EMS: History-Driven Mutation for Coverage-based Fuzzing

Chenyang Lyu Shouling Ji Xuhong Zhang Hong Liang Binbin Zhao
Kangjie Lu Raheem Beyah

NDSS 2022

Fuzzing: Automated Dynamic Vulnerability Discovery Technique

- Fuzzing is an automatic, dynamic vulnerability discovery technique.
- A fuzzer randomly employs mutation operators to generate test cases and feeds test cases to a target program in order to trigger vulnerabilities.
- Fuzzers are widely used in software testing.

OSS-Fuzz

ClusterFuzz

libFuzzer

LLVM COMPILER INFRASTRUCTURE

Existing Fuzzing Technique

Improving energy allocation strategies

- EcoFuzz
- AFL++-hier
- TortoiseFuzz

Combining fuzzing with other techniques

- QSYM
- Angora
- NEUZZ

Improving fuzzer's other implementation

- CollAFL
- MOPT
- UnTracer

……

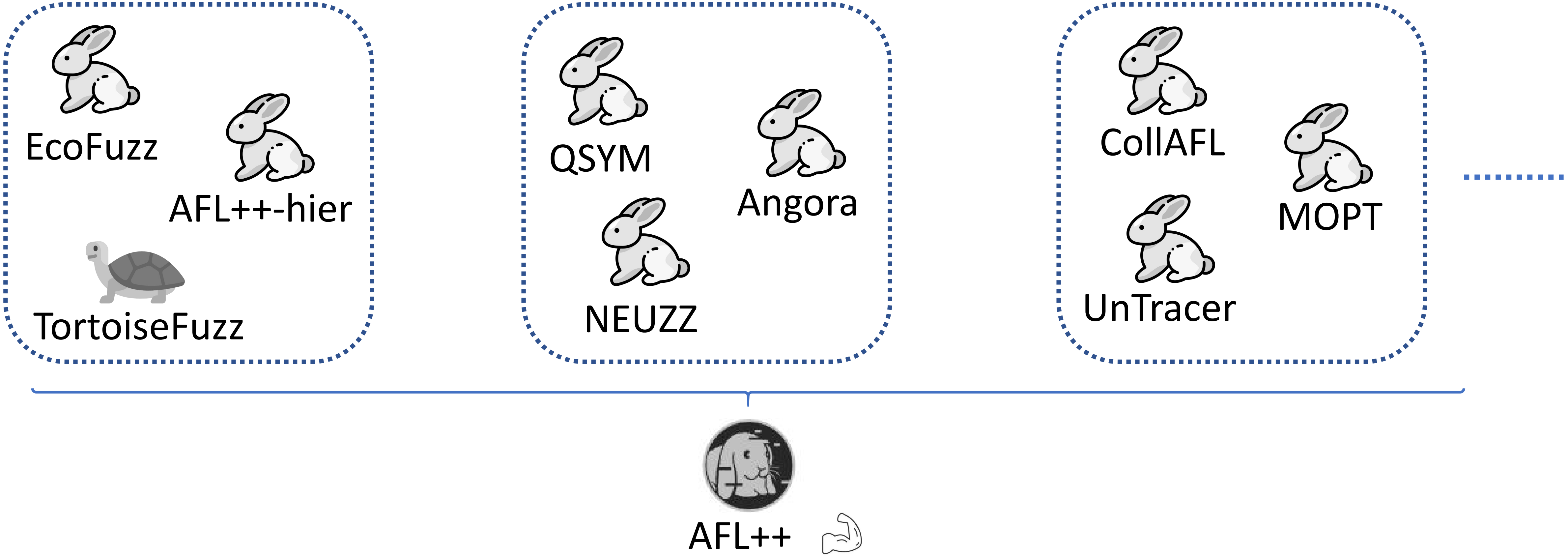
AFL++

Limitations in Existing Fuzzing Technique

Improving energy allocation strategies

Combining fuzzing with other techniques

Improving fuzzer's other implementation



Existing fuzzers cannot reuse the efficient mutation strategies, which have generated interesting test cases, learned from intra-trial and inter-trial fuzzing history.

Why Intra- and Inter-Trial History Matters

- **The efficient mutation strategies in intra-trial fuzzing history can help solve the same path constraints in different execution paths, e.g., different execution paths of a program can contain the same function call and have the same constraints.**

- **The efficient mutation strategies from inter-trial fuzzing history can help solve the path constraints because of the shared development framework and underlying libraries.**

Why Intra- and Inter-Trial History Matters

- **The efficient mutation strategies in intra-trial fuzzing history can help solve the same path constraints in different execution paths, e.g., different execution paths of a program can contain the same function call and have the same constraints.**

- **The efficient mutation strategies from inter-trial fuzzing history can help solve the path constraints because of the shared development framework and underlying libraries.**

We provide the following case studies to demonstrate the above conclusions.

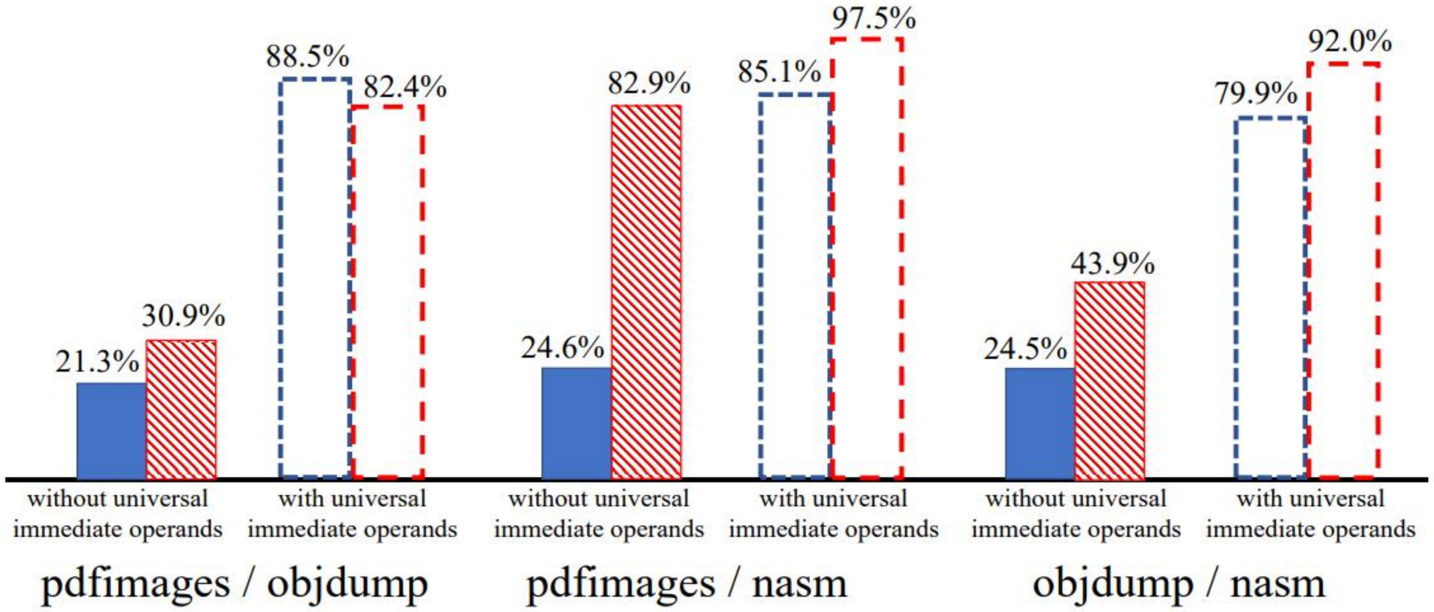
Immediate Operand Analysis

¤ **We analyze the types and usages of *immediate operands* used in the cmp assembly instructions, since they directly control branching behaviors of a program and are closely related to path constraints.**

		Singular[a]	Repetitive[b]	Total
pdfimages	Number of immediate operands	15	21	36
	Number of usages of immediate operands	15	46	61
objdump	Number of immediate operands	25	34	59
	Number of usages of immediate operands	25	195	220
nasm	Number of immediate operands	6	5	11
	Number of usages of immediate operands	6	35	41

[a]If an immediate operand is used only once, it is singular.
[b]If an immediate operand is used more than once, it is repetitive.

We do not include universal immediate operands, which are defined as interesting values in AFL.



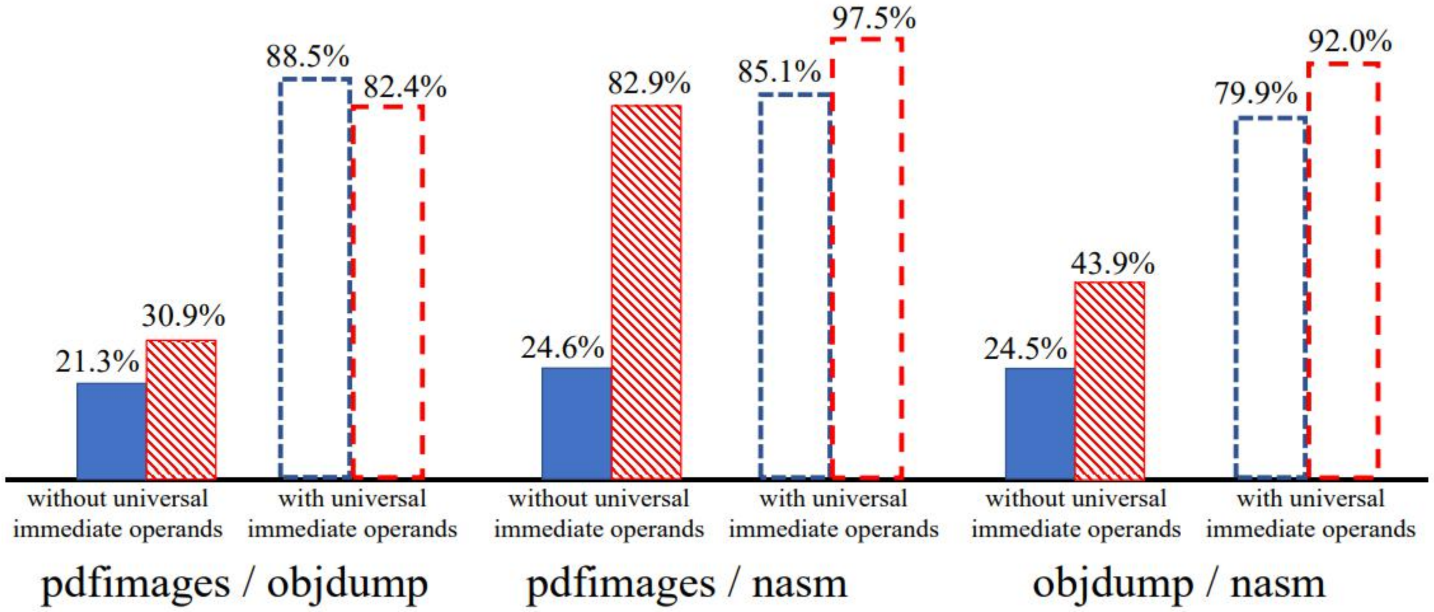
Immediate Operand Analysis

- We analyze the types and usages of *immediate operands* used in the cmp assembly instructions, since they directly control branching behaviors of a program and are closely related to path constraints.

		Singular[a]	Repetitive[b]	Total
pdfimages	Number of immediate operands	15	21	36
	Number of usages of immediate operands	15	46	61
objdump	Number of immediate operands	25	34	59
	Number of usages of immediate operands	25	195	220
nasm	Number of immediate operands	6	5	11
	Number of usages of immediate operands	6	35	41

[a]If an immediate operand is used only once, it is singular.
[b]If an immediate operand is used more than once, it is repetitive.

We do not include universal immediate operands, which are defined as interesting values in AFL.



The same immediate operand influences the control flow and data flow multiple times in a program.

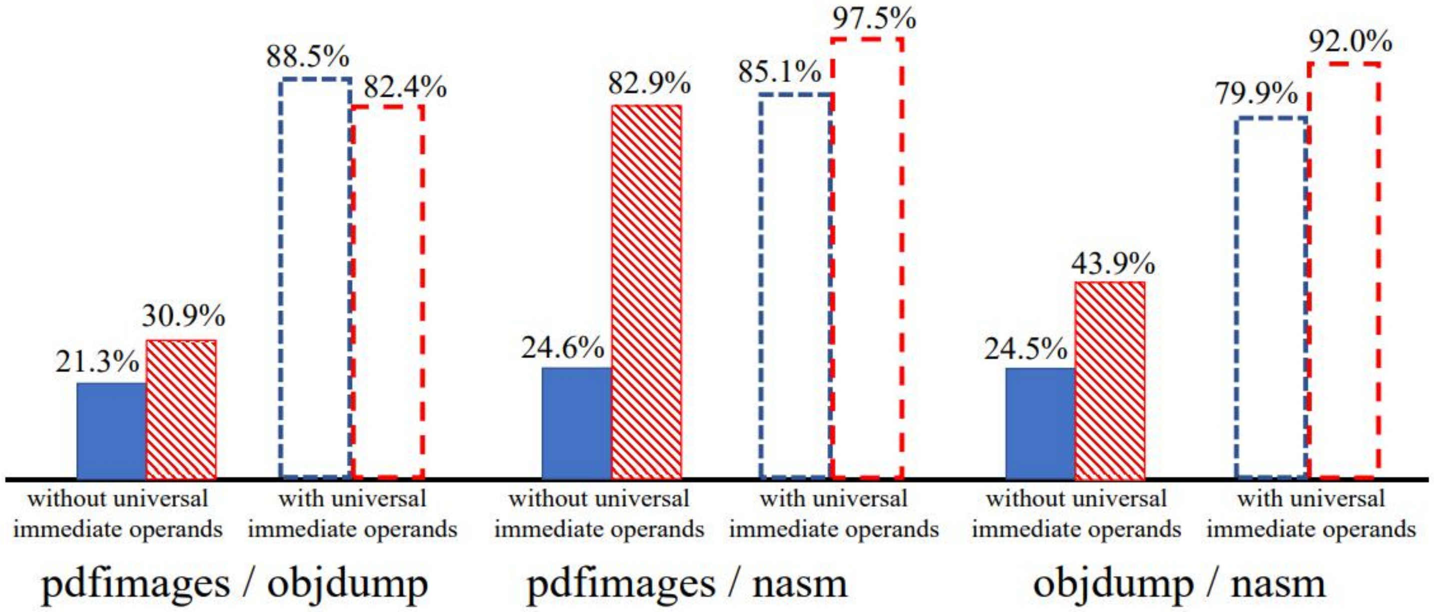
Immediate Operand Analysis

- We analyze the types and usages of *immediate operands* used in the cmp assembly instructions, since they directly control branching behaviors of a program and are closely related to path constraints.

