Neuro-Symbolic Execution: Augmenting Symbolic Execution with Neural Constraints

Shiqi Shen Shweta Shinde Soundarya Ramesh
Abhik Roychoudhury Prateek Saxena

National University of Singapore

Symbolic Execution for Bug Finding

KLEE

Pex

Angr

Kite

SAGE

jCUTE

Manticore

TRITON

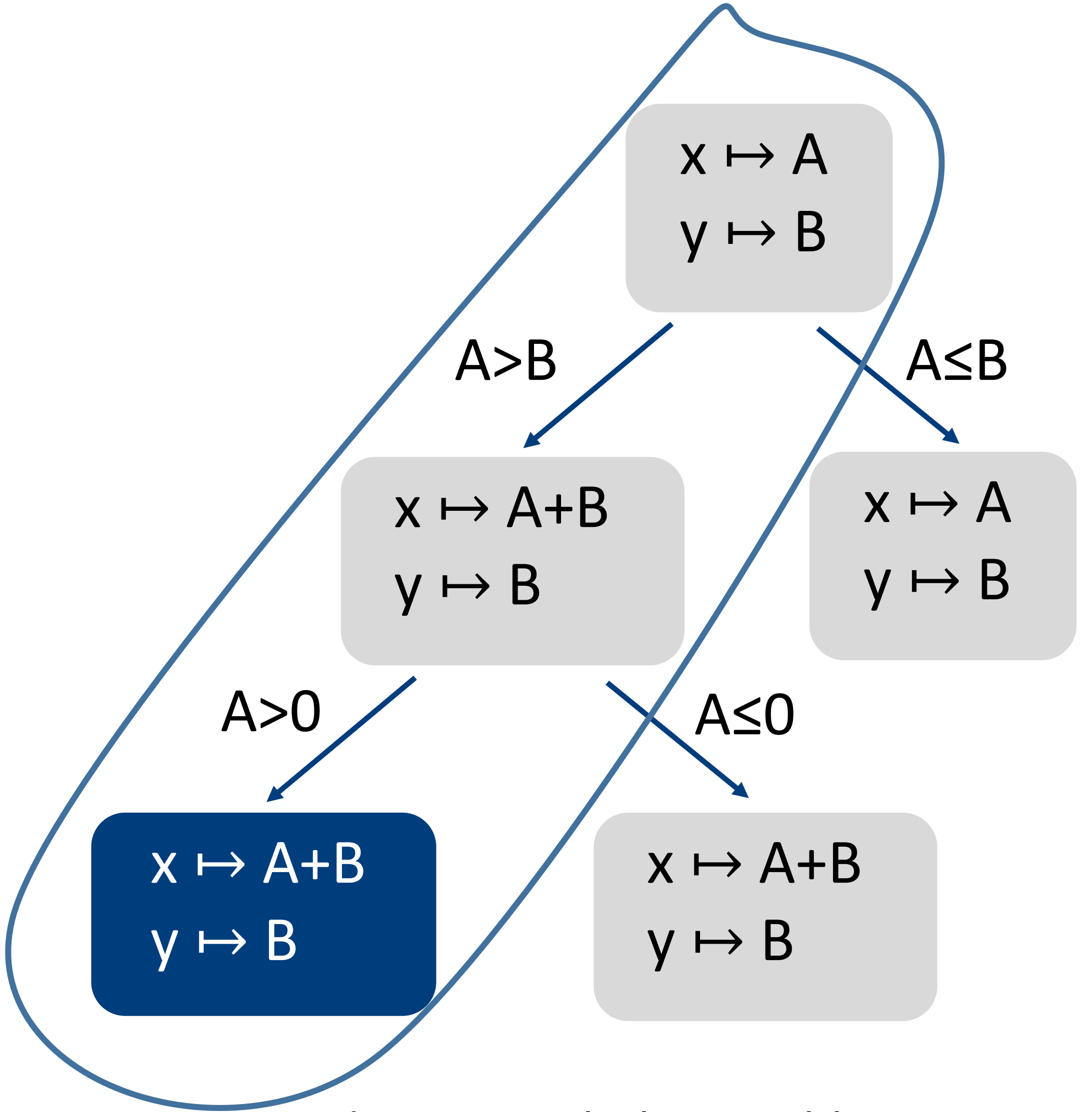
Dynamic Binary Analysis

S^{2}E

Recap: Symbolic Execution (SE)

```
def f (x, y):
 if (x>y)
    x = x+y
    if (x-y > 0)
     assert false
 return (x, y)
```

