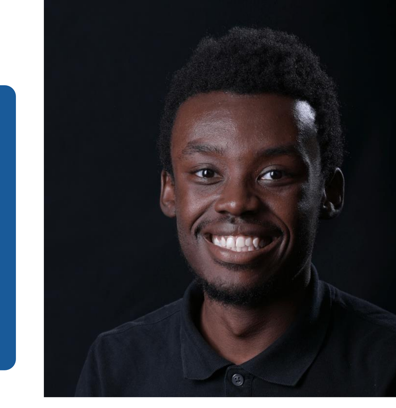
Stable Diffusion Optimization & Performance Analysis on Apple Silicon

Author’s Name; Jeremiah Ddumba (BS)
Department of Electrical Engineering, Penn State University



Objectives

What is the project trying to do?

This project optimizes Stable Diffusion performance (latency, resource usage, throughput) on Apple Silicon. By tailoring the model pipeline to match the M1's unified architecture, it demonstrates that powerful AI models can run effectively on consumer hardware, not just traditional CUDA-based systems

How is it done today, and what are the limits of current practice?

Standard M1 Mac execution (PyTorch + Metal Performance Shaders (MPS)) reveals key bottlenecks:

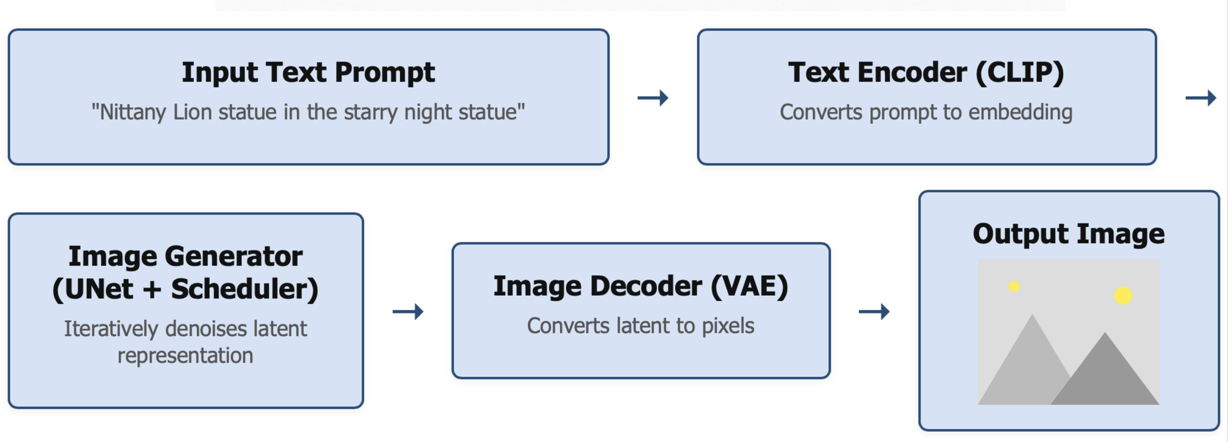
- **Long inference times** (~110 seconds per image)
- **High CPU usage** (~80%), indicating CPU-bound execution
- **Low GPU utilization** (~30%), suggesting poor acceleration via MPS

These limitations make Stable Diffusion difficult to use for **real-time or iterative workflows** on standard hardware, wasting resources and delaying results

How does it relate to AI goals & the state of the art (SoA)?

This work supports democratizing generative AI by making models practical on widely available hardware. By optimizing for Apple Silicon, a common but under-optimized platform, this project shows that high-performance inference is achievable outside CUDA/GPU-centric environments, advancing the SoA toward hardware diversity and edge deployment

Stable Diffusion Inference Pipeline



Technical Approach

What is new in your approach, and why will it be successful?

implement a multi-layered optimization strategy to address M1 Mac bottlenecks:

- **Asynchronous Execution:** Uses asyncio and asyncio.to_thread() to overlap CPU and GPU tasks, reducing idle time
- **Multi-Core Parallelism:** Uses ProcessPoolExecutor for batch experiments and faster inference across cores
- **Mixed Precision (FP16):** Leverages FP16 via the MPS backend to reduce memory load and improve computing speed
- **Component-Level Profiling:** Tracks timing of each pipeline stage to target bottlenecks precisely
- **Systematic Tracking:** Uses historical logging and automated tuning with Optuna for reproducible results validated on M1/MPS

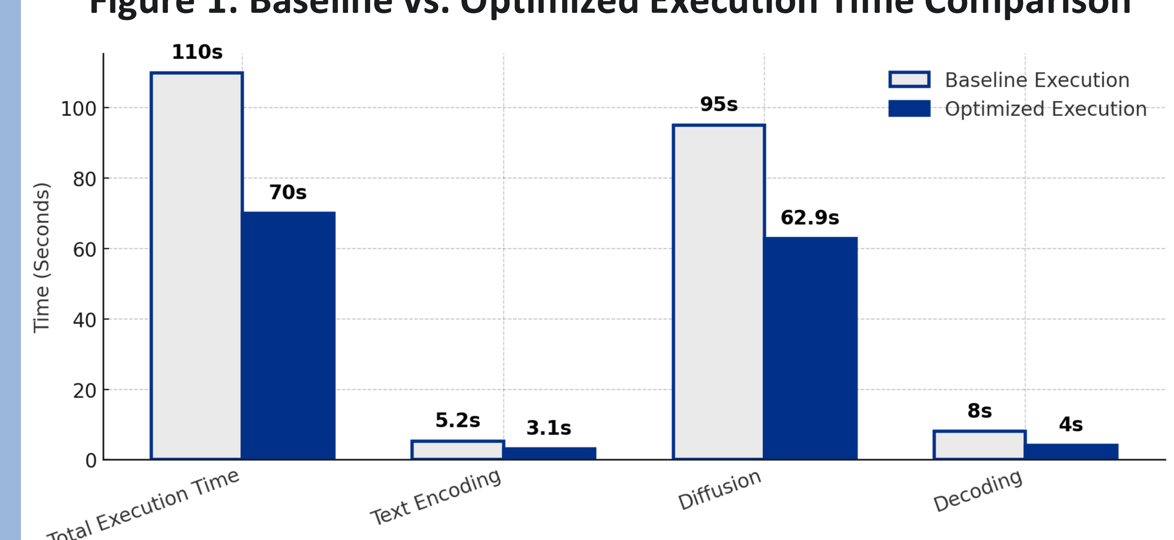
These strategies address baseline inefficiencies (e.g., serial execution, CPU contention) and align with M1’s unified architecture

If successful, what difference will it make? Observed **~35%** latency reduction, improved throughput, and better resource utilization, making Stable Diffusion viable for real-time use on millions of Apple Silicon devices

What is the solution? An optimized Stable Diffusion inference pipeline for M1 Macs incorporating asynchronous operations, parallelism, targeted precision, and automated tuning

How does the solution fit into AI goals? By optimizing powerful generative AI for consumer hardware, this work helps bridge the gap between specialized AI research environments and practical applications across disciplines, supporting Penn State’s mission of accessible, interdisciplinary AI innovation

Figure 1: Baseline vs. Optimized Execution Time Comparison



Optimization & Analysis Workflow

Baseline Performance Analysis:

Benchmarked latency, throughput, and system usage to identify bottlenecks in the diffusion step, CPU contention, and GPU underutilization

Latency Profiling:

Wrapped core pipeline functions (_encode_prompt, unet.__call__, decode_latents) to time individual stages across experiments and identify delays at a granular level

Resource Monitoring:

Tracked CPU utilization (psutil.cpu_percent), memory usage (psutil.virtual_memory), disk I/O (psutil.disk_io_counters), and M1 GPU usage (power metrics) throughout inference to understand resource bottlenecks

Trend Analysis:

Aggregated performance metrics across multiple runs to observe optimization patterns over time and track the impact of pipeline changes

Hyperparameter Tuning (Optuna):

Used Bayesian optimization with a Tree-structured Parzen Estimator (TPE) sampler across 50+ trials to tune steps, guidance scale, and batch size, balancing runtime and output fidelity