		Singular[a]	Repetitive[b]	Total
pdfimages	Number of immediate operands	15	21	36
	Number of usages of immediate operands	15	46	61
objdump	Number of immediate operands	25	34	59
	Number of usages of immediate operands	25	195	220
nasm	Number of immediate operands	6	5	11
	Number of usages of immediate operands	6	35	41

[a]If an immediate operand is used only once, it is singular.
[b]If an immediate operand is used more than once, it is repetitive.

We do not include universal immediate operands, which are defined as interesting values in AFL.



The repetitive immediate operands account for the vast majority in each program.

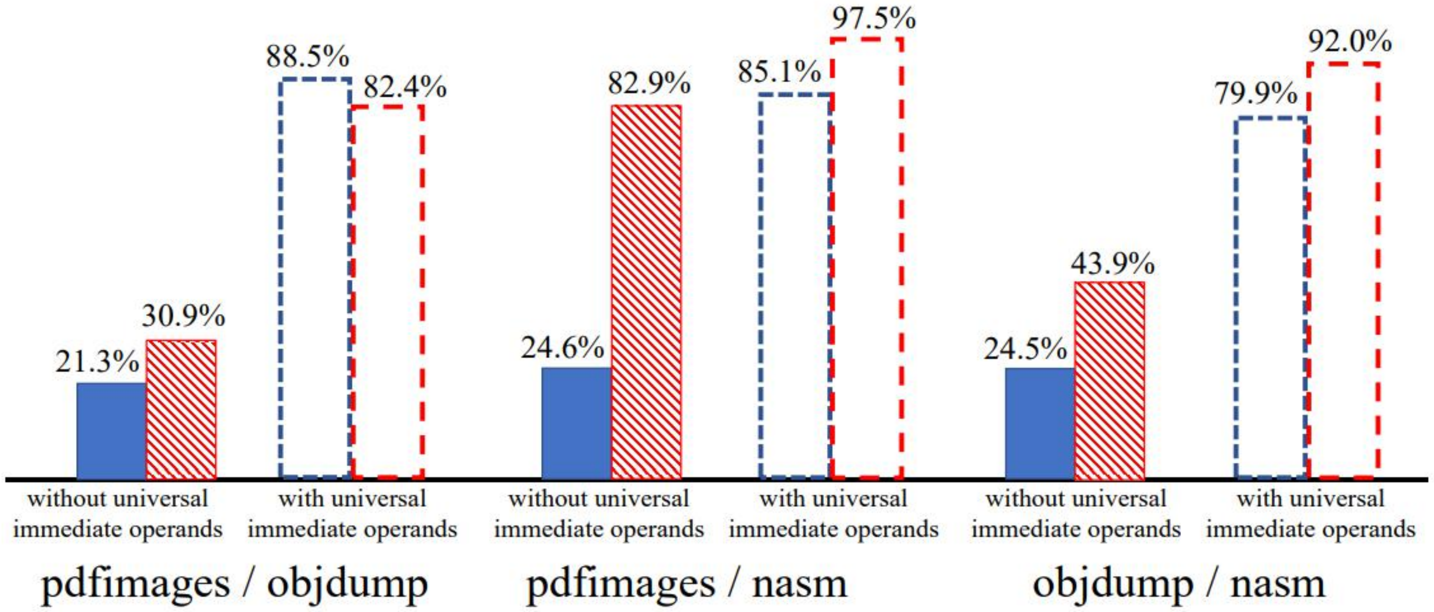
Immediate Operand Analysis

¤ **We analyze the types and usages of *immediate operands* used in the cmp assembly instructions, since they directly control branching behaviors of a program and are closely related to path constraints.**

		Singular[a]	Repetitive[b]	Total
pdfimages	Number of immediate operands	15	21	36
	Number of usages of immediate operands	15	46	61
objdump	Number of immediate operands	25	34	59
	Number of usages of immediate operands	25	195	220
nasm	Number of immediate operands	6	5	11
	Number of usages of immediate operands	6	35	41

[a]If an immediate operand is used only once, it is singular.
[b]If an immediate operand is used more than once, it is repetitive.

We do not include universal immediate operands, which are defined as interesting values in AFL.



The proportion of the usages of the same immediate operands employed in two programs cannot be ignored.

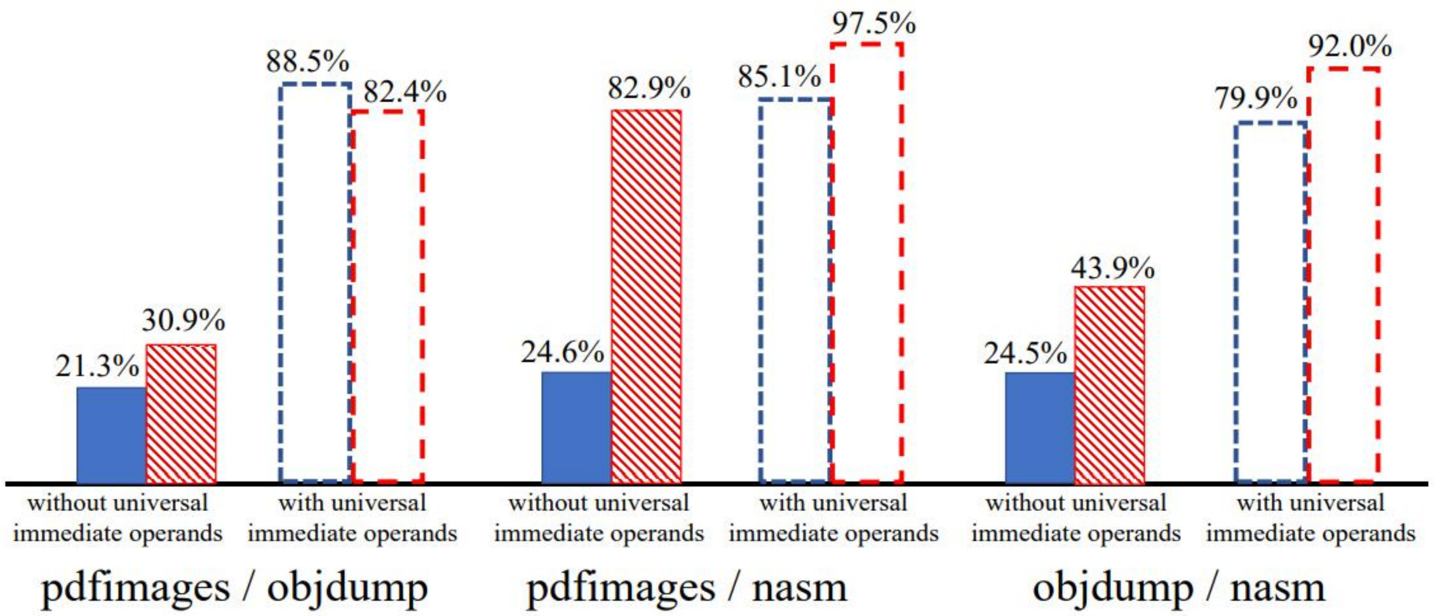
Immediate Operand Analysis

¤ **We analyze the types and usages of *immediate operands* used in the cmp assembly instructions, since they directly control branching behaviors of a program and are closely related to path constraints.**

		Singular[a]	Repetitive[b]	Total
pdfimages	Number of immediate operands	15	21	36
	Number of usages of immediate operands	15	46	61
objdump	Number of immediate operands	25	34	59
	Number of usages of immediate operands	25	195	220
nasm	Number of immediate operands	6	5	11
	Number of usages of immediate operands	6	35	41

[a] If an immediate operand is used only once, it is singular.
[b] If an immediate operand is used more than once, it is repetitive.

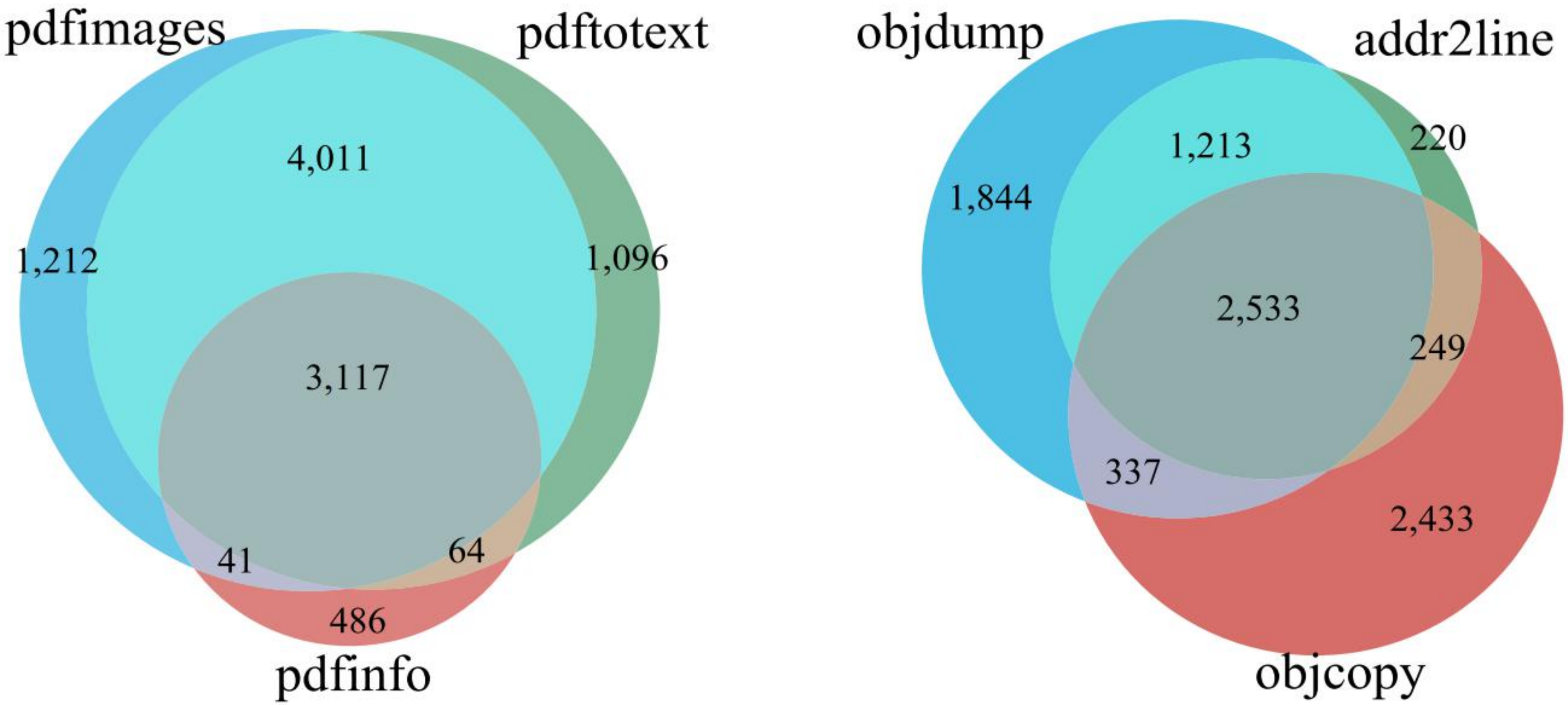
We do not include universal immediate operands, which are defined as interesting values in AFL.



Since parts of path constraints directly read values from inputs as pointed out by the state-of-the-art works, the same immediate operands in different execution paths can be solved by similar mutation strategies.

