A Comparative Study on the Accuracy and the Speed of Static and Dynamic Program Classifiers

Anderson Faustino afsilva@uem.br

Jeronimo Castrillon jeronimo.castrillon@tu-dresden.de

Fernando Pereira fernando@dcc.ufmg.br

Goal

1. Evaluate program classifiers!

Goal

1. Evaluate program classifiers!

- Malware Identification
- Plagiarism Detection
- Redundancy Elimination
- Code Lifting

Goal

1. Evaluate program classifiers!
2. Static vs Dynamic vs Hybrid

- Malware Identification
- Plagiarism Detection
- Redundancy Elimination
- Code Lifting

Goal

1. Evaluate program classifiers!
2. Static vs Dynamic vs Hybrid
3. Using the same representation

Goal

1. Evaluate program classifiers!
2. Static vs Dynamic vs Hybrid
3. Using the same representation

If we can observe program execution, can we do better?

Goal

1. Evaluate program classifiers!
2. Static vs Dynamic vs Hybrid
3. Using the same representation

br	phi	icmp	mul	add	ret
3	2	1	1	1	1

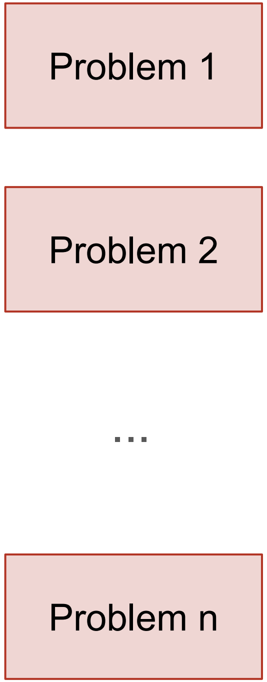
Opcode Histograms

What is a program classifier?

1. Evaluate program classifiers!
2. Static vs Dynamic vs Hybrid
3. Using the same representation

What is a program classifier?

program P



Problem 1

Problem 2

...

Problem n

What is a program classifier?

program P

Which problem does P solve?

Problem 1

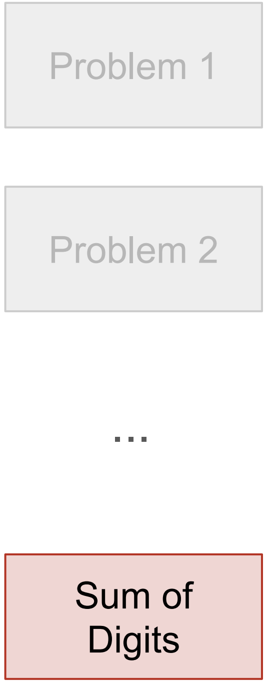
Problem 2

...

Problem n

What is a program classifier?

Statement:
Given a positive integer N, compute the sum of its digits.



Problem 1

Problem 2

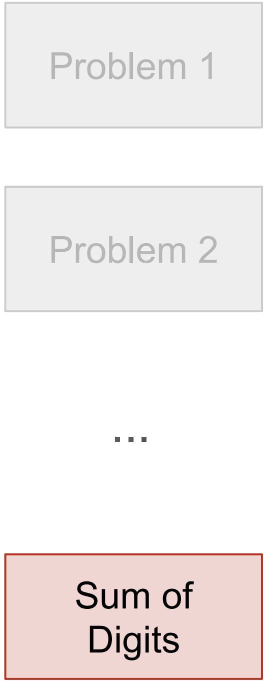
...

Sum of Digits

What is a program classifier?

Statement:
Given a positive integer N, compute the sum of its digits.

Input:
A single integer N (1 ≤ N ≤ 10^9).



Problem 1

Problem 2

...

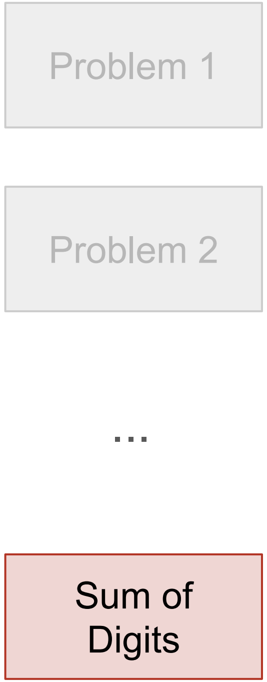
Sum of Digits

What is a program classifier?

Statement:
Given a positive integer N, compute the sum of its digits.

Input:
A single integer N (1 ≤ N ≤ 10^9).

Output:
A single integer representing the sum of the digits of N.



Problem 1

Problem 2

...

Sum of Digits

What is a program classifier?

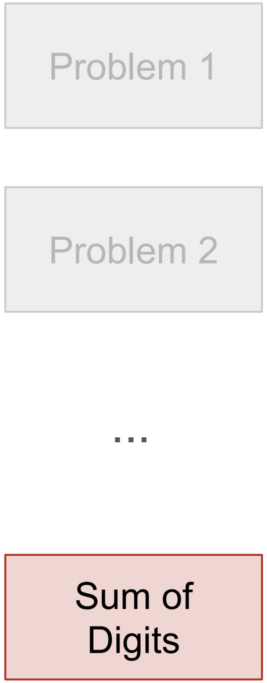
Statement:
Given a positive integer N, compute the sum of its digits.

Input:
A single integer N (1 ≤ N ≤ 10^9).

Output:
A single integer representing the sum of the digits of N.

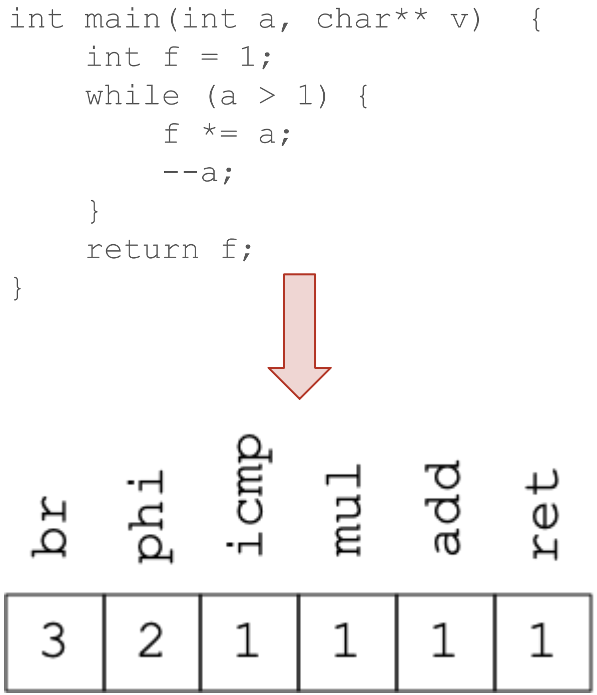
Example input:
123

Expected output:
6



How Classification Works

DCC



```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

br	phi	icmp	mul	add	ret
3	2	1	1	1	1

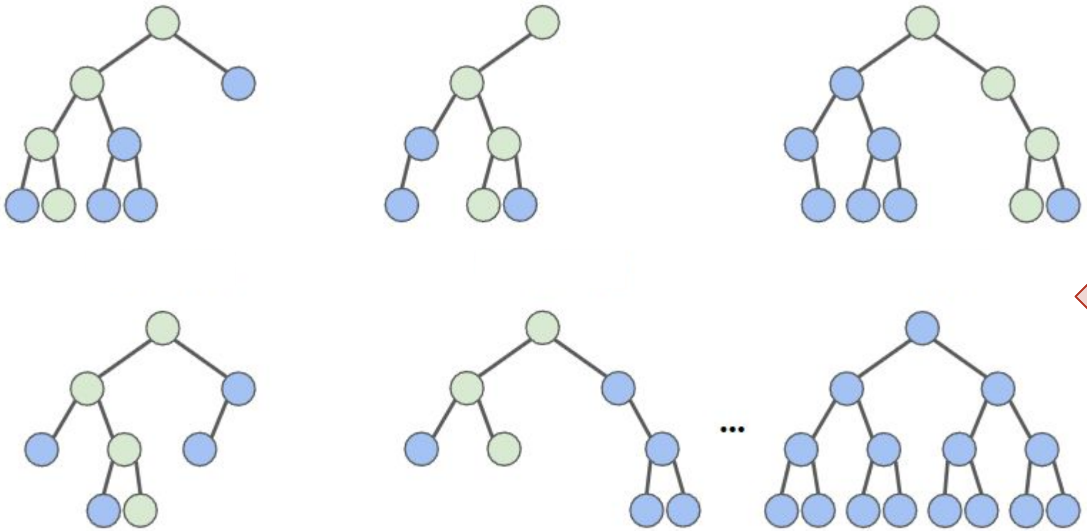
Opcode Histograms:
The embedding

How Classification Works

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

br	phi	icmp	mul	add	ret
3	2	1	1	1	1

Opcode Histograms

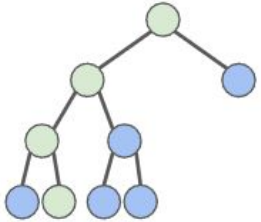


How Classification Works

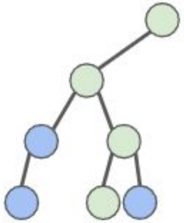
```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

br	phi	icmp	mul	add	ret
3	2	1	1	1	1

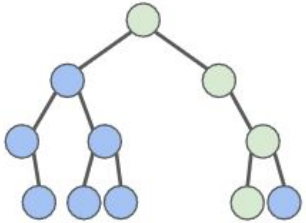
Opcode Histograms



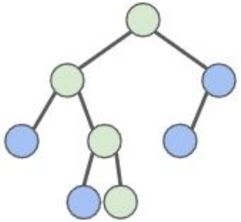
Tree 1: min sum



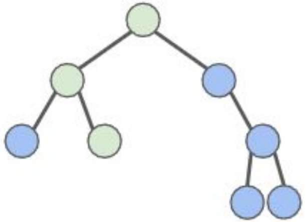
Tree 2: max subseq



Tree 3: max clique

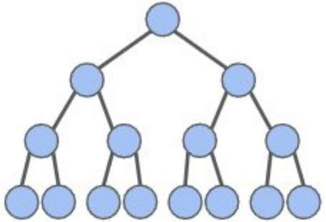


Tree 4: min cut



Tree 5: sum digits

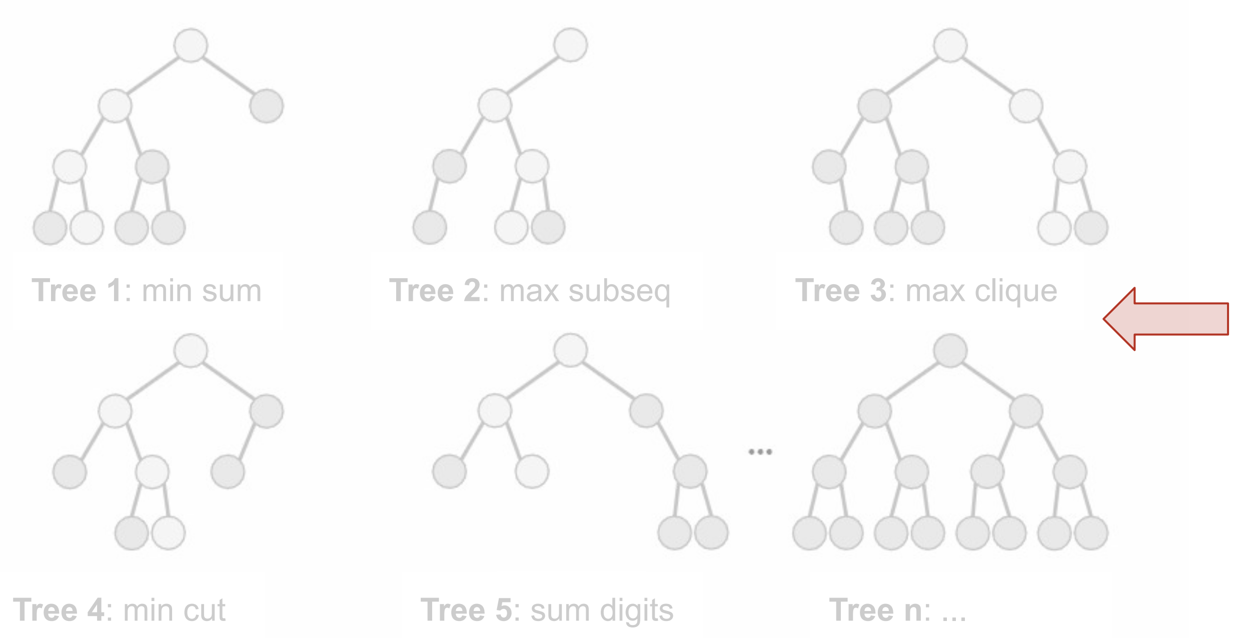
...



