# Automated Program Repair & Verification

## Coming up

- This Thursday, Beta presentations!
- Next week:
  - Tuesday, Lecture on ethics in software engineering
  - Thursday, no class — at home, online assignment/activity
    - Graded — part of the participation grade (3% of overall class grade)
    - Can optionally opt into sharing anonymized data with researchers
  - Also Thursday, Beta due!

## The Cost of Poor Software Quality in the US

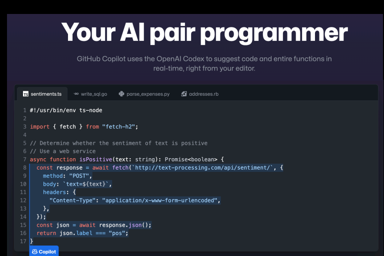
Software engineers spend 35-50% of their time validating and debugging software.

Cost of debugging, testing, and verification accounts for 50-75% of the software development budgets.

- Devon H. O'Dell, The Debugging Mindset. ACM QUEUE, http://doi.org/10.1145/3055301.3068754

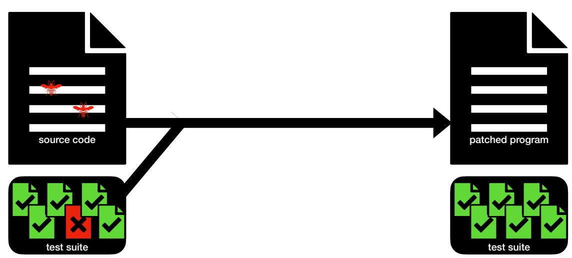
Herb Krasner, The Cost of Poor Software Quality in the US: A 2020 Report. https://www.it-cisq.org/pdf/CPSQ-2020-report.pdf

## Machine Learning in Software Engineering

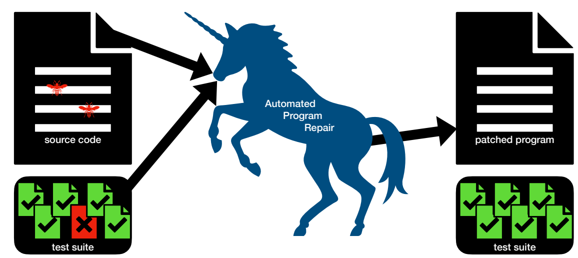



https://github.com/features/copilot

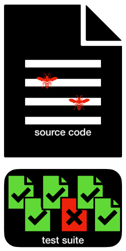
## A simpler problem: Automated program repair

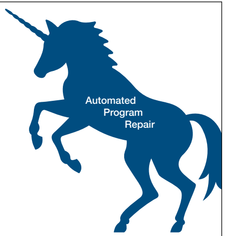



## A simpler problem: Automated program repair



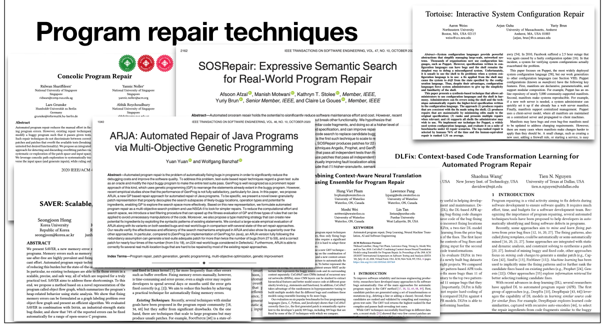
## Program repair techniques



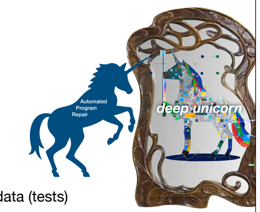

- Tweak the program
- Check if tests pass
- If not, repeat

## Program repair techniques



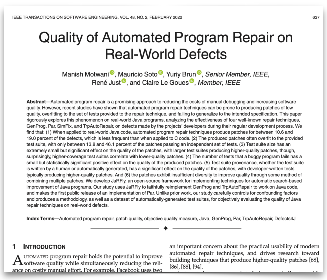

## APR is a form of machine learning

- first, many techniques rely on ML to learn
  - where to edit the code
  - how to edit the code
  - how to decide which patches are good
- second, the underlying problem is
  learning a function (program) using training data (tests)