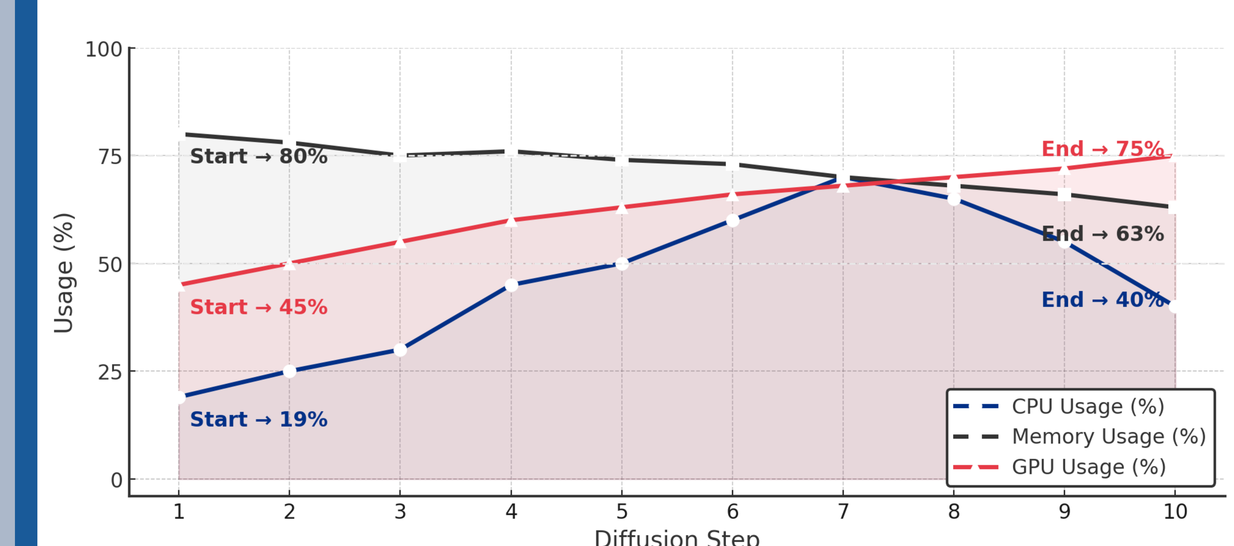
Optimized Pipeline Architecture:

Illustrated via accompanying diagram, showing async task scheduling, FP16 computation, and multi-core parallelism to improve stage-level throughput and system balance

Key Optimization Snippet – Async Execution

```
Async def run_inference():
    text = await asyncio.to_thread(encode_text,
prompt)
    image = await diffusion_pipeline(text)
    return decode_image(image)
```