Tree n: ...

How Classification Works

The model is not very important... There are many



Tree 1: min sum

Tree 2: max subseq

Tree 3: max clique

Tree 4: min cut

Tree 5: sum digits

Tree n: ...

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

br	phi	icmp	mul	add	ret
3	2	1	1	1	1

Opcode Histograms

What are the types of classifiers?

1. Static
2. Dynamic
3. Hybrid

What are the types of classifiers?

1. Static → no execution

2. Dynamic

3. Hybrid

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

What are the types of classifiers?

1. Static → no execution
2. Dynamic
3. Hybrid

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

The x86 Image

```
0000000000401110 <main>:
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

What are the types of classifiers?

1. Static → no execution
2. Dynamic
3. Hybrid

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

The x86 Image

```
0000000000401110 <main>:
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
3	2	1	1	1	1	1	1	1

What are the types of classifiers?

1. Static → no execution
2. Dynamic
3. Hybrid

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

The LLVM Image

```
define @main(%a, %v) {
bb:
  br label %bb5
bb5:
  %i2 = phi [%a:%bb], [%i13:%bb8]
  %i4 = phi [1:%bb], [%i11:%bb8]
  %i7 = icmp sgt %i2, 1
  br i1 %i7, %bb8, %bb14
bb8:
  %i11 = mul %i4, %i2
  %i13 = add %i2, -1
  br %bb5
bb14:
  ret %i4
}
```

What are the types of classifiers?

1. Static → no execution
2. Dynamic
3. Hybrid

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

The LLVM Image

```
define @main(%a, %v) {
bb:
  br label %bb5
bb5:
  %i2 = phi [%a:%bb], [%i13:%bb8]
  %i4 = phi [1:%bb], [%i11:%bb8]
  %i7 = icmp sgt %i2, 1
  br i1 %i7, %bb8, %bb14
bb8:
  %i11 = mul %i4, %i2
  %i13 = add %i2, -1
  br %bb5
bb14:
  ret %i4
}
```

br	phi	icmp	mul	add	ret
3	2	1	1	1	1

What are the types of classifiers?

1. Static
2. Dynamic → CPU observer
3. Hybrid

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

What are the types of classifiers?

1. Static
2. Dynamic → CPU observer
3. Hybrid

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

```
$> ./a.out a a

[gdb] int f = 1;
[gdb] while (a > 1) {
[gdb] f *= a;
[gdb] --a;
[gdb] while (a > 1) {
[gdb] f *= a;
[gdb] --a;
[gdb] while (a > 1) {
[gdb] return f;
```

What are the types of classifiers?

1. Static
2. Dynamic → CPU observer
3. Hybrid

```
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

What are the types of classifiers?

1. Static

2. Dynamic → CPU observer

3. Hybrid

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
4	3	1	1	2	2	2	1	1

```
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

What are the types of classifiers?

1. Static
2. Dynamic → CPU observer
3. Hybrid

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
4	3	1	1	2	2	2	1	1

```
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

What are the types of classifiers?

1. Static

2. Dynamic

3. Hybrid → ICache observer

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
3	2	1	1	1	1	1	1	1

```
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

What are the types of classifiers?

1. Static
2. Dynamic
3. Hybrid → ICache observer

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
3	2	1	1	1	1	1	1	1

```
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

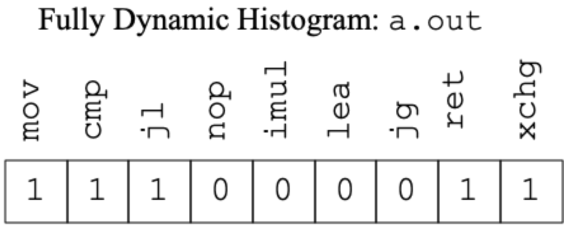
What are the types of classifiers?

1. Static
2. Dynamic → CPU observer
3. Hybrid → ICache observer

What are the types of classifiers?

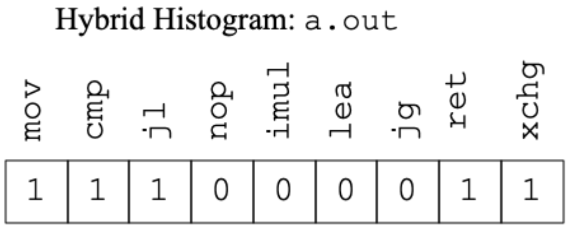
1. Static
2. Dynamic → CPU observer
3. Hybrid → ICache observer

Same result if each instruction runs once



Fully Dynamic Histogram: a.out

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1



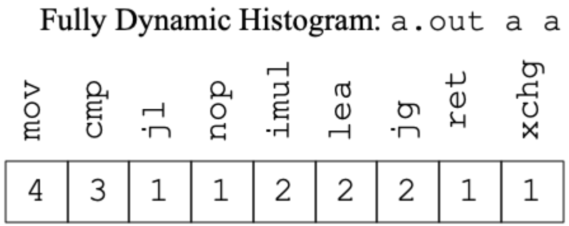
Hybrid Histogram: a.out

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1

What are the types of classifiers?

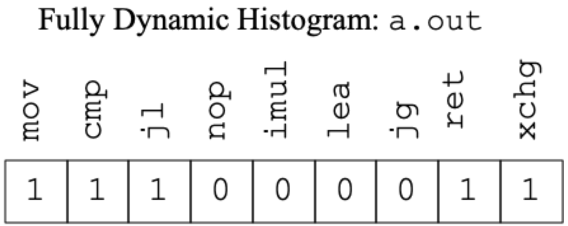
1. Static
2. Dynamic → CPU observer
3. Hybrid → ICache observer

Different results if each instruction runs more than once



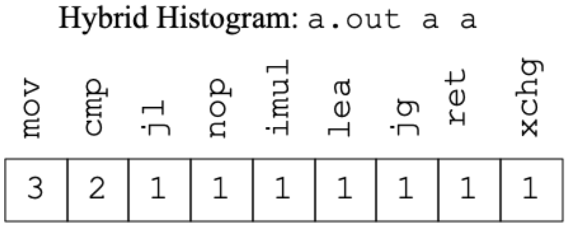
Fully Dynamic Histogram: `a.out a a`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
4	3	1	1	2	2	2	1	1



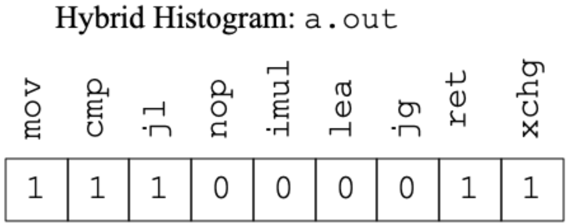
Fully Dynamic Histogram: `a.out`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1



Hybrid Histogram: `a.out a a`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
3	2	1	1	1	1	1	1	1



Hybrid Histogram: `a.out`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1

What are the types of classifiers?

1. Static
2. Dynamic → CPU observer
3. Hybrid → ICache observer

If instruction 0x401126 is fetched twice, it will be conted two times by the dynamic classifier.

Different result if each instruction runs more than once

Fully Dynamic Histogram: a.out a a

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
4	3	1	1	2	2	2	1	1

Fully Dynamic Histogram: a.out

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1

Hybrid Histogram: a.out a a

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
3	2	1	1	1	1	1	1	1

Hybrid Histogram: a.out

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1

What are the types of classifiers?

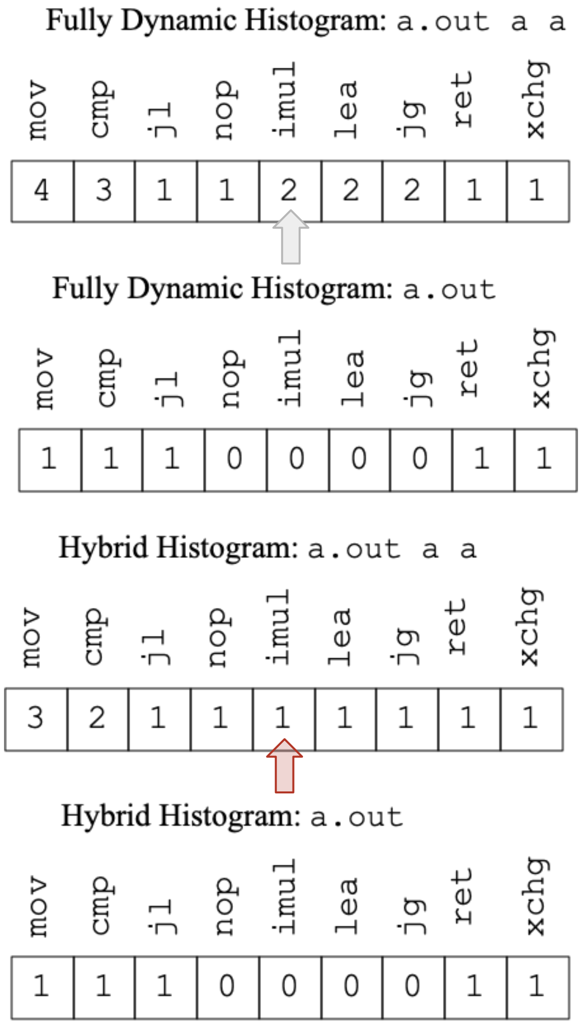
1. Static

2. Dynamic → CPU observer

3. Hybrid → ICache observer

If instruction 0x401126 is fetched twice, it will be conted two times by the dynamic classifier, but only once by the hybrid classifier.

Different result if each instruction runs more than once



Fully Dynamic Histogram: `a.out a a`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
4	3	1	1	2	2	2	1	1

Fully Dynamic Histogram: `a.out`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1

Hybrid Histogram: `a.out a a`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
3	2	1	1	1	1	1	1	1

Hybrid Histogram: `a.out`

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
1	1	1	0	0	0	0	1	1

The Rouxinol Infra-Structure

The Rouxinol Infra-Structure

https://github.com/ComputerSystemsLaboratory/Rouxinol

The Rouxinol Infra-Structure

https://github.com/ComputerSystemsLaboratory/Rouxinol

1. Histogram-based Classifiers

The Rouxinol Infra-Structure

https://github.com/ComputerSystemsLaboratory/Rouxinol

1. Histogram-based Classifiers
2. Datasets

The Rouxinol Infra-Structure

https://github.com/ComputerSystemsLaboratory/Rouxinol

1. Histogram-based Classifiers
2. Datasets
3. Games

Classification Games

3. Games

Classification Games

Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, Fernando Magno Quintão Pereira: **A Game-Based Framework to Compare Program Classifiers and Evaders**. CGO 2023: 108-121

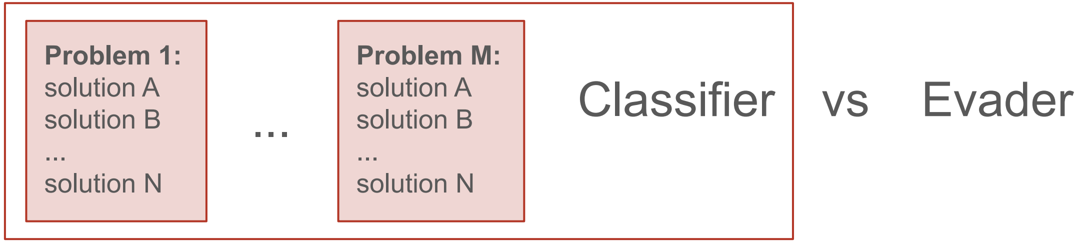
3. Games

Classification Games

Classifier vs Evader

Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, Fernando Magno Quintão Pereira: **A Game-Based Framework to Compare Program Classifiers and Evaders**. CGO 2023: 108-121

Classification Games



Problem 1:
solution A
solution B
...
solution N

...

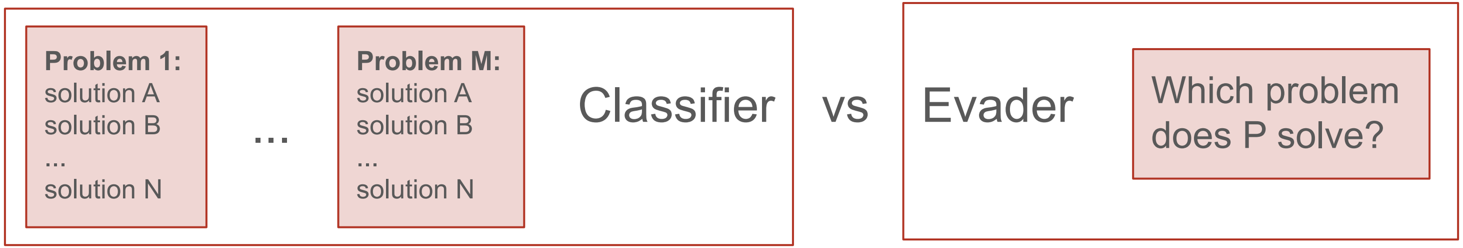
Problem M:
solution A
solution B
...
solution N

Classifier vs Evader

Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, Fernando Magno Quintão Pereira: **A Game-Based Framework to Compare Program Classifiers and Evaders**. CGO 2023: 108-121

3. Games

Classification Games



Problem 1:
solution A
solution B
...
solution N

...

Problem M:
solution A
solution B
...
solution N

Classifier

vs

Evader

Which problem does P solve?

Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, Fernando Magno Quintão Pereira: **A Game-Based Framework to Compare Program Classifiers and Evaders**. CGO 2023: 108-121

Classification Games

Game 0:

Given a program P, and N classes of problems, which problem does P solve?

Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, Fernando Magno Quintão Pereira: **A Game-Based Framework to Compare Program Classifiers and Evaders**. CGO 2023: 108-121

Classification Games

Game 0:
Given a program P, and N classes of problems, which problem does P solve?

Game 1:
Given an **obfuscated** program P, and N classes of problems, which problem does P solve?

Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, Fernando Magno Quintão Pereira: **A Game-Based Framework to Compare Program Classifiers and Evaders**. CGO 2023: 108-121

Classification Games

Game 0:
Given a program P, and N classes of problems, which problem does P solve?

Game 1:
Given an **obfuscated** program P, and N classes of problems, which problem does P solve?

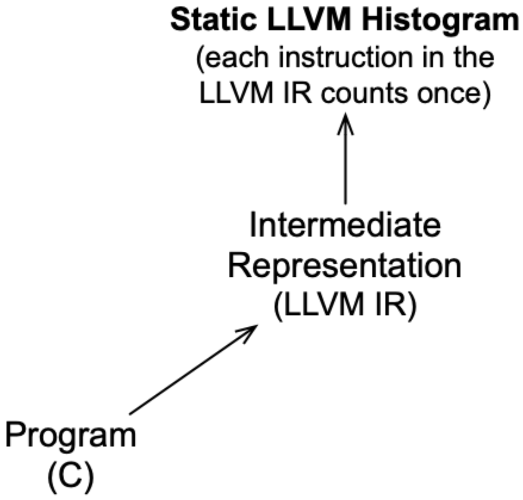
The adversary!

The Classification Methodology

Program
(C)

