System Execution Data-Driven Function-Level Regression Fuzzing

Master Thesis Defense

Presented by Youngseok Choi (Advisor: Prof. Moonzoo Kim)

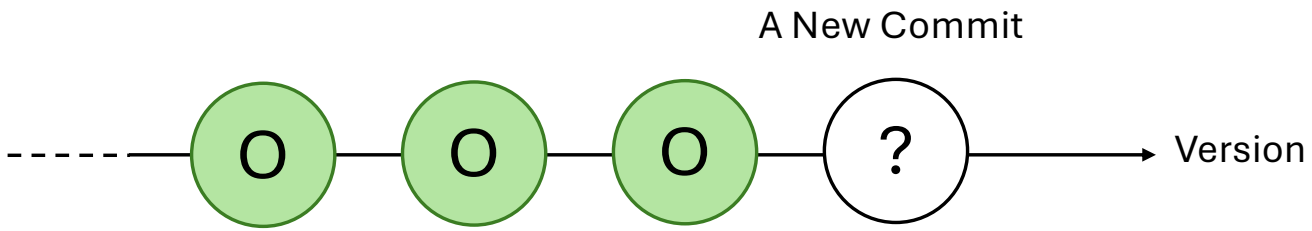
Time: 14:00, December 20th, 2024.

Location: Room #4448, E3-1, KAIST.

KAIST

Regression Fuzzing

Regression bug: A bug which causes a feature that worked correctly to stop working after a certain event (commit, system upgrade, etc.)



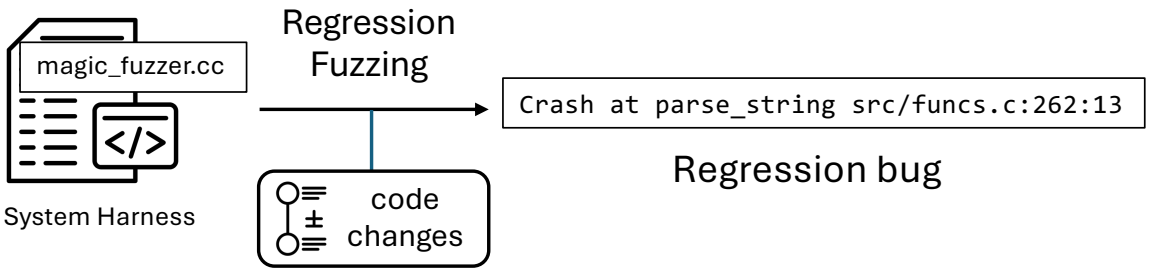
Problem Statement. (regression fuzzing)

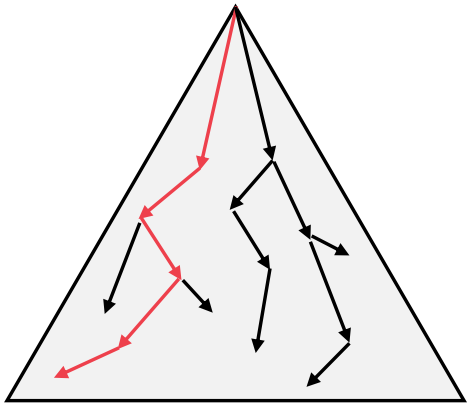
How can fuzzing be used to quickly identify regression bugs in a new version?

- Input: Project Codebase, Commit History
- Output: Regression Bug

Limitation of Existing Studies

Existing Studies: Fuzzing the entire program from the beginning.
(called as **system fuzzing**)



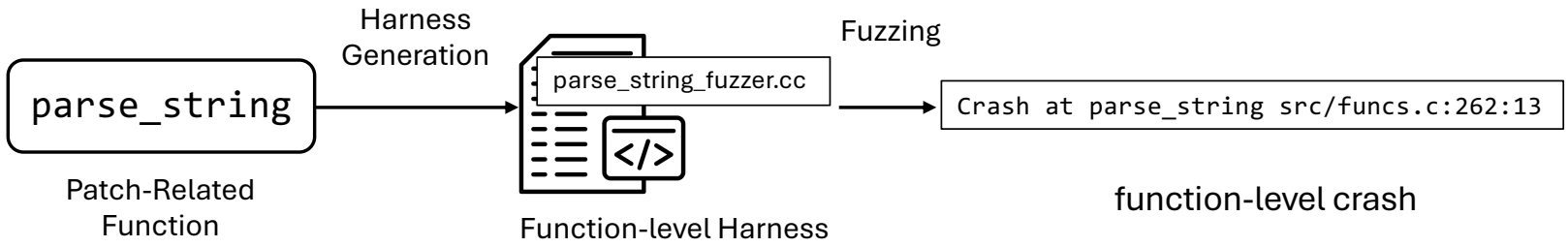


Limitations of System Fuzzing.

1. Large search space.
2. Complicated bug-revealing constraint to solve.

Function-Level Fuzzing

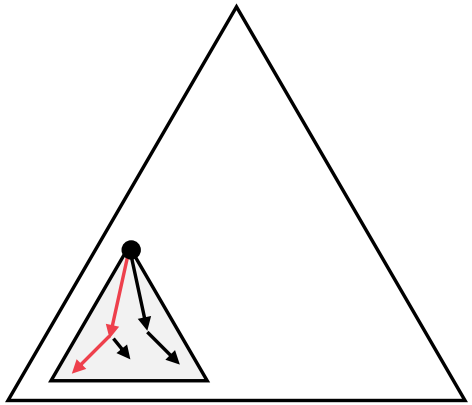
Idea: Fuzzing only patch-related functions (called as **function-level** fuzzing)



Benefits.

1. Large search space. → **Small search space**.
2. Complicated bug-revealing constraint to solve.
   → **Simpler bug constraint**.

