

CS 3551: Project Presentation

Unsafe Rust in Different Domains

Abdulaziz Alotaibi



Computer Science Department
School of Computing and Information

Project Goal

- Understand and mitigate risks in unsafe Rust code by detecting and addressing memory bugs.
- Extend Rudra's application to analyze developer-written code across different domains, beyond just Rust's official packages.
- Apply the analysis to diverse domains (e.g., distributed systems, embedded systems, database systems).
- Help the Rust community identify and mitigate memory safety vulnerabilities in unsafe Rust.

Overview of Rudra

- **Static analysis tool for Rust:** Detects memory safety bugs in unsafe Rust code.
- **Focuses on Rust's official packages:** Designed to scale and analyze large-scale Rust ecosystems.
- **Key strengths:** Efficient at pinpointing vulnerabilities in unsafe blocks with minimal false positives.
- **Foundation for this project:** Extended Rudra to analyze developer-written code across diverse domains.

Approach and Methodology

Selecting Projects:

- Identified popular open-source Rust projects across four different domains (e.g., distributed systems, embedded systems) using GitHub.

Detecting Unsafe Rust:

- Explored tools like Carrot and Geiger to identify unsafe Rust but found them ineffective for large projects with complex dependencies.
- Developed a custom Python tool to accurately detect unsafe Rust code in these projects.

Approach and Methodology

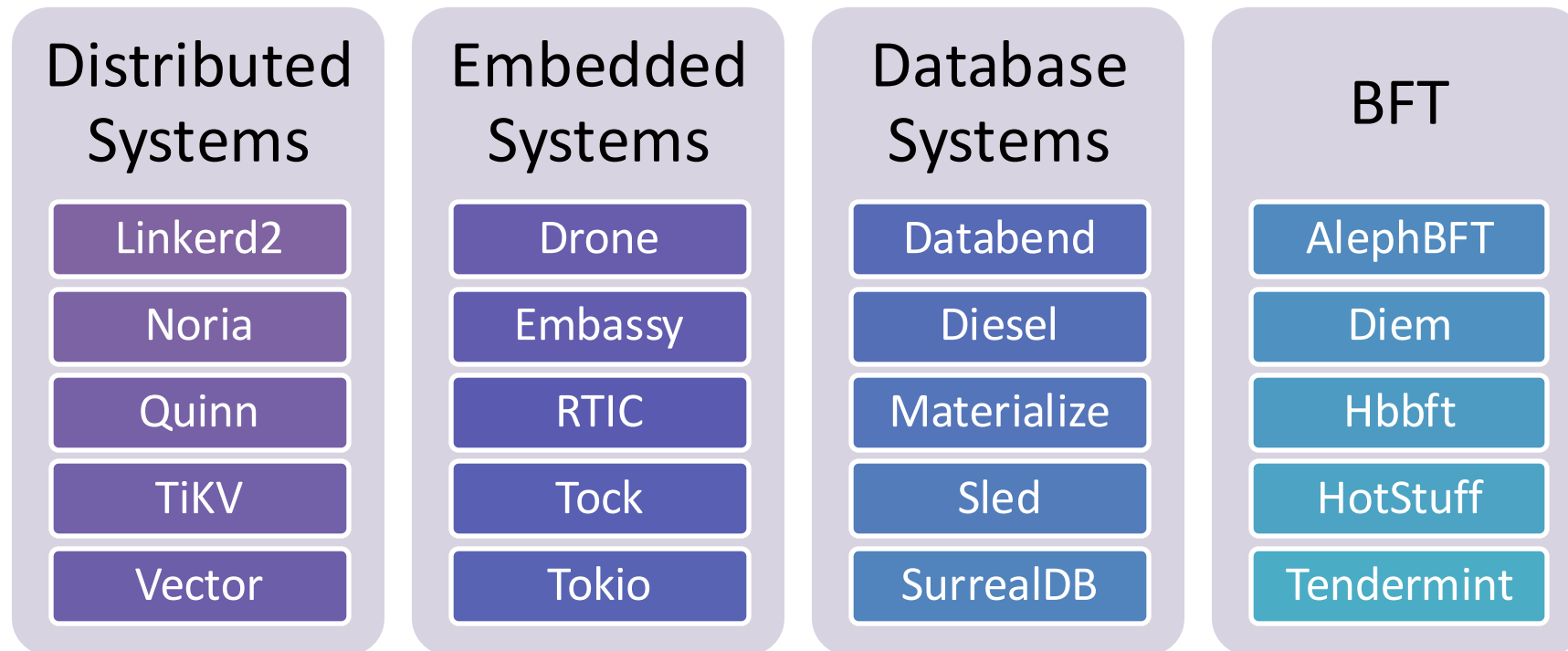
Analyzing Unsafe Patterns:

- Studied the detected unsafe Rust code to identify common patterns and categorized them into meaningful groups.
- Prioritized the most critical patterns for deeper analysis based on their potential impact and frequency.

Isolating and Testing:

- Extracted unsafe Rust code, isolated it from dependencies, and analyzed it using Rudra to detect potential memory bugs.