```
int main(int a, char** v)  {
    int f = 1;
    while (a > 1) {
        f *= a;
        --a;
    }
    return f;
}
```

The Classification Methodology



Static LLVM Histogram
(each instruction in the LLVM IR counts once)

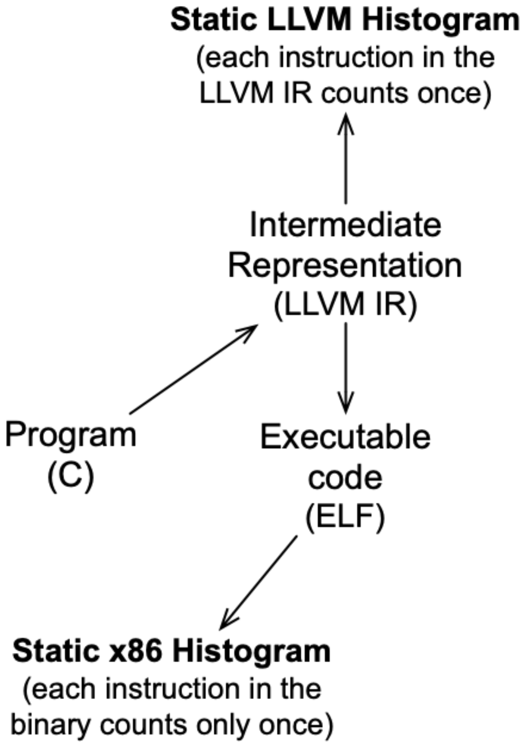
Intermediate Representation (LLVM IR)

Program (C)

```
define @main(%a, %v) {
bb:
  br label %bb5
bb5:
  %i2 = phi [%a:%bb], [%i13:%bb8]
  %i4 = phi [1:%bb], [%i11:%bb8]
  %i7 = icmp sgt %i2, 1
  br i1 %i7, %bb8, %bb14
bb8:
  %i11 = mul %i4, %i2
  %i13 = add %i2, -1
  br %bb5
bb14:
  ret %i4
}
```

br	phi	icmp	mul	add	ret
3	2	1	1	1	1

The Classification Methodology



Static LLVM Histogram
(each instruction in the LLVM IR counts once)

Intermediate Representation (LLVM IR)

Program (C)

Executable code (ELF)

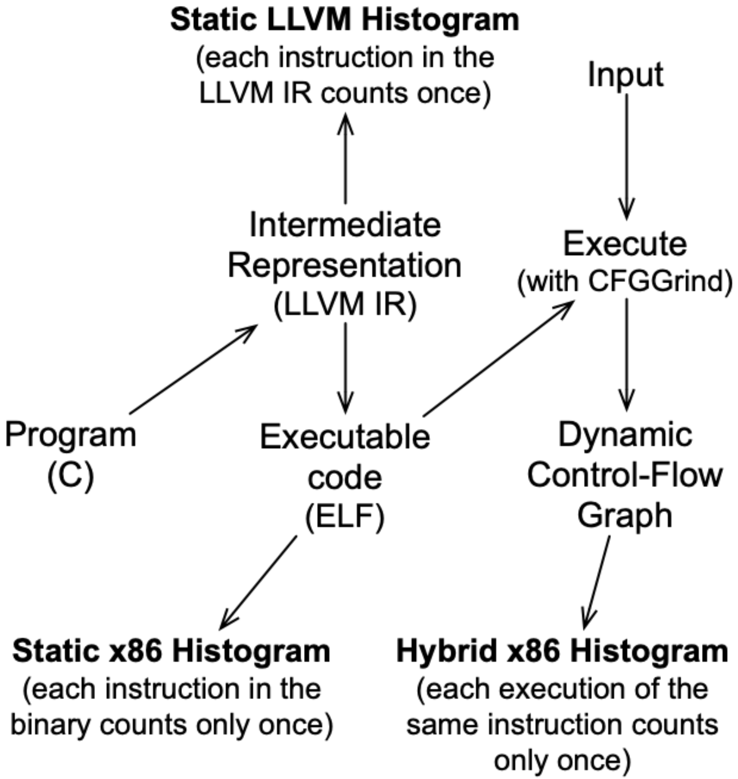
Static x86 Histogram
(each instruction in the binary counts only once)

The x86 Image

```
0000000000401110 <main>:
401110: mov $0x1,%eax
401115: cmp $0x2,%edi
401118: jl 40112d
<main+0x1d>
40111a: mov $0x1,%eax
40111f: nop
401120: imul %edi,%eax
401123: lea -0x1(%rdi),%ecx
401126: cmp $0x2,%edi
401129: mov %ecx,%edi
40112b: jg 401120
<main+0x10>
40112d: retq
40112e: xchg %ax,%ax
```

mov	cmp	jl	nop	imul	lea	jg	ret	xchg
3	2	1	1	1	1	1	1	1

The Classification Methodology



Static LLVM Histogram
(each instruction in the LLVM IR counts once)

Input

Intermediate Representation (LLVM IR)

Execute
(with CFGGrind)

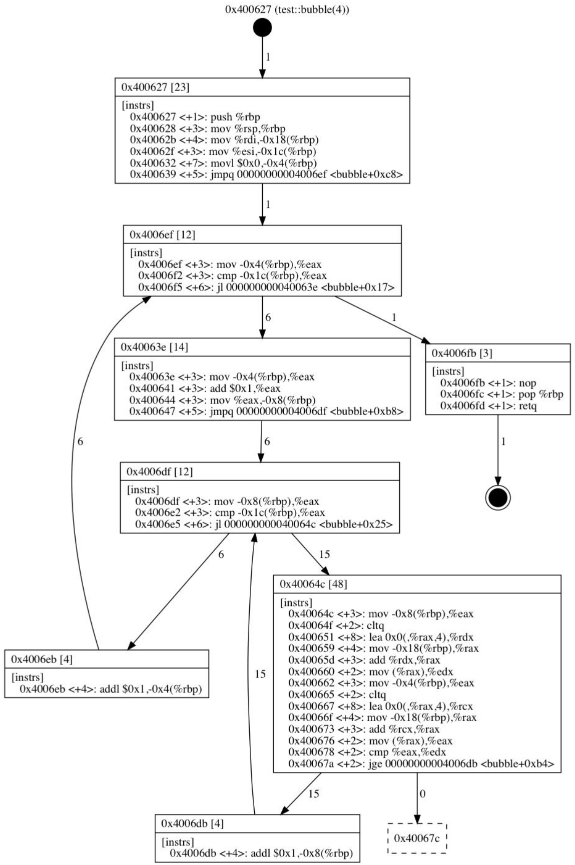
Program
(C)

Executable code
(ELF)

Dynamic Control-Flow Graph

Static x86 Histogram
(each instruction in the binary counts only once)

Hybrid x86 Histogram
(each execution of the same instruction counts only once)



0x400627 (test::bubble(4))

