GPU Programming for AI Workflow Development on AWS SageMaker: An Instructional Approach

Sriram Srinivasan
ssrinivasan@bowiestate.edu
Bowie State University
Bowie, MD, USA

Hamdan Alabsi
halabsi@bowiestate.edu
Bowie State University
Bowie, MD, USA

Rand Obeidat
robeidat@bowiestate.edu
Bowie State University
Bowie, MD, USA

Nithisha Ponnala
PONNALAN0103@students.bowiestate.edu
Bowie State University
Bowie, MD, USA

Azene Zenebe
azenebe@bowiestate.edu
Bowie State University
Bowie, MD, USA

Abstract

We present the design, implementation, and comprehensive evaluation of a specialized course on GPU architecture, GPU programming, and how these are used for developing AI agents. This course is offered to undergraduate and graduate students during Fall 2024 and Spring 2025. The course began with foundational concepts in GPU/CPU hardware and parallel computing and progressed to develop RAG and optimizing them using GPUs. Students gained experience provisioning and configuring cloud-based GPU instances, implementing parallel algorithms, and deploying scalable AI solutions. We evaluated learning outcomes through assessments, course evaluations, and anonymous surveys. The results reveal that (1) AWS served as an effective and economical platform for practical GPU programming, (2) experiential learning significantly enhanced technical proficiency and engagement, and (3) the course strengthened students' problem-solving and critical thinking skills through tools such as TensorBoard and HPC profilers, which exposed performance bottlenecks and scaling issues. Our findings underscore the pedagogical value of integrating parallel computing into STEM education. We advocate for broader adoption of similar electives across STEM curricula to prepare students for the demands of modern, compute-intensive fields.

CCS Concepts

• **Computing methodologies** → **Cloud computing**; **Massively parallel algorithms**; • **Computer systems organization** → *Hardware acceleration*; • **Software and its engineering** → Collaboration in software development.

Keywords

AWS SageMaker, GPU programming, AI workflow, Parallel computing, Cloud computing, Instructional design, Machine learning, Deep learning

ACM Reference Format:
Sriram Srinivasan, Hamdan Alabsi, Rand Obeidat, Nithisha Ponnala, and Azene Zenebe. 2025. GPU Programming for AI Workflow Development on AWS SageMaker: An Instructional Approach. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3731599.3767381

1 Introduction

We are currently entering an era where Artificial Intelligence (AI) agents are capable of reasoning, planning, and acting with unprecedented autonomy. To build these sophisticated AI agents, High-Performance Computing (HPC), and specifically Graphics Processing Unit (GPU) programming skills, play a crucial role. Traditional computing paradigms simply cannot handle the immense computational demands of modern AI. Training deep learning models and deploying real-time, high-fidelity AI solutions requires a level of parallel processing power that only GPUs can deliver. There is an exploding demand for AI engineers and data scientists [4, 9, 27] who understand how GPUs work and how to exploit parallelism in GPU architecture to design intelligent algorithms. Foundational GPU programming skills are rapidly becoming an indispensable competency for anyone pursuing a career in AI. Consequently, infusing GPU programming into graduate and undergraduate STEM curricula is a critical necessity to prepare the next generation of scientists, engineers, and data professionals. In Fall 2024 and Spring 2025, we taught an elective course (Special Topics). This course was taken by seniors majoring in Data Analytics and Information Systems who have completed an Introduction to Programming course. It's also offered as a cross-listed elective for graduate students in Information Systems. In both semesters, our focus was on developing AI workflows and agents with GPU programming using libraries like RAPIDS and Dask [25]. The first half of the 16-week course concentrated on introducing basic HPC concepts, demonstrating how to use accelerators to improve performance, and implementing simple parallel algorithms on accelerators. Previously, when this course was offered, we never introduced HPC concepts. Two factors contributed to revising the curriculum and infusing HPC for developing AI agents: 1) The support received through The U.S. National Science Foundation (NSF) funding, specifically the NSF Expand AI grant, which enabled us to procure Amazon Web Services (AWS) credits to facilitate this transition [17]. 2) The availability and accessibility of infrastructure provided by AWS, which offers affordable, on-demand access to GPU instances as a cloud service. This capability eliminated the need for maintaining on-campus HPC infrastructure to support the course. Combining both semesters, we had about thirty-nine students enrolled in this course. All students had a Python programming background, as the prerequisite course (Introduction to Programming) was taught in Python. Most students lacked C programming knowledge, and since the course's focus was on developing AI agents using HPC, they all utilized Python JIT (Just-in-Time) libraries such as Numba [6, 7, 16] and CuPy [12] for parallel programming. As most of them had no prior knowledge of parallel computing, the course began with the basics of writing simple parallel programs and running them on GPUs. Students were given AWS console/Educate access during the first class, where they set up credentials and were taught how to use Python scripts to spin up and terminate instances. Students were familiar with AWS SageMaker, which offers Jupyter Notebook, allowing them to write and run code in one place. Starting week 2, we explored into the foundations of GPU computing, where students learned the anatomy of Compute Unified Device Architecture (CUDA) threads and blocks and performed hands-on experiments with Matrix operations using CuPy. As we progressed, students focused on data flow between the Central Processing Unit (CPU) (host) and GPU (device). They started profiling and used critical thinking and problem-solving skills to address memory bottlenecks, which impact performance. Before midterm, we introduced JIT libraries, as discussed, to exploit GPU parallelism for large datasets. After the midterm, students built Graph Convolutional Networks (GCNs) and trained them on a GPU; they explored various ways to use parallel programming concepts to aggregate node features on multiple GPUs for large-scale, real-world networks. Students were also required to build Retrieval Augmented Generation (RAG) systems and experiment with GPU-tuned retrievers and generators to optimize latency and throughput. In this paper, we present the evaluation results of the course based on multiple instruments: students' performance on assessments, their end-of-semester course evaluations, and two anonymous surveys (mid and post-course). Our takeaways from offering this course are: 1) AWS or any cloud resources are great and effective tools for infusing parallel programming with modern GPUs; 2) The best way to infuse HPC concepts to STEM students with minimal programming background is through hands-on learning with a mix of lecture and in-class labs; 3) Infusing HPC concepts into the curriculum to build AI agents has a positive impact on student learning, enabling them to apply these concepts to real-world data analysis tasks. To reinforce this learning, graded activities that require independent work should be designed as extensions of in-class labs, challenging students to apply their critical thinking and problem-solving skills. Therefore, we recommend the continued offering of this course and propose exploring its designation as a General Education course, with an introductory programming course as a prerequisite. The remainder of the paper is organized as follows: Section II reviews related work, Section III outlines the course structure, Section IV presents evaluation results, and Section V concludes with future directions.