Dynamic Symbolic Execution (DSE):
A widely used variation of SE



x ↦ A
y ↦ B

A>B

A≤B

x ↦ A+B
y ↦ B

x ↦ A
y ↦ B

A>0

A≤0

x ↦ A+B
y ↦ B

x ↦ A+B
y ↦ B

A and B are symbolic variables

Fundamental Limitations of Classic DSE

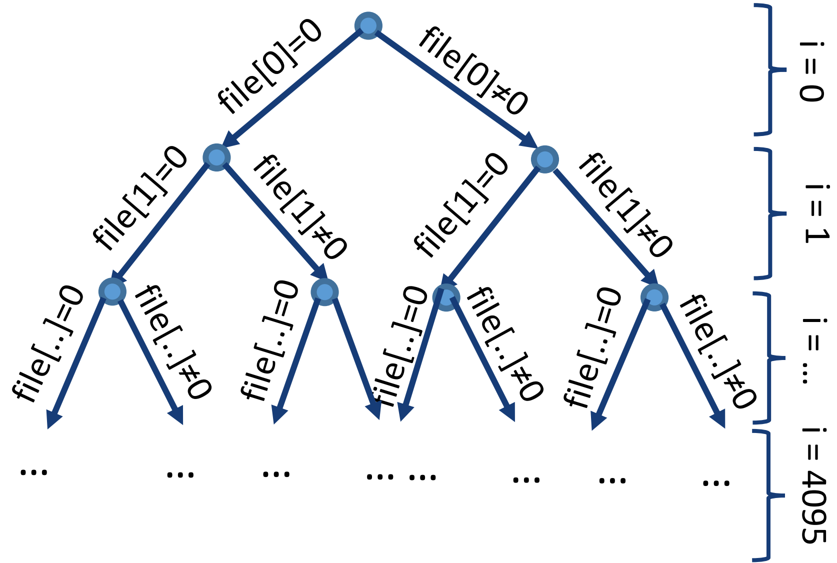
```
int main (…) {
  if (strlen(filename)>1 && filename[0]==‘-’)
    exit(1)
  copy_data(…);
  …
}
void copy_data(..., int *file,...) {
  static double data[4096], value;
  read_double_value(file, ... );
  value = fabs (data [0]);
  for(i=0; i<4096; i++)
    if(file[i] == 0.0) count++;
  data[1] /= (value+count-3);
  …
}
```

#1 Limitations of SMT Solvers

#2 Unmodeled Semantics

#3 Path Explosion

Candidate Vulerability Point (CVP): Divide-by-zero



2^{4096} **paths**

Contributions

- **Neuro-symbolic execution**
  - A new approach to tackle the limitations of DSE
  - Reasons about exact (symbolic) & approximate constraints (neural nets)
- **A Tool – NeuEx**
  - Enhances the widely used DSE engine (KLEE)
- **Evaluation**
  - Finds **94%** more bugs than KLEE in 12 hours

Problem

Inputs:

1. Source code
2. Symbolic Variables (e.g., `filename` & `file`)
3. Candidate Vulnerability Points (CVPs)
    - Divide by zero
    - Buffer overflow

Outputs:

Validated Exploits

```
int main (…) {
  if (strlen(filename)>1 && filename[0]==‘-’)
    exit(1)
  copy_data(…);
  …
}
void copy_data(..., int *file...) {
  static double data[4096], value;
  read_double_value(file, ... );
  value = fabs (data [0]);
  for(i=0; i<4096; i++)
    if(file[i] == 0.0) count++;
  data[1] /= (value+count-3);
  …
}
```

CVP: Divide-by-zero

Key Insights

Values of Symbolic Variables

Values of Vulnerable Variables in CVP

Learn an approximation with small number of I/O examples

```
int main (…) {
  if (strlen filename >1 && filename[0]==‘-’)
    exit(1)
  copy_data(…);
  …
}
void copy_data(..., int * file ...) {





  data[1] /= (value+count-3);
  …
}
```

CVP: Divide-by-zero

Key Insights

Approximation:

$$count == \Sigma_{i\in[0,4095]}\ sign(file[i] == 0)$$
$$\wedge\ a == file[0] + 256\ file[1]$$
$$\wedge\ s == sign(file[1] < 127)$$
$$\wedge\ \max == (2\times s - 1)\times a\ - 256^2 \times (s\ - 1)$$

Machine learning can learn such an approximation

```
int main (…) {
  if (strlen filename >1 && filename[0]==‘-’)
    exit(1)
  copy_data(…);
  …
}
void copy_data(..., int * file ...) {
  static double data[4096], value;
  read_double_value(file, ... );
  value = fabs (data [0]);
  for(i=0; i<4096; i++)
    if(file[i] == 0.0) count++;
  data[1] /= (value+count-3);
  …
}
```

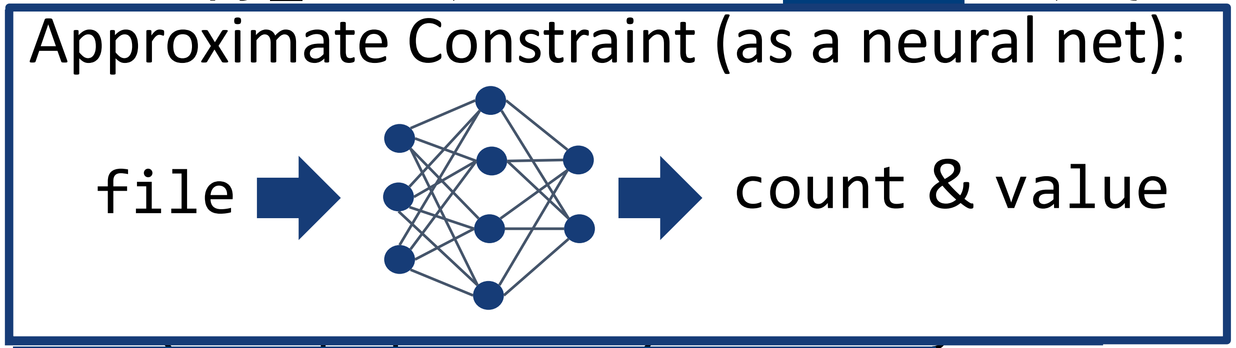
CVP: Divide-by-zero

Approach

1. Neural nets can represent a large category of functions (universal approximation theorem).

2. Multiple applications show that neural nets are learnable for many practical functions.

```
int main (…) {
  if (strlen filename >1 && filename[0]==‘-’)
    exit(1)
  copy_data(…);
  …
}
void copy_data(..., int * file ...) {
```