```
0x400627 [23]
[instrs]
  0x400627 <+1>: push %rbp
  0x400628 <+3>: mov %rsp,%rbp
  0x40062b <+4>: mov %rdi,-0x18(%rbp)
  0x40062f <+3>: mov %esi,-0x1c(%rbp)
  0x400632 <+7>: movl $0x0,-0x4(%rbp)
  0x400639 <+5>: jmpq 00000000004006ef <bubble+0xc8>
```

```
0x4006ef [12]
[instrs]
  0x4006ef <+3>: mov -0x4(%rbp),%eax
  0x4006f2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006f5 <+6>: jl 000000000040063e <bubble+0x17>
```

```
0x40063e [14]
[instrs]
  0x40063e <+3>: mov -0x4(%rbp),%eax
  0x400641 <+3>: add $0x1,%eax
  0x400644 <+3>: mov %eax,-0x8(%rbp)
  0x400647 <+5>: jmpq 00000000004006df <bubble+0xb8>
```

```
0x4006fb [3]
[instrs]
  0x4006fb <+1>: nop
  0x4006fc <+1>: pop %rbp
  0x4006fd <+1>: retq
```

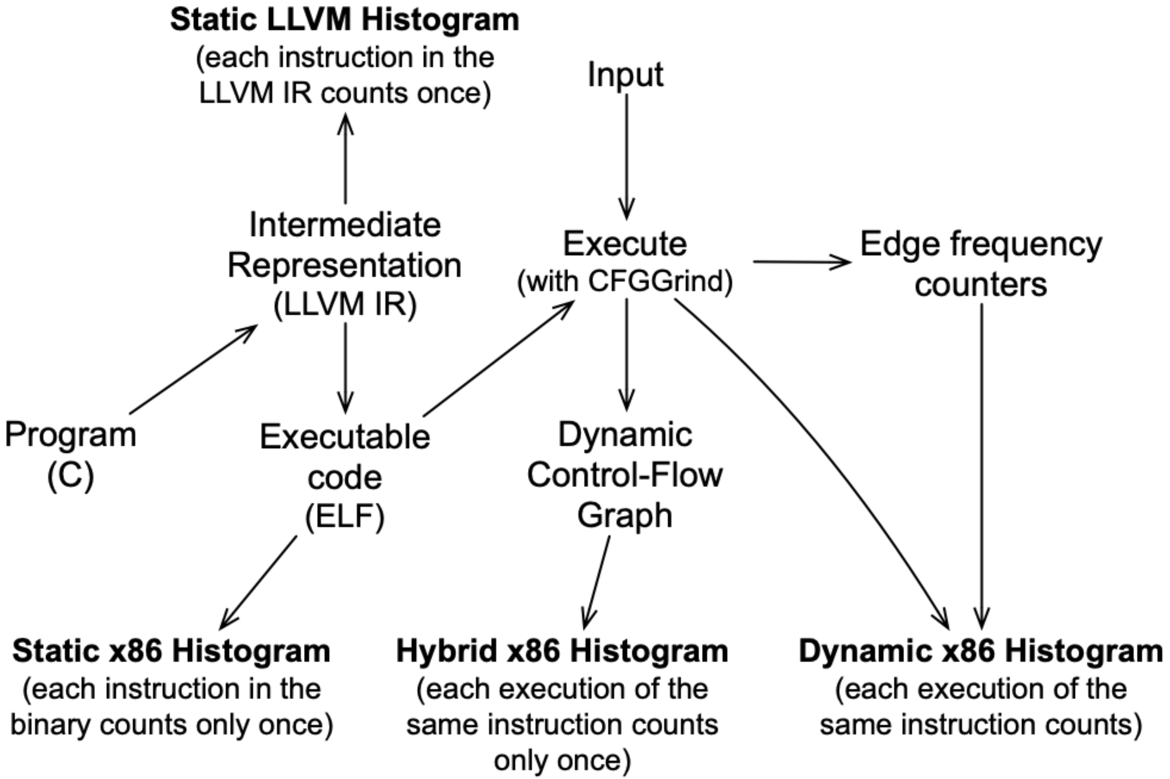
```
0x4006df [12]
[instrs]
  0x4006df <+3>: mov -0x8(%rbp),%eax
  0x4006e2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006e5 <+6>: jl 000000000040064c <bubble+0x25>
```

```
0x40064c [48]
[instrs]
  0x40064c <+3>: mov -0x8(%rbp),%eax
  0x40064f <+2>: cltq
  0x400651 <+8>: lea 0x0(,%rax,4),%rdx
  0x400659 <+4>: mov -0x18(%rbp),%rax
  0x40065d <+3>: add %rdx,%rax
  0x400660 <+2>: mov (%rax),%edx
  0x400662 <+3>: mov -0x4(%rbp),%eax
  0x400665 <+2>: cltq
  0x400667 <+8>: lea 0x0(,%rax,4),%rcx
  0x40066f <+4>: mov -0x18(%rbp),%rax
  0x400673 <+3>: add %rcx,%rax
  0x400676 <+2>: mov (%rax),%eax
  0x400678 <+2>: cmp %eax,%edx
  0x40067a <+2>: jge 00000000004006db <bubble+0xb4>
```

```
0x4006eb [4]
[instrs]
  0x4006eb <+4>: addl $0x1,-0x4(%rbp)
```

```
0x4006db [4]
[instrs]
  0x4006db <+4>: addl $0x1,-0x8(%rbp)
```

The Classification Methodology



Static LLVM Histogram
(each instruction in the LLVM IR counts once)

Input

Intermediate Representation (LLVM IR)

Execute (with CFGGrind)

Edge frequency counters

Program (C)

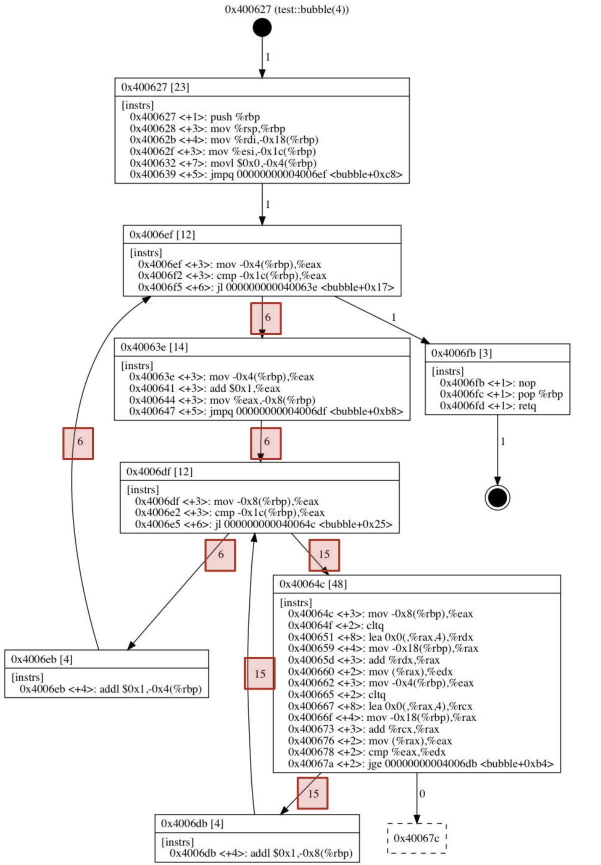
Executable code (ELF)

Dynamic Control-Flow Graph

Static x86 Histogram
(each instruction in the binary counts only once)

Hybrid x86 Histogram
(each execution of the same instruction counts only once)

Dynamic x86 Histogram
(each execution of the same instruction counts)



```
0x400627 [23]
[instrs]
  0x400627 <+1>: push %rbp
  0x400628 <+3>: mov %rsp,%rbp
  0x40062b <+4>: mov %rdi,-0x18(%rbp)
  0x40062f <+3>: mov %esi,-0x1c(%rbp)
  0x400632 <+7>: movl $0x0,-0x4(%rbp)
  0x400639 <+5>: jmpq 00000000004006ef <bubble+0xc8>
```

```
0x4006ef [12]
[instrs]
  0x4006ef <+3>: mov -0x4(%rbp),%eax
  0x4006f2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006f5 <+6>: jl 000000000040063e <bubble+0x17>
```

```
0x40063e [14]
[instrs]
  0x40063e <+3>: mov -0x4(%rbp),%eax
  0x400641 <+3>: add $0x1,%eax
  0x400644 <+3>: mov %eax,-0x8(%rbp)
  0x400647 <+5>: jmpq 00000000004006df <bubble+0xb8>
```

```
0x4006fb [3]
[instrs]
  0x4006fb <+1>: nop
  0x4006fc <+1>: pop %rbp
  0x4006fd <+1>: retq
```

```
0x4006df [12]
[instrs]
  0x4006df <+3>: mov -0x8(%rbp),%eax
  0x4006e2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006e5 <+6>: jl 000000000040064c <bubble+0x25>
```

```
0x4006eb [4]
[instrs]
  0x4006eb <+4>: addl $0x1,-0x4(%rbp)
```

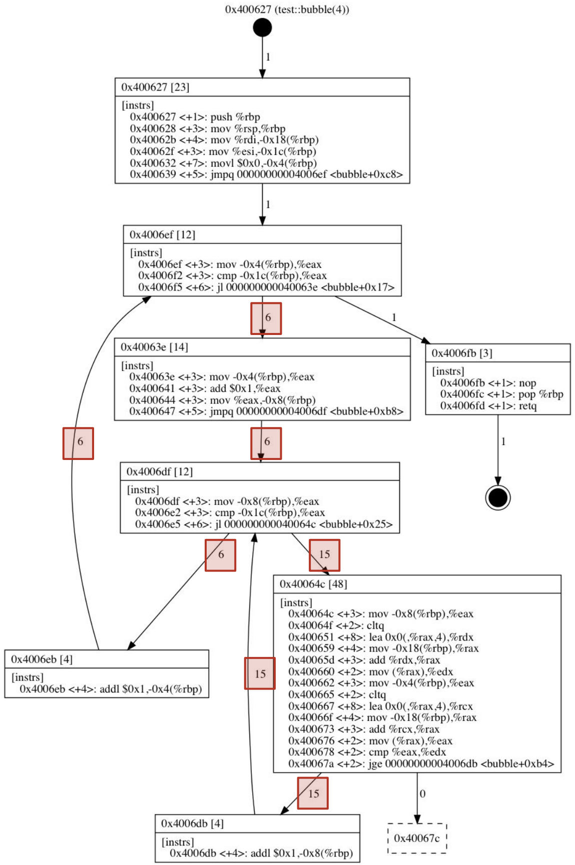
```
0x40064c [48]
[instrs]
  0x40064c <+3>: mov -0x8(%rbp),%eax
  0x40064f <+2>: cltq
  0x400651 <+8>: lea 0x0(,%rax,4),%rdx
  0x400659 <+4>: mov -0x18(%rbp),%rax
  0x40065d <+3>: add %rdx,%rax
  0x400660 <+2>: mov (%rax),%edx
  0x400662 <+3>: mov -0x4(%rbp),%eax
  0x400665 <+2>: cltq
  0x400667 <+8>: lea 0x0(,%rax,4),%rcx
  0x40066f <+4>: mov -0x18(%rbp),%rax
  0x400673 <+3>: add %rcx,%rax
  0x400676 <+2>: mov (%rax),%eax
  0x400678 <+2>: cmp %eax,%edx
  0x40067a <+2>: jge 00000000004006db <bubble+0xb4>
```