2 Related Work

The increasing pervasiveness and ubiquity of HPC systems, largely driven by the unprecedented growth and reliance on AI, requires the efficient execution of complex computational tasks. Consequently, GPUs have become indispensable components for optimizing the computing paradigm within AI applications, with major platforms such as Google Cloud, NVIDIA, and AWS SageMaker extensively utilizing GPU-accelerated and cloud-based computing resources to meet these demands. This technological evolution underscores the urgent need for academic curricula to incorporate HPC concepts, particularly parallel and GPU programming, to prepare students for current and emerging computational challenges.

Prasad et al. proposed a practical framework as outlined in the NSF/IEEE TCPP Curriculum Initiative on Parallel and Distributed Computing, specifically the Core Topics for Undergraduates [21]. This initiative provides a nationally endorsed framework for embedding parallel and distributed computing concepts into undergraduate education. Chen et al. further support this initiative, highlighting the importance of computing competency, particularly in the context of developing practical skills for high-performance computing. Their support reflects a growing consensus that undergraduate computer science education must evolve to reflect the realities of modern computational environments [8].

Arroyo echoes this need by promoting the integration of HPC and distributed programming across core computer science curricula. He proposes a progressive pedagogical framework that bridges sequential and parallel paradigms using a combination of traditional and novel teaching tools to enhance student preparedness [5]. In alignment with Arroyo's perspective, NVIDIA offers hands-on GPU-accelerated training programs in AI, HPC, and workflow automation that include teaching kits and hybrid learning formats tailored for university integration, supporting the importance of aligning academic instruction with evolving industry practices [1].

Further advancing educational strategies, Qasem et al. present innovative approaches to parallel and distributed computing education that resonate with the need for practical GPU programming skills. Their work highlights GPU programming courses modeled on frameworks such as Carpentries, alongside specialized bootcamps focused on AI workloads in HPC systems, which collectively support hands-on learning and interdisciplinary student engagement [23]. In a related context, Adams et al. analyze the rapid adoption of cloud computing services and their implications for higher education curricula. They argue that the growing demand for practical, hands-on training in cloud computing, programming, and data science requires structured teaching materials, canonical learning objectives, and modular course design. Their approach ensures that cloud computing content can be seamlessly integrated into a variety of computing courses, facilitating student exposure to this rapidly evolving field [3].

Neelima et al. provide a case study of HPC integration into an undergraduate engineering curriculum over six years, documenting a progressive expansion from shared and distributed memory programming to accelerator and hybrid programming models. Their findings reveal that deliberate pedagogical strategies and institutional support can overcome initial challenges, ultimately yielding measurable benefits to both students and stakeholders [18].

Similarly, Qasem et al. advocate for a module-driven approach to embedding heterogeneous computing topics across the computer science curriculum. This strategy addresses curricular constraints by dispersing focused modules within existing courses, a method that demonstrated increased student engagement and foundational understanding of heterogeneous architectures pertinent to HPC, mobile processing, and Internet of Things (IoT) domains [24].

Bridging theory and practice, Xu's senior elective course effectively teaches OpenMP (Open Multi-Processing), Message Passing Interface (MPI), and CUDA using Google Colab and hands-on Raspberry Pi cluster construction to teach heterogeneous and parallel computing. Xu's evaluation reports that this hybrid model significantly enhances students' comprehension of data organization and parallel programming concepts, highlighting the value of experiential learning in HPC education [29].

In parallel, the increasing prominence of AI in diverse sectors has prompted curricular innovations beyond traditional computer science programs. Xu and Babaian address the challenges of delivering AI education to interdisciplinary and non-technical audiences by proposing a graduate-level curriculum that balances foundational AI knowledge with cutting-edge developments. Their study provides actionable guidelines for curriculum design, emphasizing the need for pedagogical approaches that engage students with varying backgrounds [28].

Taken together, these works collectively argue for a curriculum that incorporates GPU programming and HPC as part of AI education, leveraging cloud-based platforms such as AWS SageMaker. The integration of GPU-accelerated cloud resources in academic training provides students with relevant, hands-on experience essential for navigating AI workflows in contemporary computational environments. Such curricular developments are vital to align educational outcomes with the technological demands of industry and research, ensuring that graduates are equipped to contribute effectively to AI-driven innovation.

3 Course Structure

3.1 Course Overview

This course plunges students into the fundamental concepts of accelerators and empowers them to leverage these powerful tools for creating sophisticated AI workflows and automated AI agents. Crucially, it integrates modules that introduce core HPC concepts, laying a robust foundation for modern AI development. The course is flexibly offered across the fall, spring, and summer semesters, contingent on enrollment. Each weekly 120-minute session is dynamically split: the initial half is dedicated to module lectures, while the subsequent half transforms into an immersive, hands-on lab. For spring and summer iterations, the course adopts a hybrid model, featuring optional in-person lab sessions to accommodate diverse student needs.

This interdisciplinary offering caters to undergraduate seniors from Data Analytics, Information Systems, and Public Health Informatics programs, as well as graduate students in Information Systems. Designed as a comprehensive 16-week journey during the standard academic year, it condenses into an intensive four-week program during the summer. The detailed evaluation components are discussed in Section IV.

Figure 1 showcases student enrollment across Fall 2024, Spring, and Summer 2025, with the Spring 2025 iteration notably seeing fifteen graduate students enroll. For the scope of this paper, we will exclusively analyze course evaluation and assessment results from the Fall 2024 and Spring 2025 sessions, as the Summer 2025 session is currently ongoing.

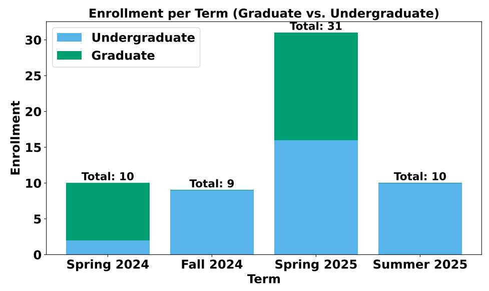


Figure 1: Enrollment per Term (Graduate vs. Undergraduate)

3.2 Infrastructure Setup

One of the cornerstones of our pedagogical approach was providing students with unparalleled hands-on access to AWS cloud resources. Each student was assigned a dedicated Identity and Access Management (IAM) role, empowering them to independently launch instances, configure NVIDIA GPUs, and provision the necessary computational power for their labs and assignments. We further augmented this practical experience by leveraging AWS Educate resources, training students not only on core AWS services but also on the intricacies of AWS SageMaker and spinning up new instances.

