

# Nathan Teshome

Cambridge, Massachusetts | [nnt@mit.edu](mailto:nnt@mit.edu) | +1 617 719 0373 | [linkedin.com/in/nathan-teshome-84b675325](https://www.linkedin.com/in/nathan-teshome-84b675325) | [github.com/nnt-git13](https://github.com/nnt-git13)

## EDUCATION

### Massachusetts Institute of Technology (MIT)

*B.S. Electrical Engineering; B.S. Mathematics; GPA: 4.60/5.00*

Cambridge, MA

*Anticipated 2028*

- Pursuing NEET Autonomous Machines Certification.
- **Currently Enrolled:** Computation Structures; Semiconductor Electronic Circuits; Advanced Real Analysis; C & Assembly; Statistics.
- **Completed:** Algorithms; Probability; Machine Learning; Circuits; Differential Equations; Linear Algebra; Electricity & Magnetism; Fundamentals of Programming (Python).

## WORK EXPERIENCE

### Satellite Swarm Reinforcement Learning Control — MIT

*Undergraduate Researcher*

Cambridge, MA

*Dec. 2026 – Ongoing*

- Ported EMFF swarm dynamics + RL controller to a fully GPU-resident pipeline; increased sample throughput by 10–15×. Implemented vectorized orbital + EMFF integrator (Clohessy–Wiltshire + dipole interactions) scaling to 4,096+ trajectories and >1M env steps/sec.
- Redesigned PPO/SAC stack with custom rollout buffers, batched normalizers, and GPU-native updates; reduced training time from ~3 days to <8 hours.
- Prototyped attention-based multi-agent policies (transformer, graph attention) achieving 20–30% faster convergence and 15% lower  $\Delta V$  usage. Integrated superconducting quench-limit constraints into reward structure; achieved >98% safe rollouts.

### MIT Teaching Assistant Positions

*Undergraduate TA*

Cambridge, MA

*Aug. 2025 – Ongoing*

- **Classical Mechanics (8.01):** Led TEAL-format discussions, simulations, and problem-solving workshops; supported conceptual mechanics learning.
- **Intro to CS using Python (6.1000/6.100A):** Taught weekly labs on testing, code organization, algorithmic reasoning; graded problem sets and provided actionable feedback.

## ACADEMIC & PERSONAL PROJECTS

### FlashFlow: GPU-Optimized Transformer Inference Runtime

*GitHub: [github.com/nnt-git13/flashflow](https://github.com/nnt-git13/flashflow)*

- Designed and trained a GPT-style Transformer matching a strong PyTorch baseline within <0.5 perplexity while maintaining numerical fidelity.
- Implemented fused CUDA/Triton kernels (attention/MLP) using tiling, shared-memory blocking, and warp reductions; achieved 1.6–2.4× kernel-level speedups and 2.3× lower memory traffic.
- Built a compiler-style C++ scheduling runtime lowering FX graphs into fused operators with autotuned tile shapes and layouts; achieved 1.4–1.9× tokens/sec over PyTorch eager and 1.2–1.5× over Inductor.
- Applied FP16/BF16 and INT8 quantization; measured up to 3.1× arithmetic throughput improvement with negligible perplexity degradation. Redesigned tensor layouts (row-major → block-swizzled) improving L2 hit rate by 28% and reducing warp div by 15–22%.

### RISC-V Vector CPU + Quant Finance GPU Accelerator + Custom PCB

*GitHub: [github.com/nnt-git13/TODO](https://github.com/nnt-git13/TODO)*

- Implemented a full RV32I soft core with custom vector opcodes and memory-mapped accelerator interface on an Artix-7 XC7A100T FPGA.
- Built a 16–32 lane SIMD Monte-Carlo pricing accelerator with RNG pipelines, LUT-based exp/sqrt, fixed-point math, and reduction trees; reached 150M+ paths/sec with ≤1% error.
- Developed heterogeneous execution model enabling warp-synchronous vector launches via BRAM-resident CSRs and custom RISC-V opcodes.
- Designed a custom 4-layer PCB (regulators, PLL, level-shifting, high-speed PMOD interfaces), in built with Python golden models + SystemVerilog benches + cycle-accurate simulation achieving 100×+ speedups over scalar software.

### Mini-TPU: MLIR Compiler + Programmable Tensor-Core Accelerator

*GitHub: [github.com/nnt-git13/TODO](https://github.com/nnt-git13/TODO)*

- Designed a programmable tensor-core accelerator (16×16 systolic array, dual-buffered SRAM, DMA engine) supporting FP16/BF16/INT8. Implemented MLIR backend with custom tensor dialect, tiling/fusion, memory planning, quantization-aware lowering.
- Achieved 2.8–4.6× INT8/INT4 speedups with ≤0.25 perplexity degradation; reached 91–95% of theoretical throughput.
- Built host runtime + FPGA prototypes validating end-to-end compute with ≤1.7% numerical error vs PyTorch eager.

## TECHNICAL SKILLS

**Programming:** Python, C, C++, SystemVerilog/Verilog, Bash; ROS 2, OpenCV, PyTorch, NumPy/Pandas, LLVM/MLIR, gem5, RLLib/Gymnasium; Django/REST, JavaScript, Playwright; CI/CD (pytest, CMake, GitHub Actions).

**Hardware:** FPGA (Zynq-7000/ZCU104), Vivado/Vitis, PYNQ; AXI4/AXI-Lite/AXI-Stream, BRAM/DSP48; cocotb, UVM-lite, riscv-dv, SymbiYosys; oscilloscopes, logic analyzers, JTAG; KiCad PCB design, soldering/reflow, SMA/LVDS.

**Tools:** Linux, Docker, Git, SLURM, Onshape, RViz2, 3D printing/rapid prototyping.