→ faster regression bug detection

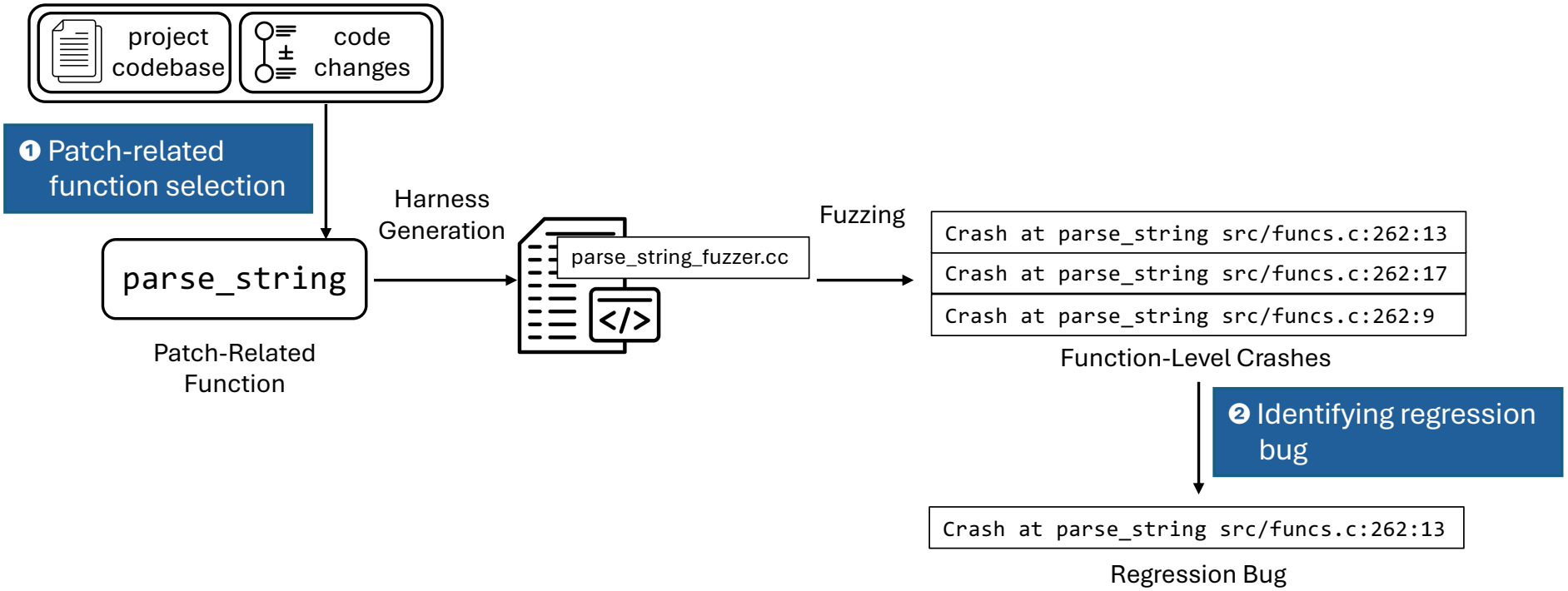


Fuction-Level Execution

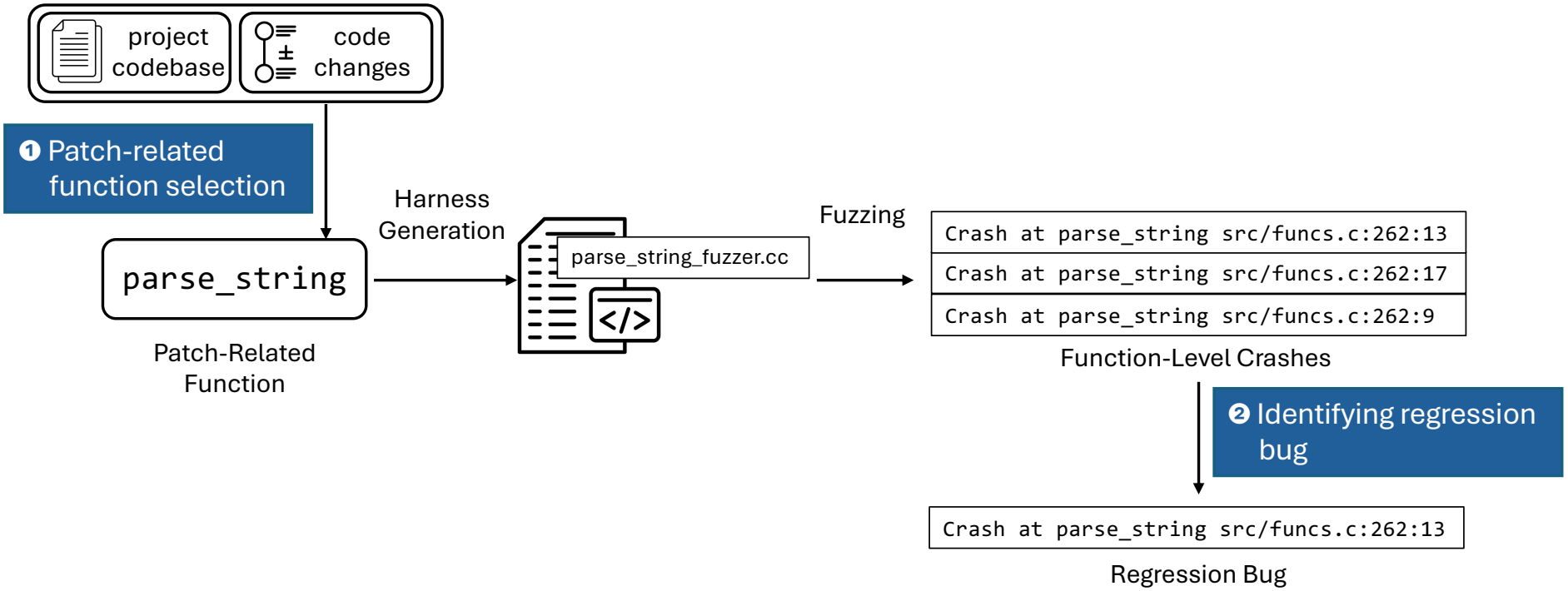
Challenges

❶ How to select **functions to test**, based on patch information?

❷ How to **identify regression bug** from collected function-level crashes?



Thesis Statement.
By utilizing **system execution data** in regression function-level fuzzing, it is possible to **❶ select patch-related functions** for fuzzing and effectively **❷ identify regression bug**.