To ensure responsible resource utilization, each student's usage was capped for all assessments, complemented by automated scripts designed to terminate idle resources. Our current budget allows us to provision ample AWS resources until July 2026, ensuring sustained access. For efficient management and monitoring of unused instances, all GPU instances are provisioned within the US East (N. Virginia) region. Students were provided with a bootstrap script that simplified resource configuration using their AWS credentials for each assessment.

3.2.1 AWS Cost. The cost efficiency of this approach is significant: the average on-demand cost for a single-GPU instance was approximately \$1.262 per student per hour, while multi-GPU instances (up to 3) averaged about \$2.314 per student per hour. Over the entire semester, students typically spent around 40–45 hours utilizing AWS resources (including GPUs, AWS SageMaker, and VPC) for their assessments, translating to an average cost of roughly \$50–60 per student for the entire semester. Appendix A provides an overview of average AWS GPU usage and cost during Fall 2024 and Spring 2025.

For certain assessments, we strategically utilized AWS Educate resources, which are provided free of charge by AWS, further enhancing cost-effectiveness and accessibility. Furthermore, we allowed students to request additional resources, capped at \$100 per student for the semester; remarkably, no one found it necessary to request additional funds, highlighting the sufficiency of the allocated resources. While we also explored Google Colab, a platform discussed in academic literature [29], its limitation to a single GPU per session made AWS a superior choice for the diverse and demanding multi-GPU AI workloads central to our curriculum.

3.3 Modules

Our primary aim is to immerse students in GPU programming concepts (GPC), enabling them to confidently navigate and utilize various powerful libraries such as RAPIDS for GPU-accelerated data science and Dask for scalable computing. Students are expected to gain a profound understanding of how GPU architecture fundamentally differs from CPU architecture, exploring into the core components of GPUs like kernels, blocks, grids, and threads.

Each course module was developed using a variety of resources, including academic peer-reviewed papers and official documentation from libraries such as Nvidia [1], Dask [2], and PyTorch [22]. The inclusion of academic papers as course materials was inspired by curriculum development articles on High-Performance Computing [15, 19]. To support student learning, we provided access to lecture slides and supplementary reference materials for each module. The textbook [10] was used to introduce students to the fundamental concepts of GPU programming and architecture.

Crucially, all labs, assignments, and exams require students to employ a Python-based interface for GPU programming, rather than native CUDA (C/C++). Our objective is to familiarize them with the architecture and empower them to create sophisticated AI workflows that are heavily supported by the rich ecosystem of Python libraries. While we acknowledge the inherent performance compromises of not using native CUDA for this course, our pedagogical focus remains on accessibility and practical application for this specific student demographic.

Table 1 outlines the modules covered during the 16-week course, alongside their corresponding student learning outcomes, and the deliverables for that week, which includes a weekly quiz for each module except week 7, and the final exam.

Students developed several distributed GPU programs, with a primary focus on partitioning large-scale, real-world networks, including PubMed [26] and Reddit [11]. This foundational step enabled the subsequent execution of Graph Convolutional Network (GCN) [14] training for node label prediction across multiple GPUs.

Algorithm 1 illustrates a classic node label problem. Here, a GCN model processes node features, $X \in \mathbb{R}^{|V| \times d}$, and the graph's structural information, typically represented by its adjacency matrix, derived from the graph $\mathcal{G} = (V, E)$. The model's objective is to predict labels for nodes, $Y \in \mathbb{R}^{|V|}$. A key strategy for infusing parallel processing into this workflow involves distributing the graph across multiple GPUs, allowing each device to train on a distinct subgraph.

As delineated in Algorithm 1, line 3, the graph is initially partitioned using the highly efficient METIS algorithm [13]. Subsequently, lines 5 and 6 detail the distribution of these subgraphs across the available GPUs, facilitating parallel training (lines 9–10). We specifically leveraged the Dask framework [25] for orchestrating this distributed training process. Within this parallel paradigm

Algorithm 1 Distributed GCN Training Using METIS Partitioning and Dask

```
Require: Graph $\mathcal{G} = (V, E)$, features $X \in \mathbb{R}^{|V| \times d}$, labels $Y \in \mathbb{R}^{|V|}$
Require: GCN model $M$ with parameters $\theta$, loss function $\mathcal{L}$, optimizer $O$
Require: Number of partitions $k$, epochs $N_{epochs}$
Ensure: Trained parameters $\theta$
 1: procedure TrainDistributedGCN($\mathcal{G}, X, Y, M, \mathcal{L}, O, k, N_{epochs}$)
 2:     Load $\mathcal{G}$, $X$, and $Y$; compute normalized adjacency $\tilde{A}$
 3:     Partition $\mathcal{G}$ into $\{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ using METIS
 4:     Initialize Dask cluster; assign one GPU per worker
 5:     Distribute each $(\mathcal{G}_i, X_i, Y_i)$ to worker $i$
 6:     Initialize global model $M_{global}$ with parameters $\theta$
 7:     for $epoch = 1$ to $N_{epochs}$ do
 8:         for each worker $i$ in parallel do
 9:             Compute forward pass on $(X_i, \mathcal{G}_i)$
10:             Compute loss $\mathcal{L}_i$ and local gradients $\nabla\theta_i$
11:         Aggregate $\{\nabla\theta_i\}_{i=1}^{k}$ across workers
12:         Update global parameters $\theta$ using $O$
13:         Report total loss $\sum_i \mathcal{L}_i$
14:     return $\theta$
```

(lines 10–12), each GPU worker independently processes its assigned partition, computes the local loss by comparing predicted node labels with ground truth, and calculates gradients. These local gradients are then efficiently combined, and a global optimizer updates the model's parameters in a synchronized fashion.

While students were encouraged to explore and devise their own orchestration methods for graph splitting and parallel training, our work illuminated several interesting challenges inherent in distributed graph processing. Furthermore, to foster a deeper understanding of partitioning effects, students were encouraged to experiment with random graph partitioning as an alternative to METIS [13] and to thoroughly analyze the resulting GPU utilization patterns.

Parallelizing large-scale, often dense graphs presents significant difficulties. In most cases, we observed that simply splitting the graph and distributing the training yielded minimal performance improvement. However, a notable outcome was the enhanced prediction accuracy scores after splitting and training, particularly when compared to sequential approaches.

4 Evaluation Instruments and Results

We utilize a multifaceted assessment approach, incorporating students' grades, their end-of-semester course evaluations, the comprehensive course project, and two anonymous surveys (mid- and post-course) to thoroughly gauge the effectiveness of our course modules and pedagogy.