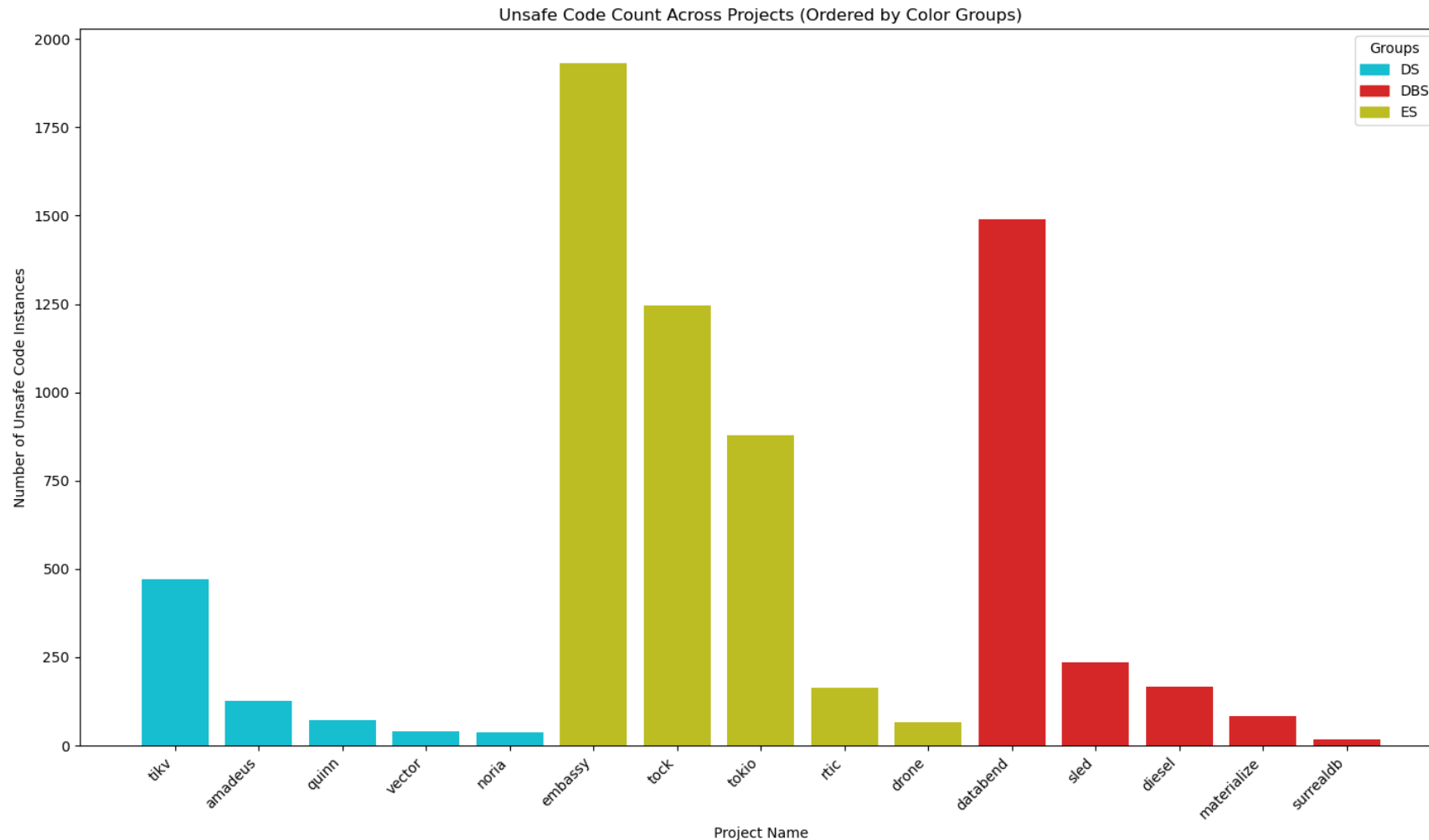
Different Domains



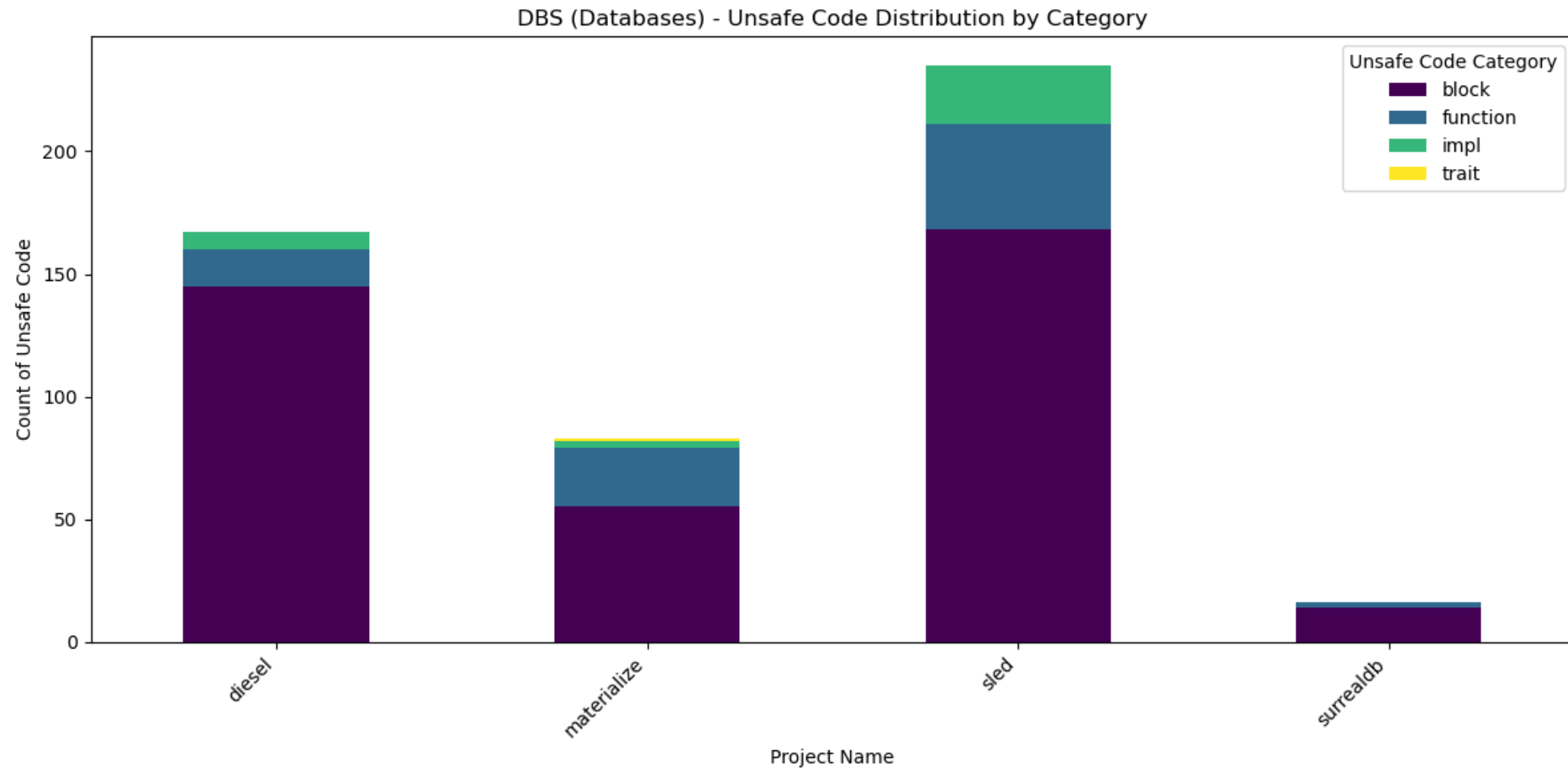
Key Results - Overview

- Unsafe Rust detection across projects in different domains.