System Execution Data

Function Coverage

List of **functions reached** during each execution.
(used in ❶ Patch-related function selection)

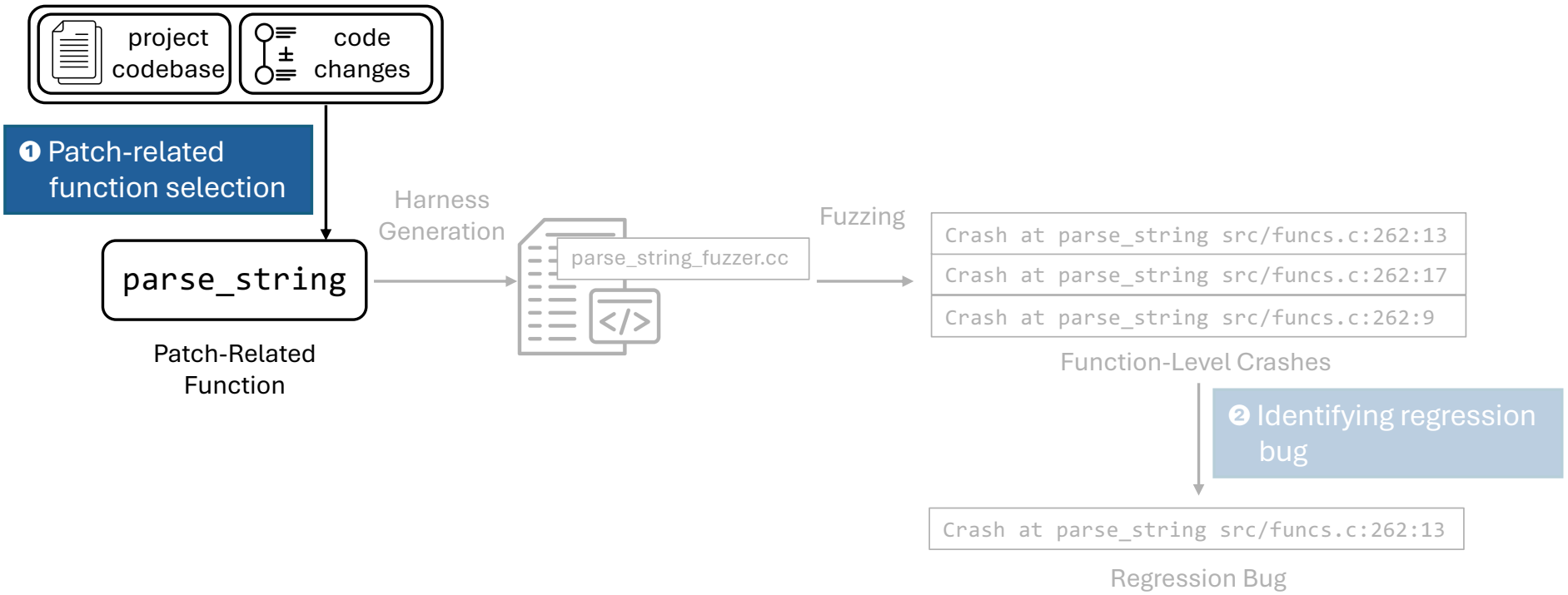
	F1	F2	F3	F4	F5
TC1	O	O	X	X	X
TC2	O	X	O	O	X
TC3	O	X	O	O	O
TC4	O	O	O	X	X

Function Input

All the **inputs** observed at the beginning of each function call during each execution.
(used in ❷ Identifying regression bug)

```
parse_string(char *str, int len)
TC1: str = “hello”, len = 5
TC2: str = “gr(a|e)y”, len = 8
⋮
TCn: str = “z{3,6}”, len = 6
```

❶ Patch-Related Function Selection



❶ Patch-Related Function Selection

Basic Approach. Select only functions updated by bug-inducing commit.

Limitation: Regression bug may occur in functions didn't updated by patch.

```
31  function main() {
        // ...
35      message, type = parse(content)
36      run(message, type)
        // ...
41  }
```

```
103  function parse(content) {
104      if (is_type0(content)) {
             // ...
108          return {message: message, type: 0}
109      } else if (is_type1(content)) {
             // ...
113          return {message: message, type: 1}
114      }
115  +   else if (is_type2(content)) {
116  +       message = parse_type2(content)
117  +       return {message: message, type: 2}
118  +   }
         // ...
121  }
```

```
72  function run(message, type) {
        // ...
75      switch type {
76          case 0:
77              run_type0(message)
78              break
79          case 1:
80              run_type1(message)
81              break
82          default:
83              panic("unreachable!") ⚠
84      }
85  }
```

❶ Patch-Related Function Selection

Idea. Select functions ‘related’ to updated functions.

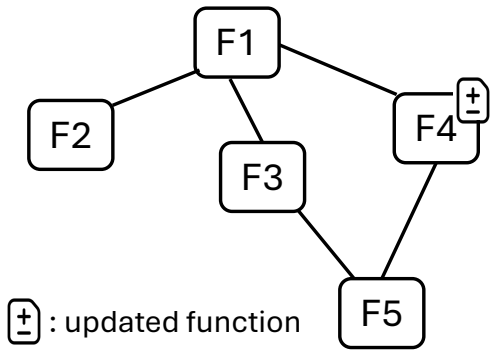
Static Call Distance

Test functions that have a **short static call distance** to the updated functions.

$$\text{StaticScore}(F) = \frac{1}{d(F, \text{updated}) + 1}$$

Test Priority of Functions

$F_4 > F_1 = F_5 > F_2 = F_3$



$d(F_1, F_4) = 1$
$d(F_2, F_4) = 2$
$d(F_3, F_4) = 2$
$d(F_4, F_4) = 0$
$d(F_5, F_4) = 1$

Dynamic Function Relevance

Test functions that are likely to **execute together with** the updated functions.

❶ Patch-Related Function Selection

Idea. Select functions ‘related’ to updated functions.

Static Call Distance

Test functions that have a **short static call distance** to the updated functions.

Dynamic Function Relevance

Test functions that are likely to **execute together with** the updated functions.

$$\text{DynamicScore}(F) = P(\text{updated}|F) \times P(F|\text{updated})$$

Test Priority of Functions

$$F_4 > F_3 > F_1 = F_5 > F_2$$

: updated function

	F1	F2	F3	F4	F5
TC1	O	O	X	X	X
TC2	O	X	O	O	X
TC3	O	X	O	O	O
TC4	O	O	O	X	X

$$P(F_1|F_4) \times P(F_4|F_1) = 100\% \times 50\% = 50\%$$
$$P(F_2|F_4) \times P(F_4|F_2) = 0\% \times 0\% = 0\%$$
$$P(F_3|F_4) \times P(F_4|F_3) = 100\% \times 67\% = 67\%$$
$$P(F_4|F_4) \times P(F_4|F_4) = 100\% \times 100\% = 100\%$$
$$P(F_5|F_4) \times P(F_4|F_5) = 50\% \times 100\% = 50\%$$

❶ Patch-Related Function Selection

Idea. Select functions ‘related’ to updated functions.