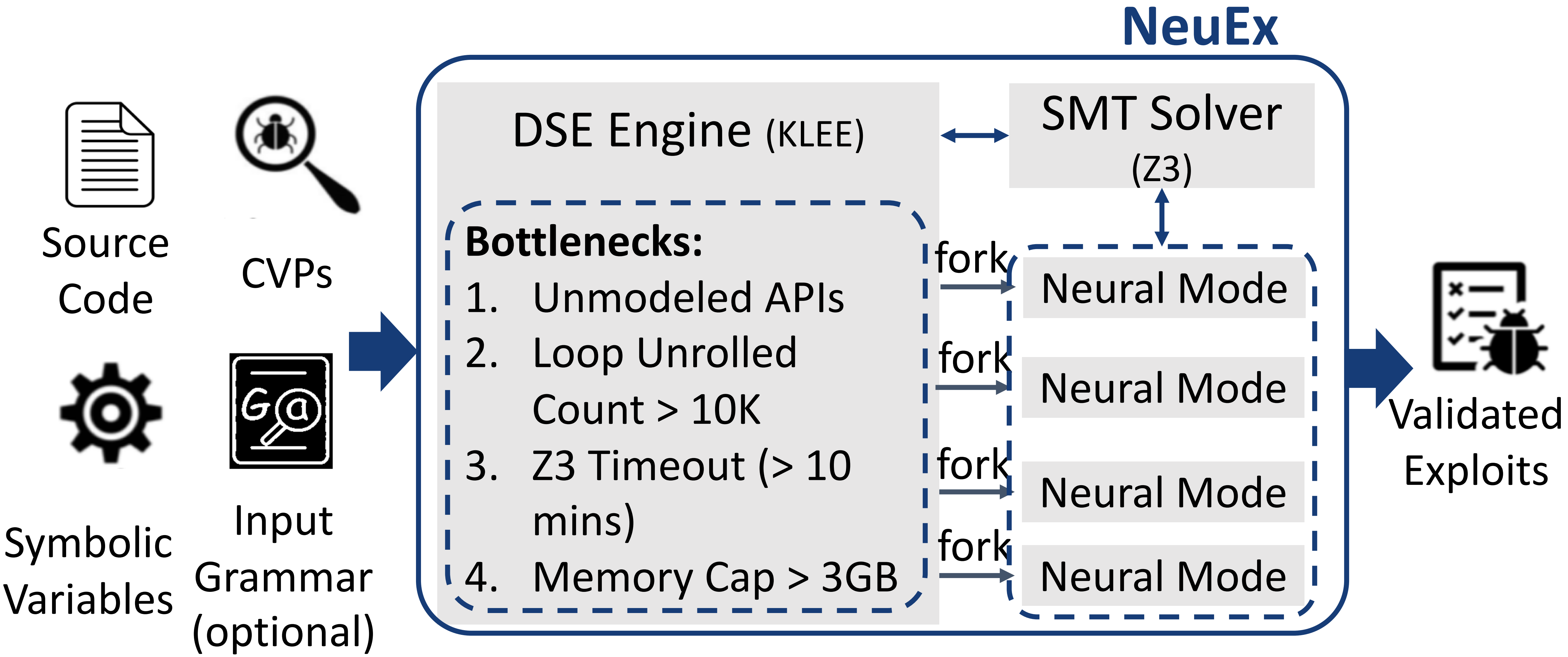
Approximate Constraint (as a neural net):

file → count & value

```
  data[1] /= (value+count-3);
  …
}
```

CVP: Divide-by-zero

NeuEx Overview



Source Code

CVPs

Symbolic Variables

Input Grammar (optional)

NeuEx

DSE Engine (KLEE)

SMT Solver (Z3)

Bottlenecks:

1. Unmodeled APIs
2. Loop Unrolled Count > 10K
3. Z3 Timeout (> 10 mins)
4. Memory Cap > 3GB

Neural Mode

Neural Mode

Neural Mode

Neural Mode

Validated Exploits

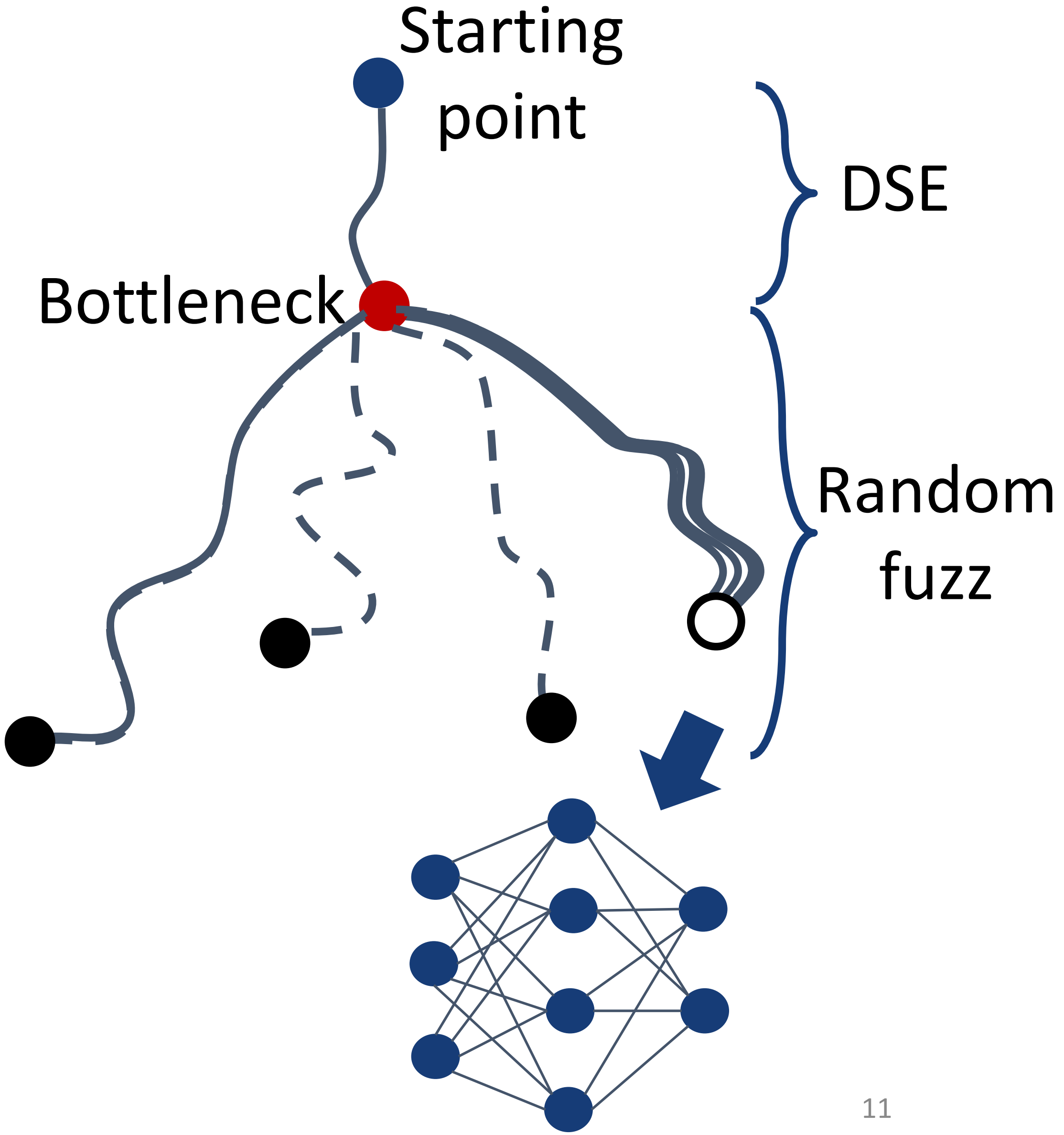
Neural Mode

Reaching CVPs

Collecting I/O Examples

Training

Solving



Generated Constraints

1. Reachability constraints:

$$strlen(filename) \leq 1$$
$$\lor\ filename \neq '-'$$

$$\land$$

$$N\colon infile \rightarrow (value, count)$$

2. Vulnerability condition:

$$value + count - 3 == 0$$

```
int main (…) {
  if (strlen(filename)>1 && filename[0]==‘-’)
    exit(1)
  copy_data(…);
  …
}
void copy_data(..., int *file,...) {
  static double data[4096], value;
  read_double_value(file, ... );
  value = fabs (data [0]);
  for(i=0; i<4096; i++)
    if(file[i] == 0.0) count++;
  data[1] /= (value+count-3);
  …
}
```

CVP: Divide-by-zero

Constraint Solving

1. Reachability constraints:

$$strlen(filename) \leq 1$$
$$\vee\ filename \neq '-'$$

$\wedge$

$$N: infile \rightarrow (\boxed{value, count})$$

2. Vulnerability condition:

$$\boxed{value + count} - 3 == 0$$

Purely symbolic constraints:
No variable shared with neural constraints

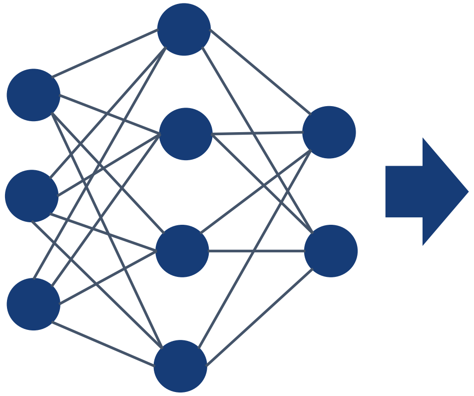
➡ SMT solver

Mixed constraints:
Including both neural constraints and symbolic constraints with shared variables

How to Solve Mixed Constraints?

Design Choice 1:

Neural net → CNF



$(X_1 \wedge Y_1) \vee (X_2 \wedge Y_2) \vee \cdots \vee (X_n \wedge Y_n)$

The number of clauses increases drastically with the neural net complexity

Design Choice 2:

Symbolic constraints → Optimization objective of the neural net

Encoding Symbolic Constraints as an Optimization Objective

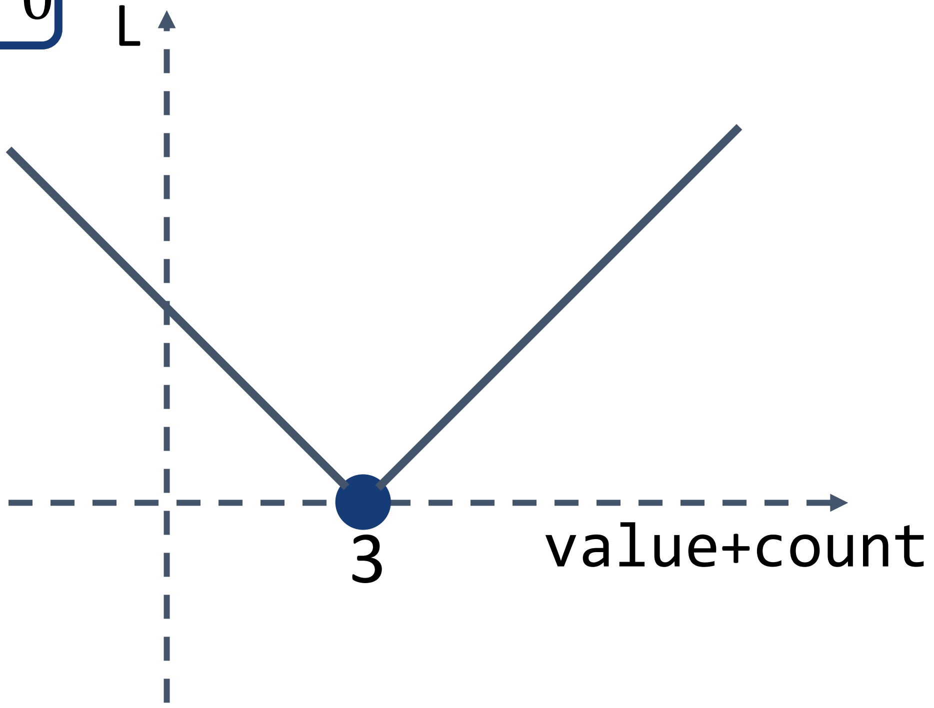
$N: infile \rightarrow (value, count) \wedge \boxed{value + count - 3 == 0}$

Symbolic constraint

Criterion for crafting the loss function:
The minimum point of the loss function satisfies the symbolic constraints.

