

Rohan N. Mengle

U.S. Citizen · rmengle@gmu.edu · github.com/Megz-Mugz · rohanmengle.com

EDUCATION

George Mason University

Fairfax, VA

M.S. in Computer Science, Thesis Track

Expected June 2027

B.S. in Applied Computer Science, Concentration in Software Engineering

December 2025

Relevant Coursework: Compilers, Advanced Computer Architecture, Operating Systems, Computer Systems and Programming, Linear Algebra, Discrete Mathematics, Data Mining, Distributed Systems (planned Spring 2027)

RESEARCH INTERESTS

- **Fault Tolerance and Reliability:** Developing techniques to detect, understand, and mitigate faults in software and computing systems, with potential applications to emerging workloads such as large language models.
- **Compilers:** Developing compiler techniques for heterogeneous systems and emerging machine learning, artificial intelligence, and deep learning workloads, with an emphasis on performance, portability, and resource efficiency.
- **Computer Architecture:** Exploring how architectural design and hardware–software interactions influence the performance, reliability, and efficiency of high-performance computing systems.
- **High-Performance Computing:** Analyzing performance, scalability, and resource-efficiency tradeoffs in parallel scientific applications and large-scale computing systems.

PUBLICATIONS

- **R. Mengle**, M. Tang, B. Zhang, Z. Zhu, G. Li, L. Yang. “From Compiler Optimizations to Code Patterns: Resilience and Performance of Compiled Programs” [**Under Review - SIGMETRICS**]

RESEARCH EXPERIENCE

George Mason University

September 2025 – Present

Researcher - Advised by Dr. Lishan Yang

Project #1 - Understanding Resilience & Performance of Compiler Optimizations

- Identified, quantified, and systematically characterized five novel, recurring x86 machine-code patterns linking compiler optimizations to resilience–performance tradeoffs: **Direct Memory Update**, **Signed Jump**, **Dead Value**, **Reuse of a Loaded Value**, and **Reuse of a Hoisted Value**. Explained how register exposure, instruction semantics, and value reuse influence fault masking and propagation.
- Developed an automated static analyzer using Python to detect the five patterns directly from x86 disassembly without fault injection. Analyzed 1,187 executables and found multiple patterns co-occurring in 79.3%, revealing why individual patterns alone cannot reliably explain whole-program resilience.
- Used Intel Pin to trace fault-induced control-flow divergence across 111 compiled versions each of **LULESH** and **heFFTe**, with 500 fault injections per executable. Identified a novel correlation between control-flow divergence and non-benign outcome rates across compiler configurations (Pearson $r = 0.980$ and 0.984 , respectively), linking greater control-flow divergence to lower resilience.
- Characterized instruction-level error masking through 116,000 targeted fault injections across 10 benchmarks, revealing substantial differences in resilience and control-flow divergence across opcodes. Faults targeting **cmp** instructions resulted in 88.3% benign outcomes and 8% control-flow divergence, compared with 22–33% benign outcomes for **add**, **sub**, and **imul**, demonstrating stronger error masking for comparison instructions.
- Conducted over 1.5 million single-bit fault injections into destination registers of dynamically executed x86 instructions. Classified outcomes as benign, silent data corruption, crashes, or hangs, and evaluated resilience–performance tradeoffs using 500 fault-free timing runs per executable.
- Built a C++, Python, and Bash pipeline using LLVM, PinFI, and Intel Pin, automating fault-injection campaigns, outcome classification, and performance measurement across hundreds of executables.

Project #2 - Selective Triple Modular Redundancy in LLVM & MLIR

- Developing a novel control-flow-guided Triple Modular Redundancy (TMR) framework in LLVM, using a weighted control-flow graph and a knapsack algorithm to select vulnerable def-use chains for protection while maximizing reliability within an execution-time overhead budget.
- Implementing LLVM IR passes in C++ for selective instruction replication and majority voting. Evaluating reliability gains and performance overhead using LLFI, an LLVM IR fault injector, and execution-time measurements.
- Extending selective TMR to MLIR to study how protection decisions interact with lowering and optimization across compiler abstraction levels.

TEACHING EXPERIENCE

George Mason University

January 2024 – May 2025

Undergraduate Teaching Assistant

SWE 437 - Software Testing (Spring 2025)

CS 262 - Introduction to Low-Level Programming (Spring 2024 & Fall 2024)

- Answered student questions on Piazza and during office hours, helping them clarify course concepts, debug assignments, and work through problems in software testing and low-level programming.

SOFTWARE ENGINEERING EXPERIENCE

Mathtech

May 2026 – August 2026

Software Engineer Intern

- Developed full-stack features for an enterprise management platform using Laravel, PHP, Blade, and JavaScript, streamlining administrative workflows through integrated user interfaces and backend functionality.
- Modernized a legacy return merchandise authorization (RMA) system by redesigning its workflow and implementing supporting features in Laravel to simplify return processing.
- Automated administrative operations using Laravel and MariaDB, implementing database migrations, complex SQL queries, and backend business logic to reduce manual processing.

Solution Street

March 2024 – December 2025

Software Engineer Intern

- Engineered a lead-ingestion pipeline using Ruby on Rails and JavaScript, delivering a public REST API with asynchronous request handling that supported over \$500,000 in revenue within six months.
- Architected a role-based scheduling and payroll system in Ruby on Rails and redesigned its PostgreSQL database to support concurrent event allocation while preserving data integrity for over 300 employees.
- Enhanced an image-hosting platform built with React, TypeScript, and Amazon S3; developed an AWS health check that uncovered a critical security vulnerability in IP-whitelist enforcement.

PROJECTS

SQLike: Heterogeneous Database Engine

- Developed an in-memory database engine in C++ using LLVM and CMake to compile SQL-like queries into native machine code. Extending the engine to support heterogeneous CPU/GPU query execution and multithreaded persistence to non-volatile storage.

Rusty Programming Language

- Developed *Rusty*, a Rust-like programming language written in C using Lex and YACC. Implemented a symbol table and type checking, with support for loops, conditionals, and heap-allocated arrays.

SKILLS

Programming Languages: C++, C, Python, CUDA, Ruby, PHP, Java, JavaScript, TypeScript, SQL

Compilers & Systems: LLVM, MLIR, Intel Pin, PinFI, gem5, x86-64 Assembly, Linux, SLURM, Wireshark

Software & Data: PyTorch, Ruby on Rails, Laravel, React, PostgreSQL, MariaDB, REST APIs, NumPy, Pandas

Developer Tools: Git, CMake, GoogleTest, GDB, LLDB, Docker