Performance Comparison of Apple Silicon, AMD Ryzen, NVIDIA RTX, and AMD Radeon RX for BERT Networking Intrusion Log Processing

Alexander Yue	**2079436**	**AZYue@cougarnet.uh.edu**
Nathaniel Nguyen	**1830317**	**ntnguyen225@cougarnet.uh.edu**

Abstract

This report presents a detailed methodology and key findings from a hardware performance evaluation of a BERT-based Natural Language Processing (NLP) model adapted for binary Network Intrusion Detection (NID). The model, originally trained on the IoT-23 dataset targeting Internet of Things (IoT) traffic, was benchmarked against the CIC-IDS2017 dataset, which represents enterprise network traffic. This domain shift initially resulted in complete classification failure, with 0% True Positives. To address this, a novel threshold optimization strategy was employed, stabilizing the decision boundary at 0.85 to achieve a security-critical Recall of 90.2%. The optimized model was then rigorously benchmarked across heterogeneous architectures, including NVIDIA CUDA, Apple Metal Performance Shaders (MPS), and high-core AMD CPUs. The NVIDIA RTX 2080 served as the high-throughput reference, processing 2.83 million flows in approximately 40 minutes at 1,261.67 logs per second using FP16 precision. These benchmarks quantify critical trade-offs in inference speed, latency, and power efficiency across modern computing platforms, providing actionable insights for deploying Transformer-based NID systems in resource-constrained environments.

1.0 Introduction

The escalating sophistication of cyber threats demands robust, high-speed Network Intrusion Detection Systems (NIDS) capable of real-time analysis. Machine Learning (ML) approaches, particularly Transformer architectures like BERT, have demonstrated efficacy in classifying network flows by interpreting structured features as sequential language data. This project extends beyond algorithmic accuracy to quantify hardware-level performance implications of such models on diverse architectures: NVIDIA CUDA-enabled GPUs, high-core AMD CPUs, and Apple Silicon MPS accelerators. By evaluating inference throughput, latency, and energy efficiency, the study highlights operational viability for edge, cloud, and enterprise deployments.

1.1 Dataset Source and Composition

1.1.1 Training Dataset: IoT-23

The BERT model was pretrained on the IoT-23 dataset, a specialized collection of labeled network traffic from IoT devices. Key characteristics include:

- Source Domain: Traffic from resource-constrained IoT endpoints, such as smart bulbs and voice assistants, exhibiting distinct communication patterns.
- Capture Environment: Derived from 20 malware executions (e.g., Mirai and Torii variants) on infected devices, plus benign traffic from three clean devices.
- Impact on Model: The training induced biases toward IoT-specific protocols and flow metrics, leading to fragility when applied to dissimilar domains like enterprise networks.

1.1.2 Evaluation Dataset: CIC-IDS2017

Benchmarking utilized the CIC-IDS2017 dataset, a benchmark for realistic enterprise traffic captured over five days, encompassing benign activity and advanced attacks.

1.1.3 Input Feature Engineering

Raw packet data was processed via CICFlowMeter to extract 80 statistical flow features. A custom preprocessing script selected 15 key attributes (e.g., Flow Duration, Packet Counts, Flow Bytes/s, and TCP Flag Counts) and serialized them into descriptive text strings, enabling BERT's NLP capabilities to model network behaviors as linguistic sequences.

1.1.4 Workload Size

The evaluation workload comprised 2,830,743 network flows, aggregated from eight CSV files. This imbalanced dataset (predominantly benign) rigorously tested computational scalability and model robustness.

1.1.5 Attack Diversity

The CIC-IDS2017 dataset includes 14 attack categories, categorized as follows:

Attack Category	Specific Attack Types Included	Primary Threat Vector
Denial of Service (DoS/DDoS)	DDOS, DOS HULK, DOS SLOWLORIS, DOS GOLDENEYE, DOS SLOWHTTPTEST	Volume/Resource Consumption
Botnet/C&C	BOT	Malicious Communication/Control
Brute Force/Credential	FTP-PATATOR, SSH-PATATOR, WEB ATTACK - BRUTE FORCE	Unauthorized Access
Web Application Exploits	WEB ATTACK - SQL INJECTION, WEB ATTACK - XSS	Code Injection/Data Theft
Reconnaissance/Other	PORTSCAN, HEARTBLEED, INFILTRATION	Information Gathering/Vulnerability Probing

This diversity facilitated per-attack Recall analysis during model stabilization.