Static Call Distance

Test functions that have a **short static call distance** to the updated functions.

- Pros: **Stable** and **fairly accurate**.
- Cons: It only considers **static** information.

$$\mathrm{StaticScore}(F) = \frac{1}{d(F, \mathrm{updated}) + 1}$$

Dynamic Function Relevance

Test functions that are likely to **execute together with** the updated functions.

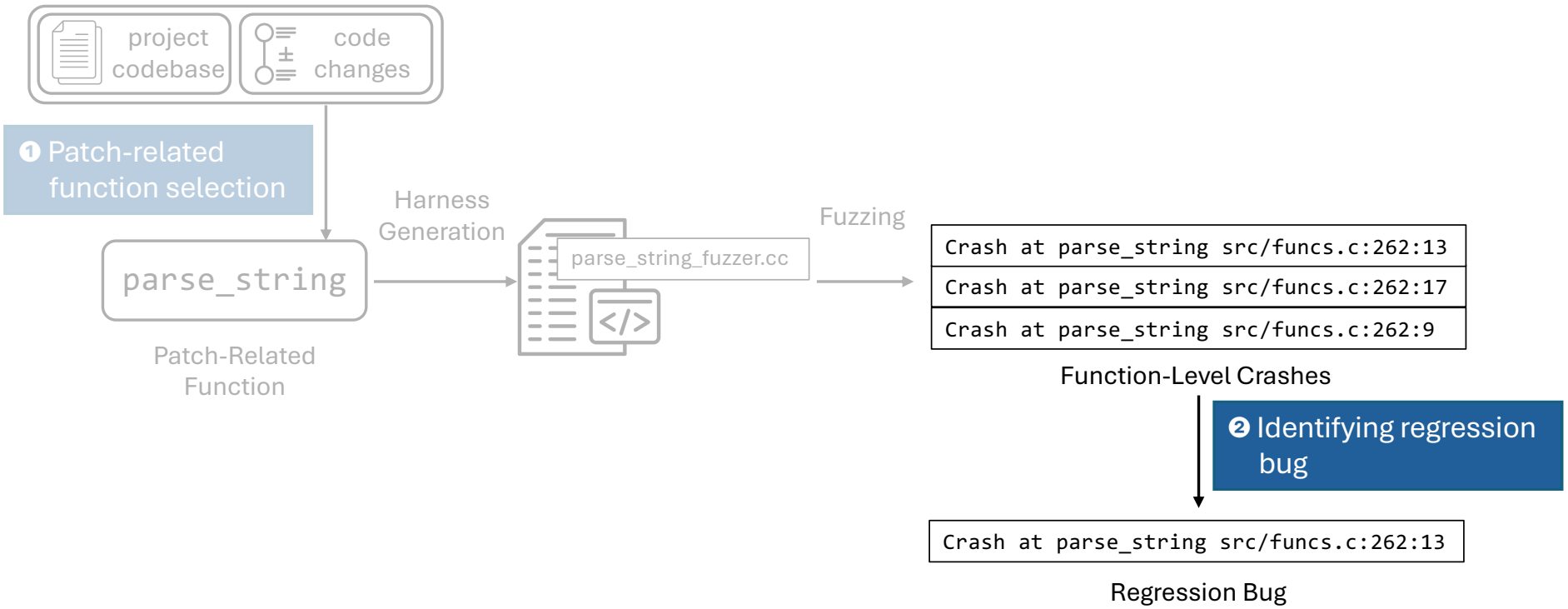
- Pros: Mesarues how two functions are related in **actual execution**.
- Cons: Depends on the **quality of system test cases**.

$$\begin{aligned}&\mathrm{DynamicScore}(F)\\ &\quad = P(\mathrm{updated}|F) \times P(F|\mathrm{updated})\end{aligned}$$

Ensemble: Select top-n functions with highest scores where

$$\mathrm{Score}(F) = \mathrm{StaticScore}(F) + \mathrm{DynamicScore}(F)$$

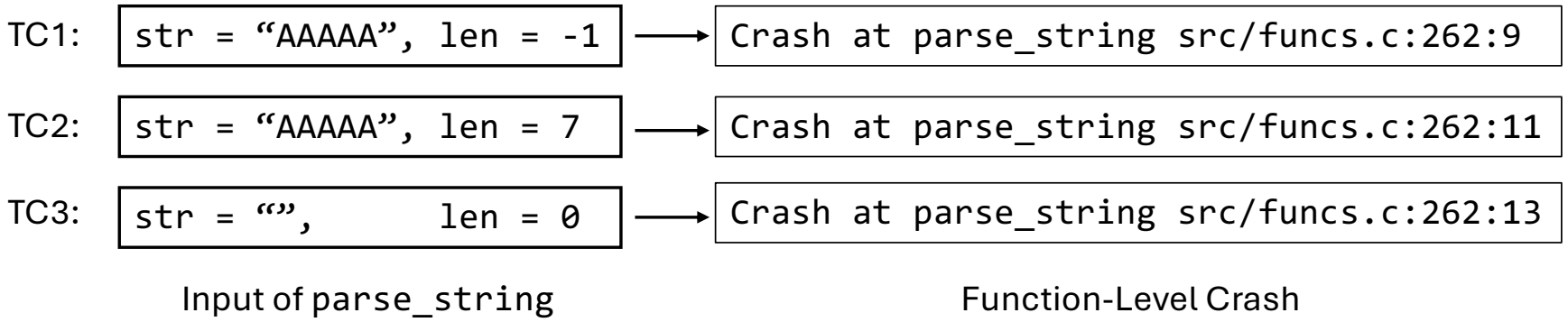
❷ Identifying Regression Bug



❷ Identifying Regression Bug

Function-level testcases are not always ‘valid’!

Suppose we collected following test cases by function-level fuzzing on `parse_string(char *str, int len)`.