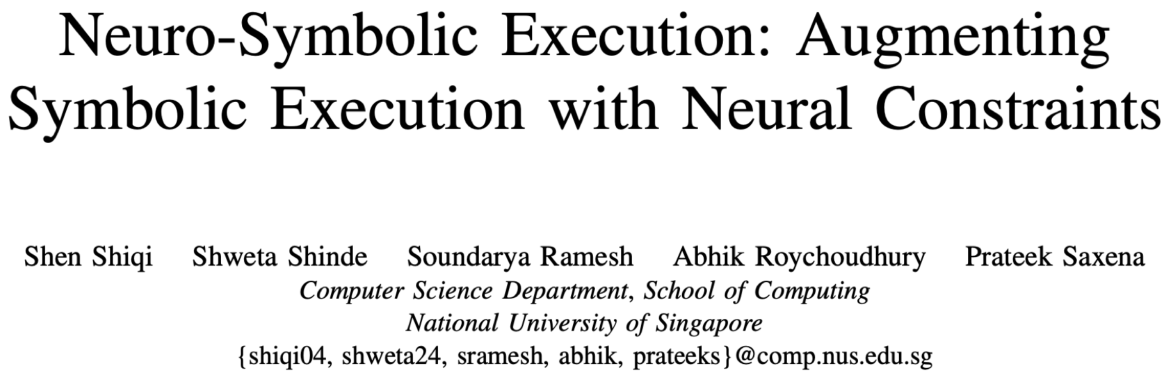
One possible encoding:

$$L = abs(value + count - 3)$$



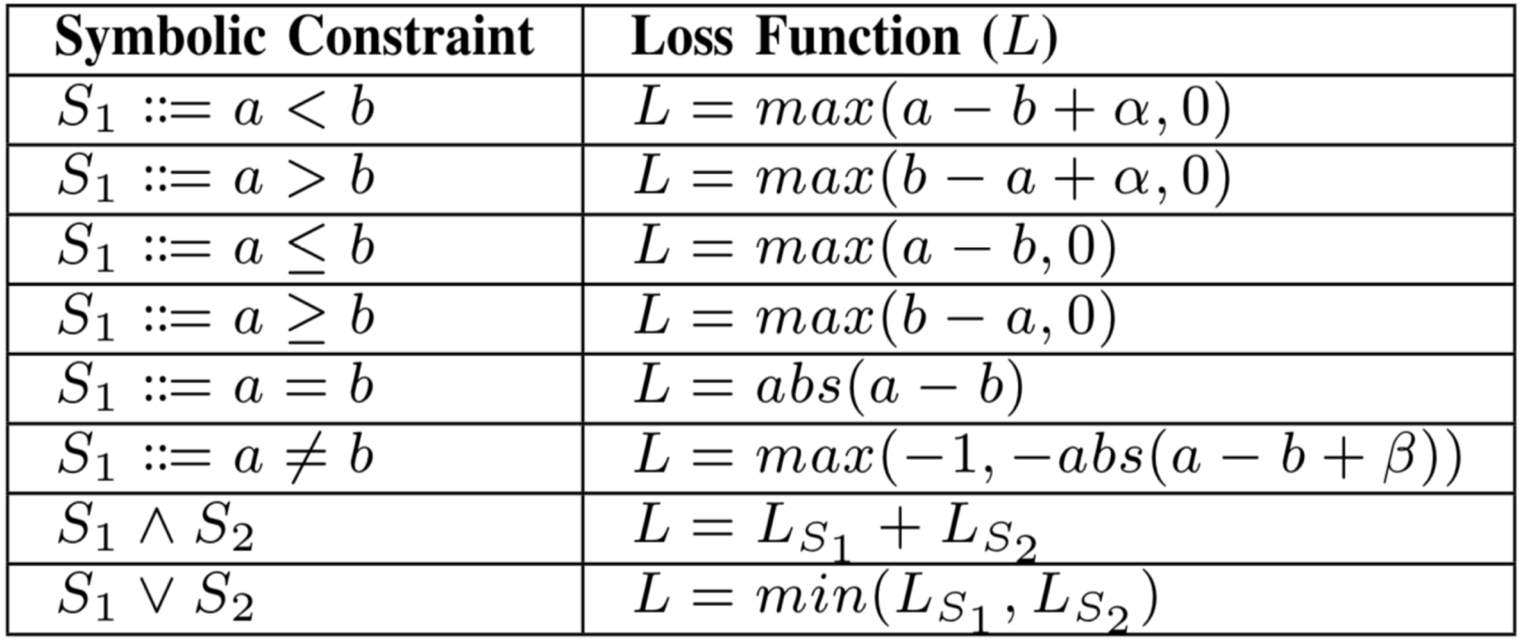
Encoding Symbolic Constraints as an Optimization Objective

NeuEx supports many symbolic constraints. Checkout the paper for the complete grammar and the corresponding loss functions.