2.0 Technical Rationale and Model Acquisition Strategy

The study adopted a BERT-based Transformer for its proficiency in sequence classification of serialized flow data. The model was sourced from a public repository (yashika0998/IoT-23-BERT-Network-Logs-Classification) to leverage pretrained weights, avoiding resource-intensive retraining.

2.1 Challenges in Custom Model Deployment

Attempts at local fine-tuning revealed critical issues:

1. **Tokenizer Dependency Failure:** The repository omitted essential tokenizer files (vocab.txt, tokenizer.json), causing AutoTokenizer.from_pretrained() to fail with a TypeError. A fallback to bert-base-uncased from Hugging Face was implemented, assuming vocabulary compatibility.
2. **Runtime Data Type Conflict (MPS Environment):** On Apple Silicon, tensor conversions to NumPy triggered TypeError: Got unsupported ScalarType BFloat16. Resolution involved explicit casting to torch.float32 prior to NumPy operations (.cpu().float().numpy()).
3. **Performance Collapse and Weight Bias**: Default threshold (0.5) yielded 0 True Positives and 0 Recall, indicating severe bias toward benign predictions. This stemmed from domain mismatch, rendering the model ineffective without adjustment.

2.2 Strategic Pivot to Threshold Optimization

Given irreproducible training, the focus shifted to inference benchmarking. A dynamic classification threshold was introduced to recalibrate the sigmoid output, optimizing for Recall while controlling False Positives.

3.0 Methodology: Data Preparation and Benchmark Implementation

3.1 Hardware and Software Environment

Benchmarks compared architectures under PyTorch, emphasizing backend-specific optimizations:

CPUs:

- AMD Ryzen 5 1600X: *6 cores/12 threads, 3.6 GHz base, 95W TDP.*
- AMD Ryzen 9 3900X: *12 cores/24 threads, 3.8 GHz base, 105W TDP.*
- Apple M1: *8-core CPU (4 performance + 4 efficiency), integrated GPU, up to 3.2 GHz.*
- Apple M3: *8-core CPU (4 performance + 4 efficiency), improved neural engine, up to 4.05 GHz.*

GPUs:

- NVIDIA RTX 2080: *8GB GDDR6, 2944 CUDA cores, 2150W TDP.*
- AMD Radeon RX 6800 XT: *16GB GDDR6, 72 Compute Units (4608 Stream Processors), 300W TDP.*
- NVIDIA A100: *40GB HBM2e, 6912 CUDA cores, 400W TDP.*

Full Hardware Specs

- Apple Macbook Air M3
- Apple Macbook Air M1
- Personal Gaming Computer (Alex):
  - OS: Windows 10 Pro
  - Processor: AMD Ryzen 5 1600 (Six Core, 3200Mhz, 6 core, 12 Logical
  - GPU: NVDIA GeForce RTX 2080
  - Memory: 24 GB 2400MHz
- Personal Gaming Computer (Nathaniel):
  - OS: Windows 10 Home
  - Processor: AMD Ryzen 9 3900X (3793 Mhz, 12 Cores, 24 Logical Processors)
  - GPU: AMD Radeon RX 6800 XT
  - Memory: 32 GB 2133 MHz
- Colab A100 (Professional Grade):
  - GPU: NVIDIA A100 (Ampere Architecture)
  - Deep Learning Cores: 6,912 CUDA Cores
  - Tensor Cores: Third-generation
  - Memory: 40GB HBM2e ($\sim 1,555$ GB/s bandwidth)
  - FP32 Performance: 19.5 TFLOPS
  - Deep Learning TFLOPS: 312 TFLOPS (TF32)

3.2 Data Preprocessing and Load Management

1. **Data Source:** 2,830,743 flows from CIC-IDS2017 CSVs.
2. **Input Pipeline:** Pandas-based load_and_preprocess_data() concatenated files and generated text inputs (e.g., "dest_port is 80. protocol is tcp. fwd_pkts is 2...").
3. **Benchmarking Workload:** Uniform batch size of 256 maximized parallel processing.

3.3 Threshold Optimization for Model Quality

Iterative testing identified an optimal threshold:

Threshold	True Positives (TP)	False Positives (FP)	Recall (TP/(TP+FN))	Precision (TP/(TP+FP))	Conclusion
0.5	~127,900	~94,800	99.90%	57.40%	Too aggressive (High FP)
0.8	~123,500	~80,800	96.50%	60.50%	High FP still unacceptable
0.9	~86,400	~53,200	67.50%	61.90%	Too conservative (Low Recall)
0.85	~115,500	~71,000	90.20%	61.90%	Selected Optimal Balance

The 0.85 threshold ensured a consistent 90.2% Recall across platforms.