```
0x4006db [4]
[instrs]
  0x4006db <+4>: addl $0x1,-0x8(%rbp)
```

```
0x40067c
```

Implementing Dynamic/Hybrid Classifiers

- CFGGrind



```
0x400627 [23]
[instrs]
  0x400627 <+1>: push %rbp
  0x400628 <+3>: mov %rsp,%rbp
  0x40062b <+4>: mov %rdi,-0x18(%rbp)
  0x40062f <+3>: mov %esi,-0x1c(%rbp)
  0x400632 <+7>: movl $0x0,-0x4(%rbp)
  0x400639 <+5>: jmpq 00000000004006ef <bubble+0xc8>
```

```
0x4006ef [12]
[instrs]
  0x4006ef <+3>: mov -0x4(%rbp),%eax
  0x4006f2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006f5 <+6>: jl 000000000040063e <bubble+0x17>
```

```
0x40063e [14]
[instrs]
  0x40063e <+3>: mov -0x4(%rbp),%eax
  0x400641 <+3>: add $0x1,%eax
  0x400644 <+3>: mov %eax,-0x8(%rbp)
  0x400647 <+5>: jmpq 00000000004006df <bubble+0xb8>
```

```
0x4006fb [3]
[instrs]
  0x4006fb <+1>: nop
  0x4006fc <+1>: pop %rbp
  0x4006fd <+1>: retq
```

```
0x4006df [12]
[instrs]
  0x4006df <+3>: mov -0x8(%rbp),%eax
  0x4006e2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006e5 <+6>: jl 000000000040064c <bubble+0x25>
```

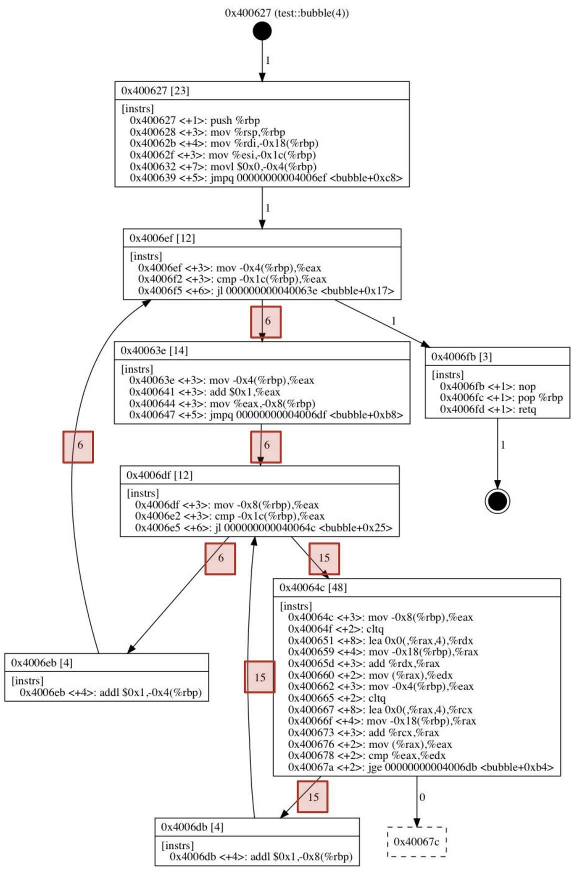
```
0x40064c [48]
[instrs]
  0x40064c <+3>: mov -0x8(%rbp),%eax
  0x40064f <+2>: cltq
  0x400651 <+8>: lea 0x0(,%rax,4),%rdx
  0x400659 <+4>: mov -0x18(%rbp),%rax
  0x40065d <+3>: add %rdx,%rax
  0x400660 <+2>: mov (%rax),%edx
  0x400662 <+3>: mov -0x4(%rbp),%eax
  0x400665 <+2>: cltq
  0x400667 <+8>: lea 0x0(,%rax,4),%rcx
  0x40066f <+4>: mov -0x18(%rbp),%rax
  0x400673 <+3>: add %rcx,%rax
  0x400676 <+2>: mov (%rax),%eax
  0x400678 <+2>: cmp %eax,%edx
  0x40067a <+2>: jge 00000000004006db <bubble+0xb4>
```

```
0x4006eb [4]
[instrs]
  0x4006eb <+4>: addl $0x1,-0x4(%rbp)
```

```
0x4006db [4]
[instrs]
  0x4006db <+4>: addl $0x1,-0x8(%rbp)
```

Implementing Dynamic/Hybrid Classifiers

- CFGGrind
- Valgrind plugin



0x400627 (test::bubble(4))

```
0x400627 [23]
[instrs]
  0x400627 <+1>: push %rbp
  0x400628 <+3>: mov %rsp,%rbp
  0x40062b <+4>: mov %rdi,-0x18(%rbp)
  0x40062f <+3>: mov %esi,-0x1c(%rbp)
  0x400632 <+7>: movl $0x0,-0x4(%rbp)
  0x400639 <+5>: jmpq 00000000004006ef <bubble+0xc8>
```

```
0x4006ef [12]
[instrs]
  0x4006ef <+3>: mov -0x4(%rbp),%eax
  0x4006f2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006f5 <+6>: jl 000000000040063e <bubble+0x17>
```

```
0x40063e [14]
[instrs]
  0x40063e <+3>: mov -0x4(%rbp),%eax
  0x400641 <+3>: add $0x1,%eax
  0x400644 <+3>: mov %eax,-0x8(%rbp)
  0x400647 <+5>: jmpq 00000000004006df <bubble+0xb8>
```

```
0x4006fb [3]
[instrs]
  0x4006fb <+1>: nop
  0x4006fc <+1>: pop %rbp
  0x4006fd <+1>: retq
```

```
0x4006df [12]
[instrs]
  0x4006df <+3>: mov -0x8(%rbp),%eax
  0x4006e2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006e5 <+6>: jl 000000000040064c <bubble+0x25>
```

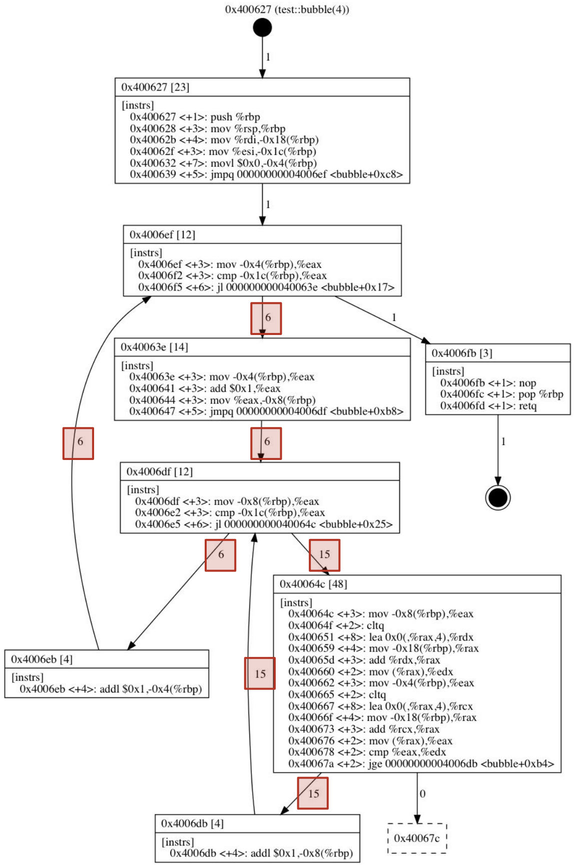
```
0x40064c [48]
[instrs]
  0x40064c <+3>: mov -0x8(%rbp),%eax
  0x40064f <+2>: cltq
  0x400651 <+8>: lea 0x0(,%rax,4),%rdx
  0x400659 <+4>: mov -0x18(%rbp),%rax
  0x40065d <+3>: add %rdx,%rax
  0x400660 <+2>: mov (%rax),%edx
  0x400662 <+3>: mov -0x4(%rbp),%eax
  0x400665 <+2>: cltq
  0x400667 <+8>: lea 0x0(,%rax,4),%rcx
  0x40066f <+4>: mov -0x18(%rbp),%rax
  0x400673 <+3>: add %rcx,%rax
  0x400676 <+2>: mov (%rax),%eax
  0x400678 <+2>: cmp %eax,%edx
  0x40067a <+2>: jge 00000000004006db <bubble+0xb4>
```

```
0x4006eb [4]
[instrs]
  0x4006eb <+4>: addl $0x1,-0x4(%rbp)
```

```
0x4006db [4]
[instrs]
  0x4006db <+4>: addl $0x1,-0x8(%rbp)
```

Implementing Dynamic/Hybrid Classifiers

- CFGGrind
- Valgrind plugin
- https://github.com/rimsa/CFGgrind



0x400627 (test::bubble(4))

```
0x400627 [23]
[instrs]
  0x400627 <+1>: push %rbp
  0x400628 <+3>: mov %rsp,%rbp
  0x40062b <+4>: mov %rdi,-0x18(%rbp)
  0x40062f <+3>: mov %esi,-0x1c(%rbp)
  0x400632 <+7>: movl $0x0,-0x4(%rbp)
  0x400639 <+5>: jmpq 00000000004006ef <bubble+0xc8>
```

```
0x4006ef [12]
[instrs]
  0x4006ef <+3>: mov -0x4(%rbp),%eax
  0x4006f2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006f5 <+6>: jl 000000000040063e <bubble+0x17>
```

```
0x40063e [14]
[instrs]
  0x40063e <+3>: mov -0x4(%rbp),%eax
  0x400641 <+3>: add $0x1,%eax
  0x400644 <+3>: mov %eax,-0x8(%rbp)
  0x400647 <+5>: jmpq 00000000004006df <bubble+0xb8>
```

```
0x4006fb [3]
[instrs]
  0x4006fb <+1>: nop
  0x4006fc <+1>: pop %rbp
  0x4006fd <+1>: retq
```

```
0x4006df [12]
[instrs]
  0x4006df <+3>: mov -0x8(%rbp),%eax
  0x4006e2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006e5 <+6>: jl 000000000040064c <bubble+0x25>
```

