Google Summer of Code

Integrated End-to-End Workflow Testing Framework for Sprocket

Applicant: Deepak Silaych

Email: deepaksilaych@gmail.com

GitHub: deepaksilaych

Time Zone: IST (UTC+5:30)

University: Indian Institute of Technology Bombay

Degree: B.Tech

Current Year: Third Year

1. Abstract

Sprocket is a workflow execution engine for bioinformatics pipelines, but it currently lacks a dedicated testing framework. This project seeks to create a testing framework integrated into Sprocket as the sprocket test subcommand, designed to perform end-to-end testing of bioinformatics workflows. The framework will harness Sprocket's execution engine to enable workflow authors to validate their pipelines efficiently, ensuring correct outputs and graceful error handling with minimal effort. Test cases will be specified in an accessible YAML or JSON format, allowing users to define inputs, expected outputs, and validation conditions—ranging from basic checks like file existence to specialized assertions for genomics file formats. Additionally, the framework will integrate seamlessly into CI/CD pipelines, offering detailed reporting and automation-friendly outputs to support continuous testing. By enhancing Sprocket's reliability and usability, this project will contribute to St. Jude Cloud's mission of advancing pediatric research through dependable genomics analyses.

2. Project Description

2.1 Preparation and Research

To craft this application, I meticulously followed the organization's guidance, diving into key resources to build a strong foundation for my proposal. My first step was exploring the Workflow Description Language (WDL) standard, which is central to how workflows are expressed in Sprocket. Understanding WDL's structure—tasks with well-defined inputs, commands, and outputs—gave me critical insight into the building blocks of bioinformatics pipelines. This knowledge shaped my approach by emphasizing the need for a testing framework that can validate both individual tasks and their integration into cohesive workflows.

I then turned to St. Jude's workflows repository, analyzing the `rnaseq-standard` pipeline as a real-world example. This pipeline, with its sequence of tasks like alignment and quality checks, revealed the complexity and interdependence of workflow components. It became clear that a robust testing solution must handle such intricacies, ensuring reliability across diverse use cases. This exploration directly influenced my focus on creating a framework tailored to bioinformatics needs.

Additionally, I evaluated the state-of-the-art tool pytest-workflow, as suggested. While `pytest-workflow` excels with its YAML-based test definitions and engine-agnostic design, its Python implementation introduces performance overhead and lacks the type safety and optimization potential of Rust. This analysis solidified my decision to propose a Rust-based framework for Sprocket, leveraging Rust's performance and safety features to deliver a superior solution tightly integrated with Sprocket's ecosystem.

2.2 Vision and Approach

My vision is to develop a testing framework that is both powerful and user-friendly, integrated as the `sprocket test` subcommand. Workflow authors will define tests using simple YAML or JSON specifications, specifying the workflow (or task) to execute, input parameters, and expected outputs. The framework will support validations such as file existence, content checks, and exit status verification, making it intuitive yet comprehensive.

To ensure the framework is technically robust and scalable, my approach includes:

- **Leveraging Sprocket's Internal APIs**: I'll prioritize using Sprocket's Rust APIs for workflow execution to maximize performance and integration. If API access is limited, I'll implement a fallback to invoke Sprocket's CLI programmatically, ensuring flexibility.
- **Isolated Test Environments**: Each test will run in a temporary, sandboxed directory to avoid interference and lay the groundwork for potential parallel execution.
- **Bioinformatics-Specific Validations**: Beyond basic checks, I'll incorporate custom assertions for genomics file formats (e.g., BAM file headers, VCF variant counts), tailoring the framework to St. Jude's domain.
- **CI/CD Optimization**: The framework will produce clear pass/fail outputs and support features like test filtering and failure preservation, streamlining automation in continuous integration pipelines.

2.3 Technical Plan

Here's how I'll execute the project with precision and technical excellence:

1. **Test Specification Parsing**: Using Rust's Serde library, I'll parse YAML/JSON test specifications into strongly typed Rust structs. This ensures reliable, error-resistant test definitions.
2. **Workflow Execution**: I'll integrate with Sprocket's internal execution functions for optimal performance. If needed, I'll use a subprocess approach to call the CLI, capturing outputs and exit codes for validation.
3. **Validation Logic**: I'll build a suite of validation functions to check file existence, content (via substring searches or checksums), and workflow status. For large files, streaming checks will prevent memory bottlenecks.
4. **Reporting**: The framework will output human-readable console logs with colored pass/fail indicators and a results summary, plus machine-readable formats (e.g., JSON) for CI/CD compatibility.
5. **Error Handling**: I'll implement robust error handling for issues like missing files or invalid specs, delivering clear, actionable error messages.