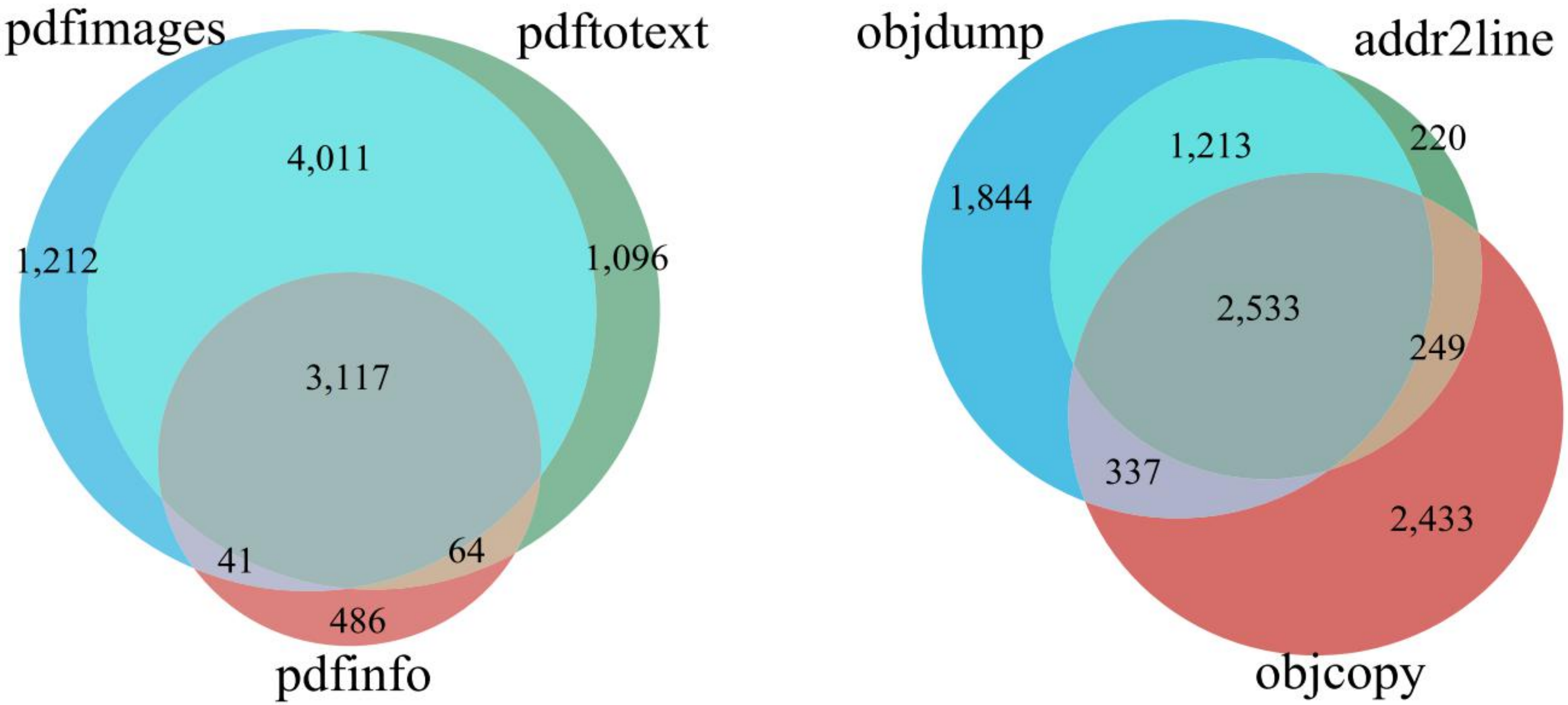
Shared Code Analysis

¤ **We analyze the number of shared basic blocks and unique basic blocks triggered in three programs from the same vendor.**



Shared Code Analysis

¤ **We analyze the number of shared basic blocks and unique basic blocks triggered in three programs from the same vendor.**



The proportion of the shared basic blocks is non-negligible in different programs from the same vendor.

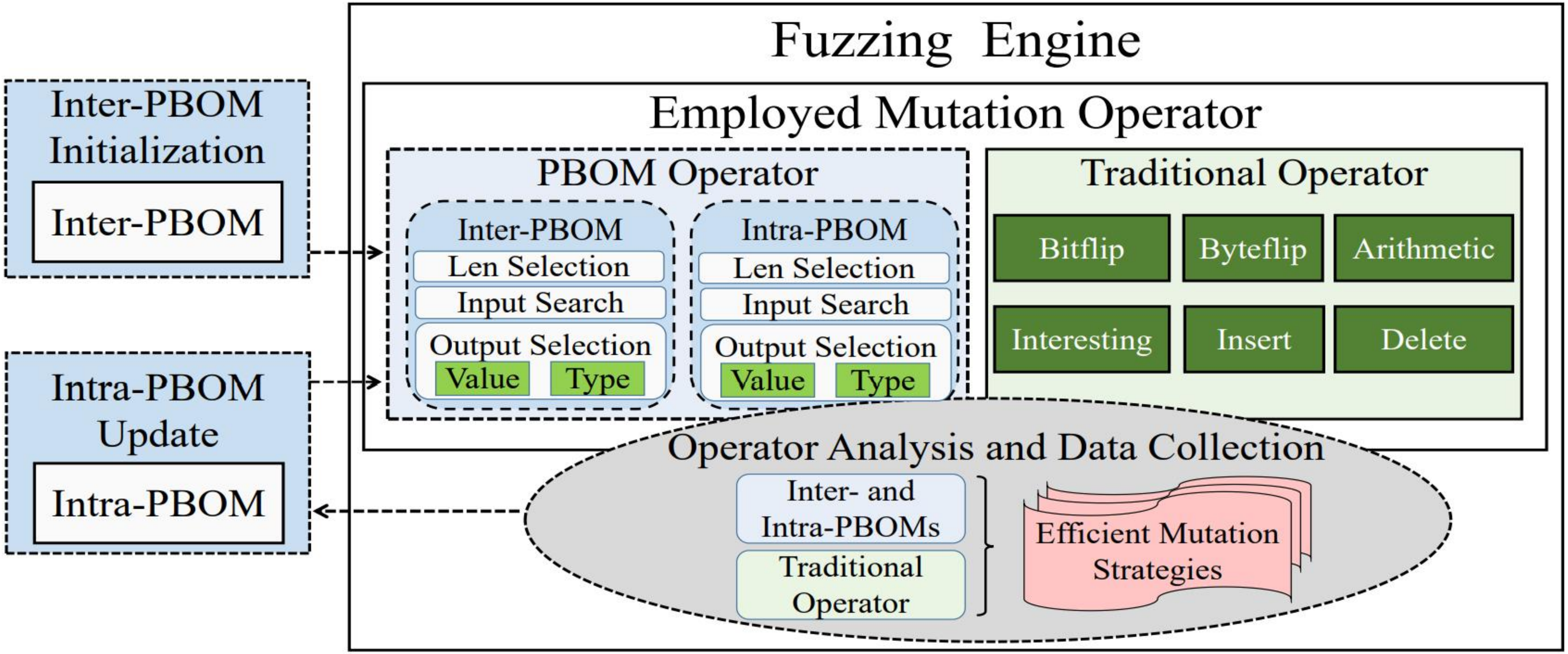
Insight

- Most of the immediate operands employed by *cmp* are repetitive in one program.
- Different programs have the same immediate operands, which are the majority of all the operands.
- Different programs developed by the same vendor invoke the same codes and contain the shared basic blocks in their execution paths, introducing more kinds of the same path constraints.

The efficient mutation strategies learned from intra- and inter-trial fuzzing history can be useful in the fuzzing process.

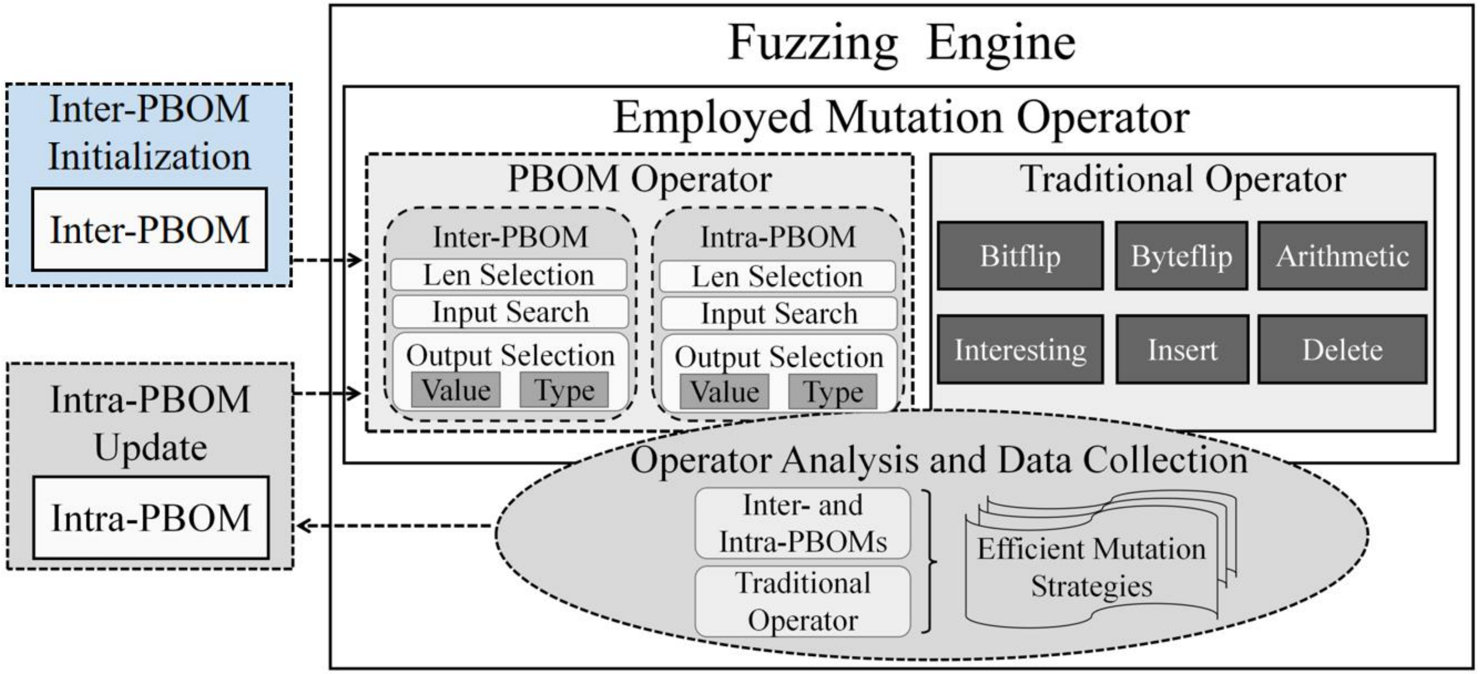
System Design

Overview of EMS



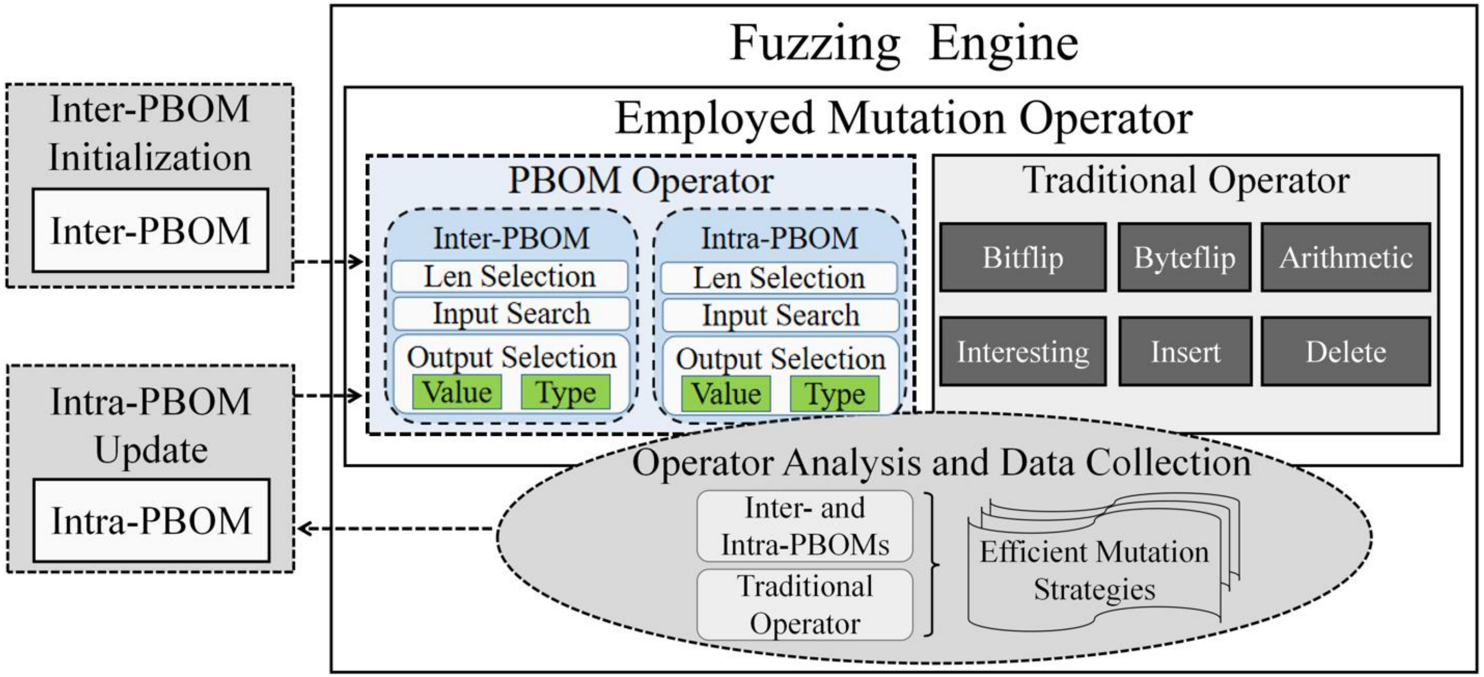
Core idea: Leveraging the proposed Probabilistic Byte Orientation Model (PBOM) to learn the efficient mutation strategies from inter and intra-trial history, respectively. Then, invoking PBOM to reuse efficient mutation strategies.

