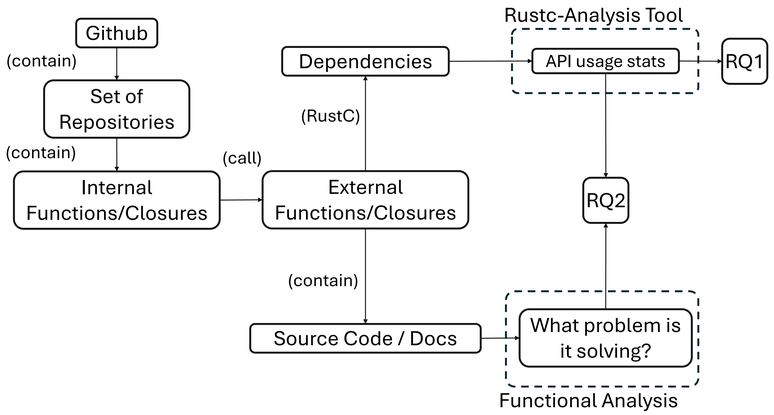
An Empirical Analysis on the Usage of Asynchronous Code by Rust Developers

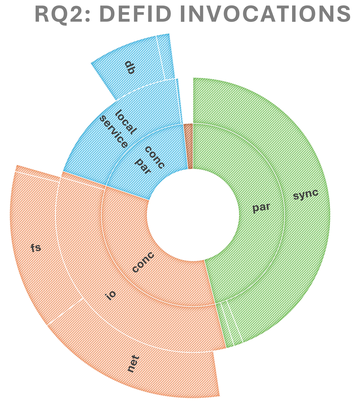
Author: Benjamin Dockx, b.h.p.dockx@student.tudelft.nl | Supervisor: Asst. Prof. Andreea Costea | Supervisor: PhD. Stud. Ruben Backx

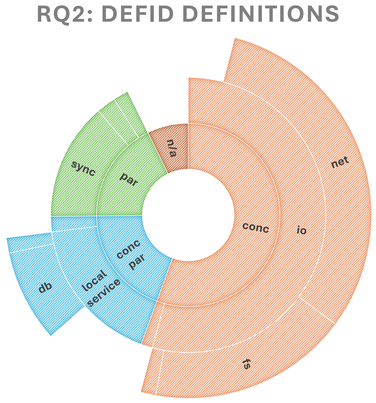
1. Motivation

Asynchronous Rust has seen widespread adoption since its release in 2019, yet little empirical evidence exists regarding how developers use asynchronous APIS and which APIs dominate the ecosystem.

RQ1: what are the most popular runtimes?
RQ2: what functionality do external async APIs provide?







2. Methodology

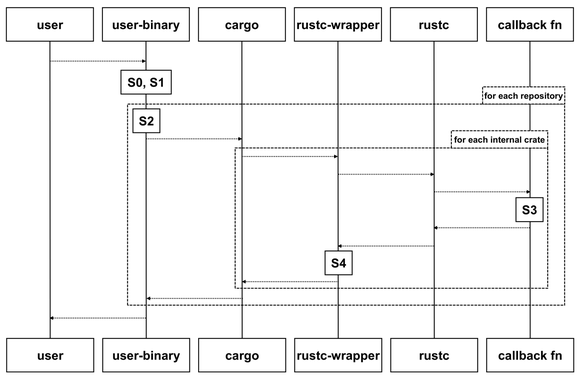
- Presenting Rustc-Analysis
- Analyzing open-source Rust code
- Using the Rust Compiler

- Top 25 GitHub Rust repositories
- 16 successfully analyzed
- 510 internal crates
- 1,649 external crates

3. Rustc-Analysis Tool

- Built on rustc MIR/HIR
- Constructs call graphs
- Extracts async function interaction
- Stores results in DuckDB

- Automatically fetches repositories
- Runs on repositories + hash (reproducible)



4. RQ1: Runtime Popularity

Tokio is dominant runtime across direct usages, indirect usages and crates.io usages.

Futures is a popular alternative also providing executor capabilities, often used with **tokio**.

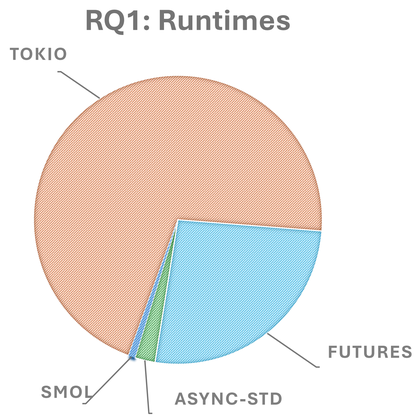
Async-std is significantly less popular and has been discontinued in lieu of **smol**.

5. RQ2: API Functionality

I/O tasks were most common usage of async APIs, namely networking and file-system operations.

Synchronization between concurrent tasks in close second

Local services made up a smaller percentage of async APIs. These local database services and language servers.



6. Key Findings

Tokio is the most widely adopted runtime

Async APIs are primarily used for **networking**, **file-system operations** and **synchronization** between concurrent tasks.

7. Future Work

- Expand dataset beyond 16 repositories
- Analyze internal async functions
- Explore LLM-assisted API categorization

Full Report

Recreation Package

Delft, Netherlands
25-06-2026