FuzzGuard: *Filtering out Unreachable Inputs in Directed Grey-box Fuzzing through Deep Learning*

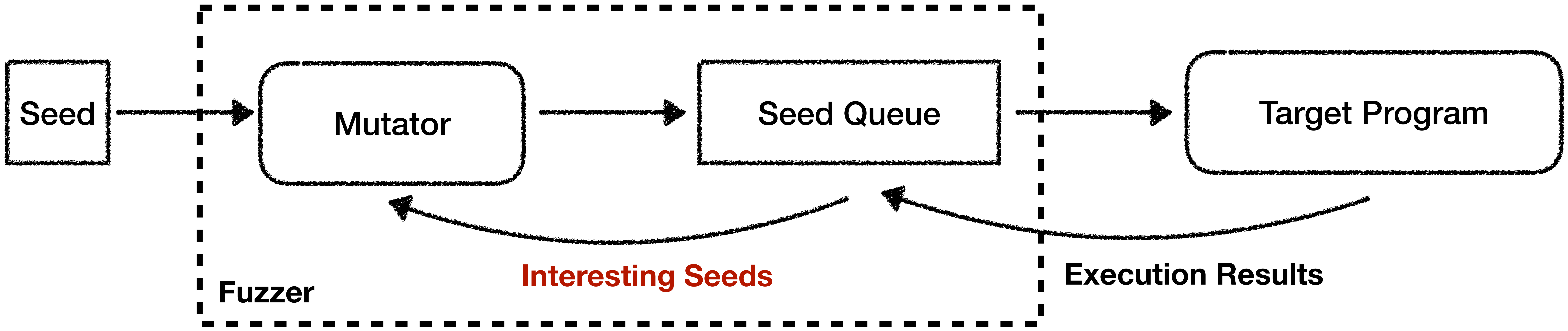
Peiyuan Zong[1,2], Tao Lv[1,2], Dawei Wang[1,2], Zizhuang Deng[1,2], Ruigang Liang[1,2], Kai Chen[1,2 *]

1 SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

2 School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

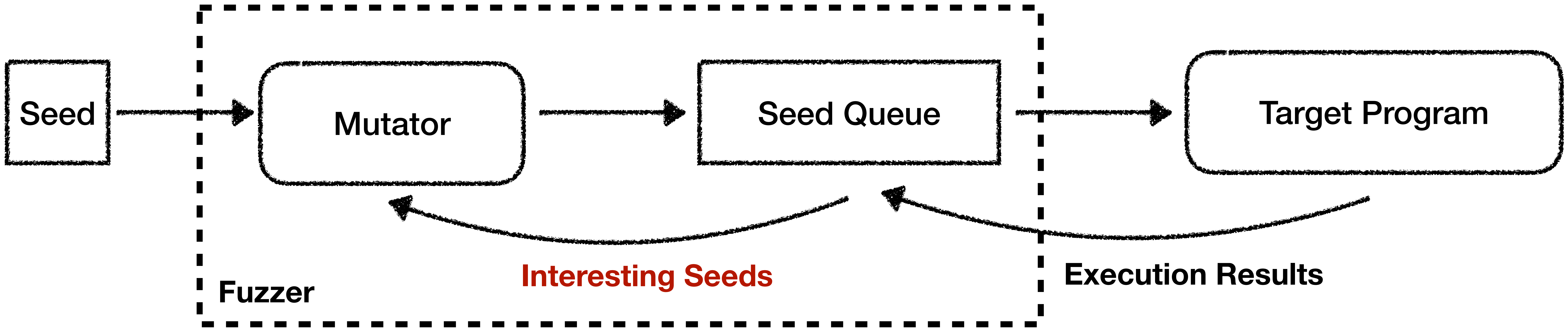
{zongpeiyuan, lvtao, wangdawei, dengzizhuang, liangruigang, chenkai}@iie.ac.cn

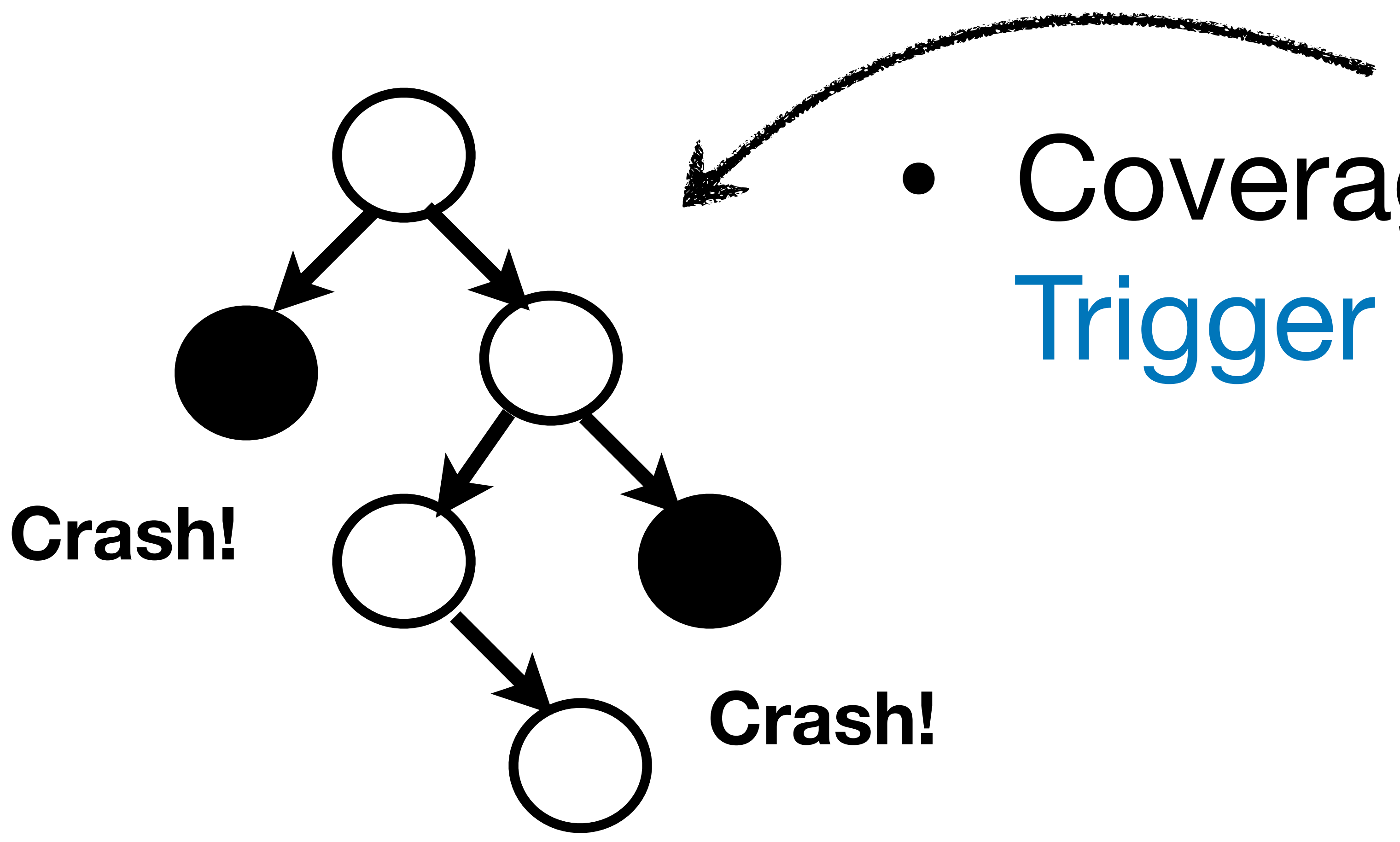
Mutation based Grey-box Fuzzing Overview



- Coverage-based Grey-box Fuzzing (CGF)
- Directed Grey-box Fuzzing (DGF)