Framework of EMS



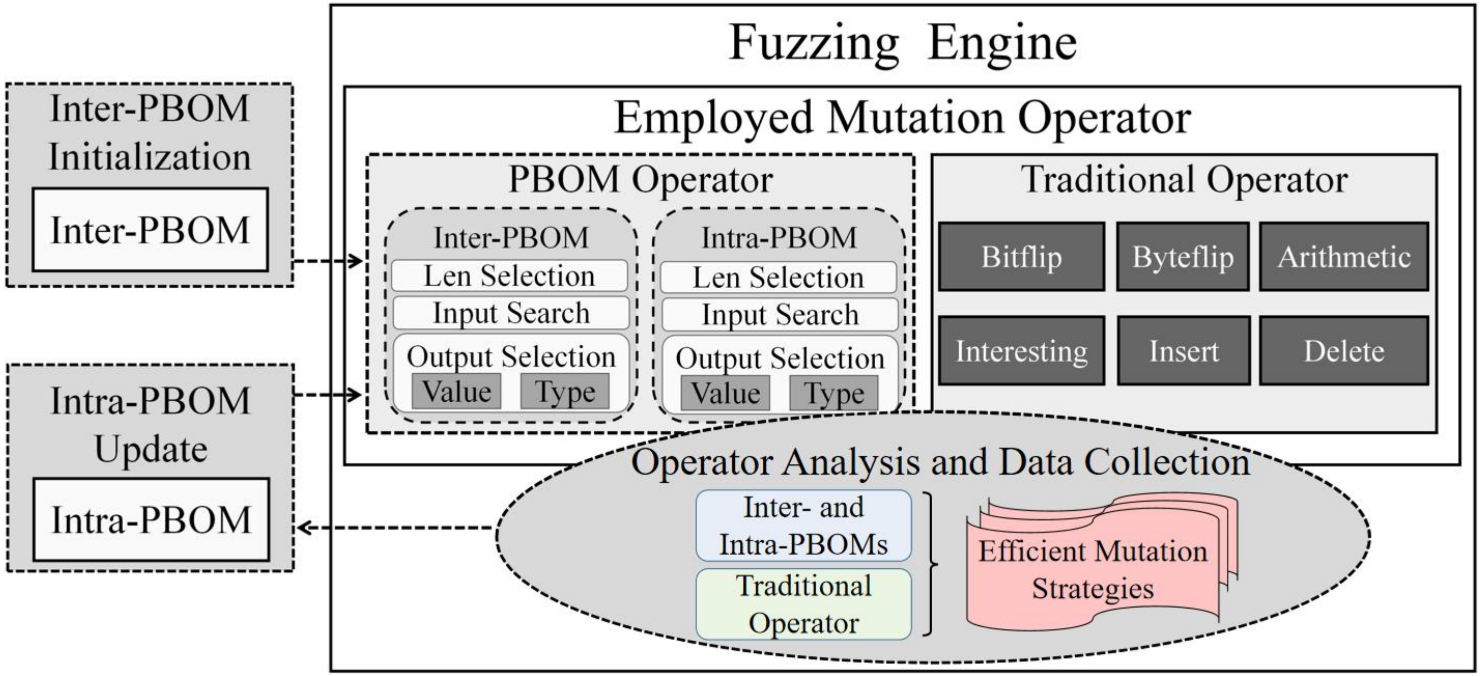
Inter-PBOM Initialization. Construct inter-PBOM at the beginning of the fuzzing process. Utilize the efficient mutation strategies from the inter-trial fuzzing history.

Framework of EMS



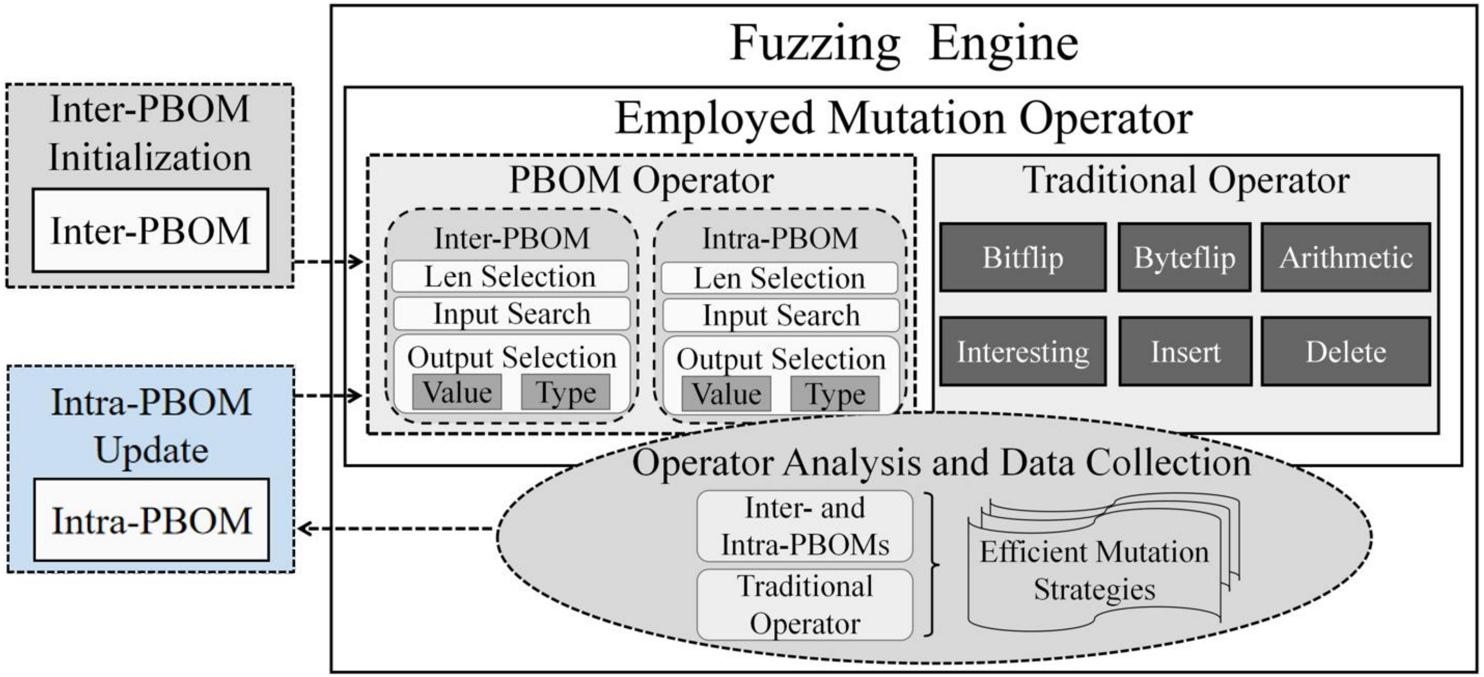
PBOM Operator. Leveraging inter-PBOM and intra-PBOM to reuse the efficient mutation strategies learned from inter- and intra-trial fuzzing history, respectively. EMS utilizes *len* and *input byte values* as the input of PBOM, and mutates seeds according to *output byte values* and *mutation type*.

Framework of EMS



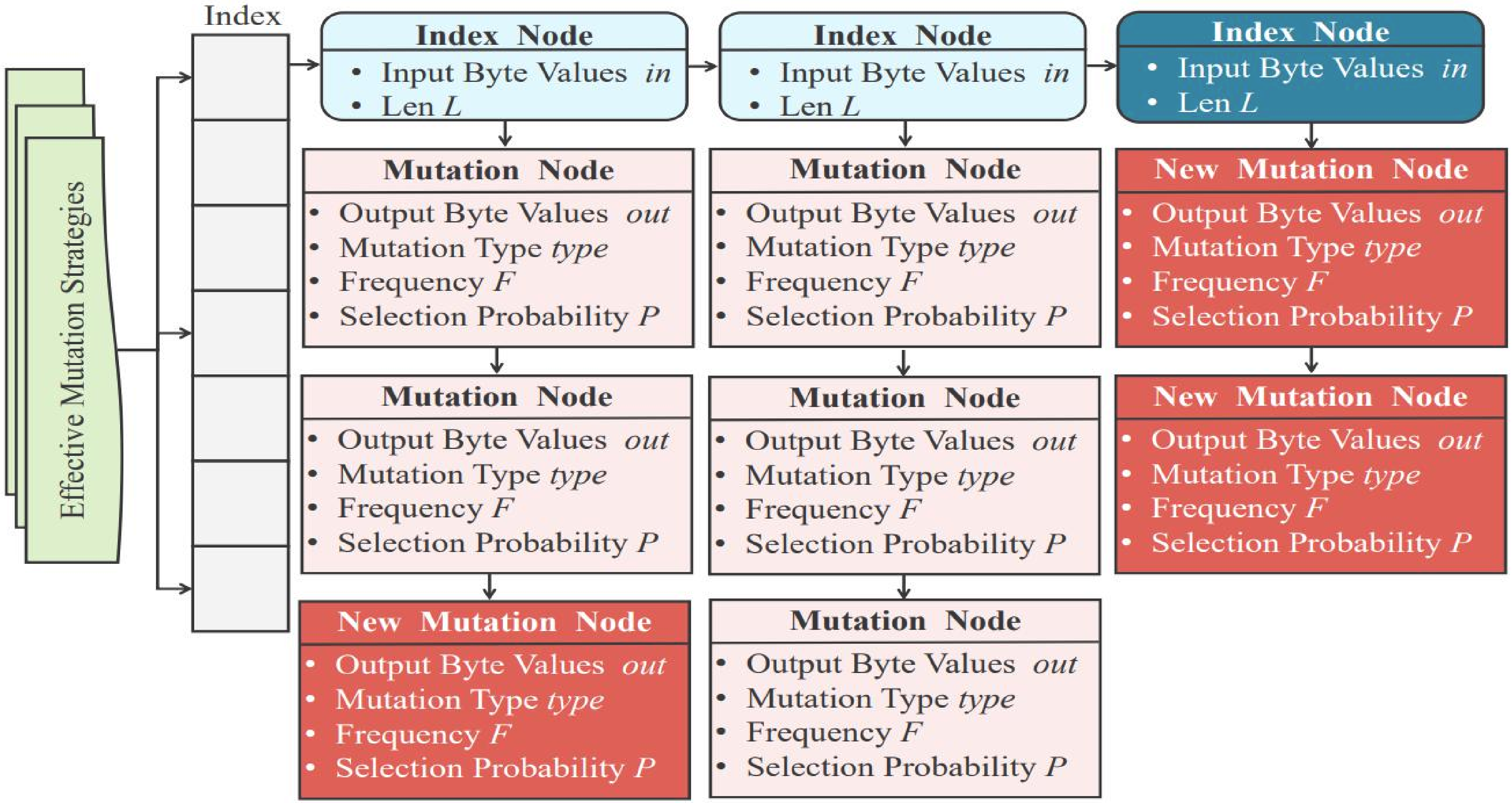
Operator Analysis and Data Collection. Record the efficient mutation strategies that generate interesting test cases and trigger unique paths and crashes on a program.

Framework of EMS



Intra-PBOM Update. Periodically construct/update intra-PBOM with the new efficient mutation strategies collected by ***Operator Analysis and Data Collection***.

Data Structure of PBOM



Construct PBOM based on a hash map to accelerate search efficiency.

Probability Algorithm in PBOM

Probability algorithm used in inter-PBOM:

$$
\begin{aligned}
p_i &= 1 - \frac{F_i}{F_1 + F_2 + \ldots + F_{n-1} + F_n} \\
&= 1 - \frac{count((out_i, type))}{\sum_{(out_k, type) \in \mathbb{MO}} count((out_k, type))} \quad . \\
P_i &= \frac{p_i}{p_1 + p_2 + \ldots + p_{n-1} + p_n} \\
&= \frac{\sum_{(out_k, type) \in \mathbb{MO}} count((out_k, type)) - count((out_i, type))}{(n-1) \times \sum_{(out_k, type) \in \mathbb{MO}} count((out_k, type))} \quad .
\end{aligned}
\tag{1}
$$

Assign more selection probability to low frequency but effective mutation strategies .

Probability Algorithm in PBOM

Probability algorithm used in inter-PBOM:

$$p_i = 1 - \frac{F_i}{F_1 + F_2 + \ldots + F_{n-1} + F_n}$$
$$= 1 - \frac{count((out_i, type))}{\sum_{(out_k, type)\in \mathbb{MO}} count((out_k, type))}.$$
$$P_i = \frac{p_i}{p_1 + p_2 + \ldots + p_{n-1} + p_n}$$
$$= \frac{\sum_{(out_k, type)\in \mathbb{MO}} count((out_k, type)) - count((out_i, type))}{(n-1) \times \sum_{(out_k, type)\in \mathbb{MO}} count((out_k, type))}. \quad (1)$$

Since inter-PBOM stores the mutation strategies from inter trials which can be extensive, the low-frequency strategies can be constructed by rare mutation operators.

Assign more selection probability to low-frequency but effective mutation strategies.