- Analysis of unsafe Rust structures: blocks, functions, traits, and implementations.
- Categorization of unsafe Rust patterns and their significance.

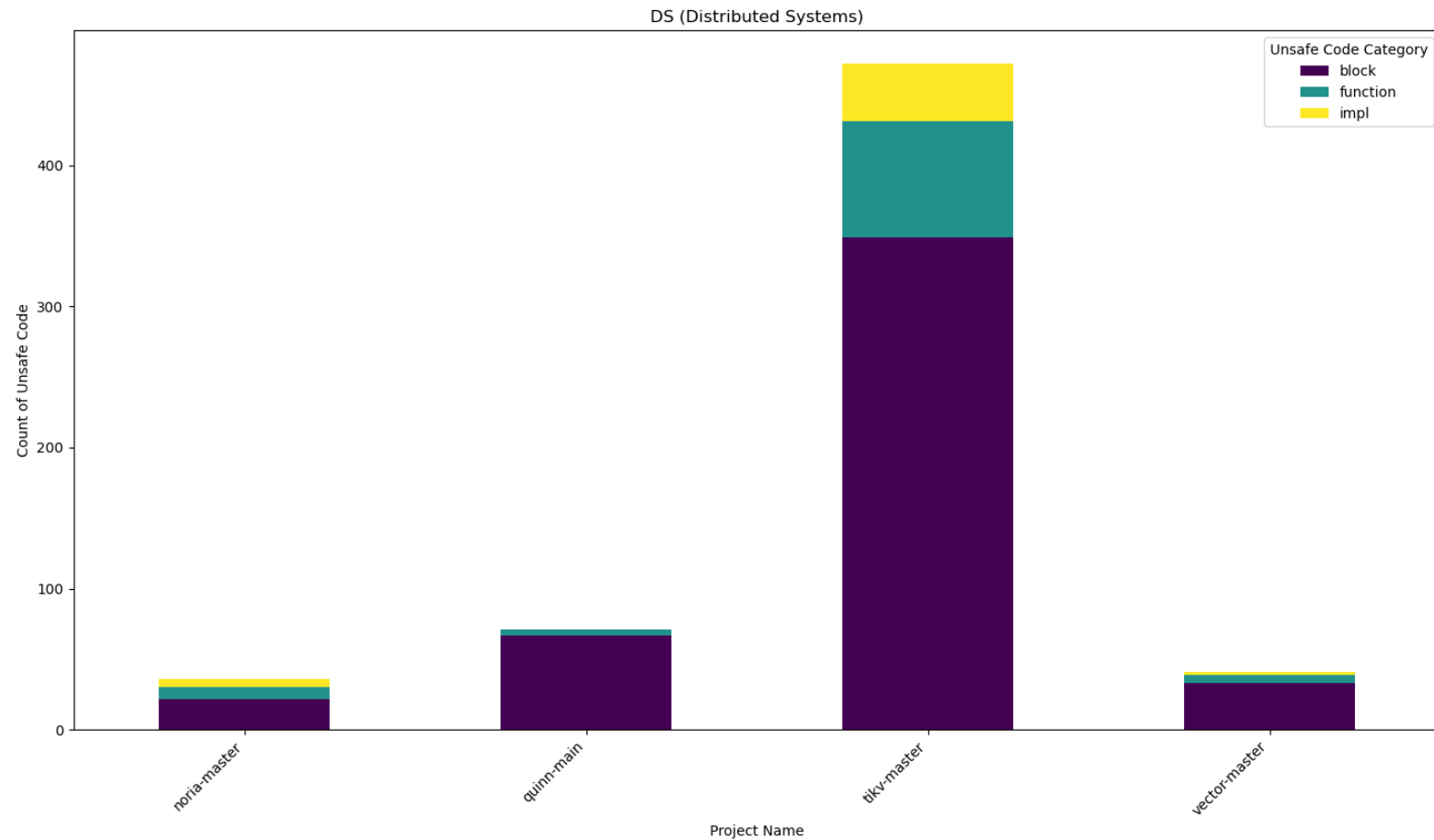
Key Results - Unsafe Rust Across Domains