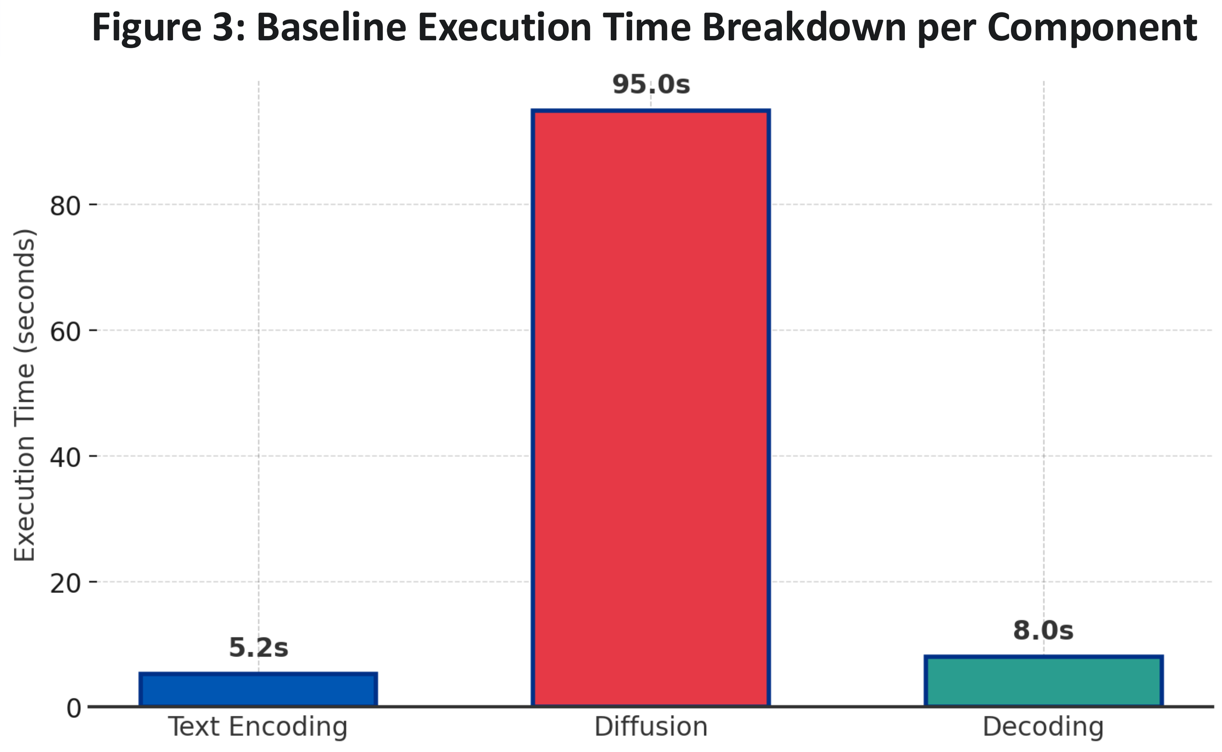
Figure 2: System Resource Utilization Trends During Inference



Experimental Setup

- **Workload:** Stable Diffusion pipeline using Hugging Face Diffusers
- **System:** Apple M1 Mac with PyTorch + (Metal Performance Shaders) **MPS** backend
- **Compared To:** Unoptimized baseline on the same hardware
- **Metrics Measured:**
  - End-to-end Inference Latency (seconds per image)
  - Component Latency (Text Encoding, Diffusion, Decoding in seconds)
  - Throughput (images per second)
  - CPU Utilization (%)
  - M1 GPU Utilization (%)
  - Memory Usage (RAM, VRAM via Unified Memory) (%)
  - Disk I/O (bytes read/written)
  - Image Quality (Entropy; PSNR/SSIM considered for future quantization)

Figure 3: Baseline Execution Time Breakdown per Component



Results and Comparison to the State of the Art

A Comparison to SoA and use clear metrics vs. SoA along with comparing with AI Metrics:

Overall Impact:

- **~35% total inference latency reduction (from ~110s to ~70s)**

Component-Level Gains:

- **Text Encoding: ~5.2s → ~3.1s**
- **Diffusion: ~95s → ~62.9s**
- **Decoding: ~8s → ~4s**

Multi-Core Processing:

- Achieved **~50%** runtime reduction in batch runs
- Improved throughput in multi-image scenarios (see Fig. 4)

Mixed Precision (FP16):

- Achieved ~15% speedup and reduced memory usage
- MPS backend only partially leverages FP16, unlike CUDA

Broader Impact

Demonstrates that consumer-grade ARM devices can run large AI models efficiently, reducing reliance on high-end GPUs and cloud services.