Probability Algorithm in PBOM

Probability algorithm used in intra-PBOM:

$$
\begin{aligned}
P_i &= \frac{F_i}{F_1 + F_2 + \ldots + F_{n-1} + F_n} \\
&= \frac{count((out_i, type))}{\sum_{(out_k, type) \in \mathbb{MO}} count((out_k, type))} \quad .
\end{aligned} \tag{2}
$$

Assign more selection probability to high-frequency mutation strategies.

Probability Algorithm in PBOM

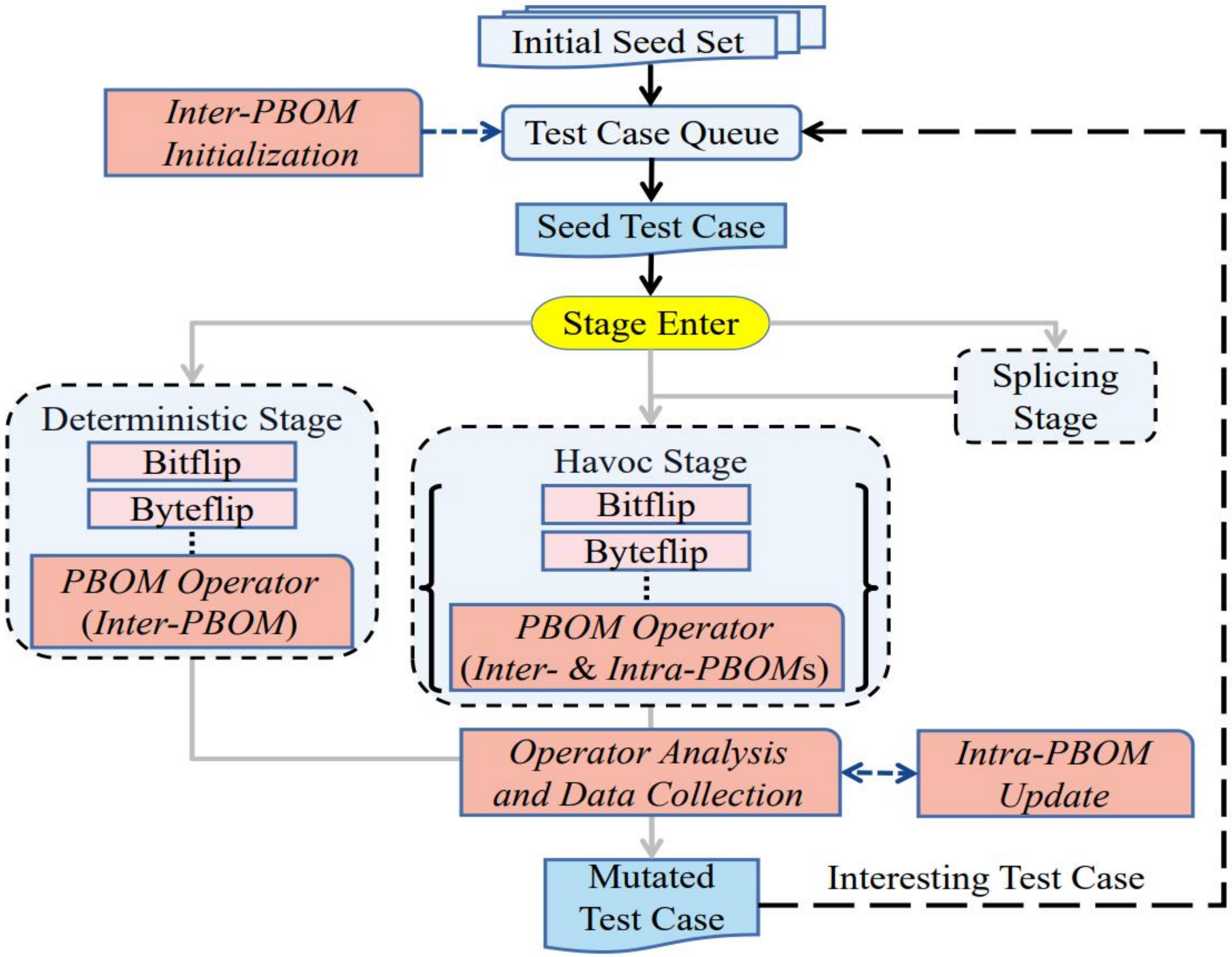
Probability algorithm used in intra-PBOM:

$$P_i = \frac{F_i}{F_1 + F_2 + ... + F_{n-1} + F_n} = \frac{count((out_i, type))}{\sum_{(out_k, type) \in \mathbb{MO}} count((out_k, type))} . \qquad (2)$$

Intra-PBOM prefers to output the mutation strategies that are the most efficient ones to generate interesting test cases in this trial.

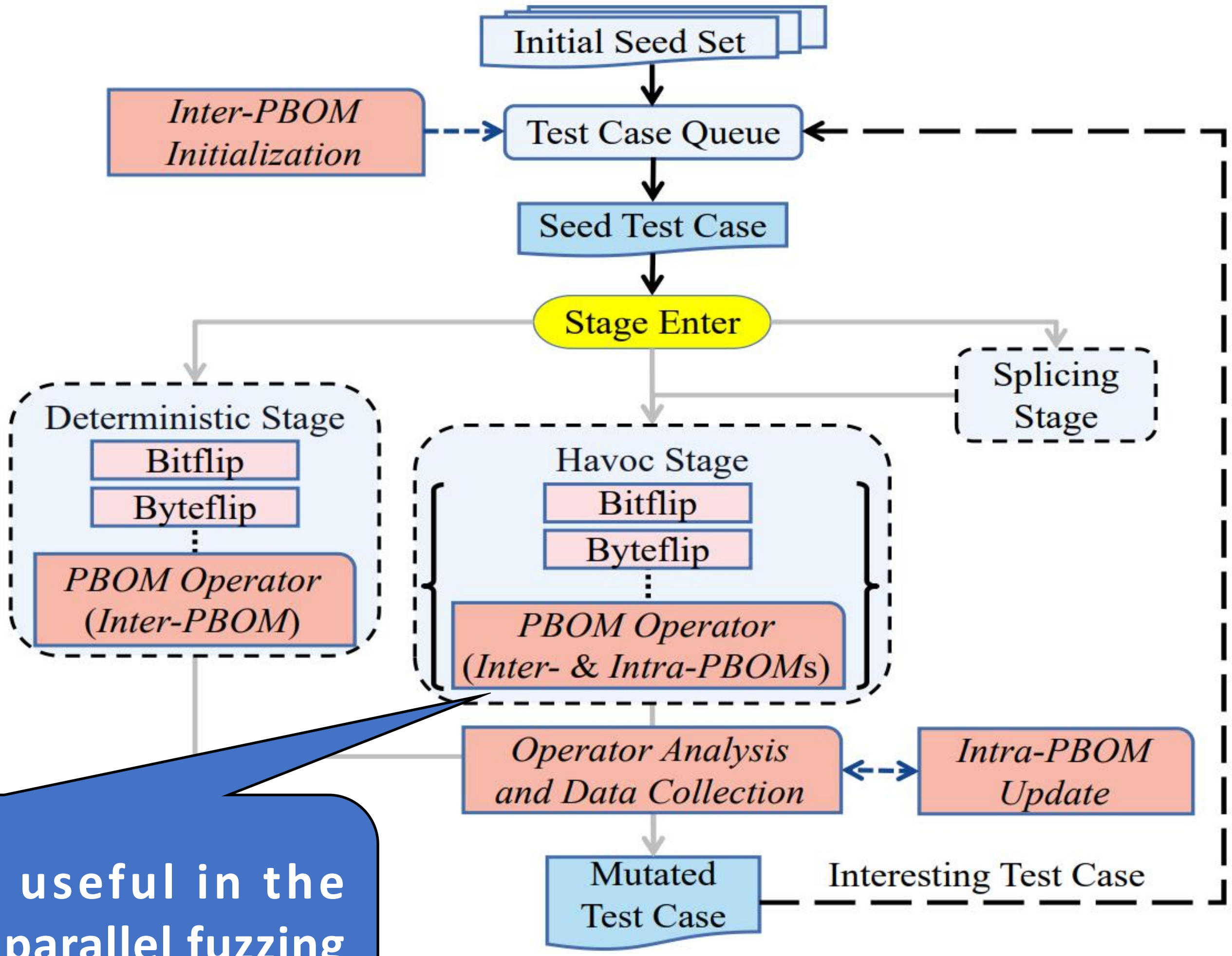
Assign more selection probability to high-frequency mutation strategies.

Workflow of EMS



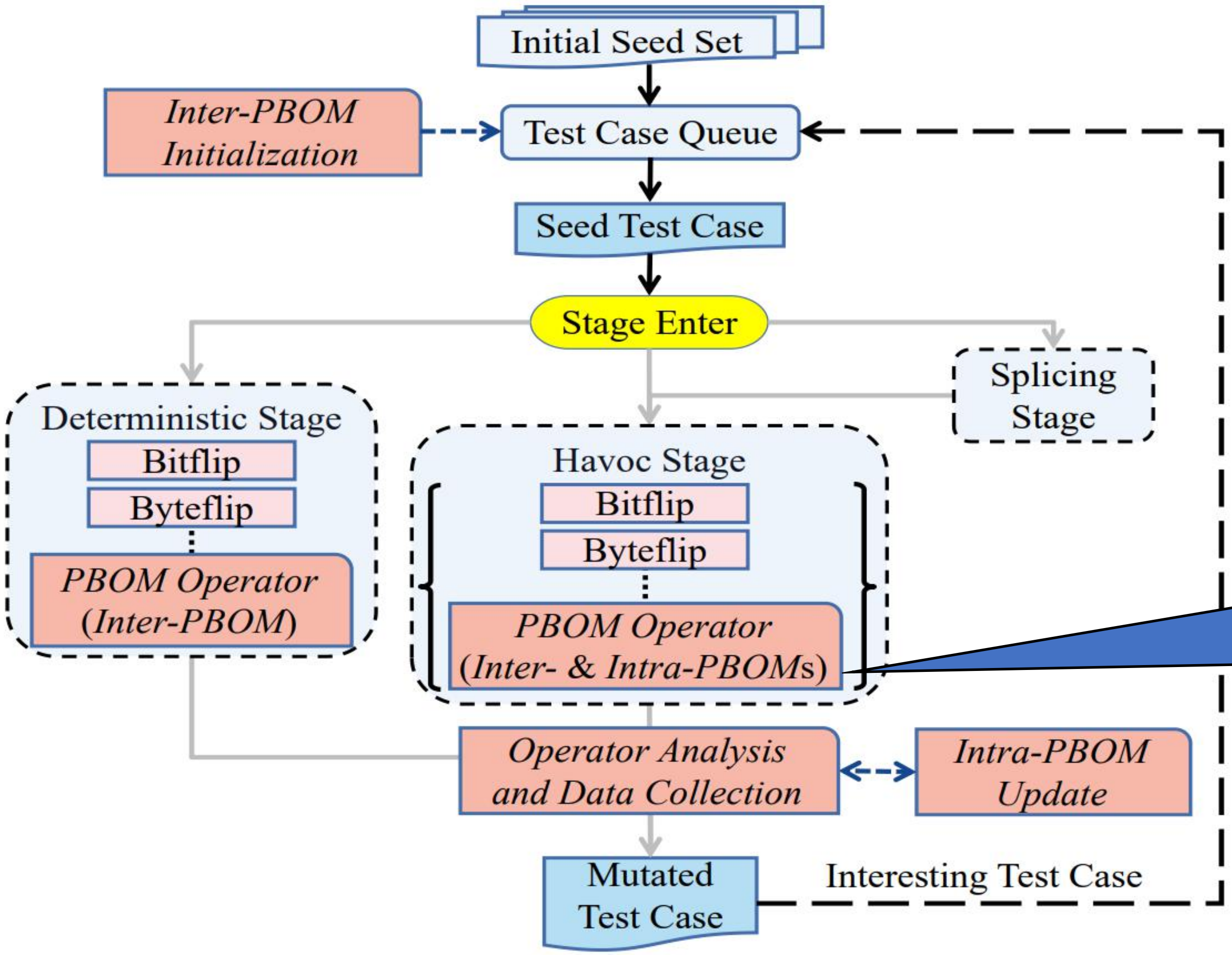
The solution of EMS can be easily extended to fuzzing tools.

Application Scenarios of PBOMs



Inter-PBOM can be useful in the fuzzing scenarios like parallel fuzzing and continuous fuzzing.

Application Scenarios of PBOMs



Intra-PBOM can be used in each trial.

Evaluation

Experiment Settings

- **Compared fuzzers: AFL, QSYM, MOPT, MOPT-dict, EcoFuzz, AFL++**
- **Target programs:**

Target	Source	Input format	Test instruction
pdfimages	xpdf-4.02	pdf	@@ /dev/null
pdftotext	xpdf-4.02	pdf	@@ /dev/null
objdump	binutils-2.28	binary	-S @@
infotocap	ncurses-6.2	txt	@@ -o /dev/null
cflow	cflow-1.6	C files	@@
nasm	nasm-2.14.03rc2	asm	-f bin @@ -o /dev/null
w3m	w3m-0.5.3	txt	@@
mujs	mujs-1.0.2	javascript	@@
mp3gain	mp3gain-1.5.2-r2	mp3	@@

Each evaluation lasts for 168 hours and is repeated 16 times.