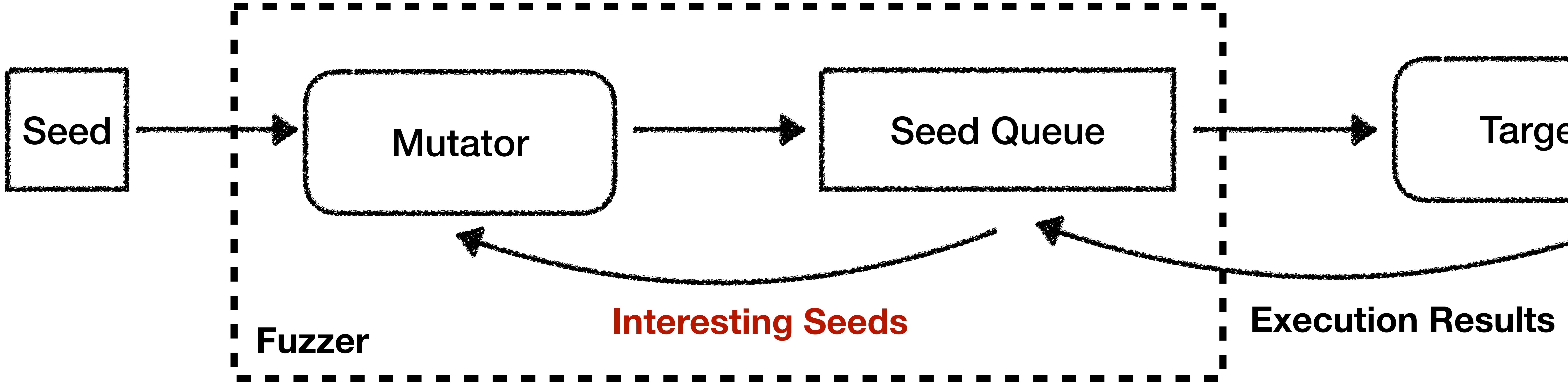
Mutation based Grey-box Fuzzing Overview

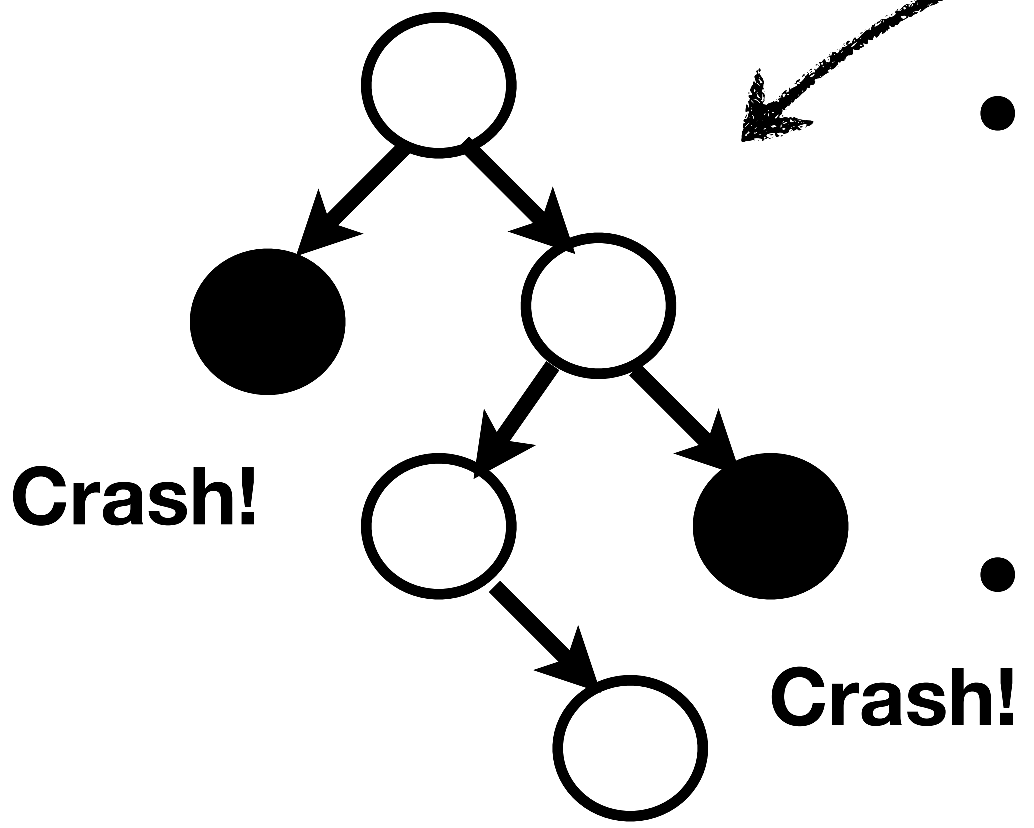




- Coverage-based Grey-box Fuzzing (CGF)
  Trigger more crashes.

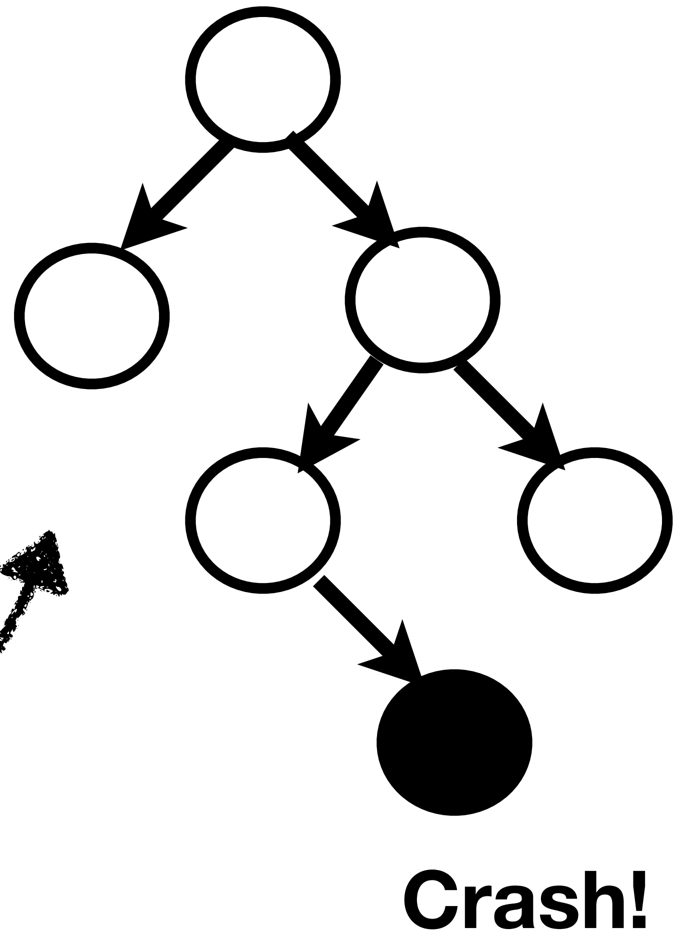
Mutation based Grey-box Fuzzing Overview





- Coverage-based Grey-box Fuzzing (CGF)
  Trigger more crashes.

- Directed Grey-box Fuzzing (DGF)
  Check whether a target code really contains a bug.



DGF: lots of inputs cannot reach the buggy code

- Crash reproduction
- Patch testing
- Potentially vulnerable code checking

```
  ThrowReaderException(...);
if (dib_info.colors_important > 256)
  ThrowReaderException(...);
if ((dib_info.image_size != 0U) && (dib_info.image_size
      > file_size))
  ThrowReaderException(...);
if ((dib_info.number_colors != 0) ||
      (dib_info.bits_per_pixel < 16)) {
    image->storage_class=PseudoClass;
```

Vulnerable code

DGF: lots of inputs cannot reach the buggy code

- Crash reproduction
- Patch testing
- Potentially vulnerable code checking

```
  ThrowReaderException(...);
if (dib_info.colors_important > 256)
  ThrowReaderException(...);
if ((dib_info.image_size != 0U) && (dib_info.image_size
      > file_size))
  ThrowReaderException(...);
if ((dib_info.number_colors != 0) ||
      (dib_info.bits_per_pixel < 16)) {
    image->storage_class=PseudoClass;
```

Vulnerable code

How to reach the vulnerable code?

- Annealing-based Power Schedules
  - Fuzz the input closer to the target longer.

DGF: lots of inputs cannot reach the buggy code

- Crash reproduction
- Patch testing
- Potentially vulnerable code checking

```
 ThrowReaderException(...);
if (dib_info.colors_important > 256)
 ThrowReaderException(...);
if ((dib_info.image_size != 0U) && (dib_info.image_size
     > file_size))
 ThrowReaderException(...);
if ((dib_info.number_colors != 0) ||
     (dib_info.bits_per_pixel < 16)) {
   image->storage_class=PseudoClass;
```

Vulnerable code

How to reach the vulnerable code?

- Annealing-based Power Schedules
  - Fuzz the input closer to the target longer.

Over 91.7% of the inputs missed the vulnerable code!

DGF: lots of inputs cannot reach the buggy code

- Crash reproduction
- Patch testing
- Potentially vulnerable code checking

```
 ThrowReaderException(...);
if (dib_info.colors_important > 256)
 ThrowReaderException(...);
if ((dib_info.image_size != 0U) && (dib_info.image_size
     > file_size))
 ThrowReaderException(...);
if ((dib_info.number_colors != 0) ||
     (dib_info.bits_per_pixel < 16)) {
  image->storage_class=PseudoClass;
```

Vulnerable code

How to reach the vulnerable code?