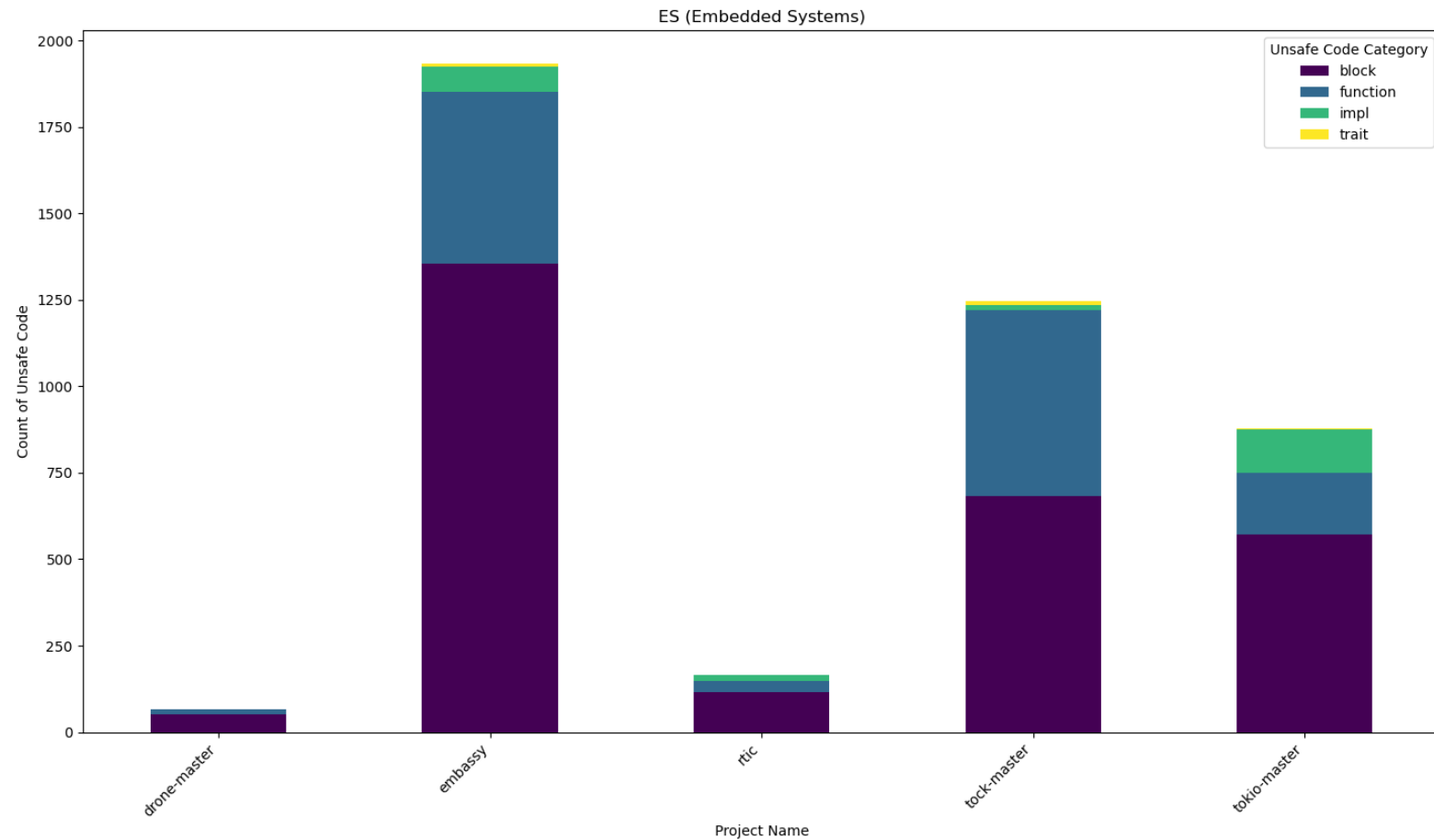
Key Results - Unsafe Rust Structures



Key Results - Unsafe Rust Structures



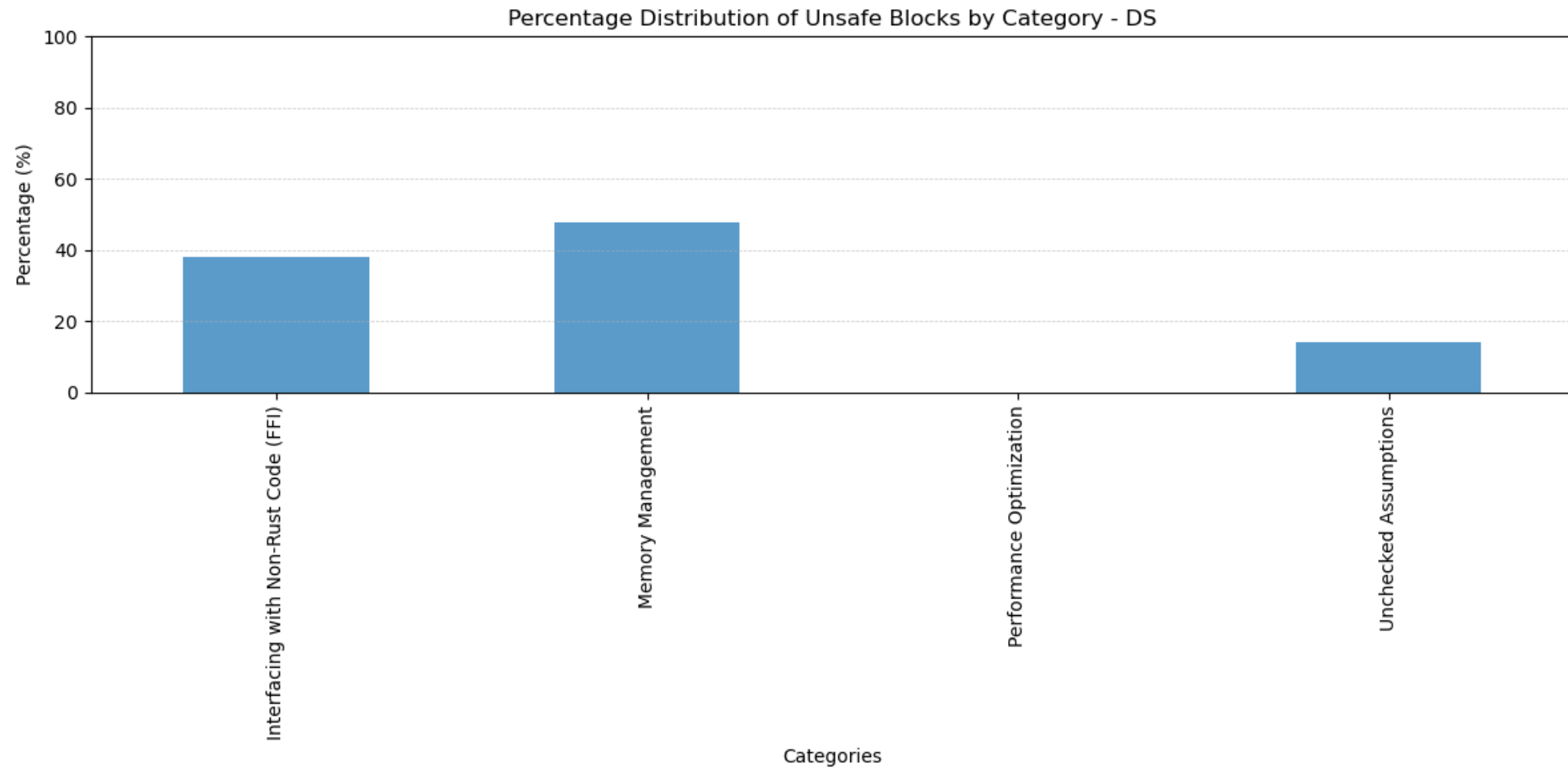
Key Results - Unsafe Rust Structures



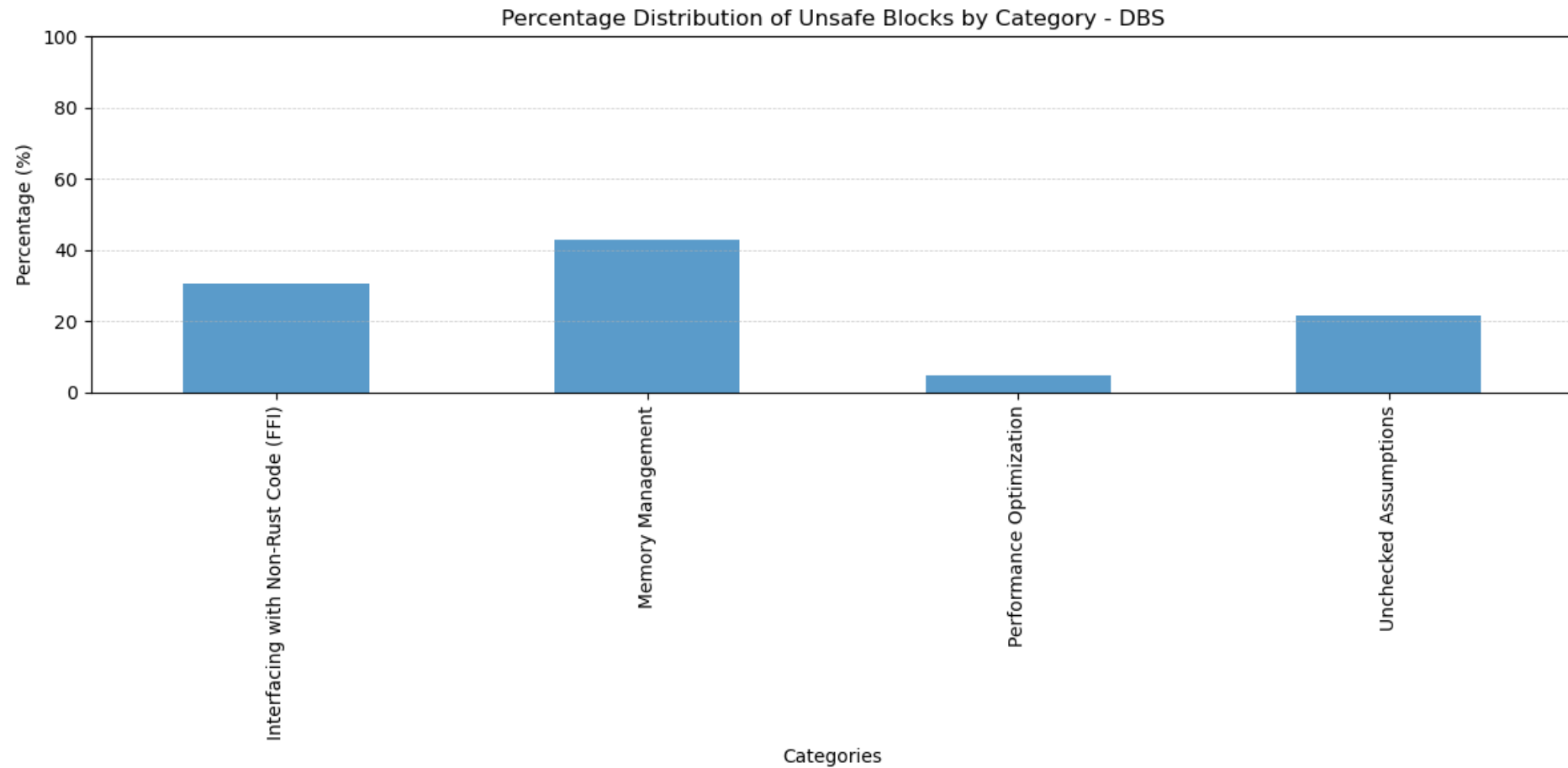
Key Results - Patterns in Unsafe Rust

- **Categories Identified:**
 - **Memory Management:** Unsafe usage for manual memory allocation and deallocation.
 - **Performance Optimization:** Unsafe blocks introduced for low-level optimizations.
 - **Interfacing with Non-Rust Code (FFI):** Unsafe code for interacting with external systems or C libraries.
 - **Unchecked Assumptions:** Unsafe blocks with unverified constraints or logic.

