ELFuzz: Efficient Input Generation via LLM-driven Synthesis Over Fuzzer Space

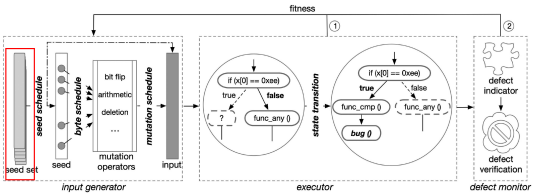
Chuyang Chen[1], Brendan Dolan-Gavitt[2], and Zhiqiang Lin[1]

[1]The Ohio State University, [2]New York University

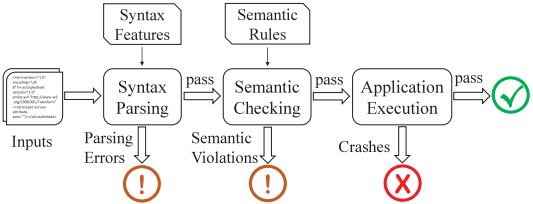
Presented at the 34th USENIX Security Symposium (USENIX Security '25), Seattle, WA, August 15, 2025

The problem

- How could we generate diverse and effective seed test cases for fuzzing?
  - Seed test cases are critical for fuzzing effectiveness
  - Efficient seed test cases require domain expertise to construct



The process of fuzzing



Test cases undergoing strict checks

Previous work

- Program analysis to mine grammars or black-box LLMs

Marker	Challenge
❶	Considering input grammars
❷	Considering semantic constraints
❸	Scaling to real-world SUTs
❹	Preventing grammar rule instantiation
❺	Human-understandable fuzzing process
❻	Easily generalizable to different domains
❼	No need for domain expertise

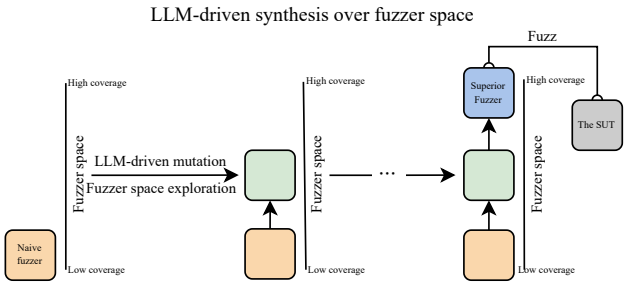
Challenges for input generation

Approach	Year	❶	❷	❸	❹	❺	❻	❼
AUTOGRAM	2016	✓				✓	✓	✓
LEARN&FUZZ	2017	✓	✓	✓	✓			✓
GLADE	2017	✓		✓		✓	✓	✓
MIMID	2020	✓				✓	✓	✓
ISLEARN	2022		✓			✓		
FUZZNG	2023	✓	✓	✓	✓	✓		
TITANFUZZ	2023	✓	✓	✓	✓			
FUZZ4ALL	2024	✓	✓	✓	✓			✓
METAMUT	2024	✓	✓	✓	✓	✓		
DY FUZZING	2024	✓	✓	✓	✓	✓		
SYZGEN++	2024	✓	✓	✓	✓	✓		
COVRL-FUZZ	2024	✓	✓	✓	✓			
FUZZINMEM	2024	✓	✓	✓	✓			
ELFuzz		✓	✓	✓	✓	✓	✓	✓

Comparison between works

Our solution: An LLM to evolve seed test case generators

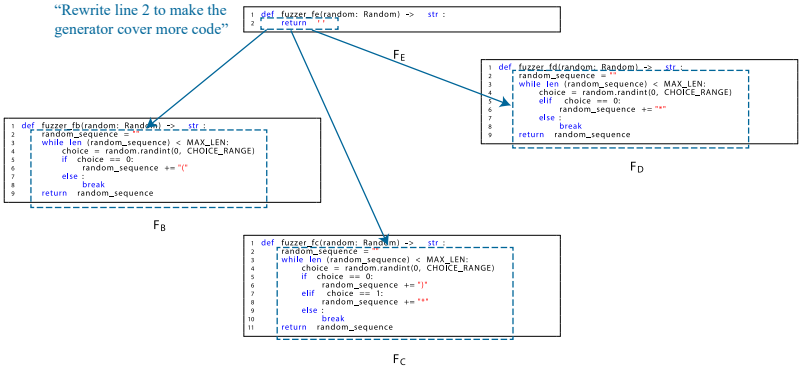
- An LLM evolves naïve seed test case generators (fuzzers for short) to efficient ones