4.1 Student Performance Assessment

Our rigorous assessment framework includes twelve to fourteen dynamic in-class labs (flexibly adjusted to class pace), four challenging assignments, and a collaborative group project (capped at two members). The project that constitutes 15% of final grade. We also evaluate attendance and class participation. Class participation is assessed through the submission of scribed notes for each lecture, along with a thoughtful question related to the module covered. Both the notes and the question are due immediately after class. This participation component is complemented by two comprehensive exams, a midterm and a final.

Table 1: Course Modules, Student Learning Outcomes (SLOs), and Deliverables

Week / Topic	Student Learning Outcome (SLO)	Deliverable
Week 1: AWS GPU Setup + Course Introduction	**Apply**: Set up AWS EC2 GPU instances and configure Python environments	Lab 1: AWS GPU instance setup with Jupyter and SSH access
Week 2: CUDA Fundamentals & GPU Parallelism	**Understand/Apply**: Explain GPU architecture, grasp CUDA programming basics, and implement parallel execution	Lab 2: CuPy vector/matrix operations & parallel processing
Week 3: Memory Management & GPU Optimization	**Analyze/Optimize**: Manage and optimize memory transfers between host and GPU	Lab 3: Matrix multiplication with memory profiling using Numba
		Assignment 1: GPU Matrix Multiplication and Profiling (Due Week 5)
Week 4: GPU Profiling Tools & Bottleneck Analysis	**Analyze/Evaluate**: Apply Nsight Systems, PyTorch profiler, and cProfile for comprehensive GPU workload analysis	Lab 4: Profiling GPU RL loop with Nsight and PyTorch profiler
		Assignment 2: Distributed GPU Data Processing (Due Week 7)
Week 5: Custom CUDA Kernels with Python	**Create/Integrate**: Write, compile, and seamlessly integrate custom CUDA kernels in Python workflows	Lab 5: Custom CUDA kernel with Numba + profiling
Week 6: RAPIDS + Dask for Scalable Data Pipelines	**Apply/Create**: Process large datasets efficiently using RAPIDS cuDF and Dask for distributed GPU workflows	Lab 6: Parallel data processing using Dask with RAPIDS cuDF
Week 7: Midterm Exam / Assessment	No SLO (Assessment Week)	Midterm Exam; Assignment 2 Due
Week 8: Deep Learning on GPUs (PyTorch Focus)	**Apply/Optimize**: Train and optimize neural networks using GPU acceleration, specifically focusing on GCNs	Lab 7: CNN model training on GPU using PyTorch
Week 9: Reinforcement Learning on GPUs	**Develop/Implement**: Develop reinforcement learning agents accelerated by GPUs	Lab 8: DQN agent training using CUDA-enabled PyTorch
Week 10: Multi-GPU Training & Parallel Strategies	**Apply/Scale**: Scale models efficiently using multi-GPU setups with Distributed Data Parallel (DDP)	Lab 9: PyTorch DDP implementation across 2 GPUs
Week 11: AI Agent Foundations & GPU Benefits	**Understand/Describe**: Describe AI agents and explain the GPU's critical role in training acceleration	Lab 10: Simple reinforcement agent using CuPy/Numba
		Assignment 3: Multi-GPU AI Agent (Due Week 13)
Week 12: Retrieval-Augmented Generation (RAG) Basics	**Understand/Describe**: Describe RAG architectures, combining retrieval and generation modules effectively	Lab 11: Basic RAG pipeline using FAISS for retrieval
Week 13: GPU-Optimized RAG Development	**Construct/Optimize**: Construct and optimize RAG models using GPU-accelerated retrievers and generators	Lab 12: Build GPU-enabled RAG with retriever + small LLM
Week 14: RAG Pipeline Optimization & Inference	**Optimize/Deploy**: Optimize end-to-end RAG pipelines for efficient real-time GPU inference	Lab 13: Deploy real-time RAG inference pipeline
		Assignment 4: End-to-End RAG System (Due Week 16)
Week 15: Project Development & Support	**Apply/Create**: Apply GPU acceleration, AI agent techniques, and RAG models in capstone projects	Lab 14: Build your own Lab (Extra Credit); Academic paper review (Extra Credit)
Week 16: Final Project Presentations & Exam	**Showcase/Demonstrate**: Showcase final projects demonstrating GPU-accelerated AI/RAG pipelines	Final Project Presentation

While all grading activities contribute to the final grade, their weighting is intentionally uneven. For labs and assignments, students will have direct access to both the instructor and the Teaching Assistant (TA), fostering a supportive learning environment where assistance is readily available. These highly interactive activities collectively constitute half of final grade, emphasizing hands-on application. The remaining half comes from independent endeavors, including the assignments and closed-book exams, designed to assess individual mastery, along with the group project.

Appendix B lists the extra credit opportunities available to students.

Figure 2 visually depicts the grade distribution across Fall 2024 and Spring 2025. In Fall 2024, the majority of students achieved a 'B' grade, with many encountering difficulties with post-midterm modules. Despite receiving assistance from the TA and instructor, a number of students were only able to submit partial assignments, suggesting challenges in independent completion. In stark contrast, Spring 2025 witnessed a significant uplift, with over 60% of students securing an 'A' grade and demonstrating enhanced independence in completing assignments. This marked improvement is directly correlated with proactive revisions to our lab instructions, meticulously designed to better prepare students for subsequent assignments. We further observed that a substantial majority of students proactively leveraged office hours throughout the semester, not only to clarify complex assignments but also to request crucial code reviews prior to submission. Conversely, students earning a 'B' or lower typically correlated with missed submissions or late lab/assignment turn-ins. The exam average remained remarkably consistent across both semesters, hovering between 75-80%. This reflects a solid foundational understanding among the students. (See Appendix C for further data analysis and findings).

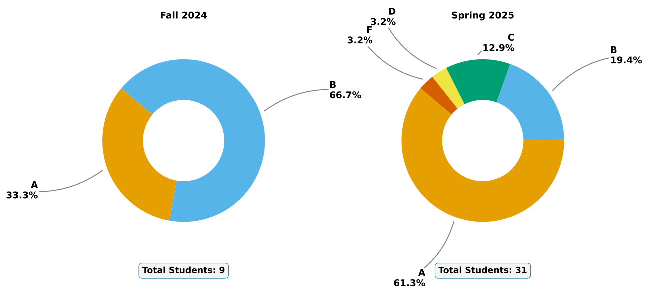


Figure 2: Grade Distribution for Fall 2024 and Spring 2025 Course Offerings

4.2 End-of-Semester Student Evaluations

A robust 85% of students completed the anonymous online evaluation form, providing valuable feedback on the course using questions presented in Table 2. All questions, which are standard for all courses and designed by the university, utilized a five-point Likert scale with response options including "Always," "Often," "Sometimes," "Seldom," "Never," and "N/A."