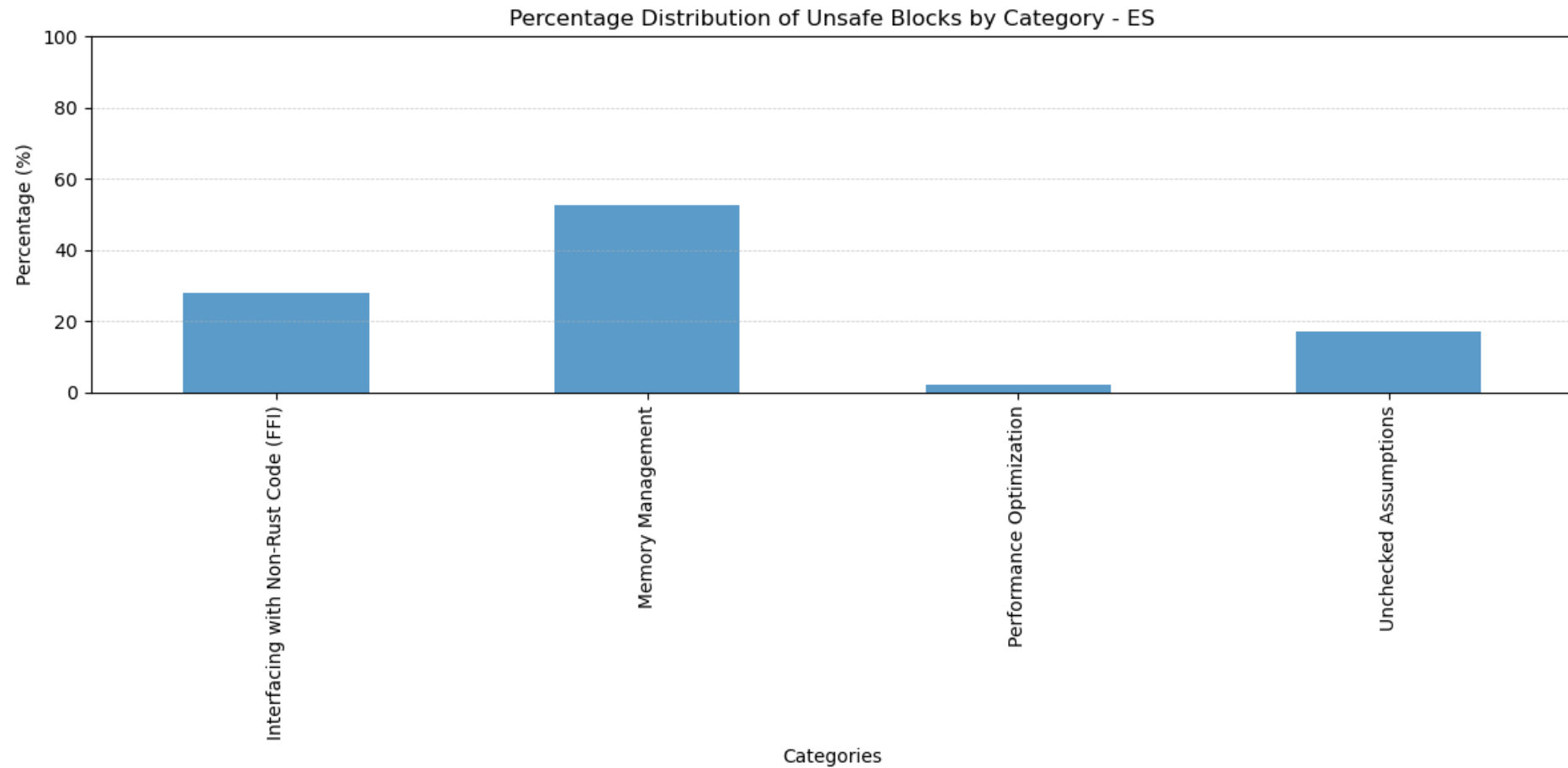
Key Results - Patterns in Unsafe Rust



Key Results - Patterns in Unsafe Rust

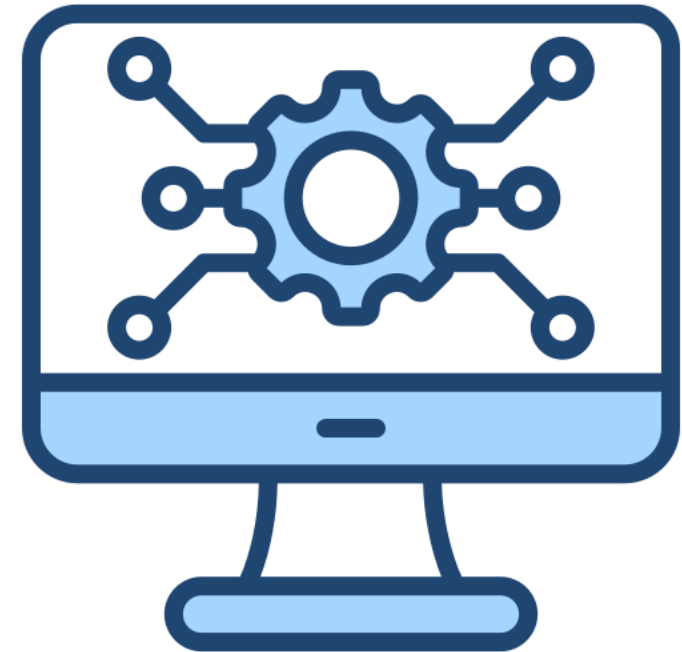


Key Results - Patterns in Unsafe Rust

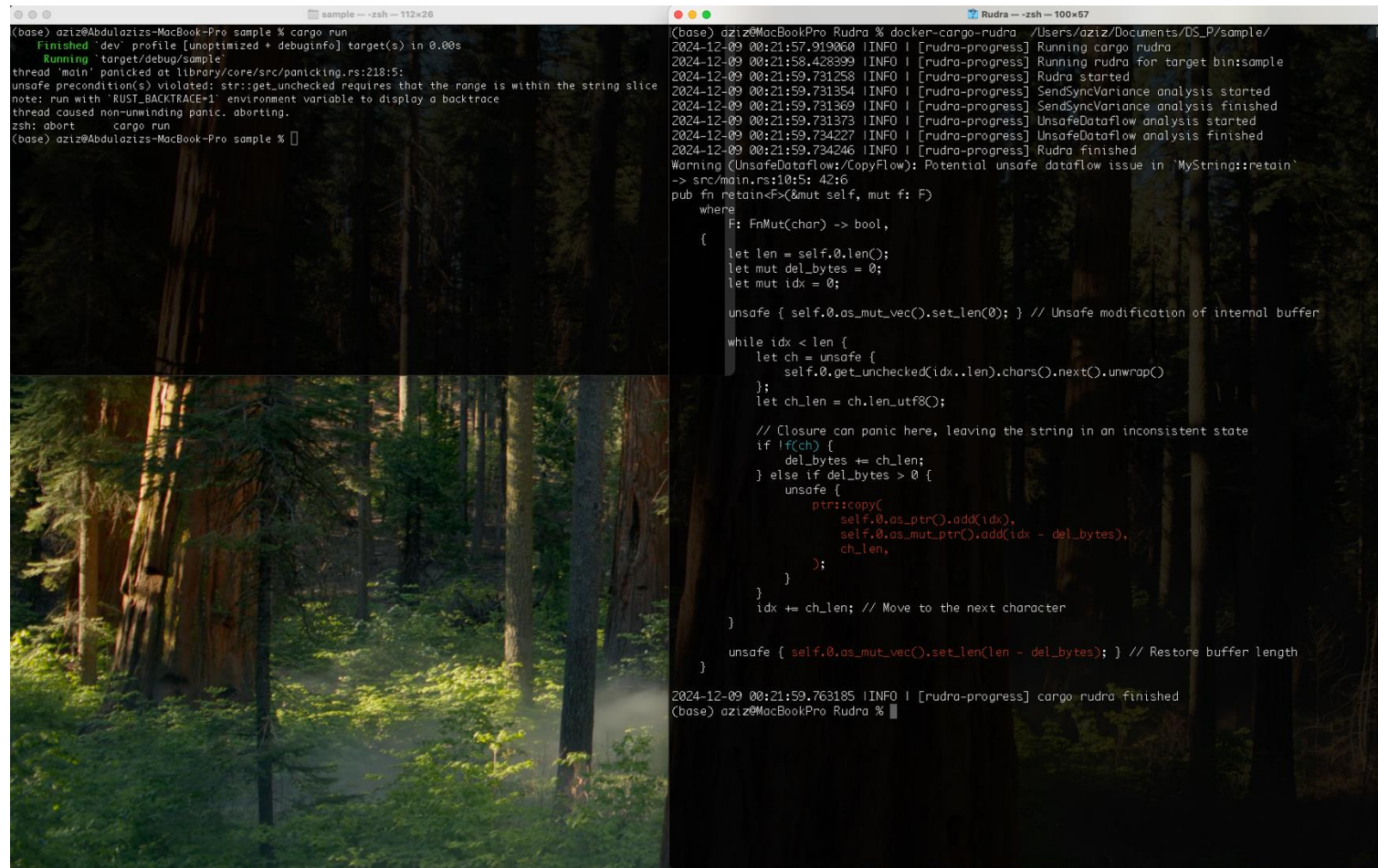


Key Results - Rudra Analysis

- Selection of Unsafe Rust
- Isolation Process
- Analysis with Rudra



Sample Analysis



```
(base) aziz@Abdulazizs-MacBook-Pro sample % cargo run
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
    Running `target/debug/sample`
thread 'main' panicked at library/core/src/panicking.rs:218:5:
unsafe precondition(s) violated: str::get_unchecked requires that the range is within the string slice
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
thread caused non-unwinding panic. aborting.
zsh: abort      cargo run
(base) aziz@Abdulazizs-MacBook-Pro sample %
```

```
(base) aziz@MacBookPro Rudra % docker-cargo-rudra /Users/aziz/Documents/DS_P/sample/
2024-12-09 00:21:57.919060 INFO [rudra-progress] Running cargo rudra
2024-12-09 00:21:58.428399 INFO [rudra-progress] Running rudra for target bin:sample
2024-12-09 00:21:59.731258 INFO [rudra-progress] Rudra started
