AI for Fuzzing: A Tale of Two Techniques

Yuekang Li

Background - Techniques for bug detection

- Software as patients
- Analyzers as doctors
- Bugs as illness

- Scan the patient
  - patients don’t need to act
  - fast but less accurate
  - static analysis

- Observe the patient
  - patients need to make actions
  - takes time but accurate / can have PoC input
  - dynamic analysis

Background - Fuzzing

- A testing technique specialized for detecting ***security*** bugs
  - can be black-box, white-box or grey-box

- Basic Idea:
  - Execute the target program with a large amount of “random” inputs and observe for abnormal behaviours of the target program (such as crashing).
  - “random” ➡☐fuzzing

- The tool used for fuzzing is called a ***fuzzer***.
  - tens of thousands of bugs found

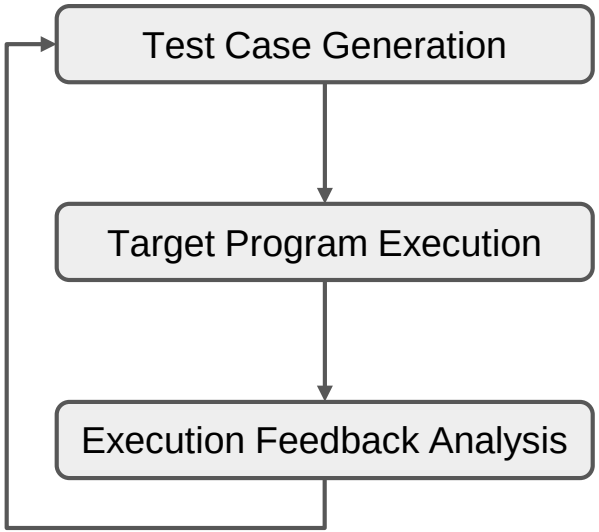
https://rabbitbreeders.us/wp-content/uploads/**american-fuzzy-lop**-rabbit-breed.jpg

Background - Grey-box fuzzing

+ Highly Scalable + Doesn’t require program analysis	+ Highly effective		+ Effective + Scalable + Jack of all trades
- Lacks effectiveness	- Lacks scalability - Relies heavily on program analysis	- Doesn’t exist !	- Master of none 🚀

The Meta Model for (Grey-box) Fuzzing

1. Generate test cases
2. Execute the target program with the generated test cases
3. Collect and analyze execution feedback
4. Use the feedback to adjust test case generation



Differentiates the “color of box”

AI for Software analysis

Let’s use AI techniques to do software analysis?

Why?

- Result is not explainable
- High accuracy requirement

AI for Software analysis

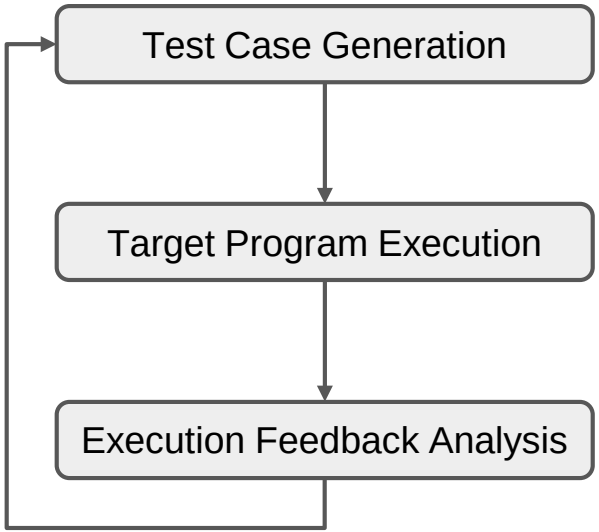
Where does AI perform well?

Why?

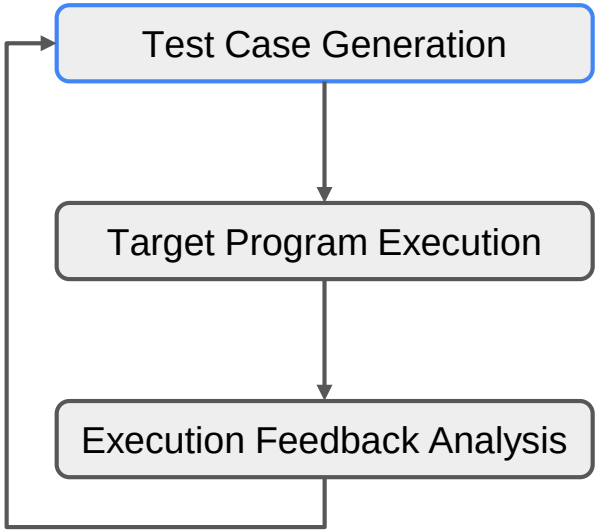
- Tolerant to “low” accuracy
- Fast -- allowing retries

The Meta Model for (Grey-box) Fuzzing

1. Generate test cases
   - Yes (SampleFuzz)
2. Execute the target program with the generated test cases
   - Yes (FuzzGuard)
3. Collect and analyze execution feedback
4. Use the feedback to adjust test case generation
   - Yes (Neuzz, MTFuzz)