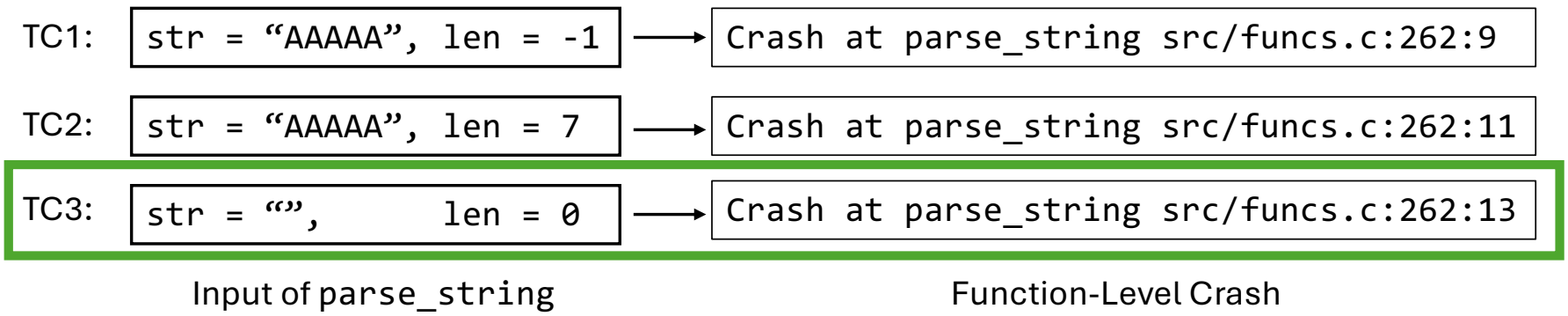


Can we say that all three crashes represent **valid** bugs?

❷ Identifying Regression Bug

Function-level testcases are not always ‘valid’!

Suppose we collected following test cases by function-level fuzzing on `parse_string(char *str, int len)`.

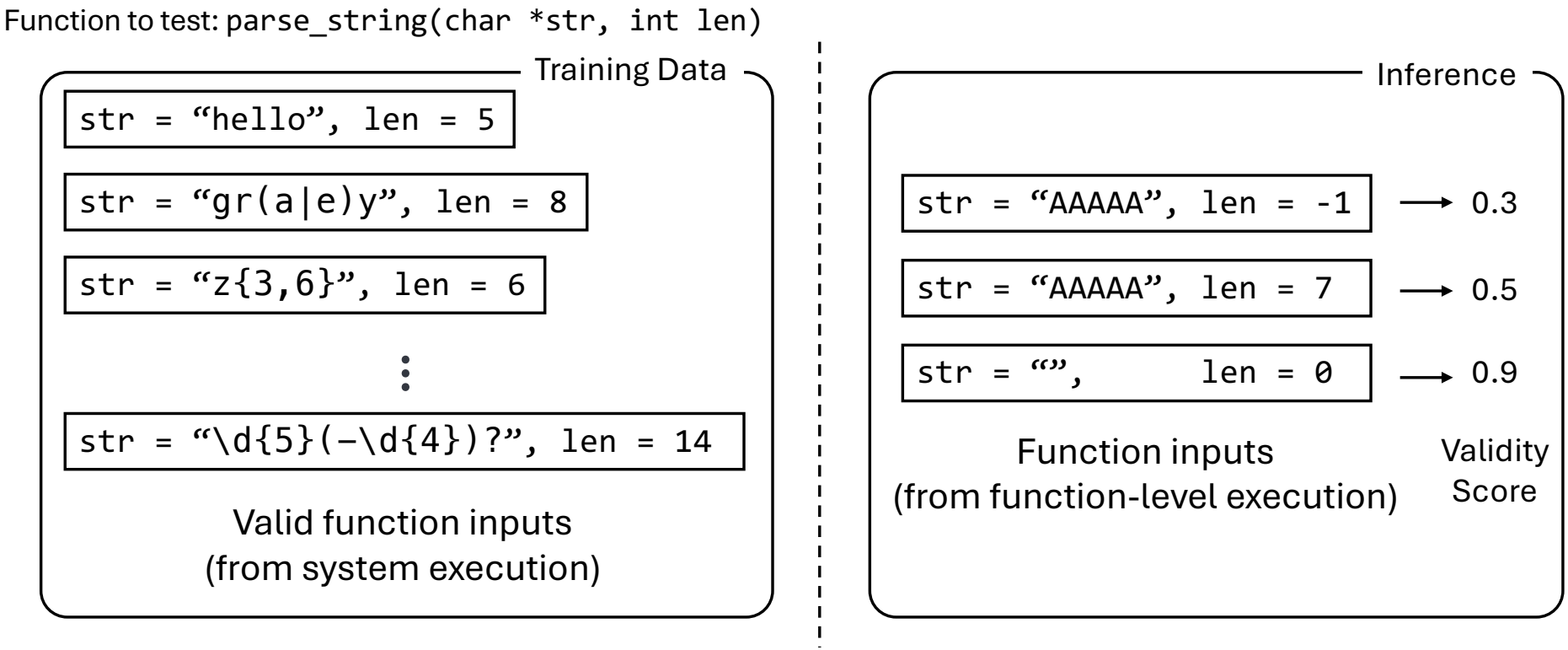


Only **TC3** is **valid** because it satisfies
the expected input constraint $\mathbf{length(str) = len}$.

❷ Identifying Regression Bug

Validity Estimation of Function Input

Idea: Function inputs observed in **system executions** are always valid.