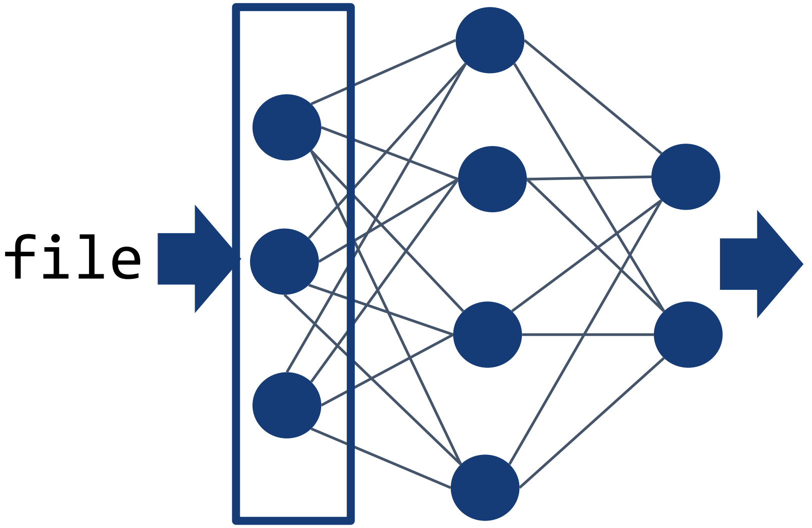
Neuro-Symbolic Execution: Augmenting Symbolic Execution with Neural Constraints

Shen Shiqi Shweta Shinde Soundarya Ramesh Abhik Roychoudhury Prateek Saxena
Computer Science Department, School of Computing
National University of Singapore
{shiqi04, shweta24, sramesh, abhik, prateeks}@comp.nus.edu.sg



Symbolic Constraint	**Loss Function** (L)
$S_1 ::= a < b$	$L = max(a - b + \alpha, 0)$
$S_1 ::= a > b$	$L = max(b - a + \alpha, 0)$
$S_1 ::= a \leq b$	$L = max(a - b, 0)$
$S_1 ::= a \geq b$	$L = max(b - a, 0)$
$S_1 ::= a = b$	$L = abs(a - b)$
$S_1 ::= a \neq b$	$L = max(-1, -abs(a - b + \beta))$
$S_1 \wedge S_2$	$L = L_{S_1} + L_{S_2}$
$S_1 \vee S_2$	$L = min(L_{S_1}, L_{S_2})$

Solving Mixed Constraints via Gradient Descent



Gradient: $\nabla_{file} L$

`file`

`count,value`

`count`	`value`	`loss`
257	20	274
1	20	18
0	20	17
…	…	…
0	3	0

`file`: 000…

Concretely validate the exploit

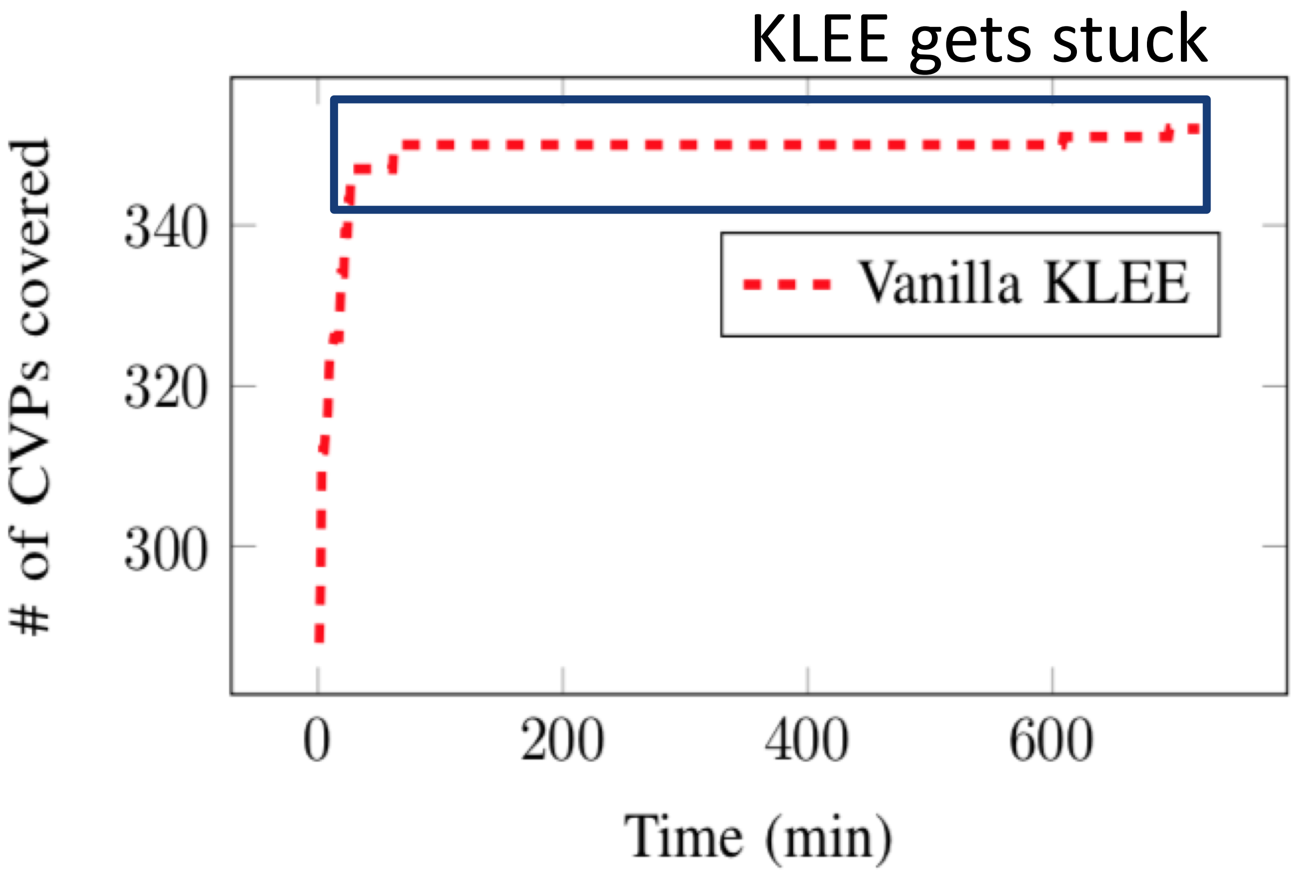
Evaluation

- **Recall**: Neural mode is only triggered when DSE encounters bottlenecks
- **Benchmarks**: 7 Programs known to be difficult for classic DSE
  - 4 Real programs
    - cURL: Data transferring
    - SQLite: Database
    - libTIFF: Image processing
    - libsndfile: Audio processing
  - LESE benchmarks
    - BIND, Sendmail, and WuFTP