Table 2: End-of-Semester Course Assessment Questions

Evaluation Questions
The course information further developed my knowledge in this area.
The course activities enhanced my learning of the course content.
The oral assignments improved my presentation skills.
The course activities improved my computer technology skills.
Lab or clinical experiences contributed to my understanding of the course theories and concepts.
The instructor clearly explained laboratory or clinical experiments or procedures.

Figure 3 provides an overview of the student feedback regarding course content, with the X-axis representing the percentage of responses for each type and the Y-axis representing the questions from Table 2. Figure 3 suggests that both undergraduate and graduate courses are largely successful in delivering positive learning experiences. However, there appear to be subtle differences in the perceived strengths, with undergraduates valuing core course content and graduates finding more significant gains in specific skill development. Further investigation into the methodologies and learning objectives of these course levels could provide deeper insights into these observed differences. The "Seldom," "Never," and "N/A" categories consistently represent a small minority of responses, indicating that strongly negative or irrelevant feedback is not prevalent. For both groups, "Lab/Clinical Experiences Contributed to Understanding" and "Instructor Clearly Explained Lab/Clinical Procedures" tend to have lower "Always" percentages compared to the course content aspects, suggesting these might be areas for instructors or course designers to explore for enhancement. We plan to experiment in Fall 2025 by revising our labs and offering a more clear explanation about labs and assignments during class sessions and to explicitly highlight how assignments serve as extensions of the lab experiences. Although the survey question wording is standardized across disciplines, in this course “lab/clinical procedures” specifically refers to programming and computational tasks rather than healthcare contexts. (See Appendix D for further data analysis on student course satisfaction.)

4.3 Anonymous Surveys

During the Fall 2024 and Spring 2025 semesters, students were invited to provide anonymous feedback. This feedback was crucial for the department to evaluate existing course offerings and strategically plan for future similar courses, particularly concerning GPU programming content and pedagogical approaches. Feedback was systematically collected via Google Forms at two key points: during the sixth week (prior to the midterm exam) and the twelfth week (before students began their group projects). The primary objective of these two surveys was to ascertain the extent to which students had grasped the course material. Each survey question utilized a five-point Likert scale. Participation in the surveys was robust, with most students completing them. The mid-course survey featured three questions assessing their confidence in using Numba, building and configuring GPUs on AWS, and utilizing HPC tools for profiling. The final survey repeated these questions and additionally included a question on their confidence level in using multiple GPUs for parallel algorithm implementation.

Figure 4a compares student self-assessment of their ability to use Numba to implement a parallel algorithm using CUDA across two semesters. In Fall 2024, responses were relatively evenly distributed: two students strongly disagreed, two disagreed, one was neutral, two agreed, and two strongly agreed. Conversely, Spring 2025 shows a greater proportion of students in the 'Neutral' to 'Strongly Agree' categories. Specifically, nine students were neutral, seven agreed, and five strongly agreed, making 'Neutral' the largest single response group. This suggests that while many students felt capable, a significant portion remained in the middle, indicating they felt neither fully proficient nor entirely lacking in this specific skill.

Figure 4b illustrates student confidence level in building and configuring GPU clusters. In Fall 2024, students initially expressed weak confidence during the midterm. We believe this stemmed from initial challenges in configuring GPUs and ensuring instances were correctly connected within the same Virtual Private Cloud (VPC)

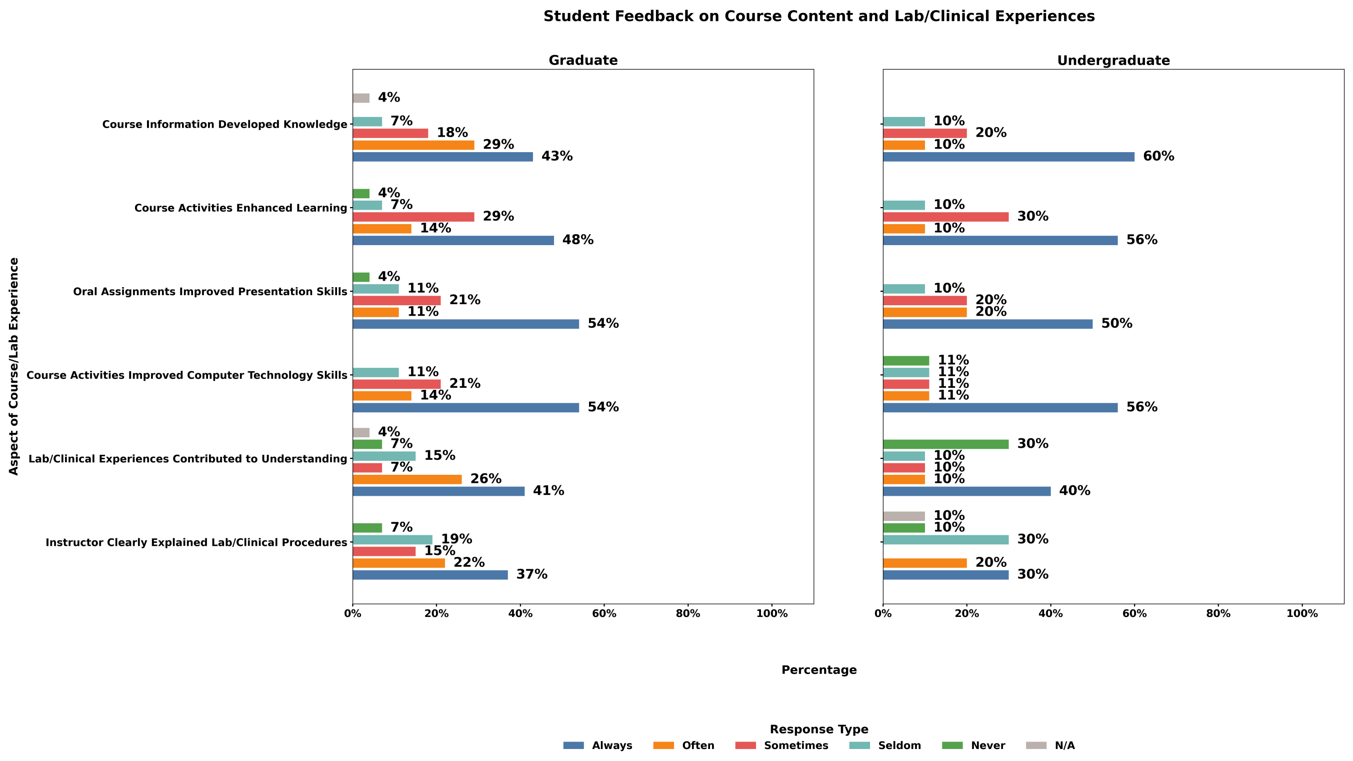


Figure 3: Student Feedback on Course Content and Lab/Clinical Experiences

with appropriate subnet addresses. By the final survey, confidence improved; we attribute this to the automation of instance setup and networking, although some students still faced challenges in updating and executing these scripts correctly.