❷ Identifying Regression Bug

Validity Estimation of Function Input

Validity estimation model

- Input: Function F, Function Input I.
- Output: Validity score $\in [0, 1]$

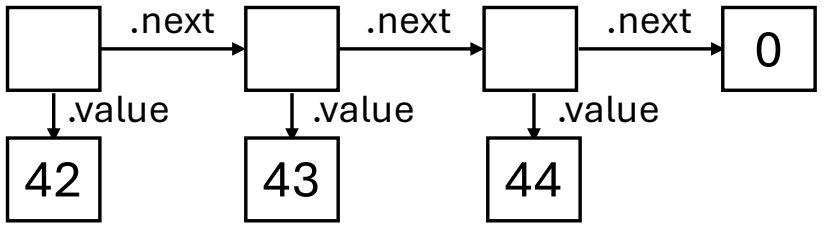
Input Features:

Function: **Text** (source code)

```
int get_size(struct node *head) {
      struct node *current = head;
      int count = 0;
      while (current != NULL) {
          count++
          current = current->next;
      }
      return count;
}
```

Function Input: **Graph**

- Node (primitive values)
- Edge (pointer / class element relation)

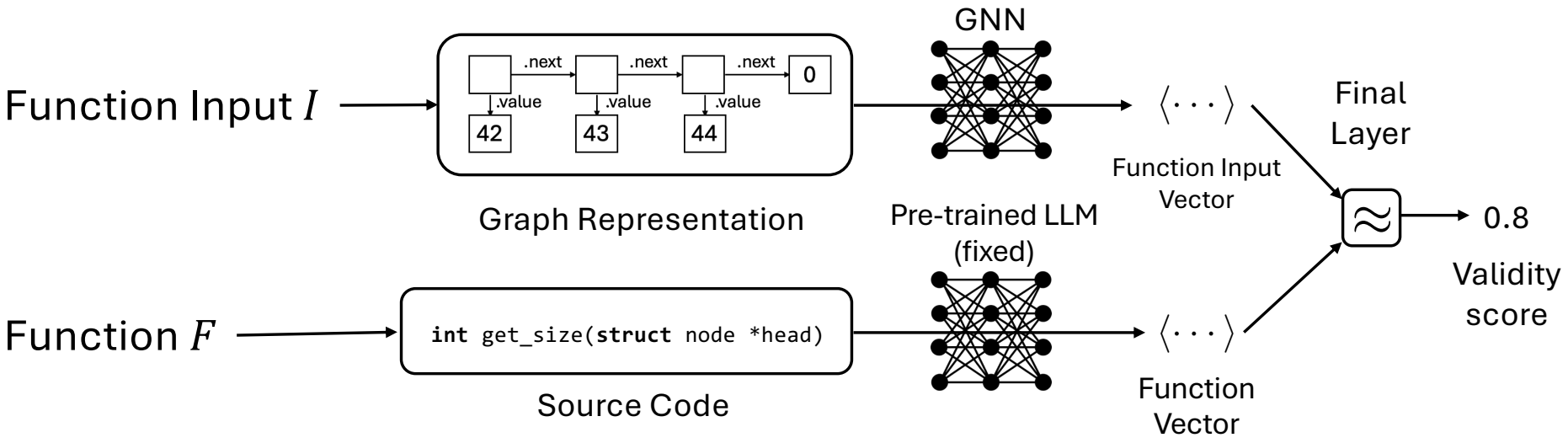


❷ Identifying Regression Bug

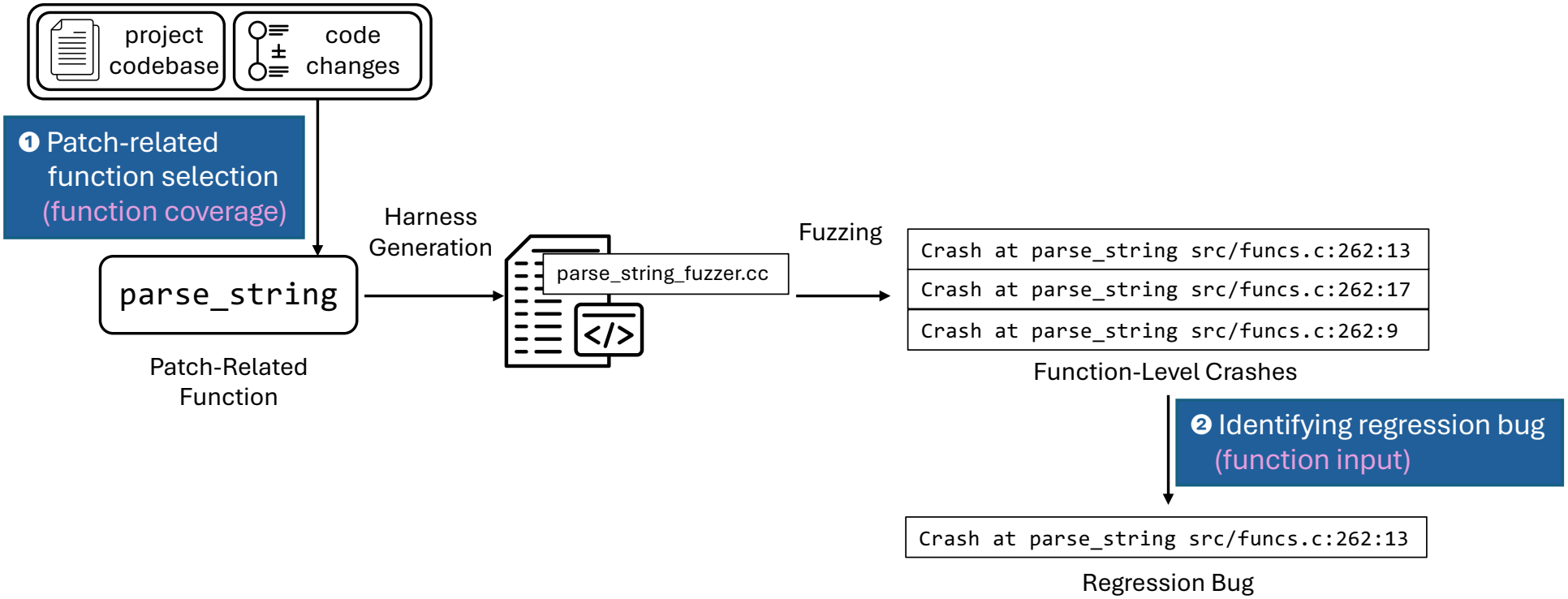
Validity Estimation of Function Input

Validity estimation model

- Input: Function F, Function Input I.
- Output: Validity score $\in [0, 1]$



Thesis Statement.
By utilizing **system execution data** in regression function-level fuzzing, it is possible to **❶ select patch-related functions** for fuzzing and effectively **❷ identify regression bug**.



Evaluation Settings

Benchmark

- Past OSSFuzz regression bugs collected from BugOSS [1] and AFLChurn [2]
- Use buggy-inducing-commit version.