Evaluation Metrics

- **The number of unique vulnerabilities found by each fuzzer, which are de-duplicated by the top three function calls reported by ASan.**
- **The number of published CVE IDs found by each fuzzer.**
- **The line coverage reported by afl-cov.**

Number of Unique Vulnerabilities After Deduplication in 16 Trials

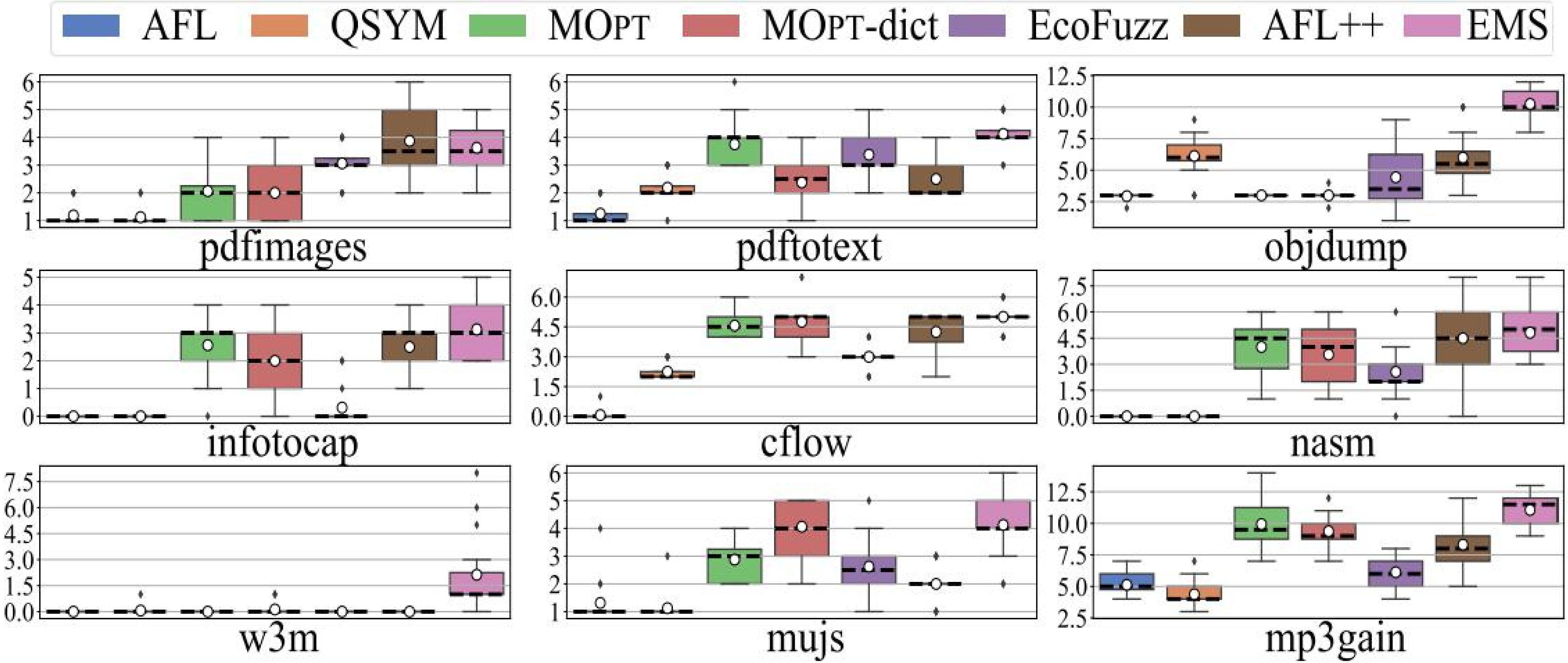
	AFL	QSYM	MOPT	MOPT-dict	EcoFuzz	AFL++	EMS
pdfimages	2	3	4	5	7	13	**15**
pdftotext	2	6	9	9	9	6	**13**
objdump	5	11	3	6	18	22	**30**
infotocap	0	0	6	6	3	**7**	**7**
cflow	1	4	6	7	6	7	**9**
nasm	0	0	11	15	13	**20**	18
w3m	0	1	0	1	0	0	**11**
mujs	4	3	4	6	6	6	**7**
mp3gain	8	11	17	18	16	18	**20**
total	22	39	60	73	78	99	**130**

Number of Unique Vulnerabilities After Deduplication in 16 Trials

	AFL	QSYM	MOPT	MOPT-dict	EcoFuzz	AFL++	EMS
pdfimages	2	3	4	5	7	13	**15**
pdftotext	2	6	9	9	9	6	**13**
objdump	5	11	3	6	18	22	**30**
infotocap	0	0	6	6	3	**7**	**7**
cflow	1	4	6	7	6	7	**9**
nasm	0	0	11	15	13	**20**	18
w3m	0	1	0	1	0	0	**11**
mujs	4	3	4	6	6	6	**7**
mp3gain	8	11	17	18	16	18	**20**
total	22	39	60	73	78	99	**130**

EMS finds the most vulnerabilities on 8 target programs after deduplication.

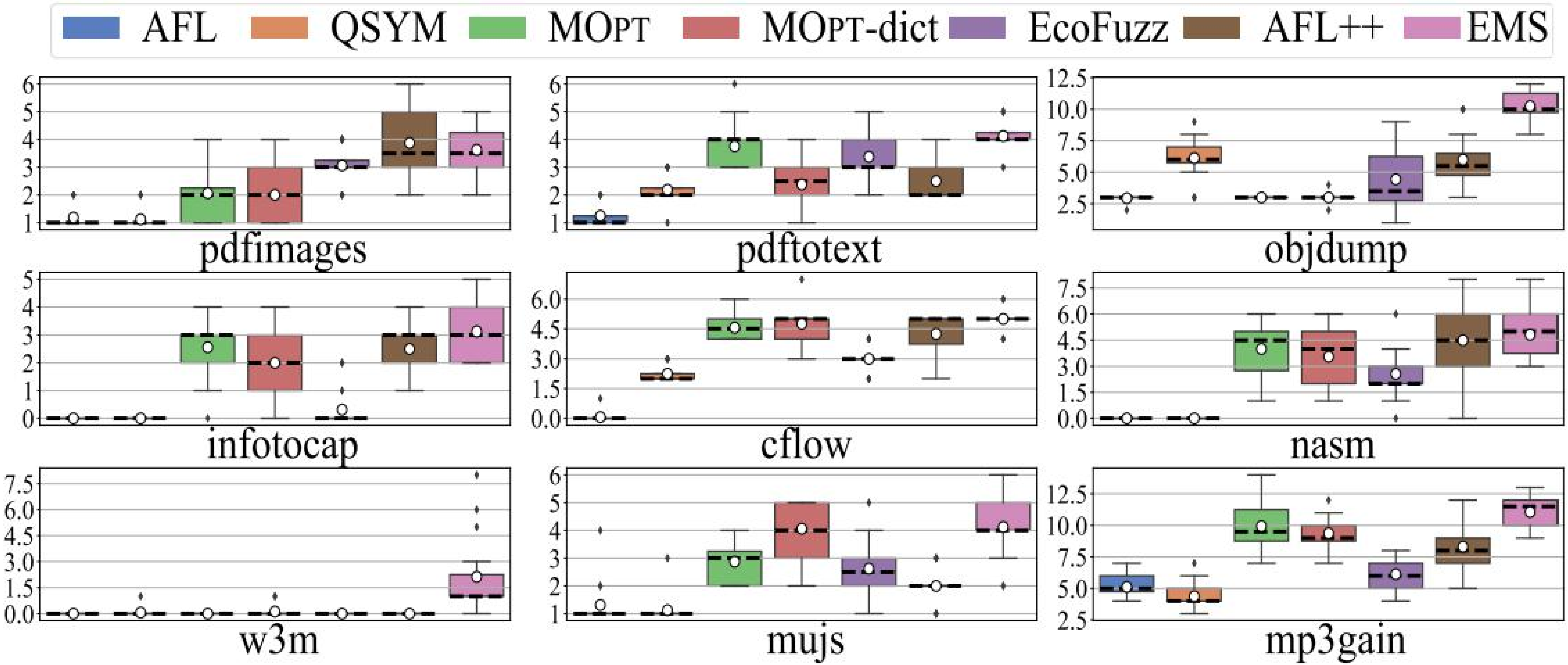
Boxplot of Number of Unique Vulnerabilities in 16 Trials



‘ ○ ’ and ‘– –’ represent the mean and median, respectively.

Y-axis: the number of unique vulnerabilities discovered in each trial

Boxplot of Number of Unique Vulnerabilities in 16 Trials



EMS can find more vulnerabilities than other fuzzers in a single trial.

Published CVE IDs Found by Each Fuzzer

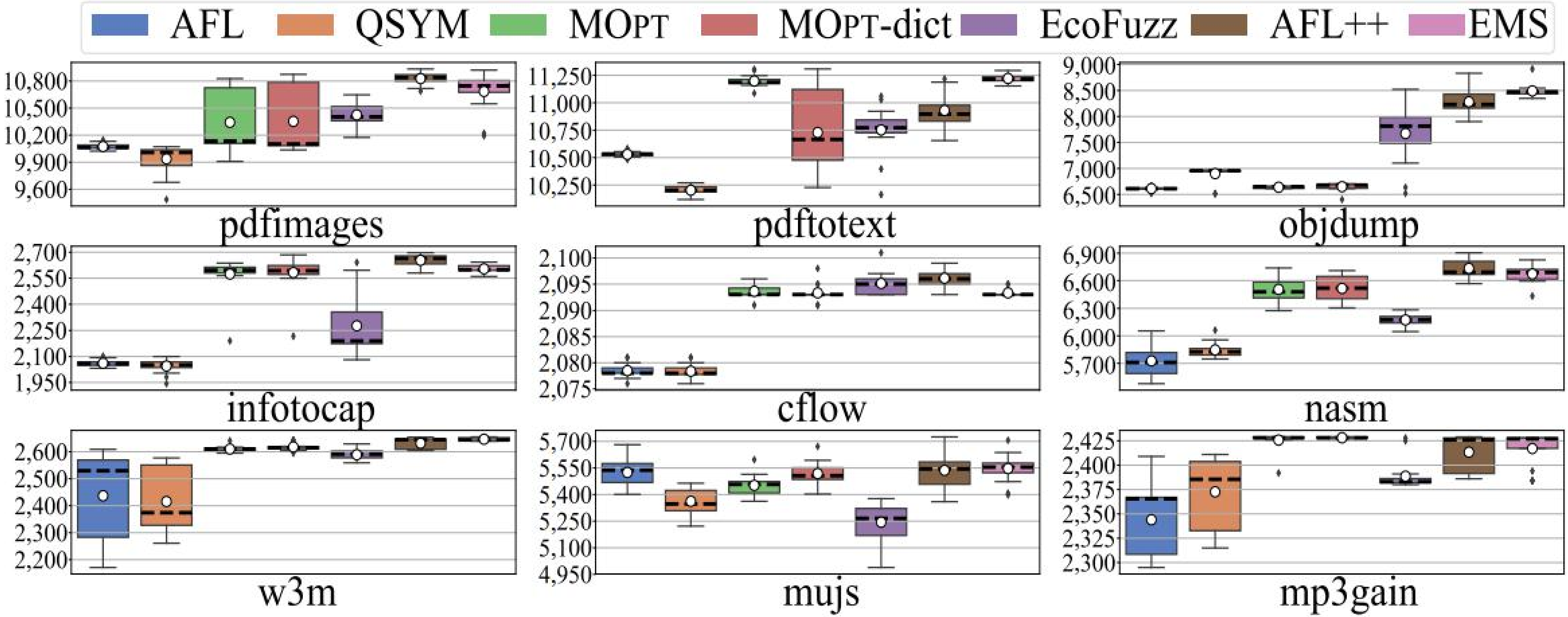
	CVE ID	AFL	QSYM	MOpt	MOpt-dict	EcoFuzz	AFL++	EMS
pdfimages	CVE-2019-17064	●	●	●	●	●	●	●
	CVE-2019-9588							●
pdftotext	CVE-2019-16088	●	●	●	●	●	●	●
	CVE-2019-9588						●	
objdump	CVE-2017-8396	●			●	●	●	●
	CVE-2017-8398		●					●
	CVE-2017-14930		●					
	CVE-2017-16831		●					
	CVE-2018-7568					●	●	●
	CVE-2018-1000876							●
	CVE-2019-9072		●					
	CVE-2019-17450		●					
cflow	CVE-2019-16165	●	●	●	●	●	●	●
	CVE-2019-16166						●	
	CVE-2020-23856			●	●	●	●	●
nasm	CVE-2018-19755			●	●	●	●	●
	CVE-2018-20535			●	●		●	●
	CVE-2018-20538			●	●			●
	CVE-2019-20334						●	●
mujs	CVE-2017-5628				●	●		●
	CVE-2018-6191	●	●	●	●	●	●	●
mp3gain	CVE-2017-14406	●	●	●	●	●	●	●
	CVE-2017-14407	●	●	●	●	●	●	●
	CVE-2017-14409						●	
	CVE-2017-14410				●		●	●
	CVE-2019-18359		●					●
	total	7	12	10	13	11	16	**19**

Published CVE IDs Found by Each Fuzzer

	CVE ID	AFL	QSYM	MOpt	MOpt-dict	EcoFuzz	AFL++	EMS
pdfimages	CVE-2019-17064	●	●	●	●	●	●	●
	CVE-2019-9588							●
pdftotext	CVE-2019-16088	●	●	●	●	●	●	●
	CVE-2019-9588						●	
objdump	CVE-2017-8396	●			●	●	●	●
	CVE-2017-8398		●					●
	CVE-2017-14930		●					
	CVE-2017-16831		●					
	CVE-2018-7568					●	●	●
	CVE-2018-1000876							●
	CVE-2019-9072		●					
	CVE-2019-17450		●					
cflow	CVE-2019-16165	●	●	●	●	●	●	●
	CVE-2019-16166						●	
	CVE-2020-23856			●	●	●	●	●
nasm	CVE-2018-19755			●	●	●	●	●
	CVE-2018-20535			●	●		●	●
	CVE-2018-20538			●	●			●
	CVE-2019-20334						●	●
mujs	CVE-2017-5628				●	●		●
	CVE-2018-6191	●	●	●	●	●	●	●
mp3gain	CVE-2017-14406	●	●	●	●	●	●	●
	CVE-2017-14407	●	●	●	●	●	●	●
	CVE-2017-14409						●	
	CVE-2017-14410				●		●	●
	CVE-2019-18359		●					●
	total	7	12	10	13	11	16	**19**

EMS achieves better CVE discovery performance than other fuzzers.