```
0x40064c [48]
[instrs]
  0x40064c <+3>: mov -0x8(%rbp),%eax
  0x40064f <+2>: cltq
  0x400651 <+8>: lea 0x0(,%rax,4),%rdx
  0x400659 <+4>: mov -0x18(%rbp),%rax
  0x40065d <+3>: add %rdx,%rax
  0x400660 <+2>: mov (%rax),%edx
  0x400662 <+3>: mov -0x4(%rbp),%eax
  0x400665 <+2>: cltq
  0x400667 <+8>: lea 0x0(,%rax,4),%rcx
  0x40066f <+4>: mov -0x18(%rbp),%rax
  0x400673 <+3>: add %rcx,%rax
  0x400676 <+2>: mov (%rax),%eax
  0x400678 <+2>: cmp %eax,%edx
  0x40067a <+2>: jge 00000000004006db <bubble+0xb4>
```

```
0x4006eb [4]
[instrs]
  0x4006eb <+4>: addl $0x1,-0x4(%rbp)
```

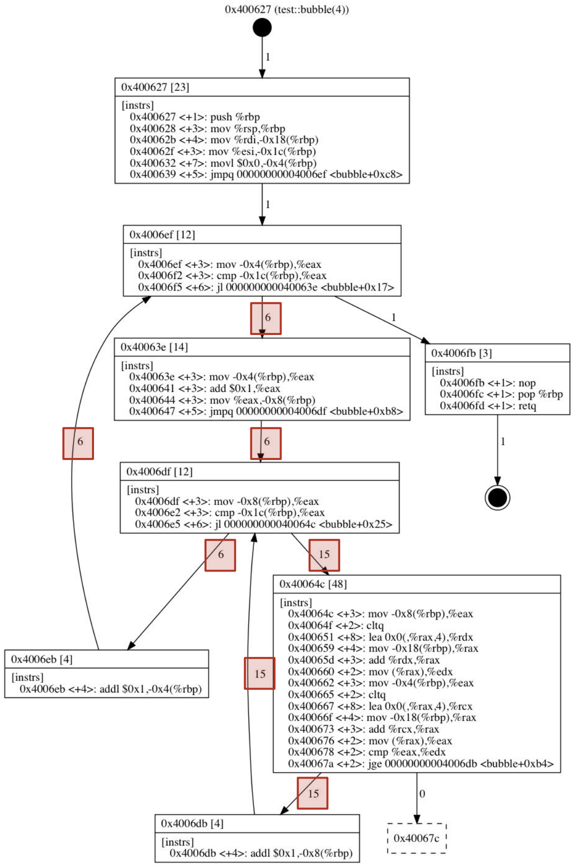
```
0x4006db [4]
[instrs]
  0x4006db <+4>: addl $0x1,-0x8(%rbp)
```

```
0x40067c
```

Implementing Dynamic/Hybrid Classifiers

- CFGGrind
- Valgrind plugin
- https://github.com/rimsa/CFGgrind

- Gradual



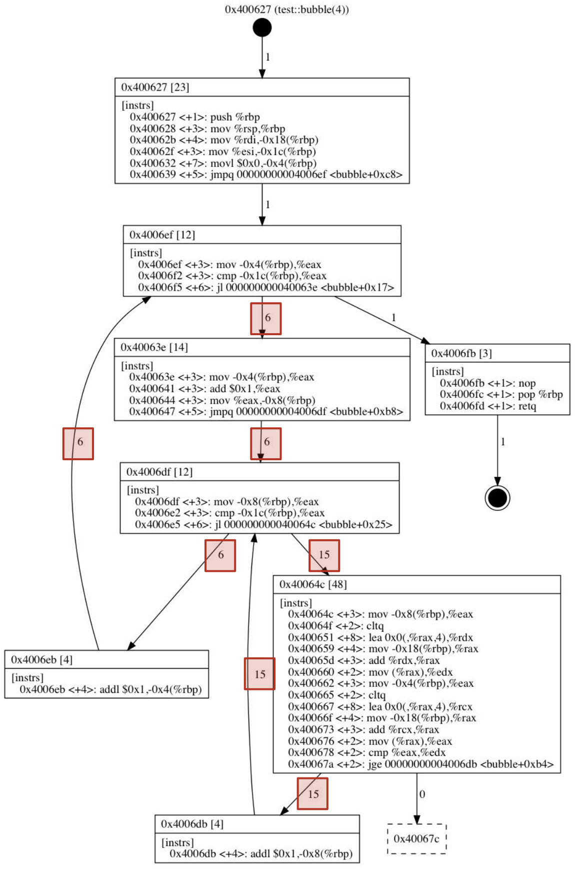
```
0x400627 (test::bubble(4))

0x400627 [23]
[instrs]
  0x400627 <+1>: push %rbp
  0x400628 <+3>: mov %rsp,%rbp
  0x40062b <+4>: mov %rdi,-0x18(%rbp)
  0x40062f <+3>: mov %esi,-0x1c(%rbp)
  0x400632 <+7>: movl $0x0,-0x4(%rbp)
  0x400639 <+5>: jmpq 00000000004006ef <bubble+0xc8>

0x4006ef [12]
[instrs]
  0x4006ef <+3>: mov -0x4(%rbp),%eax
  0x4006f2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006f5 <+6>: jl 000000000040063e <bubble+0x17>

0x40063e [14]
[instrs]
  0x40063e <+3>: mov -0x4(%rbp),%eax
  0x400641 <+3>: add $0x1,%eax
  0x400644 <+3>: mov %eax,-0x8(%rbp)
  0x400647 <+5>: jmpq 00000000004006df <bubble+0xb8>

0x4006fb [3]
[instrs]
  0x4006fb <+1>: nop
  0x4006fc <+1>: pop %rbp
  0x4006fd <+1>: retq

0x4006df [12]
[instrs]
  0x4006df <+3>: mov -0x8(%rbp),%eax
  0x4006e2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006e5 <+6>: jl 000000000040064c <bubble+0x25>

0x40064c [48]
[instrs]
  0x40064c <+3>: mov -0x8(%rbp),%eax
  0x40064f <+2>: cltq
  0x400651 <+8>: lea 0x0(,%rax,4),%rdx
  0x400659 <+4>: mov -0x18(%rbp),%rax
  0x40065d <+3>: add %rdx,%rax
  0x400660 <+2>: mov (%rax),%edx
  0x400662 <+3>: mov -0x4(%rbp),%eax
  0x400665 <+2>: cltq
  0x400667 <+8>: lea 0x0(,%rax,4),%rcx
  0x40066f <+4>: mov -0x18(%rbp),%rax
  0x400673 <+3>: add %rcx,%rax
  0x400676 <+2>: mov (%rax),%eax
  0x400678 <+2>: cmp %eax,%edx
  0x40067a <+2>: jge 00000000004006db <bubble+0xb4>

0x4006eb [4]
[instrs]
  0x4006eb <+4>: addl $0x1,-0x4(%rbp)

0x4006db [4]
[instrs]
  0x4006db <+4>: addl $0x1,-0x8(%rbp)

0x40067c
```

Implementing Dynamic/Hybrid Classifiers

- CFGGrind
- Valgrind plugin
- https://github.com/rimsa/CFGgrind

- Gradual
- Distinguish library functions

  (invisible instructions)



```
0x400627 [23]
[instrs]
  0x400627 <+1>: push %rbp
  0x400628 <+3>: mov %rsp,%rbp
  0x40062b <+4>: mov %rdi,-0x18(%rbp)
  0x40062f <+3>: mov %esi,-0x1c(%rbp)
  0x400632 <+7>: movl $0x0,-0x4(%rbp)
  0x400639 <+5>: jmpq 00000000004006ef <bubble+0xc8>
```

```
0x4006ef [12]
[instrs]
  0x4006ef <+3>: mov -0x4(%rbp),%eax
  0x4006f2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006f5 <+6>: jl 000000000040063e <bubble+0x17>
```

```
0x40063e [14]
[instrs]
  0x40063e <+3>: mov -0x4(%rbp),%eax
  0x400641 <+3>: add $0x1,%eax
  0x400644 <+3>: mov %eax,-0x8(%rbp)
  0x400647 <+5>: jmpq 00000000004006df <bubble+0xb8>
```

```
0x4006fb [3]
[instrs]
  0x4006fb <+1>: nop
  0x4006fc <+1>: pop %rbp
  0x4006fd <+1>: retq
```

```
0x4006df [12]
[instrs]
  0x4006df <+3>: mov -0x8(%rbp),%eax
  0x4006e2 <+3>: cmp -0x1c(%rbp),%eax
  0x4006e5 <+6>: jl 000000000040064c <bubble+0x25>
```

```
0x4006eb [4]
[instrs]
  0x4006eb <+4>: addl $0x1,-0x4(%rbp)
```

```
0x40064c [48]
[instrs]
  0x40064c <+3>: mov -0x8(%rbp),%eax
  0x40064f <+2>: cltq
  0x400651 <+8>: lea 0x0(,%rax,4),%rdx
  0x400659 <+4>: mov -0x18(%rbp),%rax
  0x40065d <+3>: add %rdx,%rax
  0x400660 <+2>: mov (%rax),%edx
  0x400662 <+3>: mov -0x4(%rbp),%eax
  0x400665 <+2>: cltq
  0x400667 <+8>: lea 0x0(,%rax,4),%rcx
  0x40066f <+4>: mov -0x18(%rbp),%rax
  0x400673 <+3>: add %rcx,%rax
  0x400676 <+2>: mov (%rax),%eax
  0x400678 <+2>: cmp %eax,%edx
  0x40067a <+2>: jge 00000000004006db <bubble+0xb4>
```

