# Program-based Automatic Repair Prototype

CS 6720 – Fall 2025 Project Report

**Name:** Kevin Bell
**Repository:** https://github.com/bell-kevin/PAR-tool

**Project Overview.** This project delivers a Java re-implementation of the PAR (Program-based Automatic Repair) workflow for Python programs. The tool copies a subject project into an isolated workspace, applies mutation and pattern-based operators to a designated Python file, and evaluates each candidate with a user-supplied regression test command. The prototype supports eight mutation operators, a lightweight fault/fix knowledge base, and crossover to combine promising candidates.

**Design Highlights.**

- A command-line runner (`com.par.tool.ParTool`) coordinates project cloning, mutation search, scoring, and artifact production under `_apr_results/`.
- Mutation operators include statement deletion/duplication/swapping, arithmetic and comparison replacements, conditional negation, and small-integer tweaking. A pattern-based operator injects reusable repair templates.
- The scoring system parses `pytest` output to prioritize candidates that reduce failures and stop early when a full repair is found.

**Experimental Setup.** Three self-contained Python fixtures were crafted to evaluate the repair workflow. Each fixture provides a buggy module and `pytest` suite. The prototype was executed with a search budget of 40 candidates, a 30-second per-test timeout, Python 3.11, and `pytest` 7.4.

| Subject | Failure Mode | Baseline (fail/error/pass) | Outcome |
|---|---|---|---|
| `null_guard` | Missing None guard on method call | 1 / 0 / 1 | Fixed via `NullCheckGuard`. |
| `none_equality` | Equality check triggers custom `__eq__` | 1 / 0 / 2 | Fixed via identity comparison. |
| `bounds_check` | Out-of-range list access | 2 / 0 / 1 | Fixed via bounds guard. |

Across all three benchmarks the tool synthesized a repair that restored the test suite, demonstrating end-to-end functionality and validating the mutation operators and pattern-matching components. Detailed artifacts (summary JSON, diffs, and repaired source) are included in `experiments/`.

**Challenges.** Engineering the pattern-based operator required designing compact abstractions for recurring bug/fix pairs without overfitting to a single project. Balancing search depth and runtime was addressed by tuning operator probabilities and adding cheap heuristics to score candidates early.

**Future Work.** Next steps include integrating richer statistical fault localization, expanding the fix database with more Python-specific templates, and evaluating the tool on open-source defect datasets such as QuixBugs or BugsInPy.