Oscar (Jiaye) Fang

PhD Student in Programming Languages and Program Analysis

oscar.fang@gwu.edu | Website | GitHub | LinkedIn

(530) 760-9441 | Washington, DC

Research Interests

Programming languages, program analysis, and compiler infrastructure, with an emphasis on pointer and alias analysis for Rust and LLVM.

Education

George Washington University PhD in Computer Science; advisor: Prof. Jie Zhou — 2025–2029 expected

George Washington University MS in Computer Science — 2023–2025

University of California, Davis BS in Computer and Information Science — 2018–2021

Selected Publication

Jiaye Fang, Arsalan Bin Najeeb, Dylan O'Neill, Prince Noah Johnson, Zhixiao Zhang, and Jie Zhou. "Dynamic Analysis of Unsafe Rust: Behaviors and Benchmarks." To appear in the *Proceedings of the 2026 ACM SIGOPS Annual Technical Conference* (ATC '26). Artifact.

Research Experience

Dynamic Analysis of Unsafe Rust

2023–2026

George Washington University — Accepted at ATC 2026

- Developed a compiler-based framework to study how unsafe Rust behaves at runtime. Carried source-level unsafety into LLVM IR and instrumented execution and heap accesses to relate unsafe regions to the memory objects they access.
- Characterized 100 popular crates containing unsafe code, showing that source-code prevalence alone does not capture runtime behavior: safe APIs invoke unsafe code, and incorporated standard-library code adds substantial unsafe execution.
- Curated an 18-crate benchmark for evaluating unsafe-Rust defenses, selecting workloads by observed unsafe execution, heap usage, and code coverage across diverse program types and unsafe patterns.

Pointer Analysis for Unsafe Rust

2025–Present

George Washington University — Advisor: Prof. Jie Zhou

- Investigated loss of heap-target information in Rust-generated LLVM IR using SVF's Andersen analysis integrated into `rustc`. Implemented demand-driven tracing through integer–pointer casts and stores to recover targets missed in *hashbrown*'s type-punned pointer representation.
- Evaluated recovered targets against runtime heap-access observations, separating missed targets from predictions not exercised by the test inputs. Identified type-erased, byte-offset accesses as a source of large candidate sets; integration with LLVM alias analysis is ongoing.

Characterizing LLVM Alias Analysis

2026–Present

George Washington University — Manuscript in preparation

- Built query-level instrumentation and provider-ablation experiments to investigate why LLVM returns `MayAlias` and which unresolved queries affect optimization, across Rust, C/C++, and SPEC CPU2017 compilation units.
- Used a deliberately unsound `NoAlias` intervention to compare changes in optimized IR: same-object queries had a larger load-count effect than distinct-object queries in the selected Rust corpus, with the ordering reversed in C/C++.
- Separated these code-shape experiments from executable alias-information ablations, showing why query counts and static load-count changes are insufficient proxies for optimization value and runtime performance.

Open Source Contributions

SVF — Static Value-Flow Analysis

2026

- Designed the `getMayAliases` interface and its Andersen implementation using reverse points-to sets and SCC membership; validated against exhaustive pairwise alias queries (merged #1888).
- Ported SVF to LLVM 21, recovering field constraints for opaque-pointer accesses and byte-layout memory copies (merged #1815). Fixed conservative handling of negative GEP offsets (#1805) and a `PointsTo` assignment memory leak (#1893); both merged.
- Optimized alias enumeration with direct reverse-set queries and dense-bitset accumulation: 4.78–43.64× query speedups on shared sampled LLVM values across four modules (#1894, under review).