Project	OSSFuzz	Version	Size (LoC)	Project	OSSFuzz	Version	Size (LoC)
curl	8000	`dd7521`	110619	libxml2	17737	`1fbcf4`	202998
file	30222	`6de368`	15036	ndpi	49057	`2edfae`	47302
leptonica	25212	`8fc490`	185268	picotls	13837	`7122ea`	18928
libgit2	11382	`7fafec`	159300	readstat	13262	`1de4f3`	25612
libhtp	17918	`3c6555`	14697	yara	38952	`5cc28d`	44531

System Corpus (for system execution data)

- Fuzzing with AFL++ with the default OSSFuzz setting.
- Aggregate outputs of 5 parallel fuzzing instances, executed for 12hrs

Subject	# of System Test Cases	Line Coverage	Branch Coverage
curl-8000	58004	53.90%	46.97%
file-30222	34677	21.18%	18.91%
leptonica-25212	6641	2.69%	4.62%
libgit2-11382	17064	3.38%	2.67%
libhtp-17918	78338	65.21%	60.06%
libxml2-17737	281767	19.13%	21.36%
ndpi-49057	126552	61.09%	57.62%
picotls-13837	2963	4.52%	2.75%
readstat-13262	11660	49.84%	49.91%
yara-38952	16459	19.28%	15.47%
Average	63412.5	30.02%	28.03%

[1] J.Kim and S. Hong, “Poster: BugOSS: A Regression Bug Benchmark for Empirical Study of Regression Fuzzing Techniques”, ICST ‘23.
[2] X. Zhu and M. Böhme, "Regression Greybox Fuzzing”, CCS ‘21

❶ Patch-Related Function Selection

Baseline approach

AFLChurn [2]**:** History-only based approach which gives more weight on (a) frequently updated and (b)recently updated lines

$$\text{score} = \frac{\log(\text{churn})}{\text{age}} \quad \cdots\cdots \text{(a)} \quad \cdots\cdots \text{(b)}$$

acc@n Summary

	AFLChurn	Proposed
acc@10	40%	80%
acc@5	30%	70%
acc@1	10%	50%

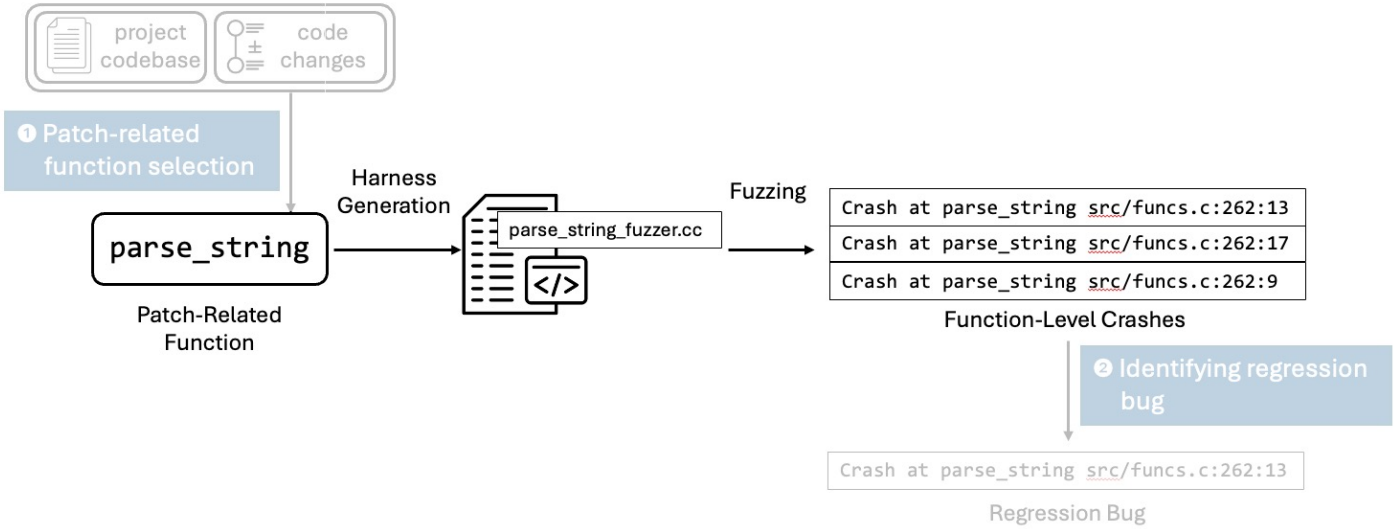
Ranking of crashing function reported by OSSFuzz

Subject	# of functions	AFLChurn	Proposed
curl-8000	1008	22	**2**
file-30222	414	**1**	**1**
leptonica-25212	2883	213	**73**
libgit2-11382	3541	**4**	**4**
libhtp-17918	428	19	**1**
libxml2-17737	2793	4	**1**
ndpi-49057	1375	14	**1**
picotls-13837	481	57	**6**
readstat-13262	196	144	**65**
yara-38952	690	6	**1**
Average	1380.9	48.4	**15.5**