3.4 Benchmark Testing Methodology

The benchmarking process was implemented using a custom Python script designed to evaluate the BERT-based NID model's inference performance across various hardware architectures. The script leverages PyTorch for model execution, Hugging Face Transformers for tokenization and model loading, and Pandas for data handling. It supports command-line arguments for device selection (e.g., --device cuda for NVIDIA GPUs, --device mps for Apple Silicon, or --device cpu for CPU fallback) and classification threshold override (e.g., --threshold 0.85). The methodology emphasizes reproducibility, efficiency optimizations (such as mixed-precision computing), and detailed metric reporting. Below is a step-by-step breakdown of the testing procedure:

- **Configuration and Argument Parsing:** Global variables define key paths (local model files, test logs directory) and defaults (batch size of 256, output files for classifications and metrics). Command-line arguments are parsed to specify the target device and threshold, allowing flexible runs without code modifications.
- **Device Detection and Setup:** The script automatically detects available hardware or enforces a specified device. For CUDA-compatible NVIDIA GPUs (e.g., RTX 2080), it enables FP16 mixed precision if the compute capability is ≥7. For Apple MPS, it uses BF16. CPU falls back to FP32. This ensures optimal data types for each backend, maximizing throughput while maintaining numerical stability.
- **Data Loading and Preprocessing:** CSV files from the CIC-IDS2017 dataset (located in the test_logs directory) are loaded iteratively using Pandas. Required feature columns (e.g., Destination Port, Flow Duration, TCP flags) are validated, and invalid files are skipped. Each flow record is transformed into a descriptive text sentence (e.g., "dest_port is 80. protocol is tcp. flow_duration is 12345...") to suit the BERT model's NLP input format. Ground truth labels are extracted for post-inference evaluation. Preprocessing time is measured separately to isolate I/O overhead from model execution.
- **Model and Tokenizer Loading:** The tokenizer is loaded from the Hugging Face hub (bert-base-uncased) to handle text encoding. The pretrained BERT model is loaded from local files, with label mappings overridden to "BENIGN" (0) and "MALICIOUS" (1). The model is moved to the selected device, converted to the appropriate dtype, and set to evaluation mode (torch.no_grad()) for inference-only operation.
- **Batch Inference Execution**: The dataset is processed in batches of 256 to leverage parallel computing on GPUs/NPUs. Inputs are tokenized (padded/truncated to max_length=512) and transferred to the device. Inference is timed precisely: CUDA uses torch.cuda.Event for millisecond accuracy, while others rely on time.time(). Autocast enables mixed-precision acceleration. Logits are computed, softened to probabilities via softmax, and classified based on the malicious class probability exceeding the threshold (default 0.85). Predictions, confidences, and ground truths are written to a JSONL file for traceability.
- **Progress Monitoring and System Metrics:** A tqdm progress bar tracks batch completion, updating with real-time throughput (logs/sec). Although system metric logging (e.g., via psutil or pynvml for CPU/GPU utilization) is commented out in the current implementation, it can be enabled for deeper hardware profiling in future runs.

- **Performance Metric Calculation and Reporting:** Post-inference, aggregate metrics are computed: total logs processed, preprocessing time, inference time, total pipeline time, throughput (logs/sec), and average latency (ms/log). These are printed to the console and saved as a structured JSON report (benchmark_report.json) for easy parsing and comparison across runs. The script recommends running a separate evaluation script (evaluate.py) on the output JSONL to derive model quality metrics like Recall, Precision, and confusion matrices.

This methodology ensures consistent testing across architectures, isolating hardware-specific performance differences while accounting for domain shifts through threshold tuning.

3.5 Encountered Issues

For initial testing with non-CUDA core systems such as the Apple ARM Architecture, and Ryzen CPU architecture, throughput was extremely slow, with the M3 architect taking roughly 4 hours to process roughly 390,000 logs. Additionally, power consumption and hardware utilization was extremely high, with GPU usage on the M3 at a consistent 98-100% throughout the entire processing time, and extremely high temperatures. For fear of damage to personal devices, and to combat the long compute times, a subsampled dataset ⅙ the size of the original (471,791 records) was used in order to off put extended execution times and hardware damage. All benchmarks were conducted on consumer-grade hardware without external optimizations like distributed processing, focusing on real-world deployability.