- Annealing-based Power Schedules
  - Fuzz the input closer to the target longer.

Over 91.7% of the inputs missed the vulnerable code!

- Symbolic execution
  - Solve path conditions for each new path.

DGF: lots of inputs cannot reach the buggy code

- Crash reproduction
- Patch testing
- Potentially vulnerable code checking

```
ThrowReaderException(...);
if (dib_info.colors_important > 256)
  ThrowReaderException(...);
if ((dib_info.image_size != 0U) && (dib_info.image_size
      > file_size))
  ThrowReaderException(...);
if ((dib_info.number_colors != 0) ||
      (dib_info.bits_per_pixel < 16)) {
    image->storage_class=PseudoClass;
```

Vulnerable code

How to reach the vulnerable code?

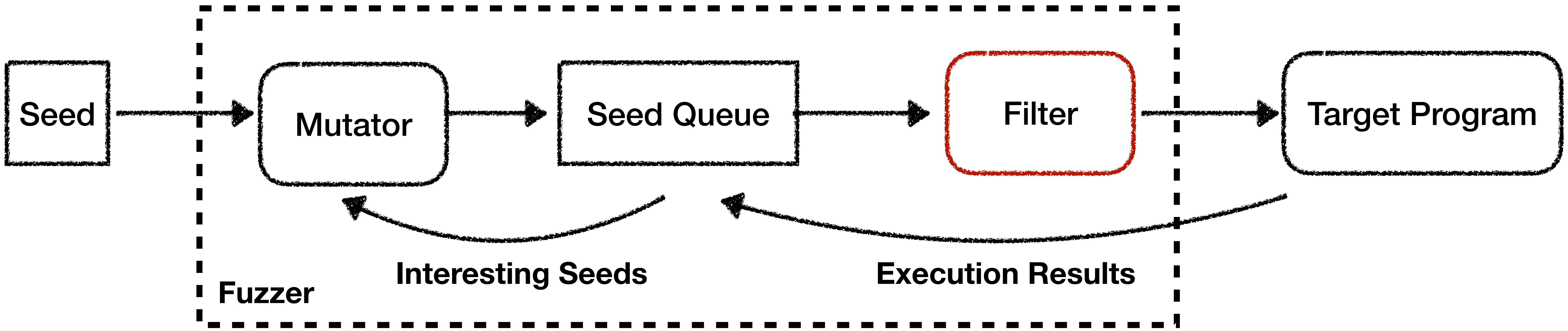
- Annealing-based Power Schedules
  - Fuzz the input closer to the target longer.

Over 91.7% of the inputs missed the vulnerable code!

- Symbolic execution
  - Solve path conditions for each new path.

High overhead requires!

Our approach: Build an input filter for the Fuzzer



- Build a **Deep Learning Model** (Filter)

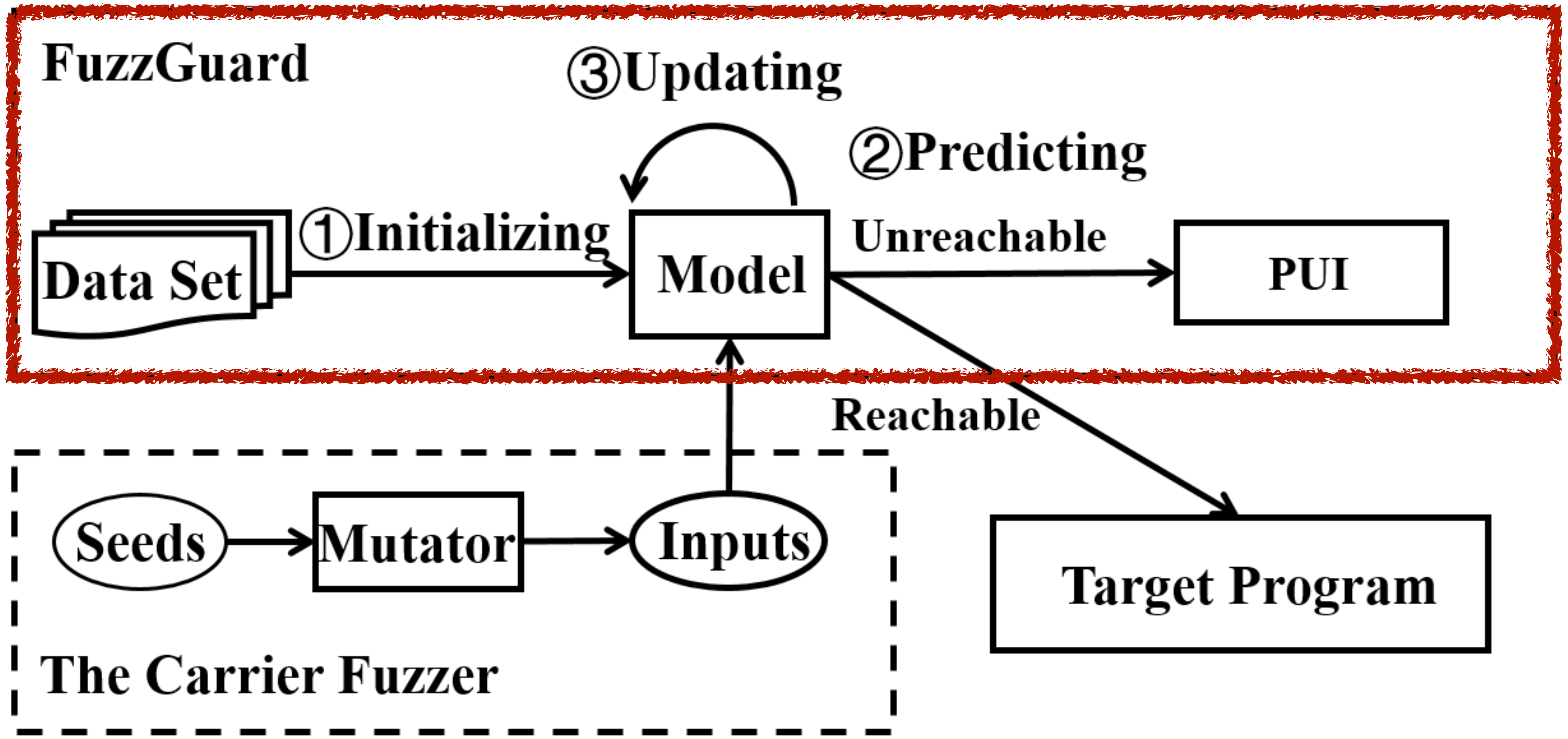
  Without running the target program

  - Learn from previous executions.
  - To identify the inputs which can reach the buggy code.

