seonjin Na, Euijun Chung, Jiashen Cao, Jing Yang, and Hyesoon Kim. 2025. Contention-Aware GPU Thread Block Scheduler for Efficient GPU-SSD. *IEEE Computer Architecture Letters* (2025).

[23] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. 2018. Nvidia tensor core programmability, performance & precision. In *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE, 522–531.

[24] Mahmood Naderan-Tahan, Hossein SeyyedAghaei, and Lieven Eeckhout. 2023. Sieve: Stratified GPU-Compute Workload Sampling. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 224–234. https://doi.org/10.1109/ISPASS57527.2023.00030

[25] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2024. A Comprehensive Overview of Large Language Models. arXiv:2307.06435 [cs.CL]

[26] NVIDIA. 2025. CUDA C++ Programming Guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#warp-shuffle-functions.

[27] NVIDIA. 2025. *CUDNN Documentation*. Retrieved Jan 17, 2025 from https://docs.nvidia.com/deeplearning/cudnn/latest/api/overview.html

[28] NVIDIA. 2025. *NVIDIA Nsight Compute*. Retrieved Jan 17, 2025 from https://developer.nvidia.com/nsight-compute

[29] NVIDIA. 2025. *NVIDIA Nsight Systems*. Retrieved Jan 17, 2025 from https://developer.nvidia.com/nsight-systems

[30] Adam Paszke, Sam Gross, and Francisco Massa et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG]

[31] Erez Perelman, Marzia Polito, J-Y Bouguet, Jack Sampson, Brad Calder, and Carole Dulong. 2006. Detecting phases in parallel applications on shared memory architectures. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 10–pp.

[32] Huanzhi Pu, Rishabh Ravi, Shinnung Jeong, Udit Subramanya, Euijun Chung, Jisheng Zhao, Chihyo Ahn, and Hyesoon Kim. 2025. Hardware vs. Software Implementation of Warp-Level Features in Vortex RISC-V GPU. *arXiv preprint arXiv:2505.03102* (2025).

[33] Alen Sabu, Harish Patil, Wim Heirman, and Trevor E Carlson. 2022. LoopPoint: Checkpoint-driven sampled simulation for multi-threaded applications. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 604–618.

[34] Hossein SeyyedAghaei, Mahmood Naderan-Tahan, and Lieven Eeckhout. 2024. GPU Scale-Model Simulation. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1125–1140.

[35] Joram Soch. 2021. *The Book of Statistical Proofs*. Retrieved Jan 15, 2025 from https://statproofbook.github.io/P/norm-lincomb

[36] Srinivas Sridharan, Taekyung Heo, Louis Feng, Zhaodong Wang, Matt Bergeron, Wenyin Fu, Shengbao Zheng, Brian Coutinho, Saeed Rashidi, Changhai Man, et al. 2023. Chakra: Advancing performance benchmarking and co-design using standardized execution traces. *arXiv preprint arXiv:2305.14516* (2023).

[37] Yifan Sun, Trinayan Baruah, Saiful A. Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, Harrison Barclay, Amir Kavyan Ziabari, Zhongliang Chen, Rafael Ubal, José L. Abellán, John Kim, Ajay Joshi, and David Kaeli. 2019. MGPUSim: enabling multi-GPU performance modeling and optimization. In *Proceedings of the 46th International Symposium on Computer Architecture* (, Phoenix, Arizona,) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 197–209. https://doi.org/10.1145/3307650.3322230

[38] Elliot Tanis and Robert V. Hogg. 1977. *Probability and Statistical Inference*.

[39] Luk Van Ertvelde, Filip Hellebaut, Lieven Eeckhout, and Koen De Bosschere. 2006. NSL-BLRL: Efficient cache warmup for sampled processor simulation. In *39th Annual Simulation Symposium (ANSS'06)*. IEEE, 8–pp.

[40] Oreste Villa, Mark Stephenson, David Nellans, and Stephen W Keckler. 2019. Nvbit: A dynamic binary instrumentation framework for nvidia gpus. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 372–383.

[41] Thomas F Wenisch, Roland E Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C Hoe. 2006. SimFlex: statistical sampling of computer system simulation. *IEEE Micro* 26, 4 (2006), 18–31.

[42] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 283–294. https://doi.org/10.1109/ISPASS57527.2023.00035

[43] Roland E Wunderlich, Thomas F Wenisch, Babak Falsafi, and James C Hoe. 2003. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th annual international symposium on Computer architecture*. 84–97.

[44] Zhibin Yu, Lieven Eeckhout, Nilanjan Goswami, Tao Li, Lizy K John, Hai Jin, Chengzhong Xu, and Junmin Wu. 2015. GPGPU-MiniBench: accelerating GPGPU micro-architecture simulation. *IEEE Trans. Comput.* 64, 11 (2015), 3153–3166.