## How well does APR work?

- Evaluated 4 techniques
  - GenProg
  - Par
  - TrpAutoRepair
  - SimFix
- Measured patch quality
- Measured what affects patch quality



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 48, NO. 2, FEBRUARY 2022 637

### Quality of Automated Program Repair on Real-World Defects

Manish Motwani, Mauricio Soto, Yuriy Brun, *Senior Member, IEEE*,
René Just, and Claire Le Goues, *Member, IEEE*

**Abstract**—Automated program repair is a promising approach to reducing the costs of manual debugging and increasing software quality. However, recent studies have shown that automated program repair techniques can be prone to producing patches of low quality, overfitting to the set of tests provided to the repair technique, and failing to generalize to the intended specification. This paper rigorously explores this phenomenon on real-world Java programs, analyzing the effectiveness of four well-known repair techniques, GenProg, Par, SimFix, and TrpAutoRepair, on defects made by the projects' developers during their regular development process. We find that: (1) When applied to real-world Java code, automated program repair techniques produce patches for between 10.6 and 19.0 percent of the defects, which is less frequent than when applied to C code. (2) The produced patches often overfit to the provided test suite, with only between 13.8 and 46.1 percent of the patches passing an independent set of tests. (3) Test suite size has an extremely small but significant effect on the quality of the patches, with larger test suites producing higher-quality patches, though, surprisingly, higher-coverage test suites correlate with lower-quality patches. (4) The number of tests that a buggy program fails has a small but statistically significant positive effect on the quality of the produced patches. (5) Test suite provenance, whether the test suite is written by a human or automatically generated, has a significant effect on the quality of the patches, with developer-written tests typically producing higher-quality patches. And (6) the patches exhibit insufficient diversity to improve quality through some method of combining multiple patches. We develop JaRFly, an open-source framework for implementing techniques for automatic search-based improvement of Java programs. Our study uses JaRFly to faithfully reimplement GenProg and TrpAutoRepair to work on Java code, and makes the first public release of an implementation of Par. Unlike prior work, our study carefully controls for confounding factors and produces a methodology, as well as a dataset of automatically-generated test suites, for objectively evaluating the quality of Java repair techniques on real-world defects.

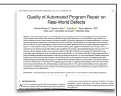
**Index Terms**—Automated program repair, patch quality, objective quality measure, Java, GenProg, Par, TrpAutoRepair, Defects4J

#### 1 Introduction

Automated program repair holds the potential to improve software quality while simultaneously reducing the reliance on costly manual effort. For example, Facebook uses two

an important concern about the practical usability of modern automated repair techniques, and drives research toward building techniques that produce higher-quality patches [68], [86], [88], [94].

## Quality vs. quantity



| technique | defects patched |
|---|---|
| GenProg | 49 (13.7%) |
| Par | 38 (10.6%) |
| SimFix | 68 (19.0%) |
| TRPAutoRepair | 44 (12.3%) |
| total | 106 (29.7%) |

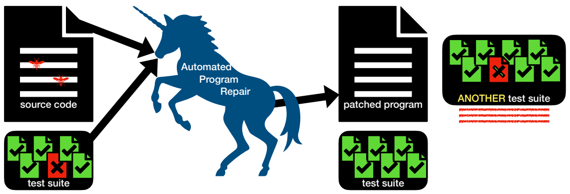
When applied to real-world Java code,
APR produces patches for 10.6-19.0% of the defects

## Quality vs. quantity

**Potential problem: Overfitting**

APR uses a set of tests to guide repair.
Tests are inherently partial.
No way APR can know if a patch captures intended behavioral constraints.