```
0x4006db [4]
[instrs]
  0x4006db <+4>: addl $0x1,-0x8(%rbp)
```

Datasets

- Programming Marathon Problems

Datasets

- Programming Marathon Problems
- CodeNet

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, and Ulrich Finkler. 2021. **Project CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks**. CoRR abs/2105.12655 (2021). arXiv

Datasets

- Programming Marathon Problems
- CodeNet (700 problems with 500 solutions each)

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, and Ulrich Finkler. 2021. **Project CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks**. CoRR abs/2105.12655 (2021). arXiv

Datasets

- Programming Marathon Problems
- CodeNet (700 problems with 500 solutions each)
- Constraint: clang `10.0 + o-llvm`

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, and Ulrich Finkler. 2021. **Project CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks**. CoRR abs/2105.12655 (2021). arXiv

Datasets

- Programming Marathon Problems
- CodeNet (700 problems with 500 solutions each)
- Constraint: clang `10.0 + o-llvm`
- At least one input, plus "`exit 0`".

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, and Ulrich Finkler. 2021. **Project CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks**. CoRR abs/2105.12655 (2021). arXiv

Adversaries (Game-1 Classification)

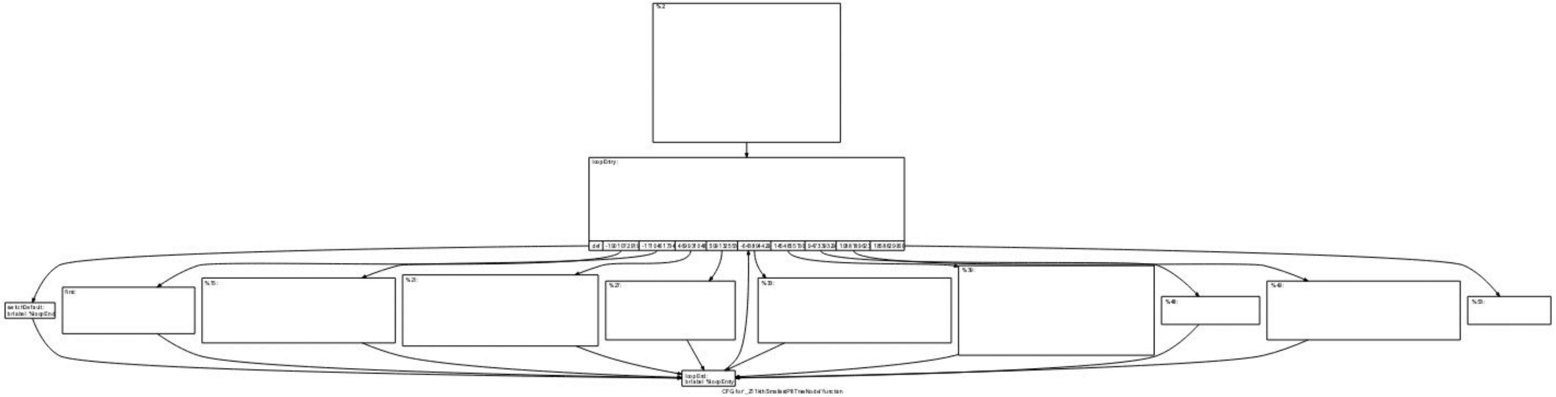
- Obfuscation techniques (via `o-llvm`)

Adversaries (Game-1 Classification)

- Obfuscation techniques (via `o-llvm`)