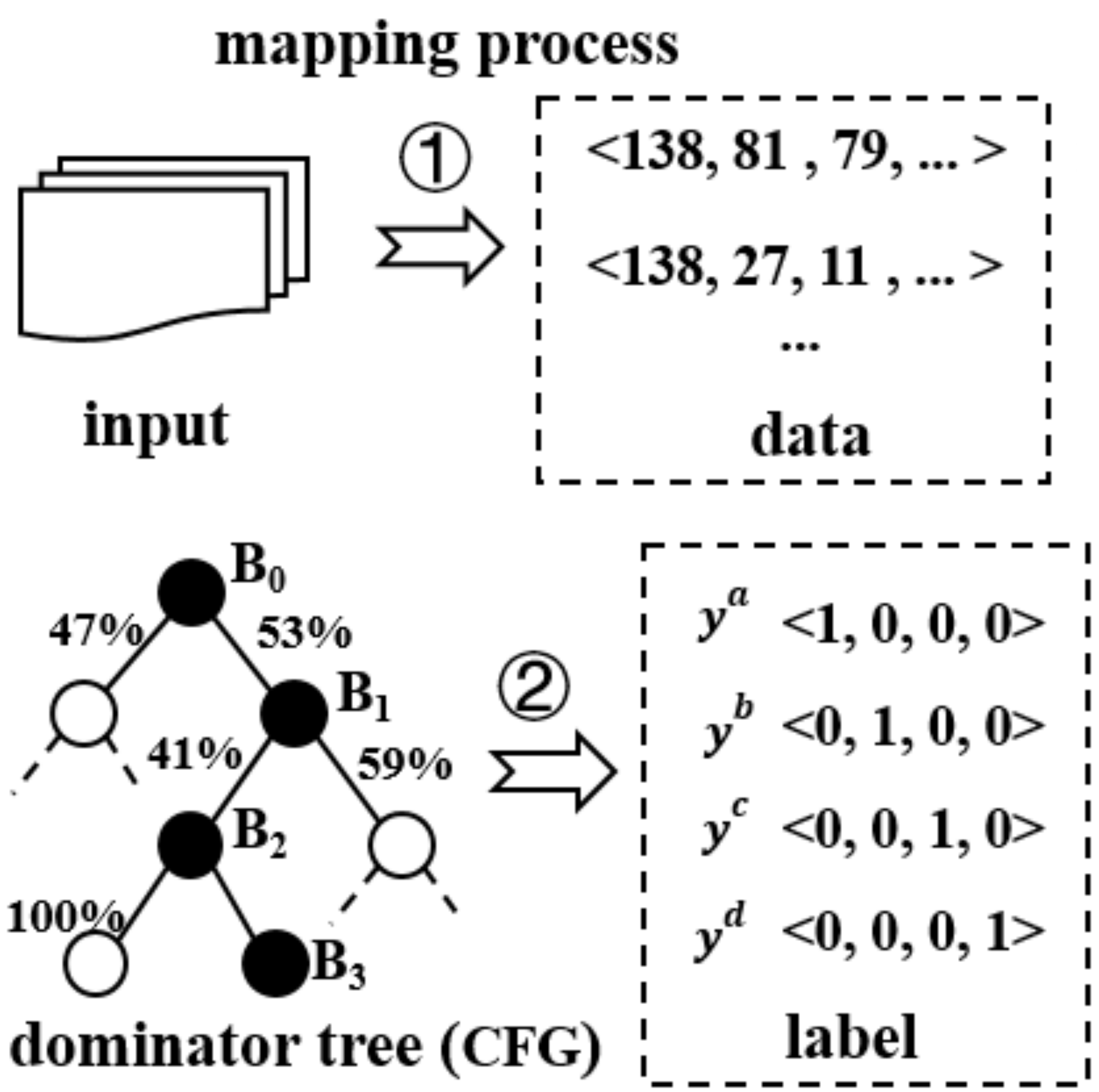
Challenges

- C1: Lack of balanced labeled data.
    - In the early stage of fuzzing, there is even no reachable input.
    - Without balanced labeled data, the trained model will be overfitting.
- C2: Lack of representative data.
    - Newly inputs look quite different from the reachable ones in the training set.
    - The trained model will fail to predict the reachability of the new inputs.
- C3: Efficiency.
    - The time cost of training and prediction should be strictly limited.