## Quality vs. quantity

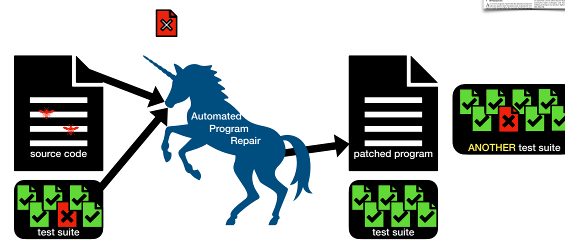



## Quality vs. quantity

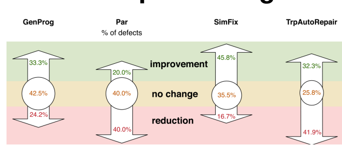
| technique | patch quality minimum | mean | median | maximum | 100%-quality patches |
|---|---|---|---|---|---|
| GenProg | 64.8% | 95.7% | 98.4% | 100.0% | 24.3% |
| Par | 64.8% | 96.1% | 98.5% | 100.0% | 13.8% |
| SimFix | 65.0% | 96.3% | 99.9% | 100.0% | 46.1% |
| TrpAutoRepair | 64.8% | 96.4% | 98.4% | 100.0% | 19.5% |

Less than half (14-46%)
of the patches are correct

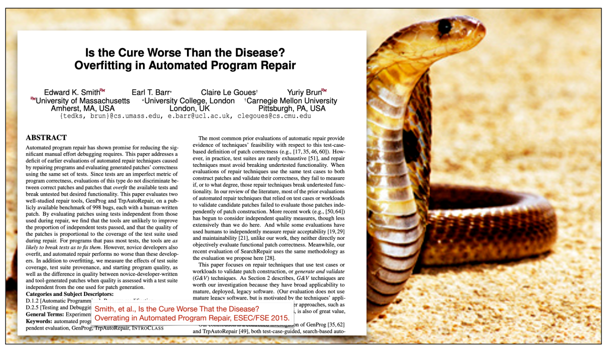
## Does APR at least improve things a bit?




## Does APR at least improve things a bit?




| technique | change in quality due to patch minimum | mean | median | maximum |
|---|---|---|---|---|
| GenProg | −30.9% | −1.7% | 0.0% | 2.6% |
| Par | −30.9% | −2.8% | 0.0% | 1.5% |
| SimFix | −24.9% | 0.2% | 0.0% | 35.0% |
| TrpAutoRepair | −30.9% | −2.1% | 0.0% | 3.8% |




Takeaway: Tests are an imperfect oracle, so APR suffers, producing low-quality patches.

Can we find a domain with better oracles?

## Formal verification allows proving software correct

## Interactive theorem provers for formal verification

Formal verification comes with a built-in oracle:
The theorem prover

Proof script

```
Proof.
intros.
induction n.
 ⋮
Qed.
```

Agda

## Industrial impact of theorem proving

AIRBUS

BedRock Systems Inc

galois

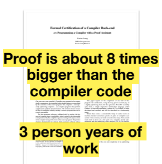
android

aws

CERTORA

## Prohibitively difficult

Verified software requires a lot of time and a lot of proofs in proportion to code

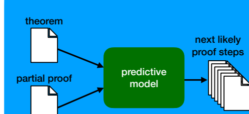



POPL 2006

Virtually all software that ships today is unverified.