Using an LLM to evolve the input generators

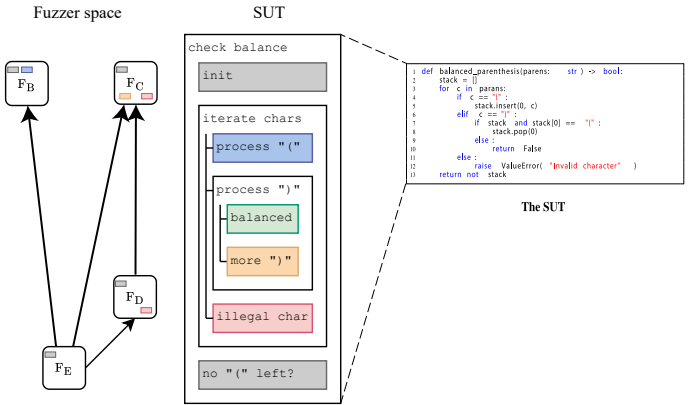
Running example: Mutating candidate fuzzers by an LLM

- Mutating a seed generator by LLM-written patches



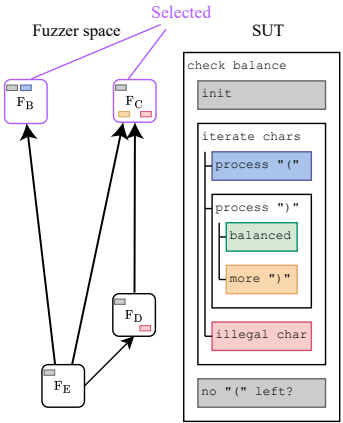
Running example: Comparing mutants in the fuzzer space

- Each mutant covers different code, forming a partial order, i.e., the *fuzzer space*



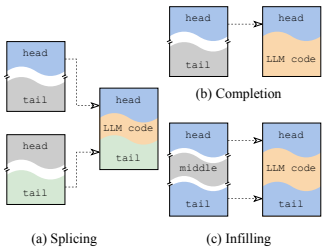
Running example: Selecting the best mutants

- Mutants that cover the most code *in together* are selected as survivors



Methodology: LLM-based mutation

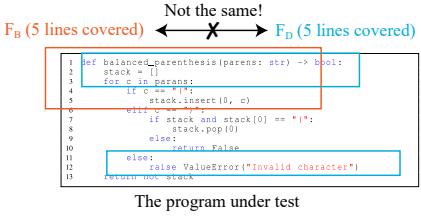
- **Splicing.** Gluing the prefix and suffix of two candidates by LLM-generated code
- **Completion.** Truncating and re-completing a candidate by LLM
- **Infilling.** Rewriting random lines of a candidate



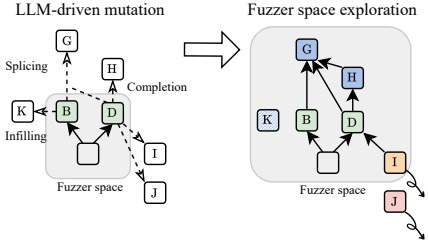
LLM-based mutators

Methodology: Fuzzer space exploration

- Mutants with the exact range of covered code form a lattice, i.e., the fuzzer space
- The lattice enables fine-grained comparison beyond single coverage values



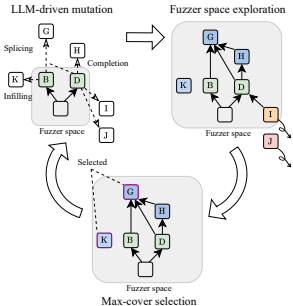
Different fuzzers with the same coverage value



Mutants are organized into the fuzzer space

Methodology: Max-cover selection

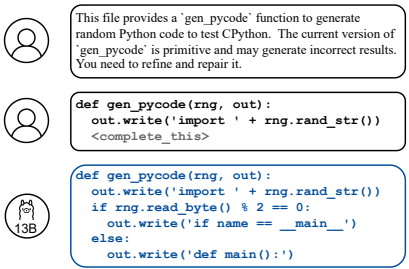
- Mutants with the maximal covered code together survive to the next iteration
  - We use an approximation algorithm to solve the $\mathcal{NP}$-complete problem



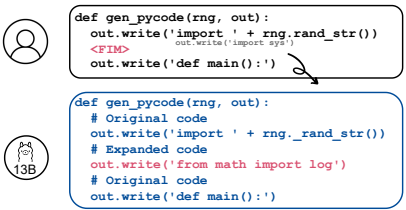
Mutants covering the most code selected

Implementation

- A local CodeLlama-13B (runnable with 26GiB VRAM) drives the evolution loop
- The evolution loop eliminates the need for complex prompt engineering