Include:

1. Complex loops
2. Floating-point variables
3. Unmodeled APIs

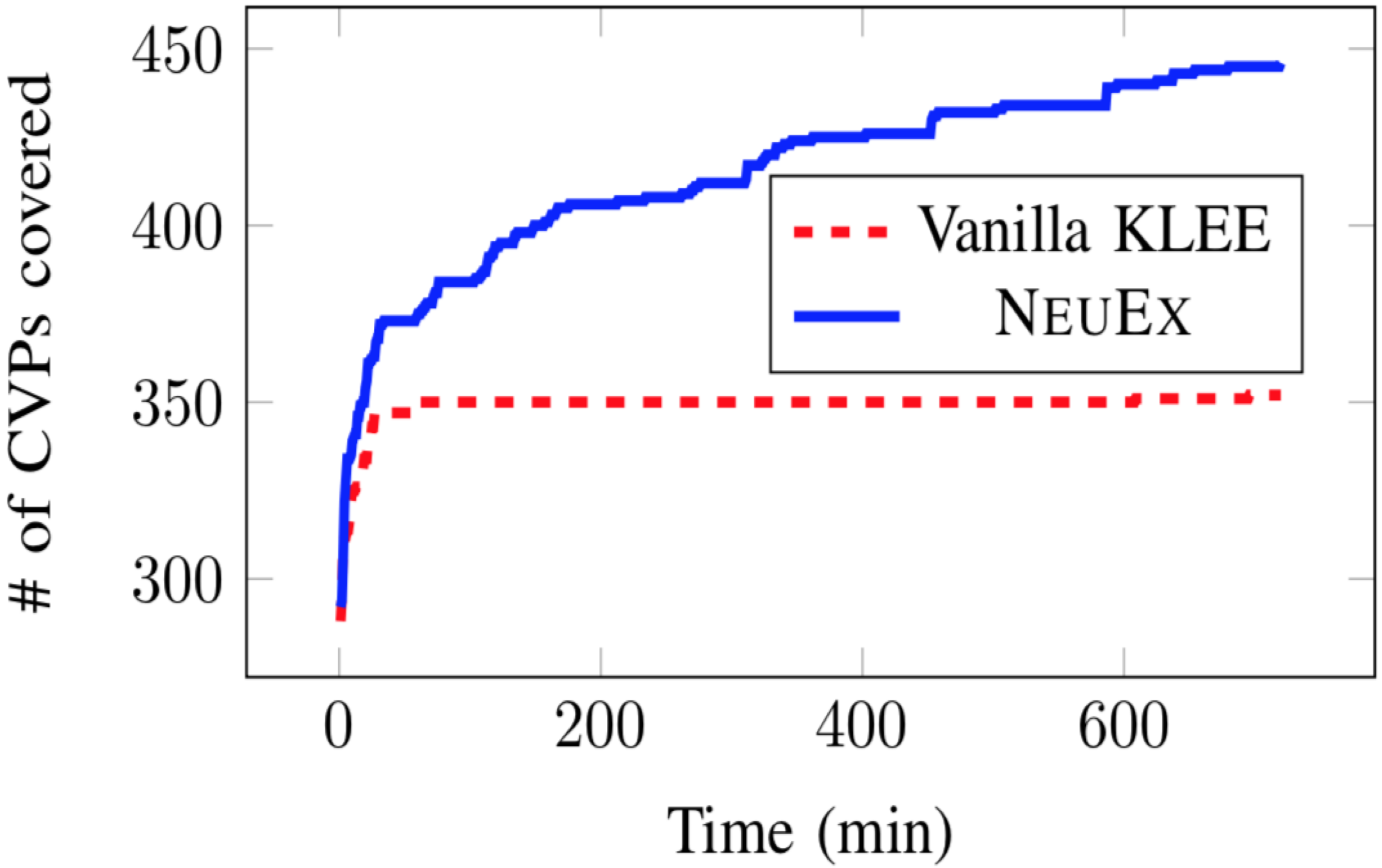
CVP Coverage & Bottlenecks for DSE



# of bottlenecks: 61

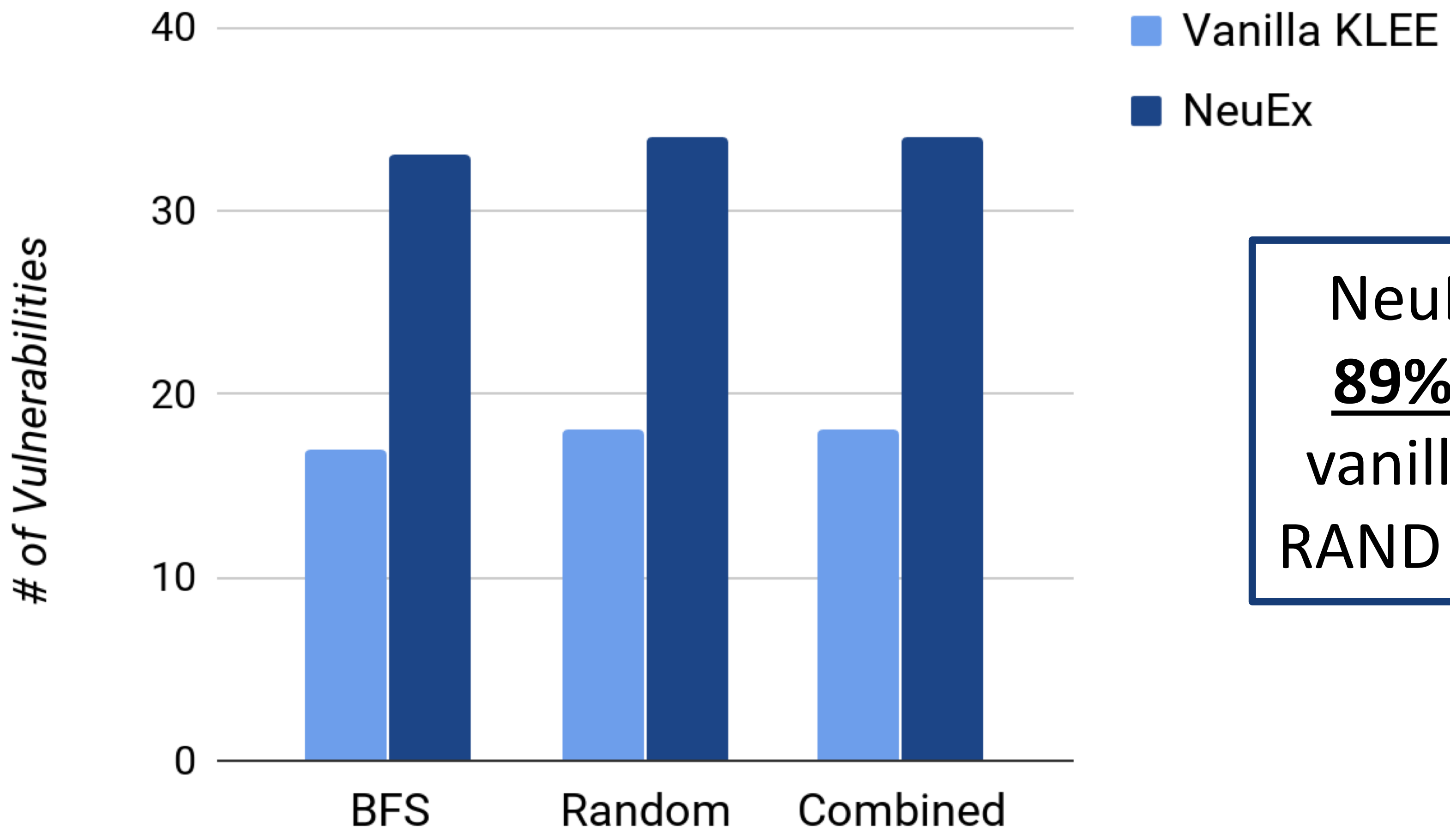
- Unmodeled APIs (6)
- Complicated loops (53)
- Z3 timeout (1)
- Memory exhaustion (1)

CVP Coverage of NeuEx vs KLEE



The number of CVPs reached or covered by NeuEx is **<u>25%</u>** higher than vanilla KLEE.

of Bugs Found by NeuEx vs KLEE



NeuEx finds **94%** and **89%** more bugs than vanilla KLEE in BFS and RAND mode in 12 hours.

Comparison to DSE Extensions

LESE