In Spring 2025, midterm confidence was mixed, with approximately twelve students expressing disagreement, eight remaining neutral, and eleven showing agreement. By the final survey, confidence had substantially improved, with students reporting strong confidence. We posit this increase in confidence resulted from the reinforcement of cluster creation tasks throughout all in-class activities.

Figure 4c illustrates student confidence level in utilizing PyTorch Profiler [22] and Nsight Systems [20] for GPU profiling. In Fall 2024, students initially expressed strong confidence during the midterm assessment; however, a clear reduction in confidence was observed in the final survey regarding their proficiency with GPU profiling tools. We suggest this decline may be attributed to the increased independent application of these profilers for project work after the midterm. A similar trend was observed in Spring 2025; however, the magnitude of the confidence dip (from "agree" to "disagree") was less pronounced compared to Fall 2024. This attenuated decline in Spring 2025 might be a consequence of incorporating additional hands-on profiling activities during in-class sessions.

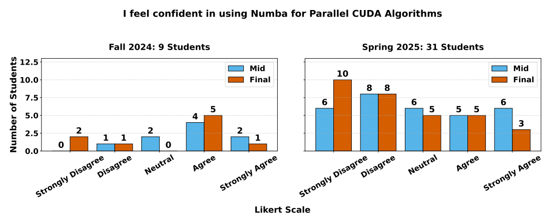
Figure 4d presents final survey results on student confidence in applying multi-GPU training and parallel computing for AI models such as GCN. Fall 2024 responses, from a small group of students, were largely positive. Spring 2025 results, from a larger group, were more varied, with ten students expressing disagreement while most reported neutral or higher confidence. The moderate confidence may reflect the assignment's difficulty, particularly issues installing PyTorch and configuring Dask (Algorithm 1).

5 Conclusions

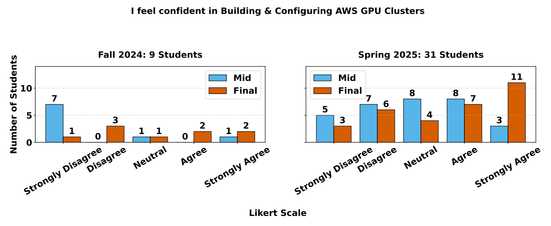
Despite the rapid advancements in AI and the growing accessibility of GPU technologies, integrating GPU programming and HPC concepts into the STEM curriculum remains a significant challenge. We contend that the most effective pedagogical approach to bridge this gap involves hands-on laboratory exercises and assessments that not only impart technical skills but also cultivate critical thinking and problem-solving abilities among STEM students.

We successfully introduced GPU programming. Our experience in delivering these special topics courses, as detailed in this paper, has been overwhelmingly positive. These offerings have provided invaluable insights into student engagement and learning outcomes.

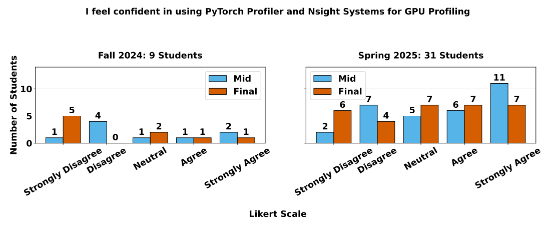
Looking ahead, we plan to refine and expand our lab modules and assessments based on these experiences. We are actively exploring various ways to integrate specific course modules into other Data Analytics offerings. To better prepare students for these advanced topics, we are considering a revision of the prerequisite course to infuse foundational HPC concepts alongside traditional sequential programming. Furthermore, we intend to explore alternative pedagogical strategies and continuously evolve the curriculum to align with the AI and HPC competencies currently in high demand in the



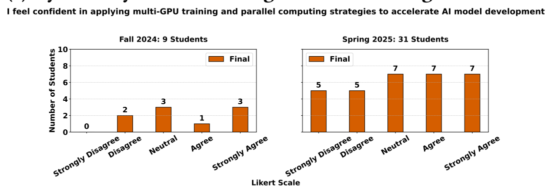
(a) Numba-CUDA Parallel Programming Ability



(b) Confidence in Using AWS GPU Cluster



(c) Python/PyTorch and Nsight GPU Profiling Confidence



(d) Multi-GPU Programming Confidence

Figure 4: Student survey results (Fall 2024 / Spring 2025)

job market. Our overarching goal is to offer this course more frequently, making it accessible to a broader and more diverse STEM student population.

Acknowledgments

This research was supported by grant funding from the National Science Foundation (NSF) for the collaborative project titled "CAP: Expanding Use-Inspired AI Research and Instruction Across Campus," Award #2401658.