## Proposal: Use APR-style technology to synthesize proofs

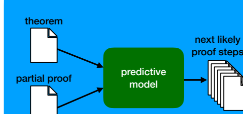
Step 1: Build a predictive model



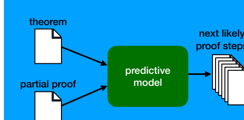

Step 2: Guide search with the model

## Proposal: Use APR-style technology to synthesize proofs

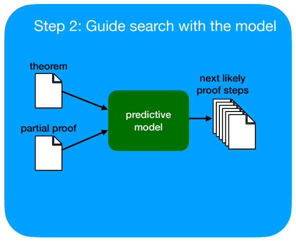
Step 1: Build a predictive model



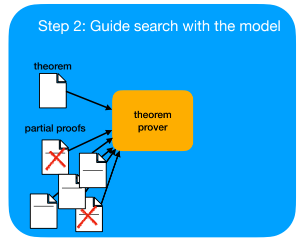

Step 2: Guide search with the model



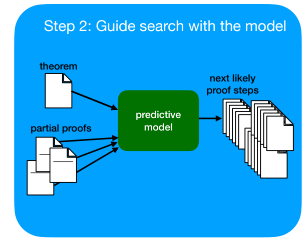
## Proposal: Use APR-style technology to synthesize proofs

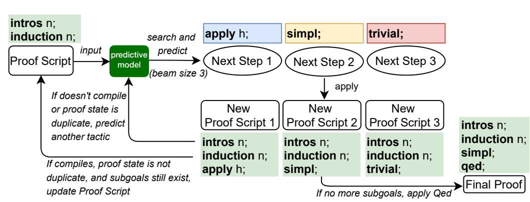



## Proposal: Use APR-style technology to synthesize proofs

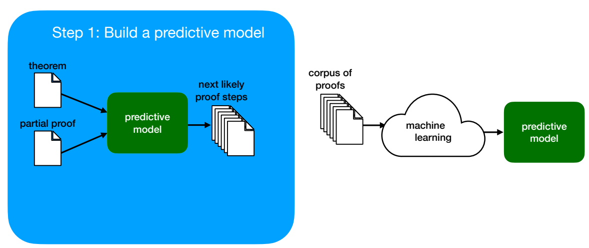



## Proposal: Use APR-style technology to synthesize proofs

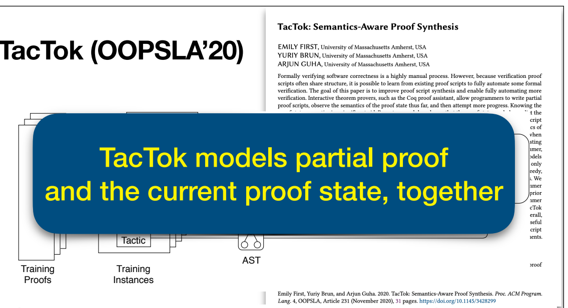



## How to learn a predictive model




## TacTok (OOPSLA'20)




ASTactic [Yang and Deng, Learning to Prove Theorems via Interacting with Proof Assistants, ICML'19] modeled just proof state.
[Hellendoorn, Devanbu, Alipour, On the naturalness of proofs, ESEC/FSE NIER'18] looked at predictability of proof sequences.

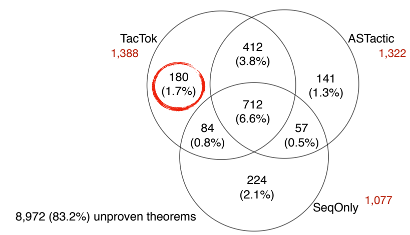
## CoqGym Dataset

- 123 open-source software projets in Coq
- 70,856 theorems
- Broken down into 96 projects (57,719 proofs) for training and 27 projects (13,137 theorems) for testing

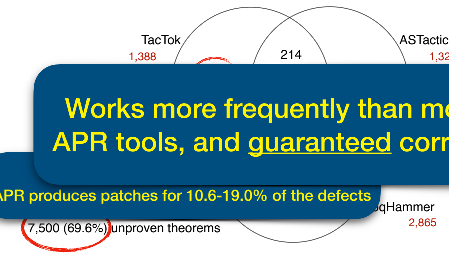
https://github.com/princeton-vl/CoqGym

[Yang and Deng, Learning to Prove Theorems via Interacting with Proof Assistants, ICML'19]

## TacTok vs. ASTactic vs. SeqOnly



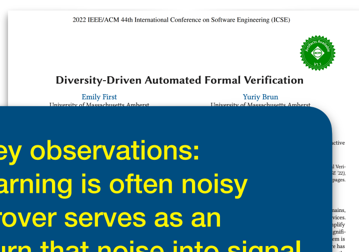

## TacTok vs. ASTactic vs. CoqHammer




Works more frequently than most APR tools, and guaranteed correct!