Overview of FuzzGuard

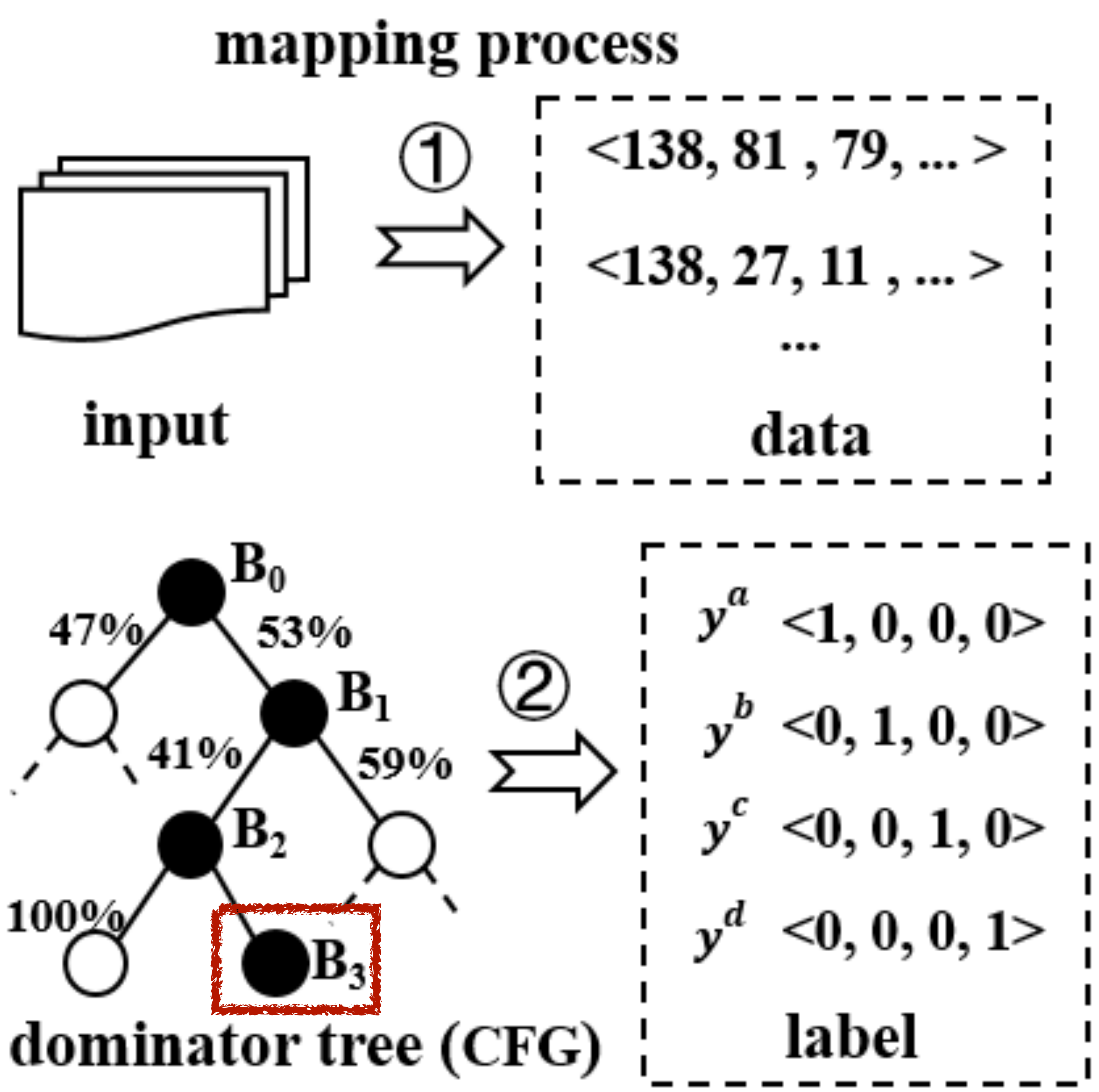


Phase 1: Model Initialization



Model Initialization

Phase 1: Model Initialization

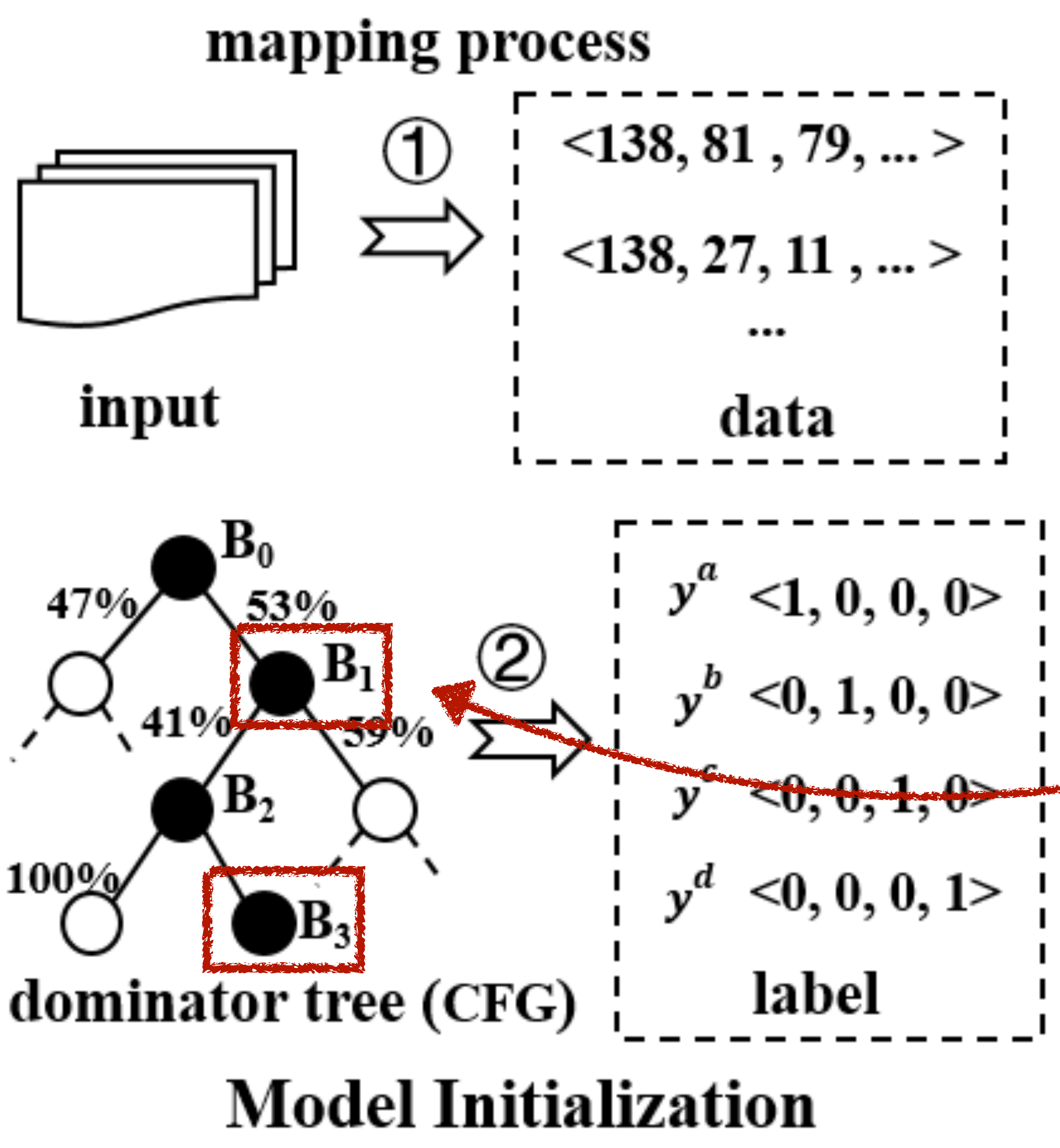


Model Initialization

C1: Lack of balanced labeled data.

- Step-forwarding approach
  - Collect and map the inputs and their execution path.
  - Choosing the dominators of the buggy code as the middle-stage targets.
  - Letting the execution reach the pre-dominating nodes first.

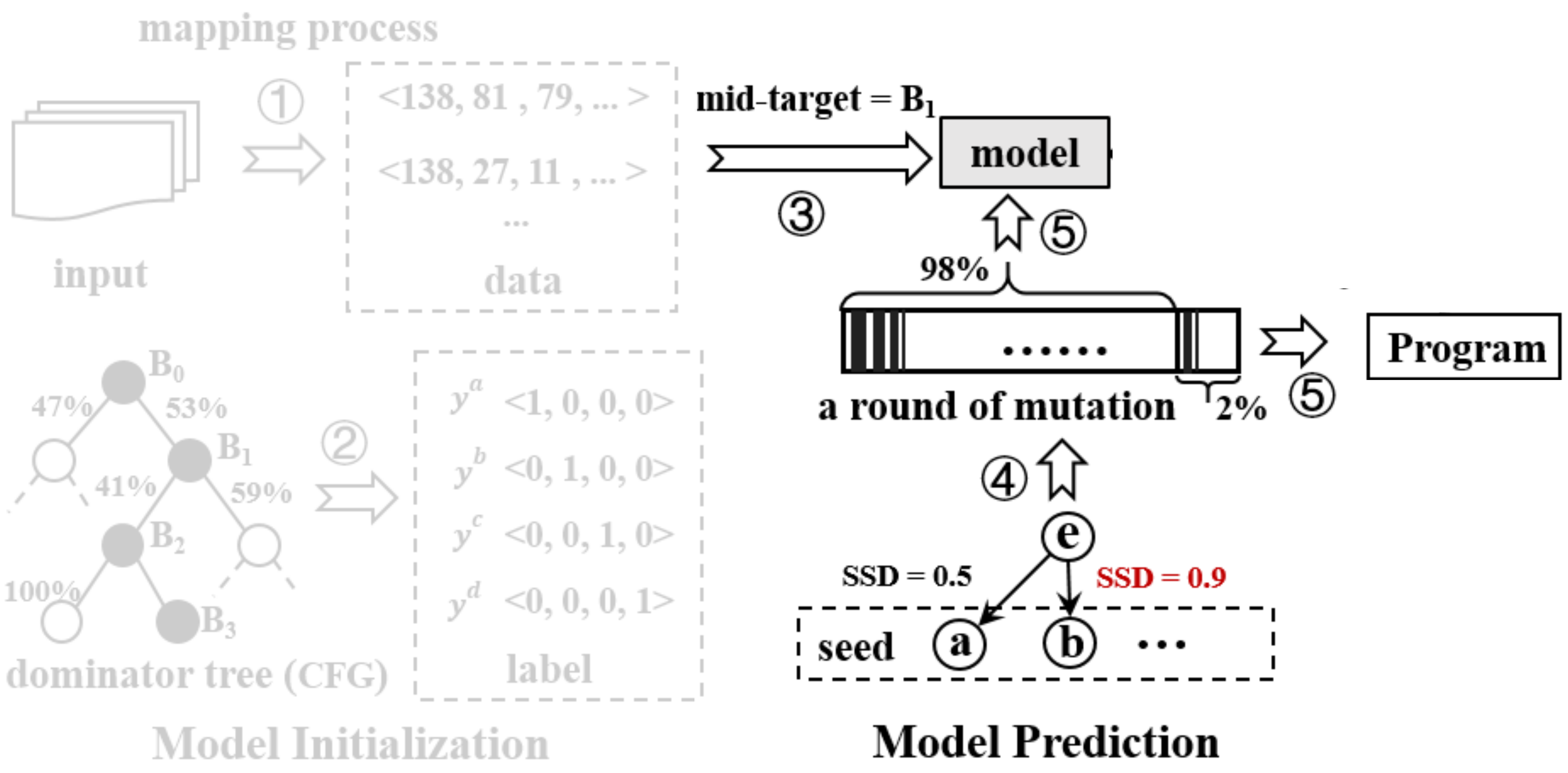
Phase 1: Model Initialization



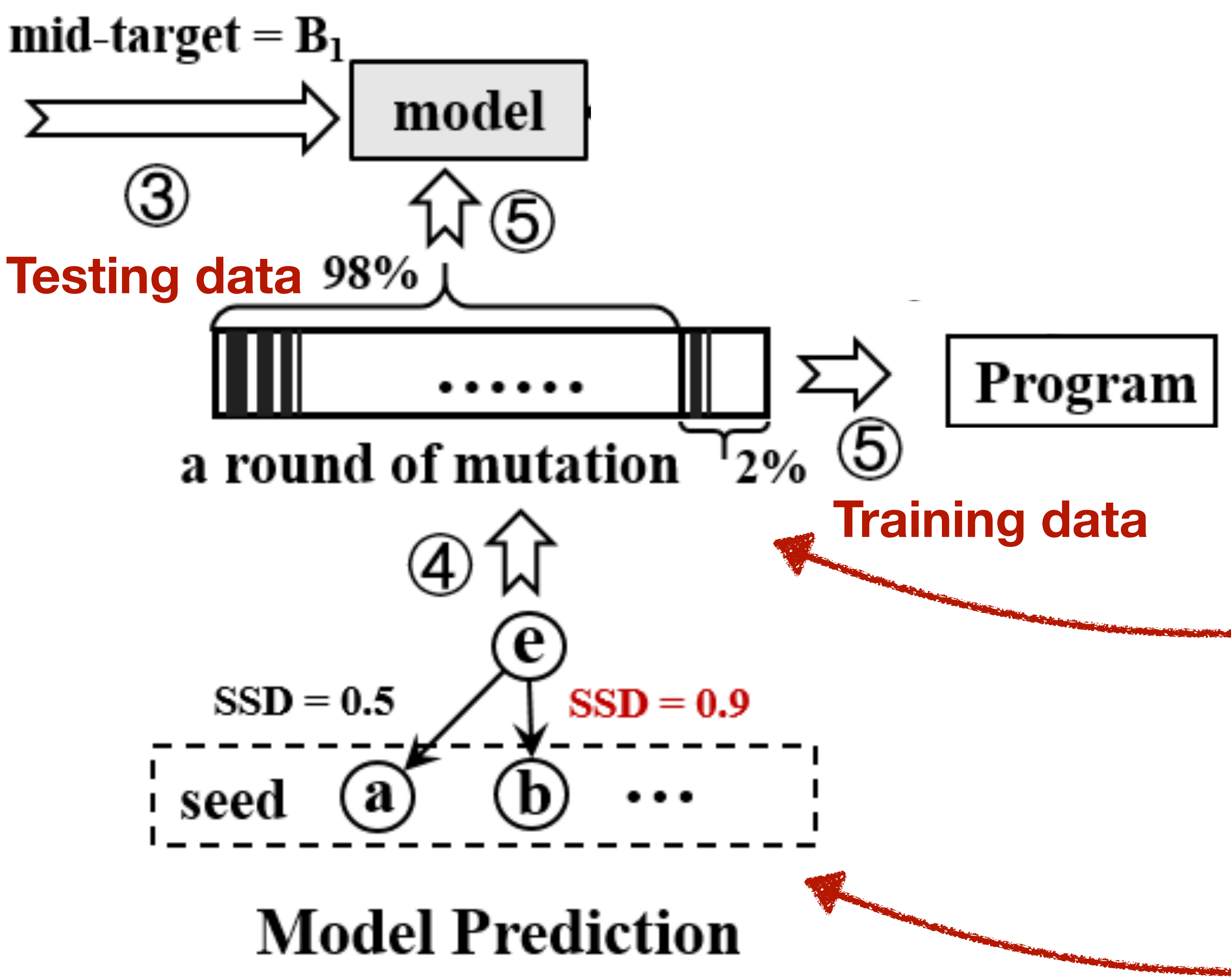
C1: Lack of balanced labeled data.