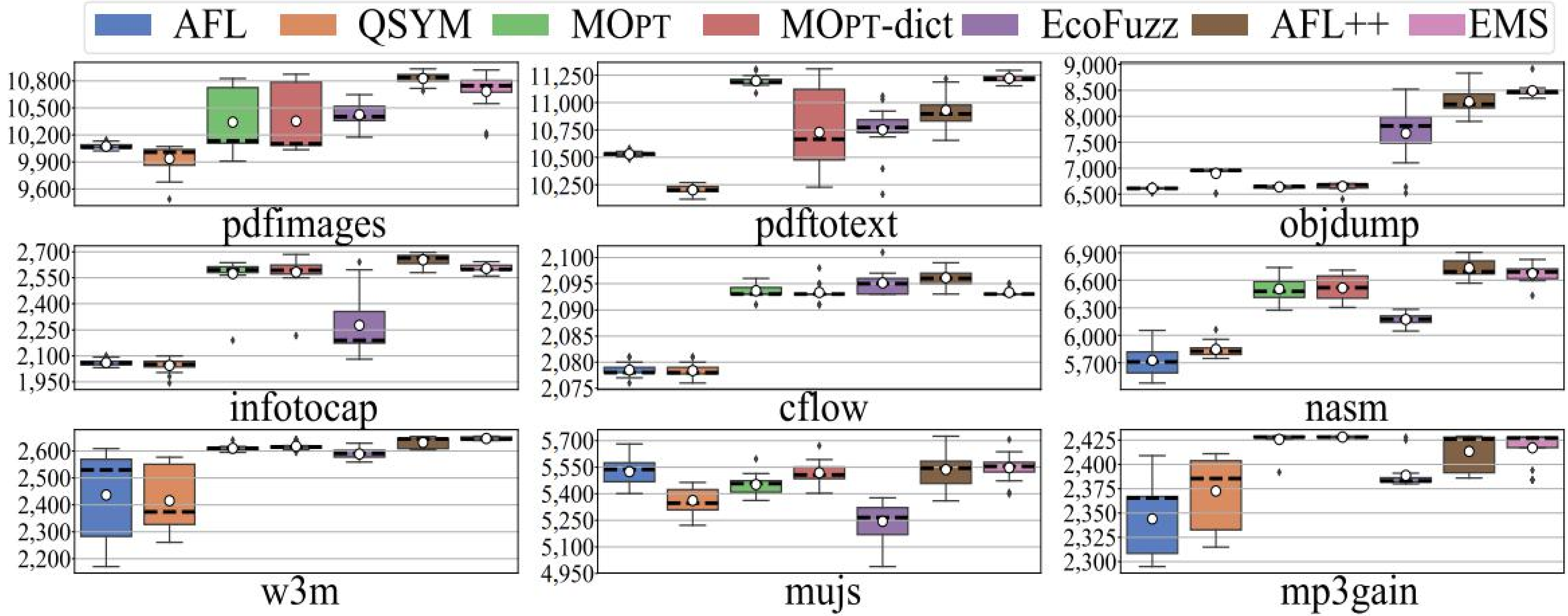
Boxplot of Number of Line Coverage in 16 Trials



‘ ○ ’ and ‘– –’ represent the mean and median, respectively.

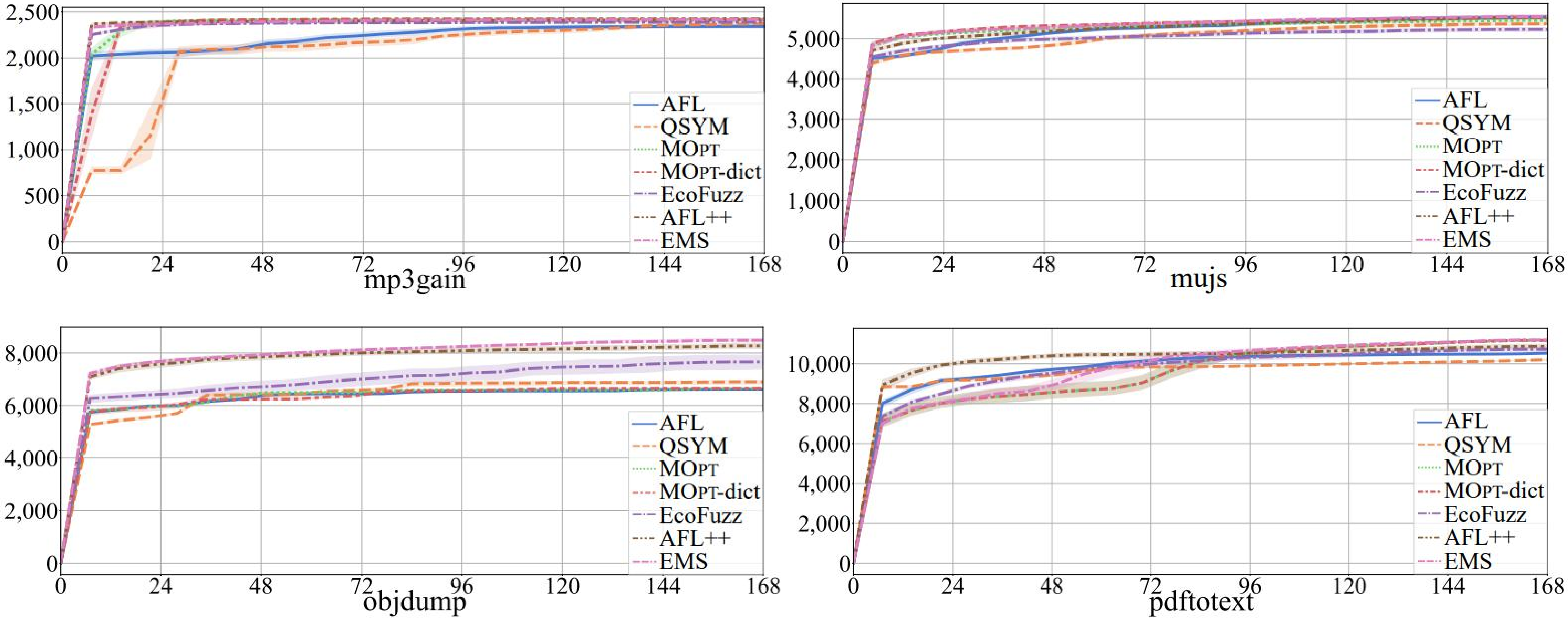
Y-axis: the line coverage discovered in each trial

Boxplot of Number of Line Coverage in 16 Trials



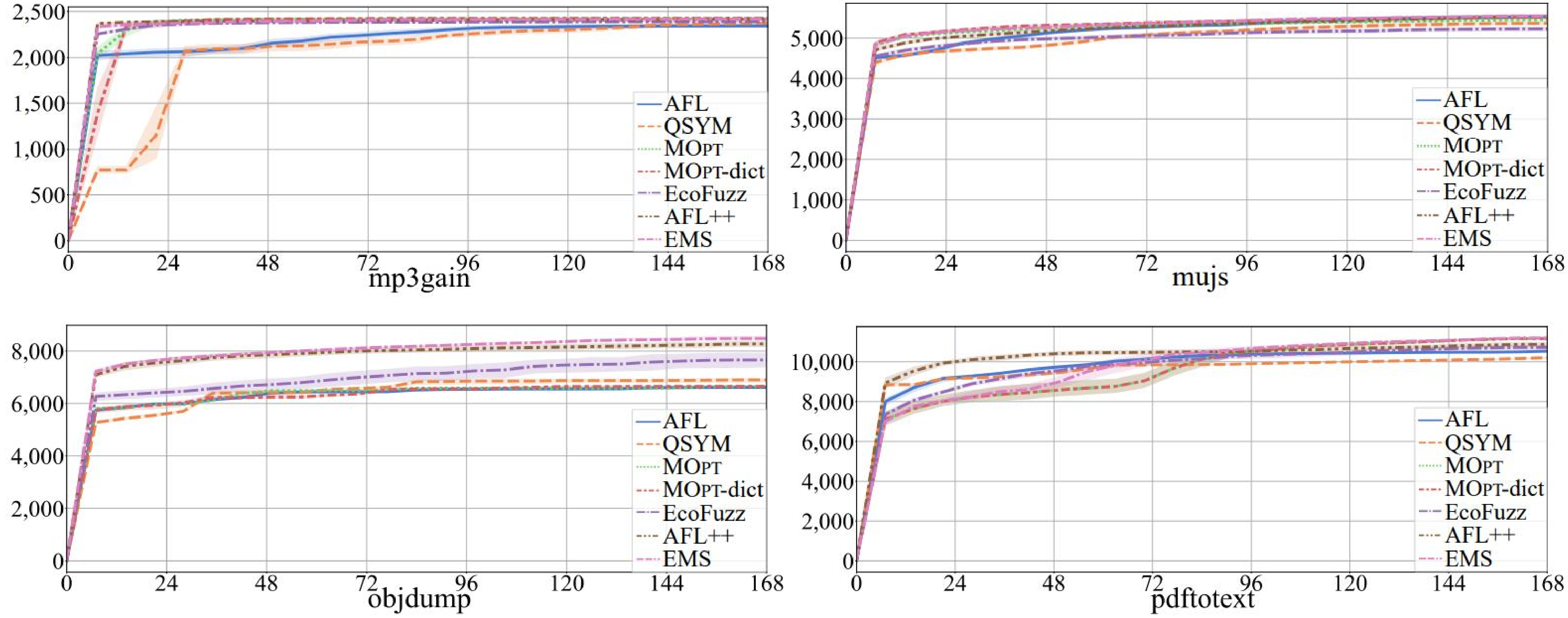
The solution of EMS can improve line coverage performance.

Line Coverage Growth over 168 Hours



Each coverage interval with a different color shows the mean and 95% confidence interval for a fuzzer. Y-axis: the number of covered code lines.

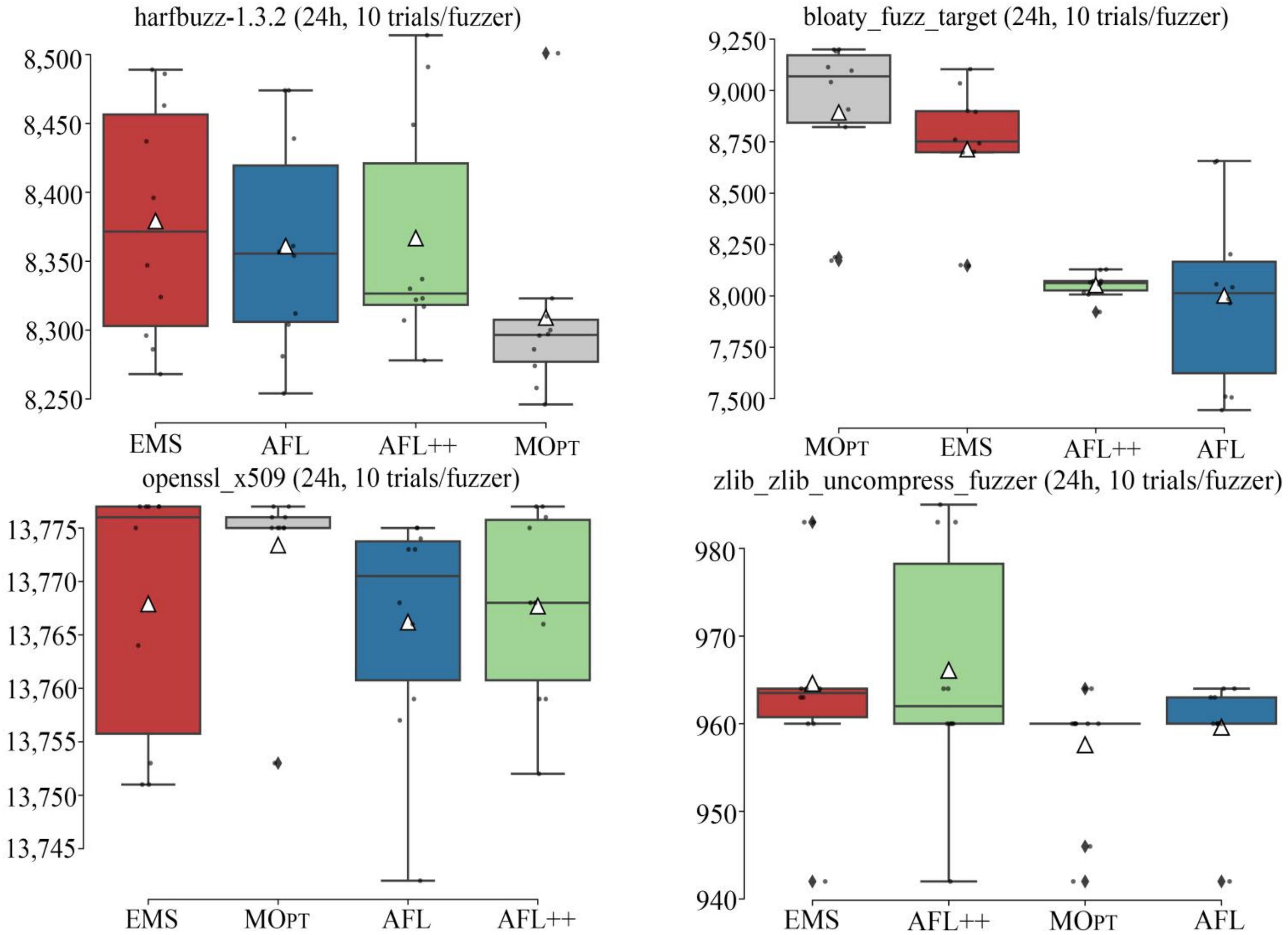
Line Coverage Growth over 168 Hours



The line coverage of EMS grows faster than other fuzzers over 168 hours.