2024-12-09 00:21:59.731354 INFO [rudra-progress] SendSyncVariance analysis started
2024-12-09 00:21:59.731369 INFO [rudra-progress] SendSyncVariance analysis finished
2024-12-09 00:21:59.731373 INFO [rudra-progress] UnsafeDataflow analysis started
2024-12-09 00:21:59.734227 INFO [rudra-progress] UnsafeDataflow analysis finished
2024-12-09 00:21:59.734246 INFO [rudra-progress] Rudra finished
Warning (UnsafeDataflow/CopyFlow): Potential unsafe dataflow issue in 'MyString::retain'
-> src/main.rs:10:5: 42:6
pub fn retain<F>(&mut self, mut f: F)
   where
       F: FnMut(char) -> bool,
   {
       let len = self.0.len();
       let mut del_bytes = 0;
       let mut idx = 0;

       unsafe { self.0.as_mut_vec().set_len(0); } // Unsafe modification of internal buffer

       while idx < len {
           let ch = unsafe {
               self.0.get_unchecked(idx..len).chars().next().unwrap()
           };
           let ch_len = ch.len_utf8();

           // Closure can panic here, leaving the string in an inconsistent state
           if !f(ch) {
               del_bytes += ch_len;
           } else if del_bytes > 0 {
               unsafe {
                   ptr::copy(
                       self.0.as_ptr().add(idx),
                       self.0.as_mut_ptr().add(idx - del_bytes),
                       ch_len,
                   );
               }
           }
           idx += ch_len; // Move to the next character
       }

       unsafe { self.0.as_mut_vec().set_len(len - del_bytes); } // Restore buffer length
   }

2024-12-09 00:21:59.763185 INFO [rudra-progress] cargo rudra finished
(base) aziz@MacBookPro Rudra %
```

Key Results - Rudra Analysis

Sample Size:

- Analyzed **5 unsafe Rust blocks** from each project.
- Analyzed a total of 60 unsafe Rust blocks: 20 blocks per domain across 3 domains.

Analysis Outcome:

- Memory Bugs Detected: None.
- Rudra flagged no memory safety issues in the selected unsafe Rust samples



Lessons Learned

- Learning Rust
- The Role of Unsafe Rust in Real-World Applications
- Capabilities and Limitations of Static Analysis Tools.
- The Complexity of Analyzing Real-World Projects
- As I've observed during this project, unsafe Rust is powerful but requires thorough analysis to ensure safety across diverse applications.

Q&A