- Step-forwarding approach
  - Collect and map the inputs and their execution path.
  - Choosing the dominators of the buggy code as the middle-stage targets.
  - Letting the execution reach the pre-dominating nodes first.

Phase 2: Model Prediction



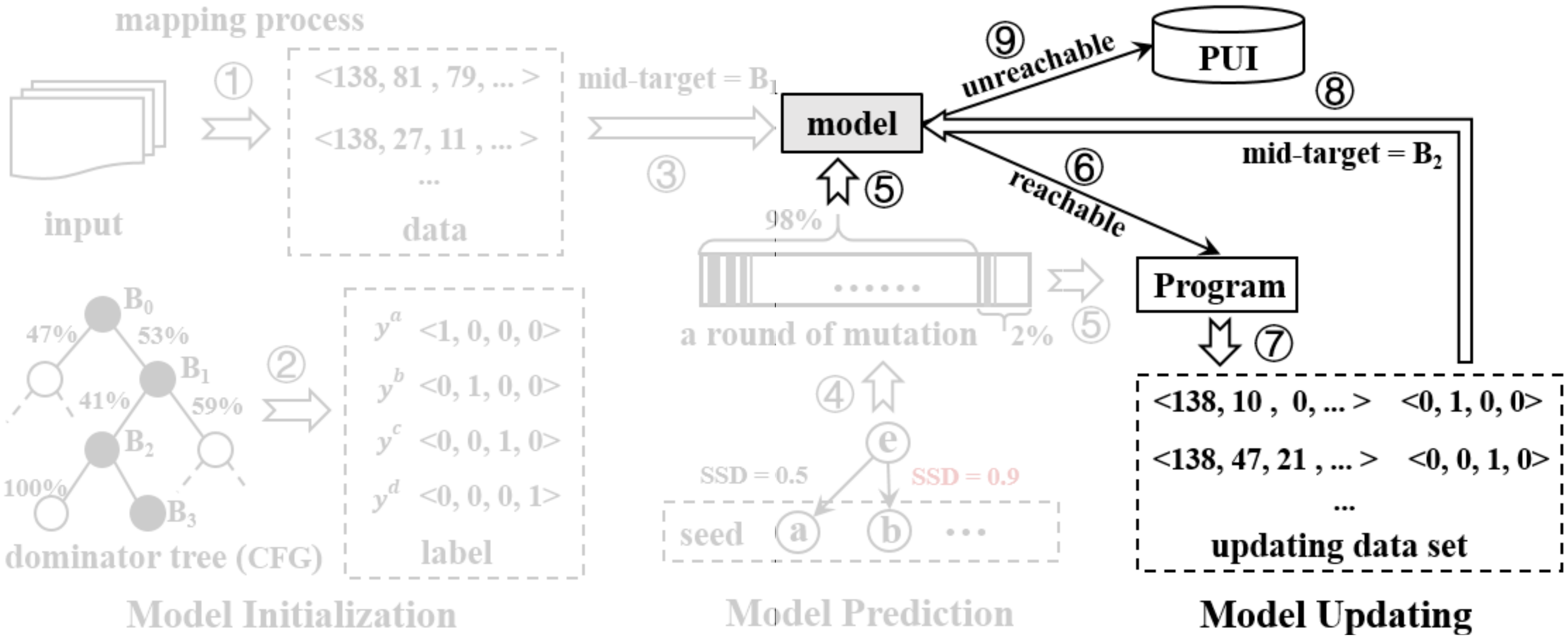
Phase 2: Model Prediction



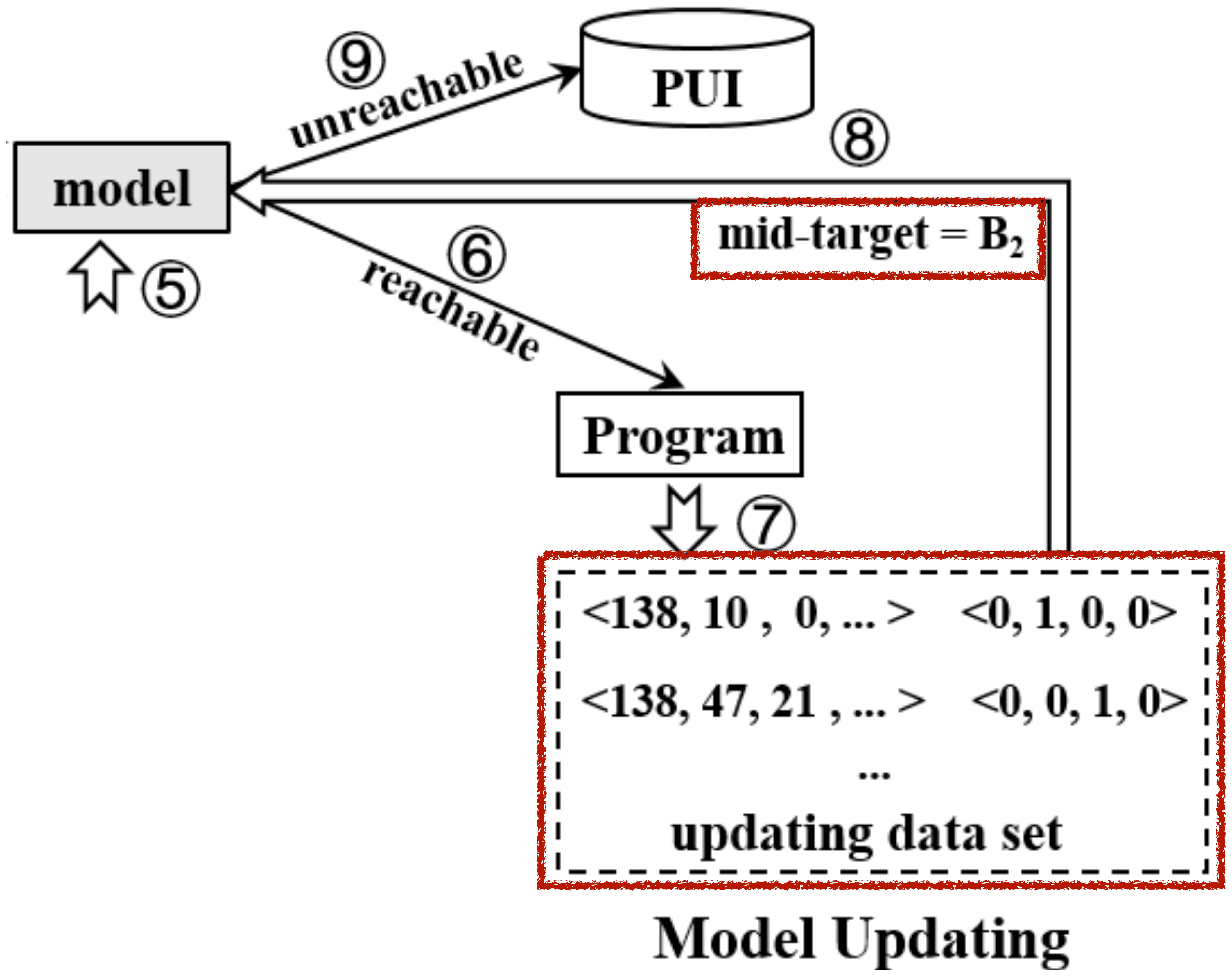
C2: Lack of representative data.