2.4 Stretch Goals

To elevate this project beyond the baseline, I'll pursue the following:

- **Custom Assertions for Genomics Data**: Extending validations to handle bioinformatics formats will make the framework uniquely valuable for St. Jude's workflows.
- **Parallel Test Execution**: Using Rust's concurrency features, I'll explore running independent tests simultaneously, with resource constraints suitable for CI environments.
- **Fake Data Generation Prototype**: As a stretch goal, I'll prototype tools to generate simple fake genomics data (e.g., small FASTQ files), setting the stage for future testing enhancements.

3. Major Contributions

Below are the key goals I plan to achieve by the end of the GSoC program, each designed to enhance Sprocket's testing capabilities for bioinformatics workflows. Each contribution includes the specific outcome and its intended target audience.

3.1 Fully functional `sprocket test` subcommand integrated into Sprocket's CLI

- **Description**: This subcommand will enable workflow authors to define, execute, and validate end-to-end tests for their workflows using YAML or JSON specifications. Built in Rust, it will leverage Sprocket's internal APIs (e.g., from the wdl and sprocket crates) for seamless execution, ensuring high performance and native integration.

- **Target Audience**: Workflow authors and developers in the bioinformatics community using Sprocket, particularly those needing reliable testing for tools like the rnaseq-standard pipeline.

3.2 User-friendly test specification schema in YAML/JSON

- **Description**: This schema will allow users to define test cases with workflows, inputs, and expected outputs (e.g., file existence, content checks), inspired by pytest-workflow but optimized for Sprocket's Rust ecosystem. It will use Rust's Serde for parsing into typed structs, ensuring robustness.
- **Target Audience**: Workflow authors seeking an accessible, declarative way to create and manage tests for their pipelines.

3.3 Comprehensive validation features for bioinformatics workflows

- **Description**: The framework will include validation tools for file existence, content (e.g., substring searches, MD5 checksums), and execution status, with custom assertions for genomics formats like BAM and VCF files (e.g., validating headers or variant counts). These will be implemented using Rust's standard library for efficiency.
- **Target Audience**: Bioinformatics researchers and developers requiring precise validation of complex workflow outputs.

3.4 Isolated execution environments for reliable testing

- **Description**: Tests will run in temporary, sandboxed directories using Rust's tempfile crate, preventing interference and supporting future parallelism. An option to preserve outputs on failure will aid debugging.
- **Target Audience**: Workflow authors and CI/CD pipeline managers needing reproducible, interference-free test environments.

3.5 Seamless CI/CD pipeline integration

- **Description**: The framework will produce automation-friendly outputs (e.g., non-zero exit codes on failure, JSON reports) and support test filtering (e.g., `--test-name`), making it ideal for tools like GitHub Actions. It will handle large outputs efficiently with streaming validation.
- **Target Audience**: DevOps engineers and workflow developers maintaining quality through continuous integration.

3.6 Detailed documentation and example test suites

- **Description**: Comprehensive guides, reference docs, and best practices will be added to Sprocket's documentation, alongside example tests for real-world workflows (e.g., a simplified RNASeq pipeline from stjudecloud/workflows). These will live in a `tests/` directory in the Sprocket repo.

- **Target Audience**: New and experienced Sprocket users needing clear onboarding and practical examples.

3.7 Prototype tools for generating fake genomics data

- **Description**: If time permits, I'll develop initial tools to generate simple fake genomics data (e.g., small FASTQ files), inspired by tools like ngs generate, using Rust for compatibility with Sprocket. This will be an exploratory step with mentor guidance.
- **Target Audience**: Workflow authors and testers needing controlled, realistic data for validation.

4. Timeline

Below is the detailed timeline for my GSoC project, outlining the work to be accomplished in each phase. The plan is structured to ensure steady progress toward delivering a testing framework for Sprocket, with specific tasks and milestones tied to the project's goals.