References

[1] [n. d.]. Hands-on GPU-Accelerated Training in AI, HPC, and Workflow Automation. https://www.nvidia.com/en-us/training/. Accessed: July 19, 2025.
[2] 2024. Dask Tutorial. https://tutorial.dask.org/. Accessed: [Insert Access Date].
[3] J. Adams, B. Hainey, L. White, D. Foster, N. Hall, M. Hills, C. Taglienti, et al. 2020. Cloud computing curriculum: Developing exemplar modules for general course inclusion. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. 151–172.
[4] Alooba. 2024. Everything You Need to Know When Assessing Parallel Computing Skills. *Alooba Blog* (2024). https://www.alooba.com/skills/concepts/devops/parallel-computing/ Accessed: [Current Date, e.g., July 21, 2025].
[5] Miguel Arroyo. 2013. Teaching parallel and distributed computing to undergraduate computer science students. (2013), 1297–1303. doi:10.1109/IPDPSW.2013.276
[6] Manuel Bruenger, Marco Cecchini, Georg Hager, and Gerhard Wellein. 2020. Lessons learned from comparing C-CUDA and Python-Numba for GPU-Computing. In *Workshop on Python for High-Performance and Scientific Computing (PyHPC)*.
[7] Marco Cecchini, Rahul Ramaswamy, Manuel Bruenger, Georg Hager, and Gerhard Wellein. 2020. Implementation and Evaluation of CUDA-Unified Memory in Numba. In *Euro-Par 2020: Parallel Processing Workshops*. Springer, 120–133.
[8] Juan Chen, Sheikh Ghafoor, and John Impagliazzo. 2022. Producing competent HPC graduates. *Commun. ACM* 65, 12 (2022), 56–65.
[9] NVIDIA Corporation. 2023. The State of AI Infrastructure: Insights from Industry Leaders. https://resources.nvidia.com/en-us-ai-infrastructure/ai-infrastructure-report White Paper.
[10] Matheus Dias and Matheus G. Dias. 2019. *Hands-On GPU Programming with Python and CUDA: Explore high-performance parallel computing with CUDA*. Packt Publishing Ltd.
[11] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*. 1024–1034. https://arxiv.org/abs/1706.02216
[12] Per Holm, Manuel Bruenger, and Gerhard Wellein. 2023. Evaluating Numba and CuPy for GPU-Accelerated Monte Carlo Transport. *Mathematics* 11, 3 (2023), 612. doi:10.3390/math11030612
[13] George Karypis and Vipin Kumar. 1998. *METIS–A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview Version 4.0.
[14] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1609.02907
[15] Nathaniel Kremer-Herman. 2023. Challenges and Triumphs Teaching Distributed Computing Topics at a Small Liberal Arts College. In *Proceedings of the Workshop on Education for High-Performance Computing (EduHPC)*. IEEE.
[16] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A Dynamic Python Compiler for Numerical Functions. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (2015), 1–6. doi:10.1145/2833157.2833162
[17] National Science Foundation. 2024. CAP: Expanding Use-Inspired AI Research and Instruction Across Campus. NSF Award #2401658. Award abstract retrieved from NSF Award Search.
[18] B. Neelima and J. Li. 2015. Introducing high performance computing concepts into engineering undergraduate curriculum: a success story. In *Proceedings of the Workshop on Education for High-Performance Computing*. 1–8.
[19] Sarah Neuwirth. 2023. Bridging the Gap between Education and Research: A Retrospective on Simulating an HPC Conference. In *Proceedings of the Workshop on Education for High-Performance Computing (EduHPC)*. IEEE.
[20] NVIDIA Corporation. 2025. *NVIDIA Nsight Systems*.
[21] Sushil K Prasad, Almadena Chtchelkanova, Sajal Das, Frank Dehne, Mohamed Gouda, Anshul Gupta, Joseph Jaja, Krishna Kant, Anita La Salle, Richard LeBlanc, et al. 2011. NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: core topics for undergraduates. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. 617–618.
[22] PyTorch Contributors. 2024. *Profiling your PyTorch Module*.
[23] A. Qasem et al. 2022. Lightning Talks of EduHPC 2022. In *2022 IEEE/ACM International Workshop on Education for High Performance Computing (EduHPC)*. IEEE, 42–49. doi:10.1109/EduHPC56719.2022.00011
[24] A. Qasem, D. P. Bunde, and P. Schielke. 2021. A module-based introduction to heterogeneous computing in core courses. *J. Parallel and Distrib. Comput.* 158 (2021), 56–66.
[25] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked Algorithms and Task Scheduling. *Proceedings of the 14th Python in Science Conference (SciPy)* (2015), 130–136. doi:10.25080/Majora-7b98e3ed-013
[26] Prithviraj Sen, Galen Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. *Collective Classification in Network Data*. Technical Report. Carnegie Mellon University. https://linqs.soe.ucsc.edu/data Technical Report.
[27] U.S. Bureau of Labor Statistics. 2024. *Data Scientists : Occupational Outlook Handbook*.
[28] J. J. Xu and T. Babaian. 2021. Artificial intelligence in business curriculum: The pedagogy and learning outcomes. *The International Journal of Management Education* 19, 3 (2021), 100550.
[29] Z. Xu. 2023. Teaching heterogeneous and parallel computing with google colab and raspberry pi clusters. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. 308–313.

A AWS GPU Computational Hours

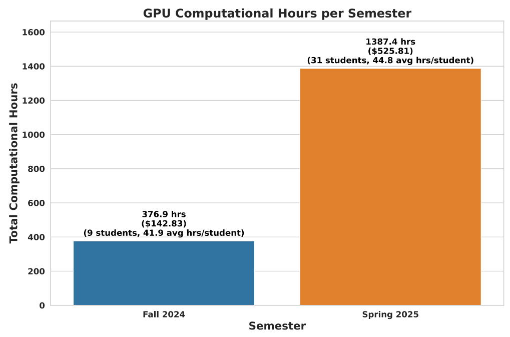


Figure 5: Average AWS GPU usage and cost for Fall 2024, and Spring 2025.

Figure 5 provides an estimated average computational hours of GPU instances for all labs and assignments in Fall 2024 and Spring 2025. The notable increase in average hours per student during Spring 2025 is due to the introduction of two additional labs. For group projects, the average usage of GPU resources was less than 2 hours in both semesters. We did not include the computational hours of GPU instances from AWS Educate because the instructor lacks access to resource usage insights for that platform.

B Extra Credit

Two distinct extra credit opportunities were offered to students.

1. Build Your Own Lab

Students were encouraged to design their own lab based on the course modules. The only requirement was that the lab could not replicate an existing one. The primary goal of this initiative was to assess students' ability to apply core concepts and design a lab from scratch. A secondary objective was to identify potential labs that could be rigorously tested during the summer of 2025 and possibly integrated into future semesters. No students attempted this extra credit in Fall 2024. In Spring 2025, three students submitted the lab; however, none of the submissions fully met the student learning outcomes. We believe this may be due to students attempting the design and preparation during finals week, which prevented them from dedicating sufficient time to research and development.

2. Academic Paper Review

In Spring 2025, an extra credit opportunity was offered where students could select a peer-reviewed academic paper published between 2020 and 2025 based on the modules discussed. They were asked to provide a one-page summary, discuss its shortcomings, and propose an extension of the research. The goal was to help students understand the current research landscape in their chosen modules. The inclusion of this extra credit was inspired by an article[19]; where attempt was made to bridge the gap between education and research. Approximately 60% of students completed this activity. While most provided excellent summaries, their explanations for expanding on the proposed research were often vague. Based on this outcome, we plan to explore integrating academic papers directly into course modules to better promote student research skills.

C Data Analysis and Findings

We evaluated potential differences in academic performance between graduate and undergraduate students enrolled in a special topics course, specifically focused on GPU programming, offered in Fall 2024 and Spring 2025. To assess this, we tested the following hypotheses:

- **Null Hypothesis (H_0)**: There is no difference in average performance between graduate and undergraduate students.
- **Alternative Hypothesis (H_1)**: There is a difference in average performance between graduate and undergraduate students.