APR produces patches for 10.6-19.0% of the defects

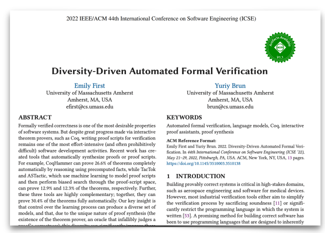
## Diva (ICSE'22)




2 key observations:

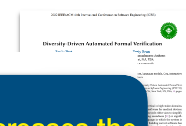
- Machine learning is often noisy
- Theorem prover serves as an oracle to turn that noise into signal.

## Diva (ICSE'22)



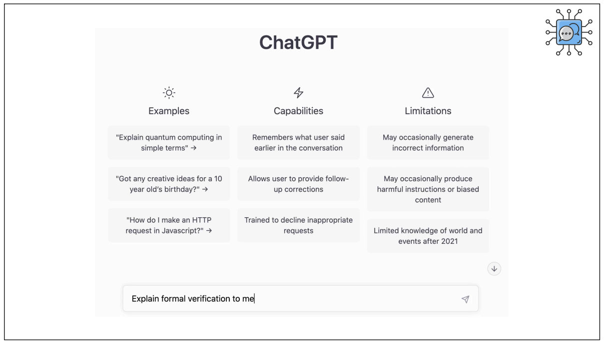

- Vary:
  - proof tactic and token depth
  - learning rate
  - embedding size
  - number of layers
  - training order
  - access to proof state, partial proof, Gallina proof term

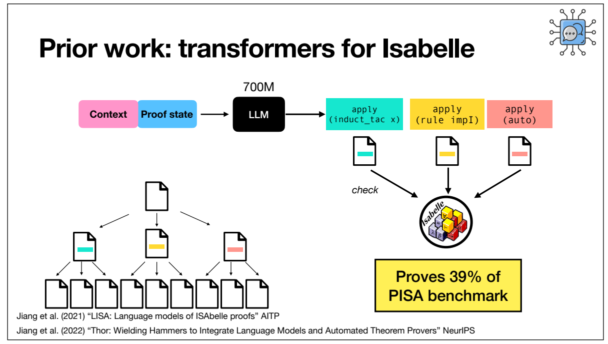
## Diva vs. state-of-the-art

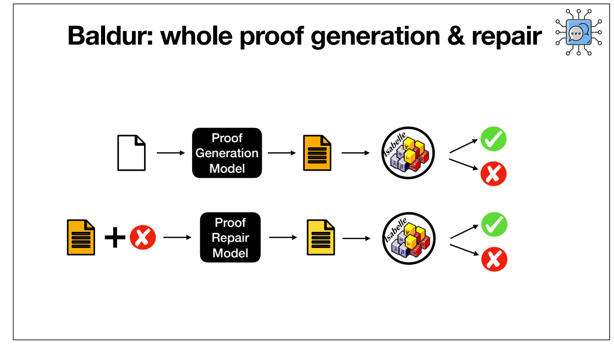


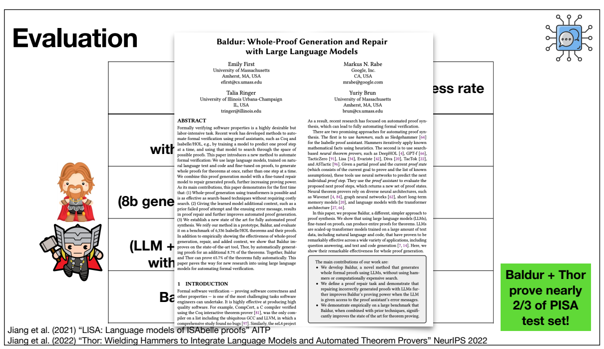

Diversity inherent in ML increases the proving power 68%-77% over prior search-based synthesis tools, and 27% over CoqHammer.