- Representative data selection
  - Sample training data from each round of mutation.
  - Calculate seed similarity degree (SSD) and sample fewer inputs for similar ones.

Phase 3: Model Updating



Phase 3: Model Updating



C3: Efficiency

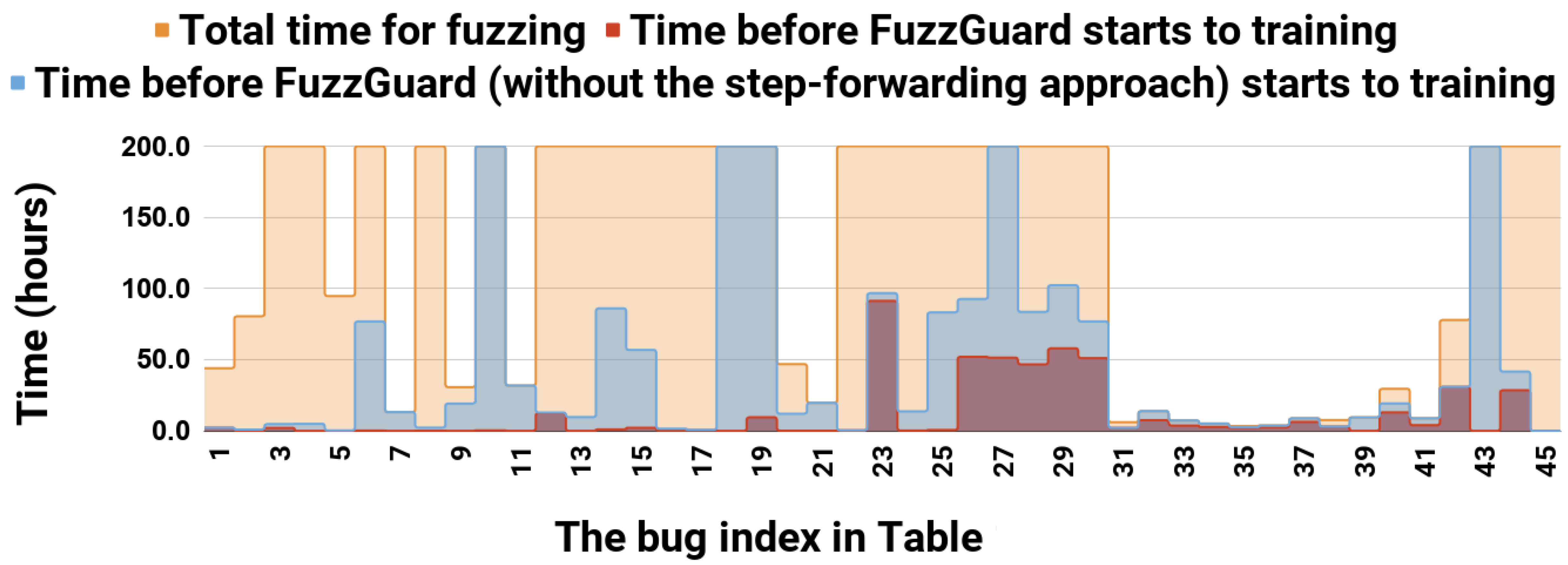
- Incremental Learning
  - Keep collect training data for updating model.
  - Incremental train the model when a new mid-target node gains balanced data.

Effectiveness Summary

- Dataset
  - 45 bugs in 10 real-world programs with different file formats.
- Results
  - 1.3x -17.1x speedup (5.4x averagely)
  - The earlier the model is trained, the more time could be saved.
  - The more reachable inputs generated by the carrier fuzzer, the less effective FuzzGuard is.

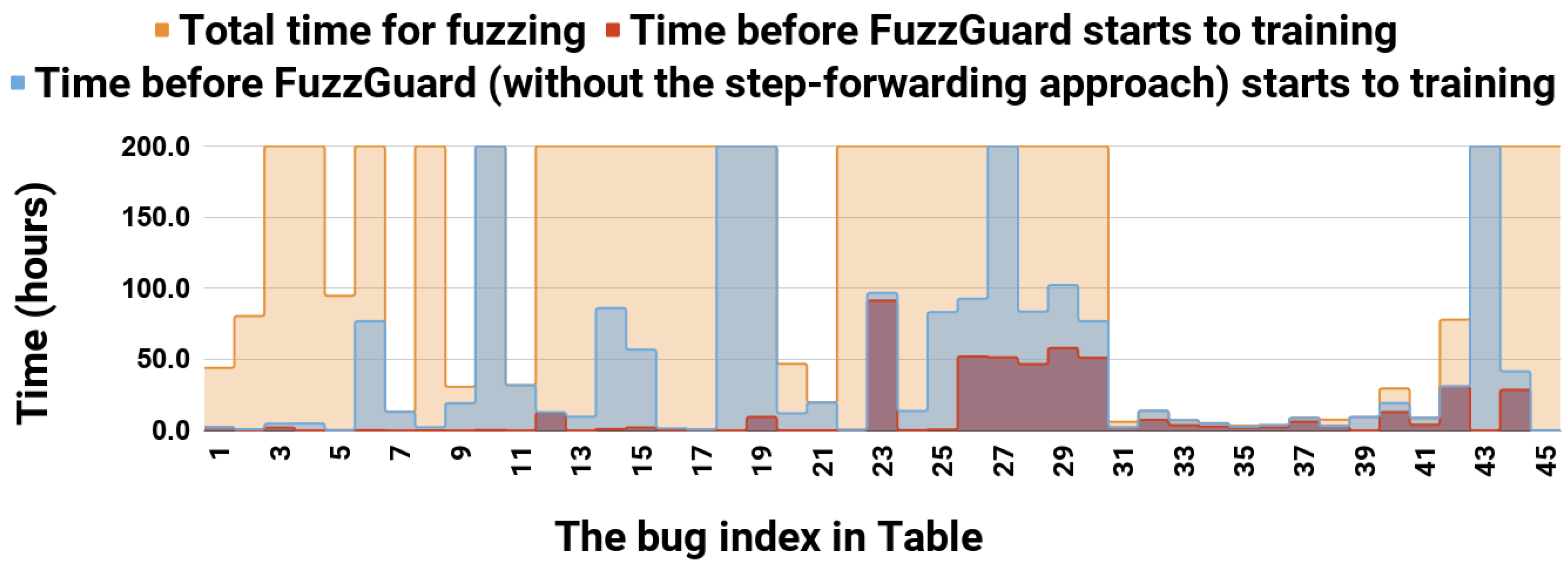
No.	Program	Vuln. Code	$N_{Functions}$	$N_{Constraints}$	N_{Inputs}	$UR.$	$Filtered$	T_{AFLGo}	T_{+FG}	Speedup FG	Speedup FG_1	Speedup FG_2
18	ImageMagick v7.0.7-1	tiff.c:1934	149.1 K	1.2 M	9.4 M	98.5%	92.5%	200 h	15.2 h	**13.1**	1.6	8.7
19	ImageMagick v7.0.5-5	bmp.c:[illegible]	102.4 K	926.5 K	12.9 M	64.3%	59.9%	200 h	80.5 h	**2.5**	1.7	2.5
20	Jasper v2.0.14	jp2[illegible]	13.9 K	17.7 M	28 M	99.4%	50.9%	200 h	99 h	**2.0**	1.7	1.9
21	Jasper v2.0.10	[illegible]	740	9.7 K	11.3 M	99.7%	94.3%	46.9 h	3.7 h	**12.7**	1.4	11.1
22	Jasper v2.0.10	[illegible]	1.7 K	36.8 K	6.1 M	99.9%	94.0%	19.7 h	1.6 h	**12.0**	1.0	10.0
23	Jasper v2.0.10	[illegible]	1.1 K	11.8 K	22.3 M	62.4%	56.0%	200 h	89 h	**2.2**	2.0	2.2
24	Libming v0.4.8	[illegible]	104	5.3 K	38.6 M	99.9%	70.2%	200 h	63 h	**3.2**	1.0	3.1
25	Libming v0.4.7	pa[illegible]	75	4.7 K	32.3 M	99.8%	94.7%	200 h	11.7 h	**17.1**	8.5	14.1
26	Libming v0.4.7	parser.c[illegible]	170	2.7 K	16.1 M	91.9%	86.6%	200 h	27.2 h	**7.3**	1.7	6.2
27	Libming v0.4.7	parser.c:3381	79	790	38.4 M	99.7%	69.9%	200 h	61.3 h	**3.3**	2.0	3.2
28	Libming v0.4.7	parser.c:3095	25	217	46.8 M	92.9%	65.7%	200 h	70 h	**2.9**	1.9	2.8
29	Libming v0.4.7	parser.c:2993	22	386	45.9 M	97.2%	64.8%	200 h	71.8 h	**2.8**	1.7	2.7
30	Libming v0.4.7	parser[illegible]	24	294	77 M	92.9%	63.6%	200 h	75.3 h	**2.7**	2.0	2.5
31	Libming v0.4.7	pa[illegible]	55	423	12.6 M	99.8%	61.3%	6.1 h	2.8 h	**2.2**	2.0	1.8
32	Libming v0.4.7	[illegible]	38	308	13 M	99.9%	43.2%	14 h	8.2 h	**1.7**	1.0	1.6
33	Libming v0.4.7	[illegible]	32	340	16.6 M	99.9%	46.0%	7.3 h	4.4 h	**1.7**	1.0	1.4
34	Libming v0.4.7	[illegible]	36	396	19.6 M	99.9%	43.3%	5.2 h	3.4 h	**1.5**	1.0	1.1
35	Libming v0.4.7	[illegible]	37	637	18.9 M	99.8%	37.2%	3.4 h	2.5 h	**1.4**	1.0	1.1
36	Libming v0.4.7	p[illegible]	34	1.1 K	17.6 M	99.9%	33.6%	3.8 h	2.9 h	**1.3**	1.0	1.1
37	Libming v0.4.7	pars[illegible]	34	402	30.7 M	99.9%	27.7%	8.9 h	6.9 h	**1.3**	1.0	1.2
38	Libming v0.4.7	outputtxt.c:143	64	2.2 K	27.3 M	65.5%	24.6%	7.7 h	6.1 h	**1.3**	1.1	1.1
39	Libtiff v4.0.9	tif_dirwrite.c:1901	728	14.4 K	8.6 M	99.9%	91.4%	9.6 h	1.3 h	**7.4**	1.0	4.8
40	Libtiff v4.0.7	tif_sw[illegible]	631	13.1 K	44.7 M	99.7%	52.8%	29.6 h	15 h	**2.0**	1.1	1.3
41	Libtiff v4.0.7	tiffc[illegible]	728	13.3 K	15.6 M	99.9%	51.7%	8.9 h	4.6 h	**1.9**	1.0	1.7
42	Libtiff v4.0.7	tif_[illegible]	416	11.6 K	60.6 M	79.5%	36.3%	77.9 h	49.8 h	**1.6**	1.4	1.5
43	Libxml2 v2.9.4	SA[illegible]	418	15.7 K	92.6 M	99.9%	94.4%	200 h	17.6 h	**11.3**	1.0	5.2
44	Podofo v0.9.5	Pdf[illegible]	19.8 K	44.1 K	2.6 M	99.3%	79.7%	200 h	40.7 h	**4.9**	4.8	1.8
45	Tcpreplay v4.3.0-beta1	get.c:174	23	1.1 K	203.3 M	53.2%	49.5%	200 h	105.4 h	**1.9**	1.7	1.9
Avg.			15.5K	315.9 M		91.7%	65.1%			**5.4**	2.6	4.4