Evaluation on FuzzBench

Each evaluation lasts for 24 hours and is repeated 10 times.



Further Analysis

PBOM Contribution Analysis

P+T: the interesting test cases generated by the mutations from *PBOM Operator* and traditional mutation operators.

T: the shadow versions of interesting test cases, which are generated by only replaying the mutations from traditional mutation operators at the same locations.

pdfimages						
Trial	Unique vulnerabilities found by T	Unique vulnerabilities found by P + T	Contribution	Edge coverage triggered by T	Edge coverage triggered by P + T	Contribution
1	2	3	1	1,825	2,303	+26.2%
2	1	2	1	1,766	2,281	+29.2%
3	2	2	0	1,747	2,234	+27.9%
4	3	3	0	1,659	2,170	+30.8%
5	3	5	2	1,836	2,344	+27.7%
6	2	2	0	1,776	2,289	+28.9%

PBOM Operator **can improve the performance of vulnerability discovery and edge coverage.**

PBOM Contribution Analysis

P+T: the interesting test cases generated by the mutations from *PBOM Operator* and traditional mutation operators.

T: the shadow versions of interesting test case

the mutations from traditional mutation opera

Most mutations on an interesting test case are provided by traditional mutation operators, the shadow test cases have only a very small percentage of mutations removed. Thus, *PBOM Operator* provides the key mutations to find unique vulnerabilities and edge coverage.

pdfima

Trial	Unique vulnerabilities found by T	Unique vulnerabilities found by P + T	Contribution	Edge coverage triggered by T	Edge coverage triggered by P + T	Contribution
1	2	3	1	1,825	2,303	+26.2%
2	1	2	1	1,766	2,281	+29.2%
3	2	2	0	1,747	2,234	+27.9%
4	3	3	0	1,659	2,170	+30.8%
5	3	5	2	1,836	2,344	+27.7%
6	2	2	0	1,776	2,289	+28.9%

***PBOM Operator* can improve the performance of vulnerability discovery and edge coverage.**

Efficient Mutation Strategy Analysis

The similarities and differences between the efficient mutation strategies learned on different programs

Program A	Duration	N_{n1} (pct.)	N_{n2} (pct.)	N_y (pct.)	N_t	N_{n1} (pct.)	N_{n2} (pct.)	N_y (pct.)	N_t	Program B
pdfimages	5 hours	2,755 (33.0%)	565 (6.8%)	5,020 (60.2%)	8,340	5,971 (37.6%)	4,876 (30.7%)	5,020 (31.6%)	15,867	nasm
	1 day	3,331 (29.4%)	821 (7.3%)	7,168 (63.3%)	11,320	8,021 (36.9%)	6,553 (30.1%)	7,168 (33.0%)	21,742	
	2 days	2,824 (25.7%)	754 (6.9%)	7,400 (67.4%)	10,978	9,388 (38.1%)	7,861 (31.9%)	7,400 (30.0%)	24,649	
	7 days	2,906 (28.5%)	525 (5.1%)	6,775 (66.4%)	10,206	9,098 (39.2%)	7,361 (31.7%)	6,775 (29.2%)	23,234	
objdump	5 hours	3,977 (53.2%)	2,007 (26.9%)	1,487 (19.9%)	7,471	2,446 (50.3%)	925 (19.0%)	1,487 (30.6%)	4,858	infotocap
	1 day	3,941 (50.8%)	1,795 (23.2%)	2,015 (26.0%)	7,751	3,530 (50.9%)	1,394 (20.1%)	2,015 (29.0%)	6,939	
	2 days	3,645 (51.0%)	1,294 (18.1%)	2,210 (30.9%)	7,149	4,878 (53.6%)	2,010 (22.1%)	2,210 (24.3%)	9,098	
	7 days	5,732 (54.5%)	2,049 (19.5%)	2,733 (26.0%)	10,514	5,176 (53.7%)	1,722 (17.9%)	2,733 (28.4%)	9,631	
cflow	5 hours	1,637 (44.8%)	844 (23.1%)	1,174 (32.1%)	3,655	3,566 (37.8%)	4,678 (49.7%)	1,174 (12.5%)	9,418	w3m
	1 day	1,576 (37.9%)	902 (21.7%)	1,676 (40.4%)	4,154	4,337 (33.6%)	6,894 (53.4%)	1,676 (13.0%)	1,2907	
	2 days	1,802 (44.5%)	649 (16.0%)	1,598 (39.5%)	4,049	3,701 (29.5%)	7,226 (57.7%)	1,598 (12.8%)	12,525	
	7 days	1,661 (44.0%)	733 (19.4%)	1,385 (36.6%)	3,779	3,550 (31.4%)	6,367 (56.3%)	1,385 (12.3%)	11,302	

N_t: The total number of efficient mutation strategies collected from the current experiment.

N_{n1}: The number of mutation strategies whose input byte values appear in both experiments, while their output byte values and mutation types only appear in the repective experiment.

N_{n2}: The number of mutation strategies whose input byte values only appear in the repective experiment.

N_y: The number of mutation strategies whose input byte values, output byte values and mutation types appear in both experiments.

The same inter-PBOM can be useful on different programs.

Efficient Mutation Strategy Analysis

The similarities and differences between the efficient mutation strategies learned on different programs

Program A	Duration	N_{n1} (pct.)	N_{n2} (pct.)	N_y (pct.)	N_t	N_{n1} (pct.)	N_{n2} (pct.)	N_y (pct.)	N_t	Program B
pdfimages	5 hours	2,755 (33.0%)	565 (6.8%)	5,020 (60.2%)	8,340	5,971 (37.6%)	4,876 (30.7%)	5,020 (31.6%)	15,867	nasm
	1 day	3,331 (29.4%)	821 (7.3%)	7,168 (63.3%)	11,320	8,021 (36.9%)	6,553 (30.1%)	7,168 (33.0%)	21,742	
	2 days	2,824 (25.7%)	754 (6.9%)	7,400 (67.4%)	10,978	9,388 (38.1%)	7,861 (31.9%)	7,400 (30.0%)	24,649	
	7 days	2,906 (28.5%)	525 (5.1%)	6,775 (66.4%)	10,206	9,098 (39.2%)	7,361 (31.7%)	6,775 (29.2%)	23,234	
objdump	5 hours	3,977 (53.2%)	2,007 (26.9%)	1,487 (19.9%)	7,471	2,446 (50.3%)	925 (19.0%)	1,487 (30.6%)	4,858	infotocap
	1 day	3,941 (50.8%)	1,795 (23.2%)	2,015 (26.0%)	7,751	3,530 (50.9%)	1,394 (20.1%)	2,015 (29.0%)	6,939	
	2 days	3,645 (51.0%)	1,294 (18.1%)	2,210 (30.9%)	7,149	4,878 (53.6%)	2,010 (22.1%)	2,210 (24.3%)	9,098	
	7 days	5,732 (54.5%)	2,049 (19.5%)	2,733 (26.0%)	10,514	5,176 (53.7%)	1,722 (17.9%)	2,733 (28.4%)	9,631	
cflow	5 hours	1,637 (44.8%)	844 (23.1%)	1,174 (32.1%)	3,655	3,566 (37.8%)	4,678 (49.7%)	1,174 (12.5%)	9,418	w3m
	1 day	1,576 (37.9%)	902 (21.7%)	1,676 (40.4%)	4,154	4,337 (33.6%)	6,894 (53.4%)	1,676 (13.0%)	1,2907	
	2 days	1,802 (44.5%)	649 (16.0%)	1,598 (39.5%)	4,049	3,701 (29.5%)	7,226 (57.7%)	1,598 (12.8%)	12,525	
	7 days	1,661 (44.0%)	733 (19.4%)	1,385 (36.6%)	3,779	3,550 (31.4%)	6,367 (56.3%)	1,385 (12.3%)	11,302	

N_t: The total number of efficient mutation strategi

N_{n1}: The number of mutation strategies whose inp r

output byte values and mutation types only appea

N_{n2}: The number of mutation strategies whose inp nt.

N_y: The number of mutation strategies whose inpu s

appear in both experiments.

N_{n1} and N_y account for the majority, which implies that **using input byte values as the index of efficient mutation strategies is reasonable**.

The same inter-PBOM can be useful on different programs.

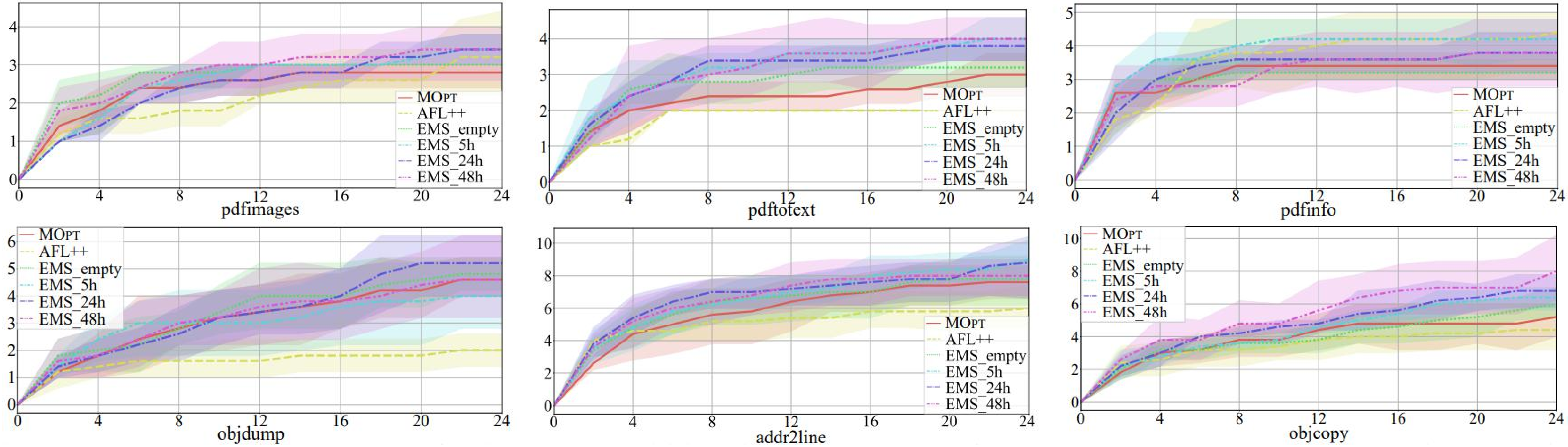
Evaluation on Programs from the Same Vendor

- **Compared fuzzers: MOPT, AFL++, EMS_empty, EMS_5h, EMS_24h, EMS_48h (EMS with different inter-PBOMs)**
- **Target programs:**

Source	Target	Input format	Test instruction
xpdf-4.02	pdfimages	pdf	@@ /dev/null
	pdftotext	pdf	@@ /dev/null
	pdfinfo	pdf	@@
binutils-2.28	objdump	binary	-S @@
	addr2line	binary	s -e @@
	objcopy	binary	−−debugging -p -D @@ /dev/null

Each evaluation lasts for 24 hours and is repeated 5 times.

Evaluation on Programs from the Same Vendor



Each coverage interval with a different color shows the mean and 95% confidence interval for a unique fuzzer. Y-axis: the number of the unique vulnerabilities reported by ASan.

The results demonstrate the contribution of the inter-PBOM to different programs developed by the same vendor.

Conclusion

Conclusion

- We discover that both intra- and inter-trial fuzzing history contain rich knowledge of key mutation strategies that lead to the discovery of unique paths or crashes.
- we propose PBOM to capture the mutation strategies that trigger unique paths and crashes from the intra- and inter-trial history.
- We present a novel history-driven mutation framework EMS that employs PBOM as one of the mutation operators to probabilistically provide the desired mutation byte values and mutation types according to the input ones.
- The evaluation results demonstrate the significant fuzzing performance of EMS and the contribution of PBOM to the generation of interesting test cases.
- https://github.com/puppet-meteor/EMS

Q&A

puppet@zju.edu.cn