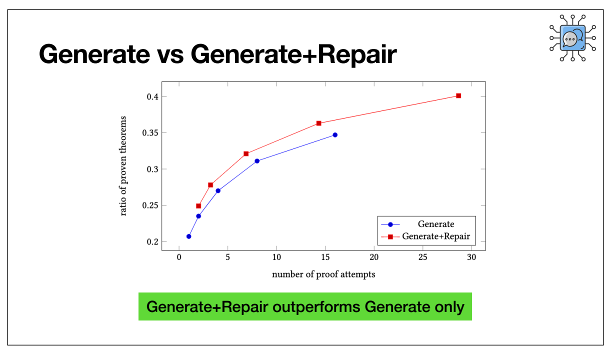
https://github.com/LASER-UMASS/Diva/

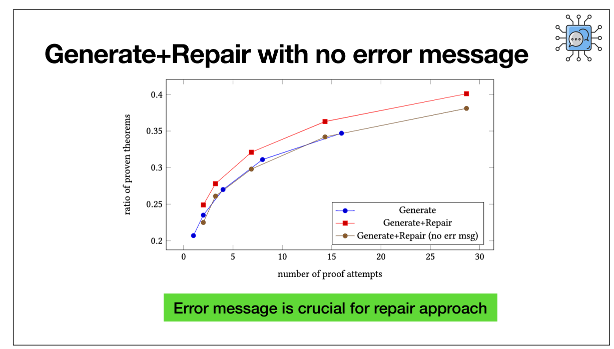

ChatGPT
Examples
Capabilities
Limitations
"Explain quantum computing in simple terms" →
Remembers what user said earlier in the conversation
May occasionally generate incorrect information
"Got any creative ideas for a 10 year old's birthday?" →
Allows user to provide follow-up corrections
May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?" →
Trained to decline inappropriate requests
Limited knowledge of world and events after 2021
Explain formal verification to me


Prior work: transformers for Isabelle
700M
Context
Proof state
LLM
apply (induct_tac x)
apply (rule impI)
apply (auto)
check
Proves 39% of PISA benchmark
Jiang et al. (2021) "LISA: Language models of ISAbelle proofs" AITP
Jiang et al. (2022) "Thor: Wielding Hammers to Integrate Language Models and Automated Theorem Provers" NeurIPS

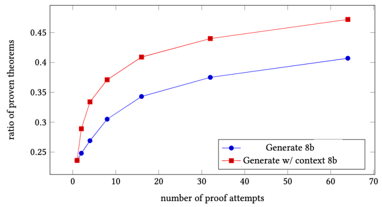

Baldur: whole proof generation & repair
Proof Generation Model
Proof Repair Model


Evaluation
ess rate
wit
(8b gene
(LLM +
with
Ba
Baldur: Whole-Proof Generation and Repair with Large Language Models
Baldur + Thor prove nearly 2/3 of PISA test set!
Jiang et al. (2021) "LISA: Language models of ISAbelle proofs" AITP
Jiang et al. (2022) "Thor: Wielding Hammers to Integrate Language Models and Automated Theorem Provers" NeurIPS 2022


Generate vs Generate+Repair
ratio of proven theorems
number of proof attempts
Generate
Generate+Repair
Generate+Repair outperforms Generate only


Generate+Repair with no error message
ratio of proven theorems
number of proof attempts
Generate
Generate+Repair
Generate+Repair (no err msg)
Error message is crucial for repair approach

## Generate with context




Proof context helps improve proof generation

## Fully Automated Formal Verification

Machine learning and meta-heuristic search
can fully automate
some bug-repair and formal verification.

While APR underperforms because it is driven by an unreliable oracle,
formal verification is a killer app for APR because
the theorem prover provides a reliable oracle.