[2] X. Zhu and M. Böhme, "Regression Greybox Fuzzing", CCS '21

❷ Identifying Regression Bug

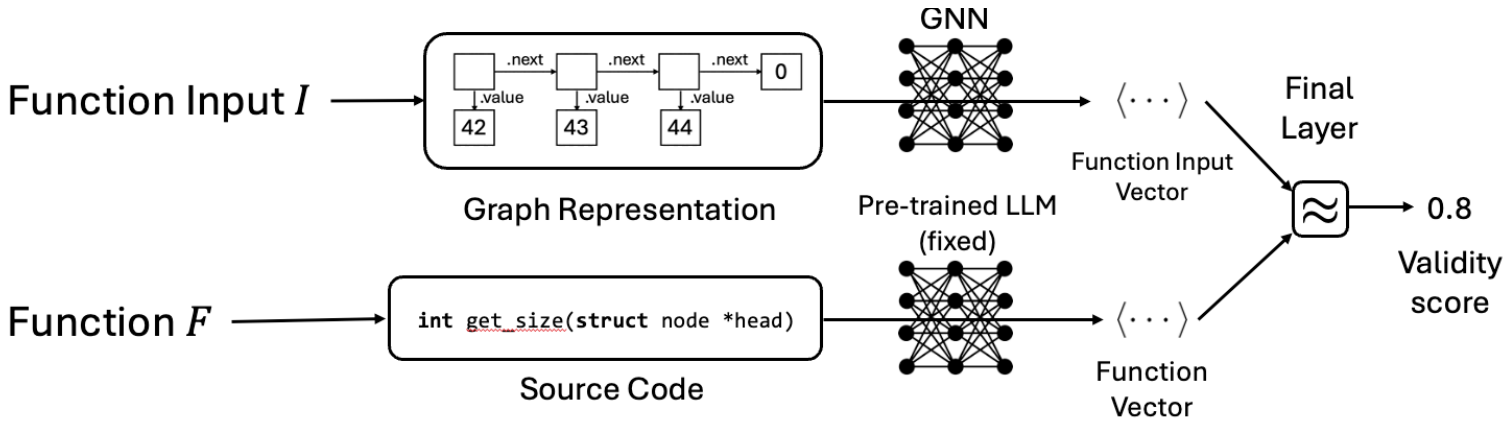
Function-Level Fuzzing Settings



- Target functions: **Top-10** functions from **❶ Patch-Related Function Selection** (Exclude the subjects that failed to find buggy function in top-10 ranking)
- Harness Generation: CROWN 2.0.
- Function-level Fuzzing: Run AFL++ for **1hr** for each function.

❷ Identifying Regression Bug

Validity Estimator Model Settings



- Training Data: **1,000 valid function inputs** for each function extracted from system executions.
- GNN: GCN (3 layers, $d_{hidden} = 1024$).
- Pre-trained LLM (fixed): CodeGen-350M-multi.
- Final layer: Cosine similarity.

❷ Identifying Regression Bug

Evaluation Settings

- Function-level crashes are deduplicated by crash location.
- Metric: Ranking of crash which have crash location identical to the OSSFuzz report.

Testcase	Crash Location	Validity Score
TC1	**Crash at A**	**0.84**
TC2	Crash at A	0.52
TC3	Crash at A	0.32
TC4	Crash at B	0.23
TC5	**Crash at B**	**0.45**
TC6	**Crash at C**	**0.86**
TC7	Crash at C	0.73

Total # of unique crashes: 3
Ranking of TC1 (crash at A): 2nd
Ranking of TC5 (crash at B): 3rd
Ranking of TC6 (crash at C): 1st

❷ Identifying Regression Bug

Evaluation Settings

- Function-level crashes are deduplicated by crash location.
- Metric: Ranking of crash which have crash location identical to the OSSFuzz report.

Results

Subject	# of Unique Crashes	Rank	Subject	# of Unique Crashes	Rank
curl-8000	44.2	8.4	libxml2-17737	74.6	8.0
file-30222	25.4	2.2	ndpi-49057	29.0	-
libgit2-11382	23.8	6.8	picotls-13837	76.0	12.4
libhtp-17918	102.8	10.6	yara-38952	57.8	32.4
Average				59.4	11.54

- By checking average **top 11.54** crashes, regression bug can be found.
- By checking top 10 crashes in 4/7 subjects (acc@10 = 57.14%), regression bug can be found.

Thesis Statement.
By utilizing **system execution data** in regression function-level fuzzing,
it is possible to **❶ select patch-related functions** for fuzzing
and effectively **❷ identify regression bug**.

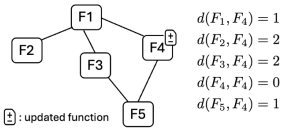
❶ Patch-Related Function Selection

Static Call Distance

Test functions that have a short static call distance to the updated functions.

$$\text{StaticScore}(F) = \frac{1}{d(F_1, \text{updated}) + 1}$$

Test Priority of Functions
$F_4 > F_1 = F_5 > F_2 = F_3$



$d(F_1, F_4) = 1$
$d(F_2, F_4) = 2$
$d(F_3, F_4) = 2$
$d(F_4, F_4) = 0$
$d(F_5, F_4) = 1$

Dynamic Function Relevance

Test functions that are likely to execute together with the updated functions.

$$\text{DynamicScore}(F) = P(\text{updated}|F) \times P(F|\text{updated})$$

Test Priority of Functions
$F_4 > F_3 > F_1 = F_5 > F_2$

: updated function

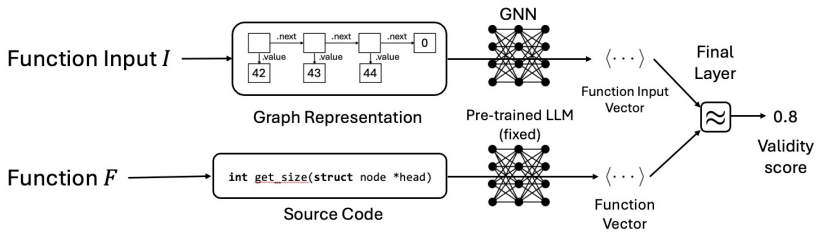
	F1	F2	F3	F4	F5
TC1	O	O	X	X	X
TC2	O	X	O	O	X
TC3	O	X	O	O	O
TC4	O	O	O	X	X

$P(F_1|F_4) \times P(F_4|F_1) = 100\% \times 50\% = 50\%$
$P(F_2|F_4) \times P(F_4|F_2) = 0\% \times 0\% = 0\%$
$P(F_3|F_4) \times P(F_4|F_3) = 100\% \times 67\% = 67\%$
$P(F_4|F_4) \times P(F_4|F_4) = 100\% \times 100\% = 100\%$
$P(F_5|F_4) \times P(F_4|F_5) = 50\% \times 100\% = 50\%$

❷ Identifying Regression Bug

Validity Estimation of Function Input

Validity estimation model
- Input: Function F, Function Input I.
- Output: Validity score ∈ $[0, 1]$



19

acc@10 of buggy function ranking: 80%

Regression bug in top 11.54 crashes

Future Direction.
1. Deep reasoning and study on ❷ validity estimation of function input .
2. Validity-guided automated harness generation and fuzzing.

Publication

Y. Choi, A. Lee, M. Kim, “Dynamic Unit State Data-Driven False Alarm Filtering for Regression Unit Testing”, *Journal of KIISE, Vol.51, No.11, 2024*.

* this thesis proposes a GNN-based approach for crash validity estimation while the publication introduced an LLM-based crash validity estimator.