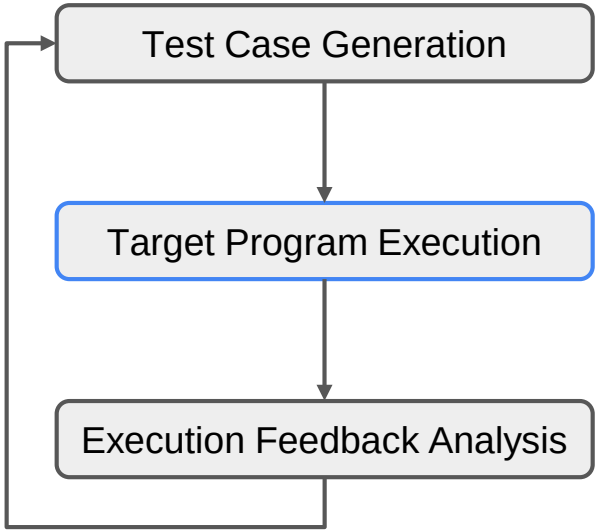


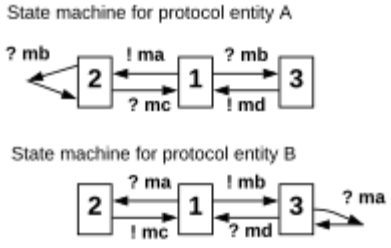
AI for test case generation



- Learn input format
- Generate new inputs with models

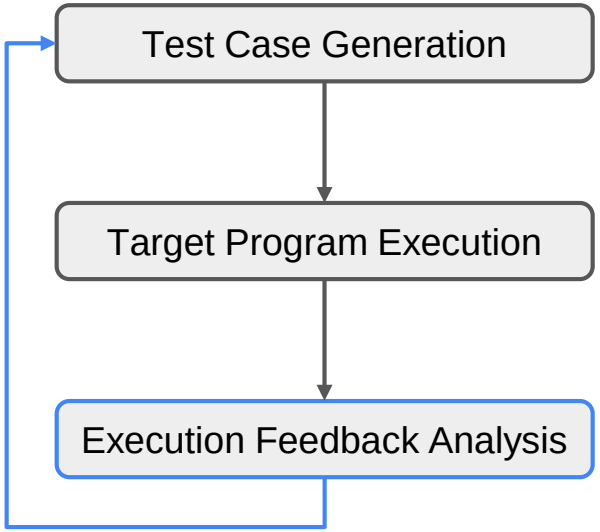
AI for faster program execution

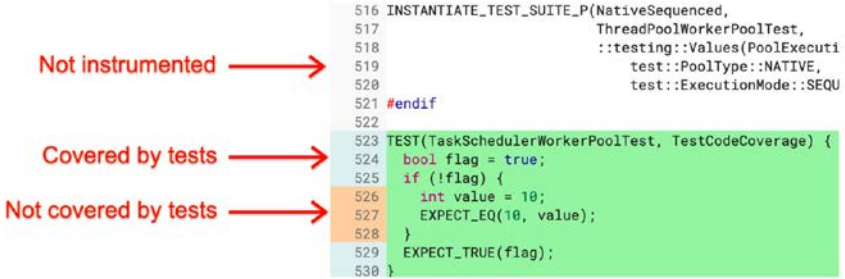




- Learn input reachability information
- Predict input reachability
- Filter out unreachable inputs