Contribution of Individual Techniques

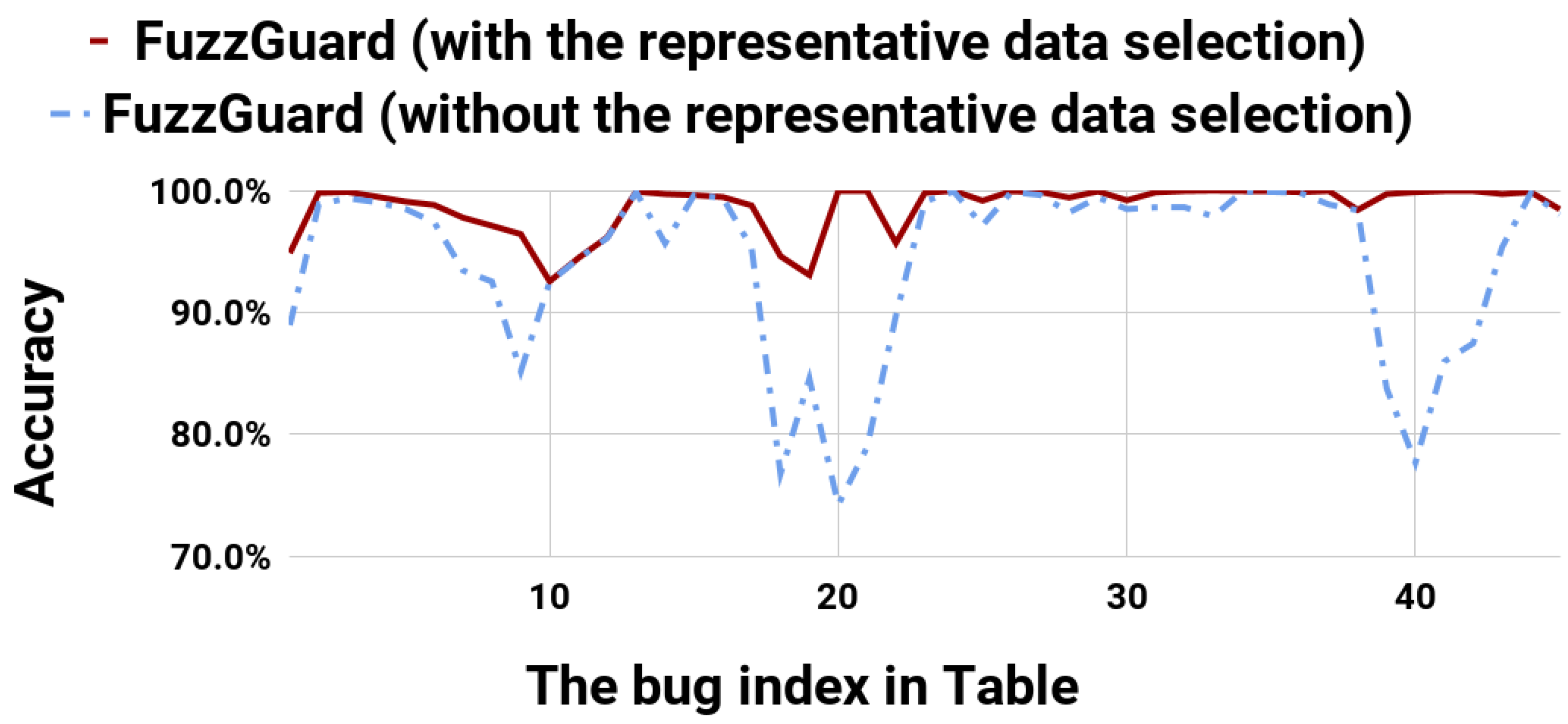


- Without the step-forwarding approach
  - Gain only 2.6x speedup averagely.
  - 14/45 bugs cannot be trained.

Contribution of Individual Techniques



- Without the step-forwarding approach
  - Gain only 2.6x speedup averagely.
  - 14/45 bugs cannot be trained.



- Without the representative data selection
  - Gain only 4.4x speedup averagely.
  - The accuracy dramatically decreases in some cases.

Understanding & Future Work

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
													number_colors		bits_per_pixel	
00000000h:	**28**	00	00	00	**04**	00	00	00	**01**	00	00	00	**01**	**00**	**20**	**00**
00000010h:	00	**26**	00	00	**17**	00	00	00	00	00	00	00	00	00	00	00
00000020h:	**01**	00	00	00	00	00	00	00	00	00	26	26	26	26	26	26
00000030h:	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26
00000040h:	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26
00000050h:	26	26	26	28	36											

```
  ThrowReaderException(...);
if (dib_info.colors_important > 256)
  ThrowReaderException(...);
if ((dib_info.image_size != 0U) && (dib_info.image_size
    > file_size))
  ThrowReaderException(...);
if ((dib_info.number_colors != 0) ||
    (dib_info.bits_per_pixel < 16)) {
  image->storage_class=PseudoClass;
```

Vulnerable code

- Understanding
  - The key features learned by the model are related to the key bytes in the PoC.
- Future Work
  - The benefit to input mutation.

Conclusion

- FuzzGuard： A deep-learning-based approach to predict reachability of program inputs without execution.
    - Step-forwarding approach for handling unbalanced data training.
    - Representative data selection for training data collection.
    - Incremental learning for the dynamic model.
- Increase the runtime performance of the vanilla AFLGo from 1.3x to 17.1x.

Thanks for Listening!

Q&A

Code Release: https://github.com/zongpy/FuzzGuard.