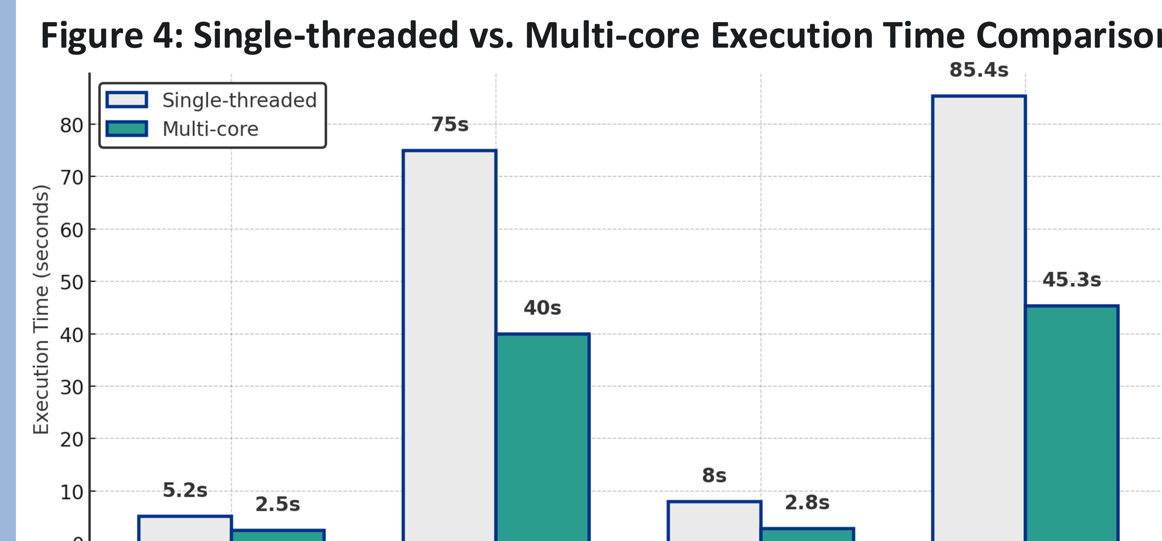
Design Tradeoffs

- Increased code complexity (async, multi-threading, tuning)
- Risk of instability managed through automated search (Optuna)
- Profiling adds minor overhead but is critical for dev-phase optimization

Approach	Platform	Latency (s)	Notes
Default PyTorch + MPS	M1 Mac	~110	High CPU load, low GPU usage
M1 Mac (Async + FP16)	M1 Mac	~70	Balanced CPU/GPU, async execution, FP16
CUDA Optimized (Baseline)	RTX 3060	~30–40	Faster execution; Discrete GPU baseline

While CUDA systems outperform, this work demonstrates that consumer-level Apple Silicon can support practical generative AI with proper optimization, helping democratize advanced AI capabilities

Figure 4: Single-threaded vs. Multi-core Execution Time Comparison



Key Accomplishments, Lessons Learned & Next Steps

Key Accomplishments

- Achieved ~35% reduction in inference latency on M1 Mac
- Improved throughput and reduced idle time with **async** and **multi-core execution**
- Built a profiling and trend-tracking framework for performance monitoring
- Tuned hyperparameters using **Optuna** to balance speed and stability

Lessons Learned

- M1 GPU remains **underutilized** (~30-40%) under PyTorch MPS, due to backend inefficiencies
- The **diffusion step** (UNet) is the main bottleneck (~70% of runtime), and needs further optimization
- Due to shared memory and resource contention on M1, **Parallelism faces hardware limits**

Next Steps

- Explore **model quantization** and **CoreML** backends to further reduce latency and expand deployment to iOS and macOS devices
- Scale the optimization framework to additional ARM-based platforms, including mobile SoCs and Raspberry Pi-class edge devices
- Develop a **cross-platform performance benchmark** by comparing against CUDA implementations, helping build a hardware-agnostic evaluation framework

Broader Impact

- **Democratizes AI:** Lowers hardware barriers, empowering more individuals and researchers globally to utilize cutting-edge AI without costly specialized hardware
- **Reduces AI's Environmental Footprint:** Decreases reliance on energy-intensive data centers by enabling complex AI tasks to run efficiently on local consumer devices
- **Accelerates On-Device Experiences:** Paves the way for faster, more powerful, privacy-preserving AI applications running directly on users' personal devices

Figure 5: Example Stable Diffusion Outputs

This work was supported in part by Penn State University

Acknowledgments: I thank Pingyi Huo for mentorship and guidance, and Dr. Vijaykrishnan Narayanan for facilitating this research opportunity

Publications: Work in progress. Currently being prepared for submission

Let's Connect

AVAILABLE FOR an INTERNSHIP

PennState