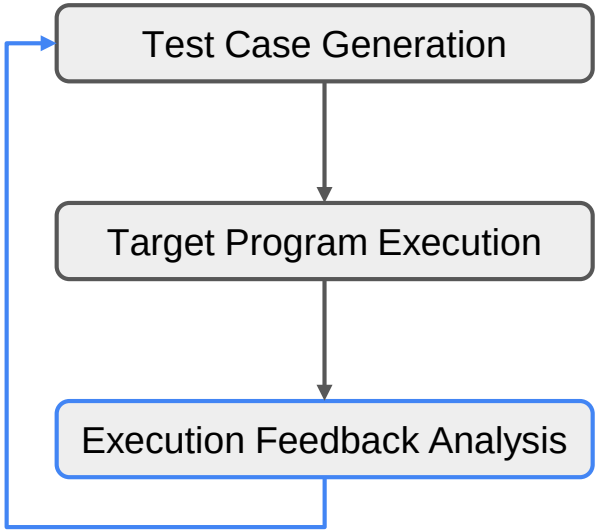
AI for feedback usage

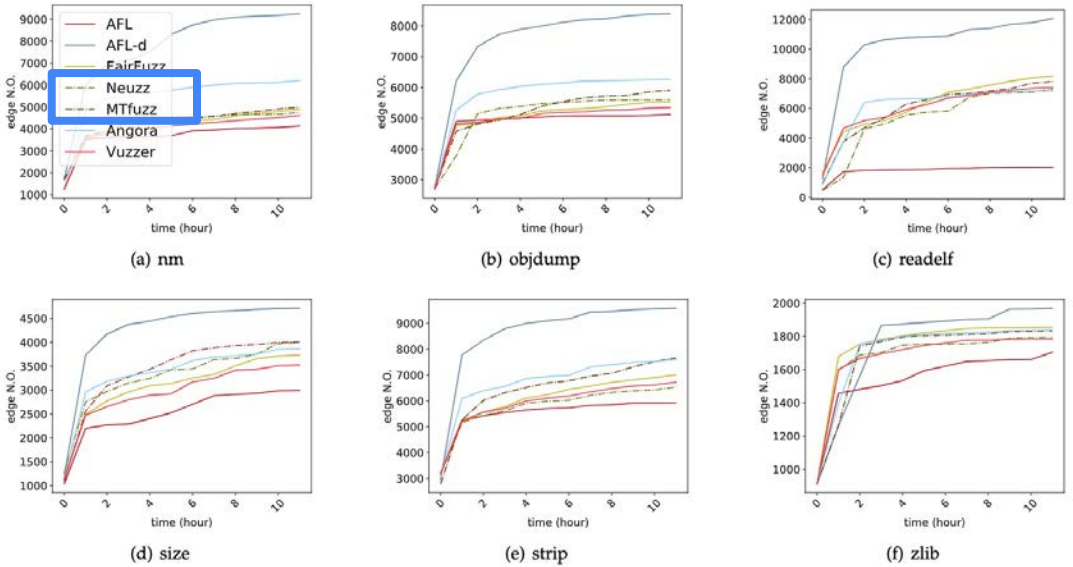




- Learn input-byte & coverage relations
- Mutate interesting/safe bytes (preserve key coverage etc.)

AI for feedback usage - Study





We observe that the performance of DL-based fuzzers do not significantly outperform other fuzzers. Overall, the AFL default mode does not perform as well as others and AFL-d outperforms the other approaches in covering moreprogram branches.

AI for feedback usage - Study

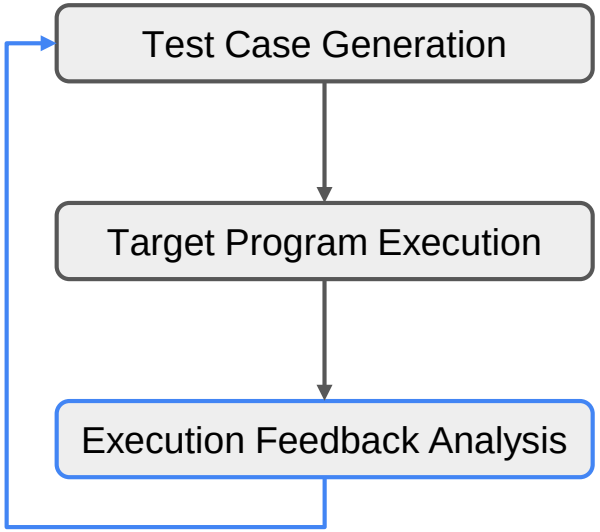
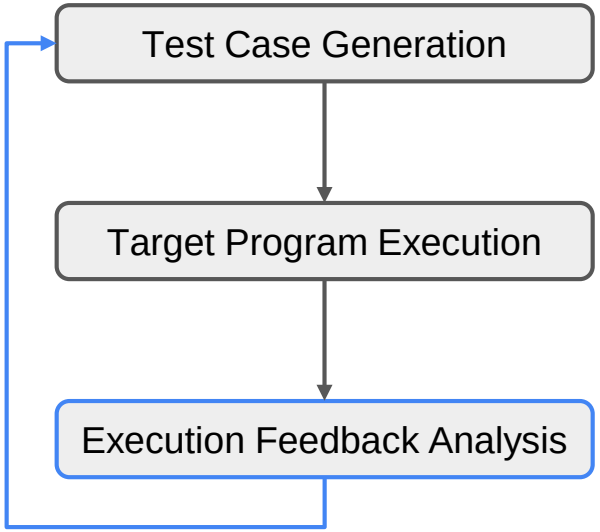


TABLE 5: Results of accuracy on different projects (%)

		nm	objdump	readelf	strip	size	zlib
Neuzz	Acc	92.47	95.01	96.15	88.07	89.87	84.77
	Rec	55.47	61.45	51.97	61.36	63.37	41.00
MTFuzz	Acc	91.23	94.65	95.49	89.79	89.32	87.33
	Rec	57.06	59.84	50.82	62.76	56.97	36.72

We observe that DL-based fuzzers can often suffer from the problem of imbalanced training data, which makes the model predict branches as “uncovered” blindly, simply based on statistical evidence.

AI for feedback usage - Study




```
1  int status = 0;
2  else if (str.find('ABC') == 'True') {
3      status = 1;
4  }
5  else {
6      status = 2;
7  }
```

An example of limited expressive model

We observe that some models lack expressiveness. The models are position dependent.

Future Directions

Fuzzing Aspect

- Combine with hybrid fuzzing techniques

AI Aspect

- Training data balancing
- Develop more expressive models

Thank you!