Period	GSoC Phase	What I'll Aim to Accomplish
May 8 - June 1	Community Bonding Period	- Get to know mentors and the Sprocket community. - Study the Sprocket codebase (e.g., wdl and sprocket crates). - Review St. Jude's workflows repository and pytest-workflow for testing insights. - Finalize the test specification schema with mentor feedback.
June 2 - June 8	Coding (Week 1)	- Set up my development environment and build Sprocket from source. - Implement the `sprocket test` subcommand skeleton using [Clap](https://crates.io/crates/clap). - Define Rust structs for parsing YAML/JSON test specs with [Serde](https://serde.rs/). - Write unit tests for spec parsing.

Period	GSoC Phase	What I'll Aim to Accomplish
June 9 - June 15	Coding (Week 2)	- Explore Sprocket's internal APIs for workflow execution (e.g., from the wdl crate). - Prototype executing a simple WDL workflow programmatically. - Implement a fallback to invoke Sprocket's CLI via subprocess if needed. - Capture workflow output and exit codes.
June 16 - June 22	Coding (Week 3)	- Implement validation functions for file existence and content (e.g., substring search, checksums). - Integrate validation with workflow execution: run a test, check outputs, and report pass/fail. - Test with a simple WDL workflow (e.g., a "hello world" task).
June 23 - June 29	Coding (Week 4)	- Support multiple test cases in a single spec file. - Use tempfile for isolated execution environments. - Add basic reporting: pass/fail summary with colored output.
June 30 - July 5	Coding (Week 5)	- Add validation for bioinformatics outputs (e.g., BAM/VCF file checks). - Implement the `expect_failure` option for tests designed to fail. - Test with a real-world WDL task from St. Jude's repository (e.g., a simplified RNASeq task).
July 6 - July 12	Coding (Week 6)	- Prepare for mid-term evaluation: write initial documentation and ensure code is review-ready. - Demo the `sprocket test` subcommand for mentors. - Incorporate feedback and plan adjustments for the second half.

Period	GSoC Phase	What I'll Aim to Accomplish
July 13 - July 19	Coding (Week 7) *Midterm evaluation: July 18*	- Apply mid-term feedback (e.g., adjust spec format, improve performance). - Add quality-of-life features: test filtering (e.g., `--test-name`), preserve failure outputs. - Explore parallel test execution with Rust's async features (with resource limits).
July 20 - July 26	Coding (Week 8)	- Test the framework with a medium-complexity workflow (e.g., a small RNASeq pipeline). - Implement streaming validation for large files to avoid memory issues. - Enhance error handling for missing files or invalid specs.
July 27 - August 3	Coding (Week 9)	- Finalize user documentation: guides, reference docs, and best practices. - Write a tutorial for adding new tests, using St. Jude's workflow examples. - Start drafting the GSoC final report.
August 4 - August 10	Coding (Week 10)	- Polish the CLI: improve help text, error messages, and output formatting. - Add machine-readable output (e.g., JSON) for CI/CD integration. - Plan post-GSoC maintenance and community engagement.
August 11 - August 17	Coding (Week 11)	- Prototype simple fake data generation tools (e.g., small FASTQ files) as a stretch goal. - Write unit tests for new features and ensure good code coverage. - Address any remaining mentor feedback.

Period	GSoC Phase	What I'll Aim to Accomplish
August 18 - August 24	Coding (Week 12)	- Conduct final testing with real-world workflows. - Complete the GSoC final report and presentation. - Ensure all code and docs are merged or ready for review.
August 25 - September 1	Coding (Final Week)	- Wrap up any remaining tasks and hand over the project to mentors. - Celebrate a successful GSoC project completion!

Possible Changes in Timeline Due To...

- **Unexpected Issues :** I have included buffer time in the timeline to account for unforeseen issues. If any major roadblocks arise, I will communicate with my mentors promptly to discuss solutions and adjust the timeline as needed. I am also prepared to prioritize tasks and focus on the most critical aspects of the project if necessary.
- **Integration Challenges :** Integrating with Sprocket's existing systems may present challenges. I will work closely with my mentors to understand the codebase and address any integration issues effectively. I will also proactively seek help from the Sprocket community if needed.
- **Feedback from Mentors:** I am committed to incorporating feedback from my mentors throughout the development process. This may require adjustments to the timeline, but I will prioritize delivering a high-quality product. I will also strive to implement feedback in an efficient and timely manner.
- **Dependency Issues :** Problems with external libraries. I will research alternatives and consult with mentors.
- **Personal Issues :** I will communicate any personal issues to my mentors as soon as possible and try to minimize disruption.