Several-sentence prompts for mutation



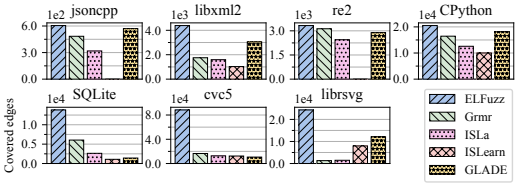
Query to generate glue code

Evaluation: Research questions

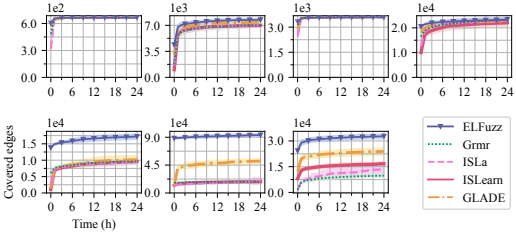
- **RQ1: Effectiveness in fuzzing.** How effective is ELFuzz in fuzzing?
- **RQ2: Effectiveness in bug finding.** How effective is ELFuzz in finding bugs?
- **RQ3: Ablation study.** How do different components of ELFuzz contribute to its effectiveness? Which is the most important one?
- **RQ4: Interpretability and extensibility.** Do the synthesized fuzzers catch interesting characteristics of the programs under test in an interpretable and extensible way?

RQ1: Effectiveness in fuzzing

- ELFuzz can generate test cases with high coverage (up to 434.8% more compared to other techniques). These test cases, as seeds, bring significant advantages for later mutation-based fuzzing compared to other techniques.



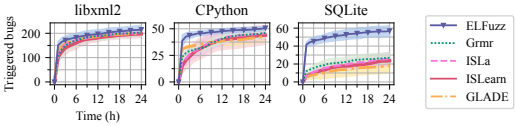
Coverage of the inputs generated within 10min



Coverage trends during mutation-based fuzzing

RQ2: Effectiveness in bug finding

- ELFuzz is effective in finding more bugs. In the controlled experiments, it finds more artificially injected bugs. In the real-world experiment, ELFuzz finds five new bugs, including severe and complex vulnerabilities, even with a short time budget.



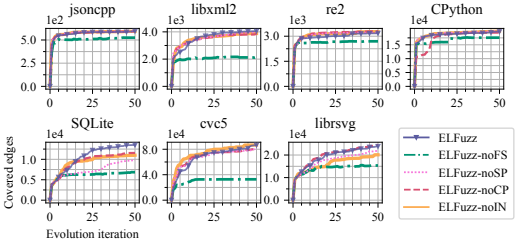
Trends of artificially injected bugs triggered

- **Bug 1.** Format string injection in error messages of `set-logic`
- **Bug 2.** Dead loop during parsing the `set-option` command
- **Bug 3.** Dead loop when rewriting arithmetic expressions related to division
- **Bug 4.** NULL-pointer dereference if multiple output files are set
- **Bug 5.** NULL-pointer dereference if `nullable` checks are not provided with arguments

Real-world bugs in `cvc5` found within 14 days

RQ3: Ablation study

- Fuzzer space contributes the most to the performance of ELFuzz compared to the three LLM-based mutators.



Edge coverage of survivors during the evolution

RQ4: Interpretability and extensibility

- Fuzzers synthesized by ELFuzz catch diverse and correct grammatical and semantic requirements of the input format of the programs under test and can be extended with other techniques (Zest in our case study).

```
testcase.write('CREATE TABLE IF NOT EXISTS ')
testcase.write(TABLE_NAME + '\n')
for col in COLUMN_NAMES[:-1]:
    testcase.write(col)
    testcase.write(', ')
testcase.write(COLUM_NAMES[-1])
testcase.write(
    'INTEGER PRIMARY KEY AUTOINCREMENT);\n'
)
... # Insert values into the table
orderby_columns = []
for _ in range(3):
    orderby_columns.append([
        COLUMN_NAMES[rand() % len(COLUMN_NAMES)],
        ['ASC', 'DESC'][rand() % 2]
    ])

testcase.write('SELECT * FROM ')
testcase.write(TABLE_NAME)
testcase.write(' ORDER BY ')
for col in orderby_columns[:-1]:
    testcase.write(col[0])
    testcase.write(' ')
    testcase.write(col[1])
    testcase.write(', ')

testcase.write(orderby_columns[-1][0]
testcase.write(' ')
testcase.write(orderby_columns[-1][1])
testcase.write(';\n')
```

A snippet from the synthesized fuzzers (simplified for illustration)

Thank you!

Full paper

Source code

My homepage