4.0 Results and Performance Analysis

Initial attempts to train a custom model on CIC-IDS2017 yielded inconsistent classification reports across epochs, likely due to suboptimal hyperparameter tuning or feature interpretation. Consequently, the pretrained IoT-23 model was adopted, stabilized via threshold optimization, and benchmarked.

4.1 Timing and Efficiency Metrics

NVIDIA RTX 2080 (Baseline)

DEVICE	cuda
DTYPE	torch.float16
TOTAL_LOGS	2830743
PREPROCESS_TIME (S)	163.7877486
INFERENCE_TIME (S)	2243.650566
TOTAL_PIPELINE_TIME (S)	2407.438315
THROUGHPUT (Logs/S)	1261.668391
LATENCY (S)	0.7926012946

AMD R9 3900X (⅙ dataset)

DEVICE	cpu (fallback)
DTYPE	torch.float32
TOTAL_LOGS	471791
PREPROCESS_TIME (S)	16.4071829
INFERENCE_TIME (S)	22747.6292951
TOTAL_PIPELINE_TIME (S)	22764.0364780
THROUGHPUT (Logs/S)	20.7402272
LATENCY (S)	48.2154795

Colab A100

DEVICE	cuda
DTYPE	torch.float16
TOTAL_LOGS	2830743
PREPROCESS_TIME (S)	114.9595907
INFERENCE_TIME (S)	469.372613
THROUGHPUT (Logs/S)	6030.907901
LATENCY_MS (S)	0.1658125139

Apple M3 (⅙ dataset)

DEVICE	mps
DTYPE	torch.bfloat16
TOTAL_LOGS	471791
PREPROCESS_TIME	10.30924392
INFERENCE_TIME	3254.913923
TOTAL_PIPELINE_TIME	3265.223167
THROUGHPUT	144.9473046
LATENCY	6.899058955

Apple M1 (⅙ dataset)

DEVICE	mps
DTYPE	torch.bfloat16
TOTAL_LOGS	471791
PREPROCESS_TIME	10.784521102905273

INFERENCE_TIME	14516.34091424942
TOTAL_PIPELINE_TIME	14527.125435352325
THROUGHPUT	32.50068338756664
LATENCY	30.768583788689103

4.2 Analysis of GPU Performances

The RTX 2080 achieved sub-four-minute processing for 2.83 million flows, driven by CUDA parallelism and FP16 tensor core acceleration, halving memory usage and doubling throughput. The Colab 100 (Google Colab CPU) provided improvements to every metric, while still analyzing the same amount of logs as our benchmark. This is due to the fact that the Colab 100 is the NVIDIA A100 GPU, which uses Tensor Cores instead of Cuda Cores. Tensor Cores are processing units specifically designed to accelerate AI/Deep-Learning workloads at a significantly higher rate (about 20 times more maximum) than any consumer Cuda Core GPU.

Future phases will extend comparisons to AMD CPU and Apple MPS, assessing scalar vs. vectorized efficiency and power-per-inference metrics.

4.3 Analysis of CPU Performances

The CPU performances were based on the dataset being cut by a factor of six. Due to the original dataset causing maximum memory usage, we cut the dataset in order to properly run our benchmarks. Our results are based on the benchmarks run on Apple M1 and M3 chipsets, and the AMD Radeon 6800XT CPU fallback.

Given the processing of the CPUs for a flow of ~471,000 flows, the Mac M3 chipset performed the best in every single category, while the AMD Radeon 6800XT CPU fallback performed the worst in every category. This is due to the fact that RDNA2 is not optimized to run any sort of modeling due to the lack of architecture for it. To run machine learning tasks on AMD's RDNA2 architecture, it requires a significant amount of performance drawbacks in everything else to solely focus on our benchmarks (just for the performance to be at most, a fourth as efficient as any NVIDIA GPU running CUDA), which would lead to other problems for the PC.

5.0 Conclusion

This benchmarking study underscores the feasibility of deploying BERT-based NID classifiers on heterogeneous hardware, despite domain-shift challenges. Threshold optimization mitigated initial failures, enabling reliable 90.2% Recall at the cost of moderate Precision. The NVIDIA RTX 2080 baseline highlights GPU superiority for high-throughput scenarios, while upcoming Apple Silicon and AMD CPU evaluations will illuminate trade-offs in energy efficiency and scalability. These insights inform optimized NIDS deployments, balancing detection efficacy with computational constraints in dynamic threat landscapes. Future work could explore quantization, distillation, or federated learning to further enhance cross-domain robustness and edge compatibility.