5. Time Period/Working Hours

I am excited to outline my time commitment for the Google Summer of Code (GSoC) project, ensuring I can deliver a high-quality outcome. Here's how I plan to allocate my time:

- **Total Hours**: I estimate the project will require **350 hours** to complete. This aligns with GSoC guidelines for a large project and covers all key aspects, including development, testing, documentation, and additional features.

- **Weekly Commitment**: During the coding period, I plan to dedicate **25-30 hours per week** to the project. This consistent effort will allow me to make steady progress while maintaining flexibility to address challenges or refine my work.

This schedule reflects my commitment to the project's success, and I'm prepared to adjust as needed to meet milestones. Let me know if you'd like to move to the next section!

6. My Experience in Rust

I am proud of my contributions to various projects, and below are some examples of code samples that demonstrate my abilities. These include pull requests (PRs) to existing codebases where my contributions are clearly defined, as well as projects I've worked on independently.

- **Image Processing Service in Rust:** I designed and implemented a high-performance, modular image processing service in Rust, tailored for face detection, alignment, quality assessment, feature extraction, and anti-spoofing detection. This project integrates seamlessly with the NVIDIA Triton Inference Server for efficient model serving and is deployable via Docker.
- **Sprocket #72**: This PR implements the ability to override input values for WDL workflows via command-line arguments. This feature allows users to modify values in their input JSON/YAML files without needing to edit the files directly.
- **Mprocks #160**: This PR adds two major features to mprocs: terminal content search and copy-all functionality similar to vim.
- **Rust-clippy #14464**: This PR adds a new lint that detects unnecessary parentheses in negated if conditions like if !(condition) and suggests the cleaner if !condition form. Check failures are due to improper test implementation which is communicated with the community and will be resolved.

7. About Me

I am a third-year B.Tech student at IIT Bombay with a strong foundation in software development and a passion for creating robust and reliable systems. My academic coursework in Data Structures and Algorithms, Operating Systems, Low level informatics, Introduction to Computation Biology and Networks has provided me with a solid understanding of the principles behind software testing and quality assurance. Beyond academics, my projects and industry experience have equipped me with the skills necessary to successfully complete this project.

Projects

In addition to the Rust projects/PRs listed in the next section, I've led several impactful open-source projects:

- **MumbaiFlood.in :** I led the technical development of a critical city-wide flood monitoring platform,

which served over 10,000 weekly users. This project was instrumental in helping the city's residents navigate the challenges of the monsoon season. My contributions were pivotal to its success; I architected and deployed a full-stack data processing and alert generation. I also developed a machine learning prediction system, integrating GFS weather data with historical measurements.

Later, this project was made semi-open source on demand and **currently a team of volunteers is working on major upgradation and preparation for upcoming monsoon season thus its services are down**

- **ITC SSO**: I developed a student-run Single Sign-On (SSO) system using Django to address limitations of official sso to the institute's network. This project prioritized accessibility with security and has been adopted by 17 student clubs.
- **AptoTrade :** My team and I built a futures trading platform for the Inter-IIT Tech Meet 12.0 , solving a problem statement given by Aptos Foundation. We initially developed the core engine in Rust but later switched to Move due to ecosystem preferences. Our team won first place in the competition.
- **Financial Document Analysis System :** I led the development of an RAG pipeline for analyzing financial documents, securing first place at InterIIT Tech Meet 13.0 (again). I developed a vector store client with timeout mechanisms using Pathway (problem sponsor) for optimized retrieval, and created an evaluation pipeline with LLM-based evaluators and Entity Precision metrics for benchmarking.

Why I'm Excited About This Project

I am thrilled about contributing to this GSoC project because it combines my expertise in Rust with my interest in bioinformatics. Working on Sprocket offers a unique opportunity to enhance genomics workflows and support pediatric research at St. Jude, a mission I deeply admire. I am eager to collaborate with the community, learn from experienced mentors, and deliver reliable software solutions.