As we did not specify a direction for the difference, a two-tailed test was employed to assess whether the observed performance differences reflected random variation or a systematic distinction between the two cohorts. Before conducting hypothesis testing, we evaluated whether the data satisfied the assumptions required for parametric tests—namely, normality and homogeneity of variance.

Figure 6 displays histograms, while Figure 7 and Figure 8 present Q–Q plots. These visualizations indicate clear departures from normality, particularly in the graduate group, whose scores were tightly clustered near the upper end of the distribution and exhibited noticeable skewness.

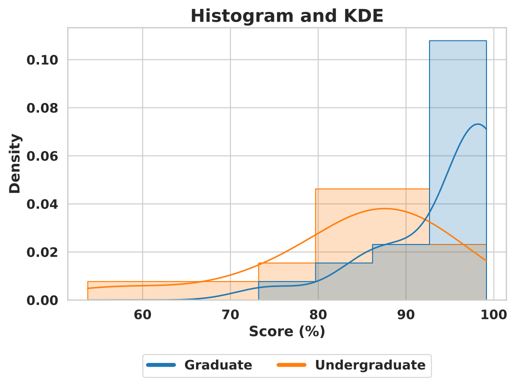


Figure 6: Histogram comparison of academic scores between graduate and undergraduate student groups.

To statistically assess normality, we conducted Shapiro–Wilk tests. The graduate group showed a substantial deviation from normality ($W = 0.722$, $p < .001$), while the undergraduate group also deviated, though less severely ($W = 0.898$, $p = .037$). In contrast, Levene's test for equality of variances revealed no significant difference between the two groups ($F = 2.437$, $p = .127$), suggesting that the assumption of homogeneity of variance was met. These results are summarized in Table 3.

In addition to assumption testing, descriptive statistics provided insight into the distribution and central tendency of scores in each group, as shown in Table 4. Graduate students achieved remarkably higher mean scores and demonstrated a more compact distribution, reflected in lower variability compared to undergraduates.

Although the variances were homogeneous, the pronounced non-normality, especially in the graduate scores, made parametric tests (e.g., independent samples t-test) inappropriate. Given the relatively

Table 3: Results of Assumption Tests for Normality and Homogeneity of Variance

Assumption Test	Statistic	p-value
Shapiro–Wilk (Graduate)	0.722	$< .001$
Shapiro–Wilk (Undergraduate)	0.898	.037
Levene's Test	2.437	.127

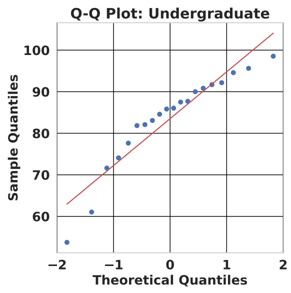


Figure 7: Q–Q plots of academic scores for undergraduate students.

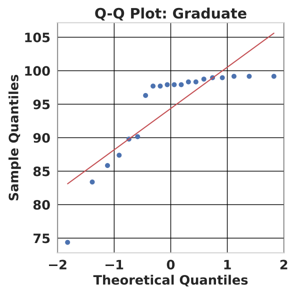


Figure 8: Q–Q plots of academic scores for graduate students.

Table 4: Descriptive statistics for academic performance scores by group

Group	Mean	Std Dev	Min	Q1	Median	Q3	Max	Count
Graduate	94.36	6.91	74.38	90.06	97.92	98.80	99.17	20
Undergraduate	83.51	11.33	53.75	80.79	85.94	91.05	98.54	20

small sample sizes ($n = 20$ per group) and non-normal distributions, we employed the Mann–Whitney U test, a non-parametric alternative suitable for comparing central tendencies between two independent groups. The Mann–Whitney U test revealed a statistically significant difference in performance between the two groups, with graduate students outperforming undergraduates ($U = 332.00$, $p = .0004$). Figure 9 which show a higher median and a more compact score distribution among graduate students compared to undergraduates.

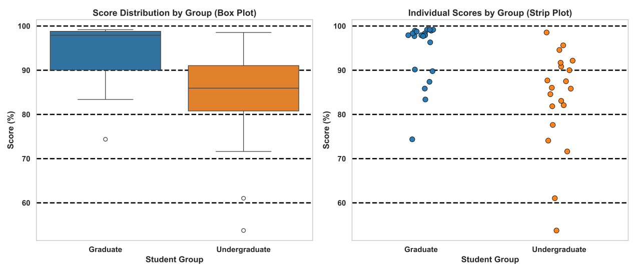


Figure 9: Boxplot and Stripplot of academic scores for graduate and undergraduate student groups.

In conclusion, the data provide strong evidence to reject the null hypothesis. Graduate students performed significantly better than undergraduate students in terms of their weighted total scores.

By testing assumptions and applying a suitable non-parametric method, we ensured the validity of our findings and aligned our analytical approach with the data's distributional characteristics.

D Descriptive Statistics of Student Satisfaction

We analyzed student satisfaction for the GPU programming course across Fall 2024 and Spring 2025 based on course evaluations collected anonymously ($n = 18$) and found overall positive results with some differences between the semesters. In Fall 2024, most students (87.5%) reported "Very High" satisfaction, while one student (12.5%) reported "Very Low" satisfaction. In Spring 2025, 60% of students rated their satisfaction as "Very High," and the remaining 40% rated it as "High," with no "Very Low" ratings.

As shown in Figure 10 we counted the number of responses in each satisfaction category by semester. When we examined the proportional breakdown in Figure 11 we saw that Fall 2024's responses clustered heavily around "Very High," whereas Spring 2025 had a more balanced split between "Very High" and "High."

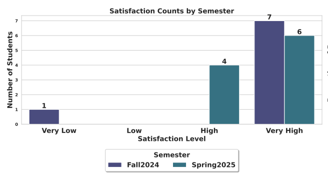


Figure 10: Bar Plot of Satisfaction Counts by Semester.

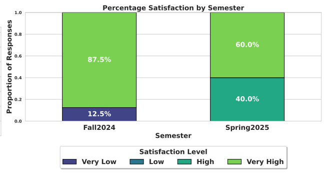


Figure 11: Stacked Bar Chart of Percentage Satisfaction by Semester.

Overall, we conclude that students were very satisfied with the GPU programming course in both semesters. While Fall 2024 included an isolated low satisfaction rating, Spring 2025 showed consistently positive and moderately varied satisfaction levels. These findings suggest that the GPU programming course effectively facilitated the acquisition of valuable new skills, which contributed to high levels of student satisfaction, reflecting the course's successful delivery and instructional quality.