- Structured Approach for Loop Reasoning
  - **Two orders** of magnitude slower on average

Veritesting

- Combination of DSE + static analysis
  - **Fails to find bugs** in 12 hours
  - Poor instruction coverage

Key Takeaways

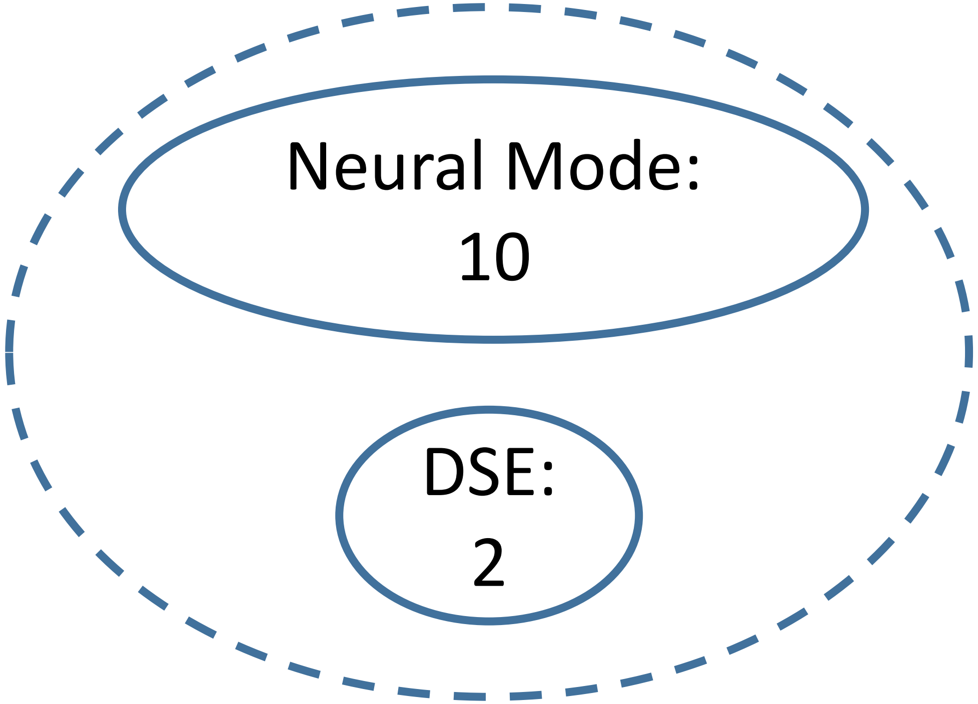
A new approach: Neuro-Symbolic Execution

- Resolves fundamental bottlenecks of DSE
- First to learn unstructured representation not amenable to SMT solver
- Unique use of optimization techniques and SMT for solving constraints

Evaluation

- Finds **94%** more bugs than KLEE within 12 hours

Backup: New Bugs



NeuEx: 12

Neural Mode:
10

DSE:
2