  - Instruction substitution
  - Bogus control flow
  - Control-flow flattening

Adversaries (Game-1 Classification)

- Obfuscation techniques (via `o-llvm`)
    - Instruction substitution
    - Bogus control flow
    - Control-flow flattening



Adversaries (Game-1 Classification)

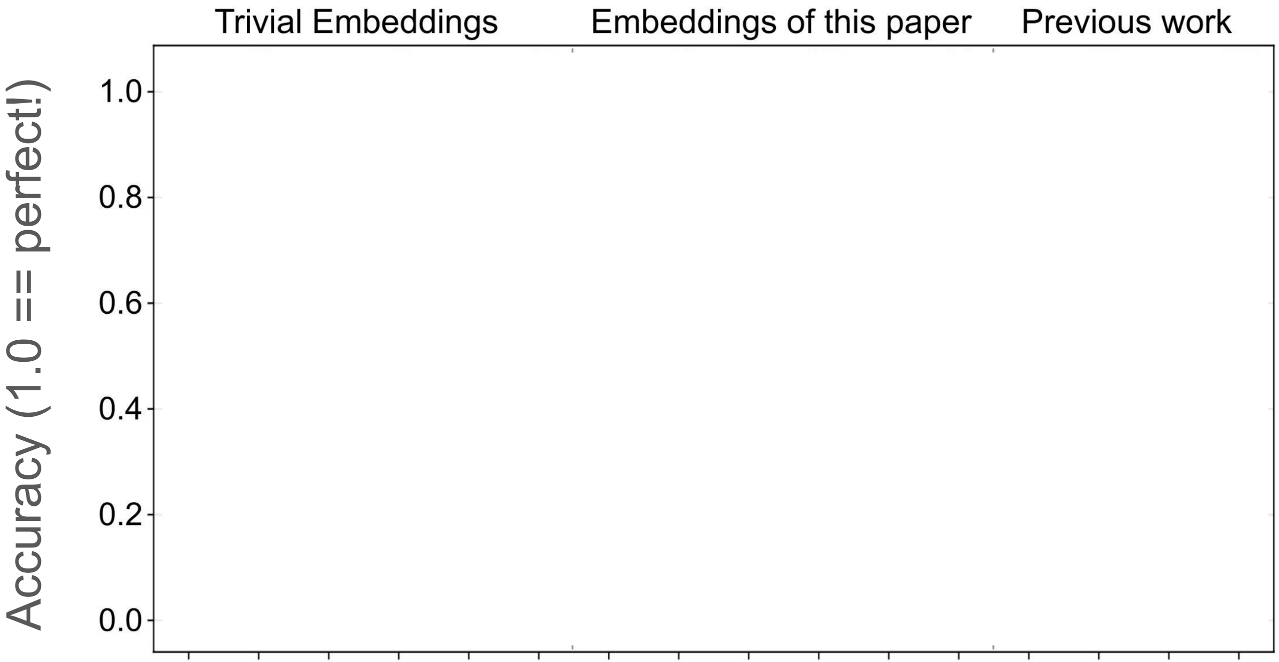
- Obfuscation techniques (via `o-llvm`)
- Optimizations (`clang -O1`, `-O2`, `-O3`)

Xiaolei Ren, Michael Ho, Jiang Ming, Yu Lei, Li Li: **Unleashing the hidden power of compiler optimization on binary code difference: an empirical study**. PLDI 2021: 142-157

RQ1: The Strawman Hypothesis

What is the accuracy of the classifiers evaluated in this paper compared to either trivial classifiers or to state-of-the-art techniques discussed in the literature?

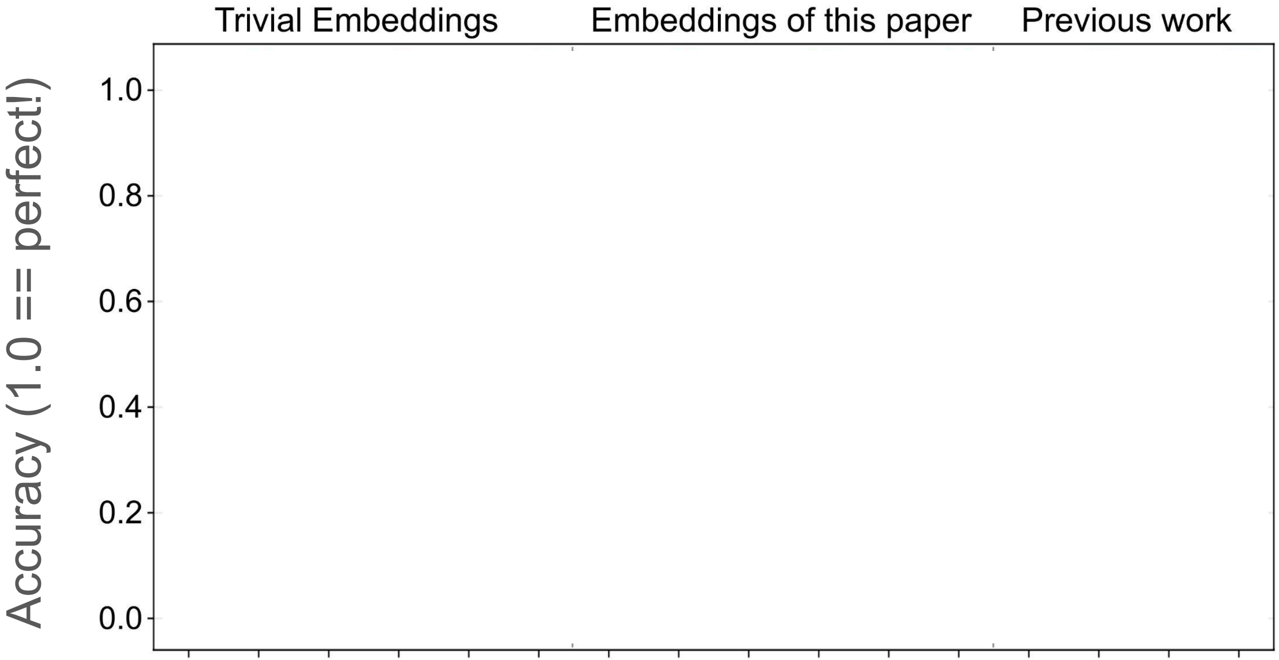
RQ1: The Strawman Hypothesis



RQ1: The Strawman Hypothesis

Trivial embeddings:

Count instructions

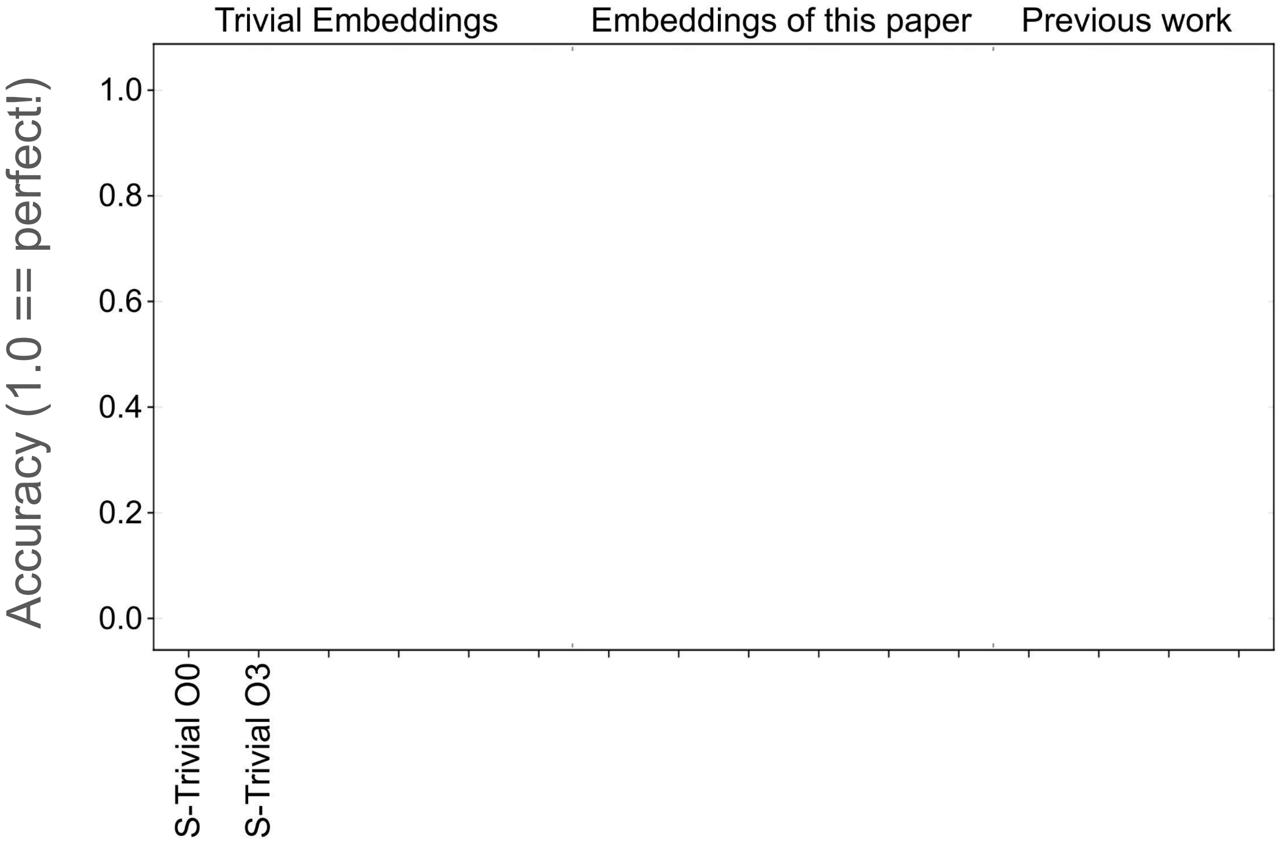


RQ1: The Strawman Hypothesis

Trivial embeddings:

Count instructions

- Static (LLVM)

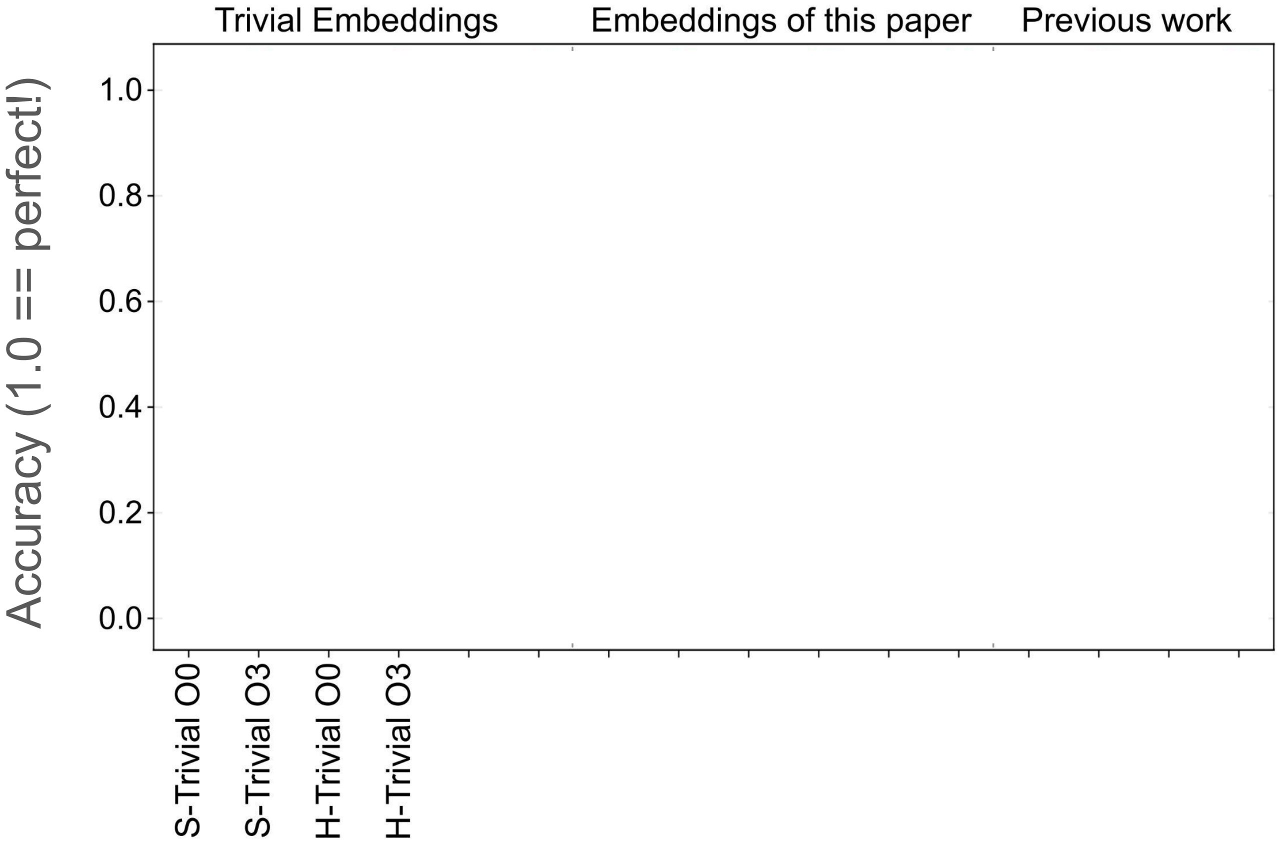


RQ1: The Strawman Hypothesis

Trivial embeddings:

Count instructions

- Static (LLVM)
- Hybrid

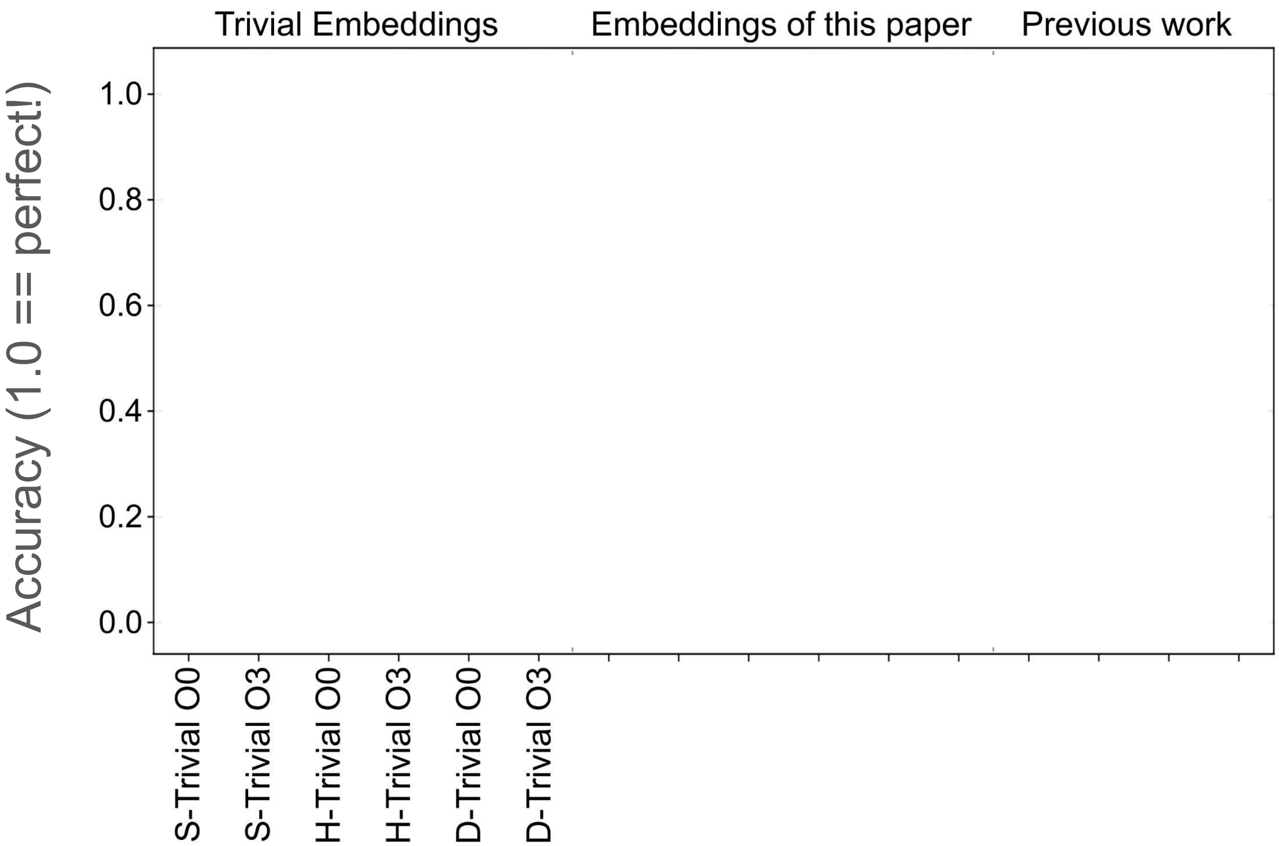


RQ1: The Strawman Hypothesis

Trivial embeddings:

Count instructions

- Static (LLVM)
- Hybrid
- Dynamic



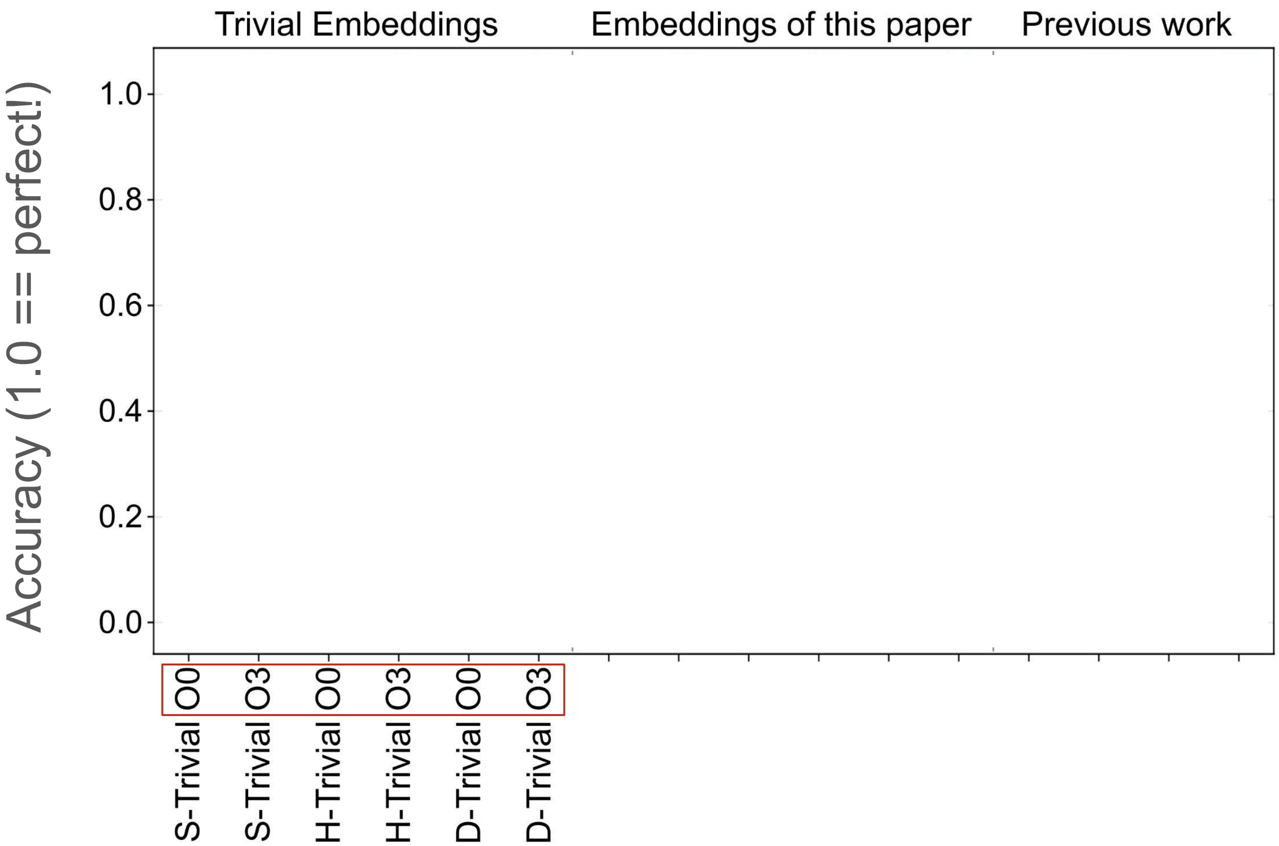
RQ1: The Strawman Hypothesis

DCC

Trivial embeddings:

Count instructions

- Static (LLVM)
- Hybrid
- Dynamic

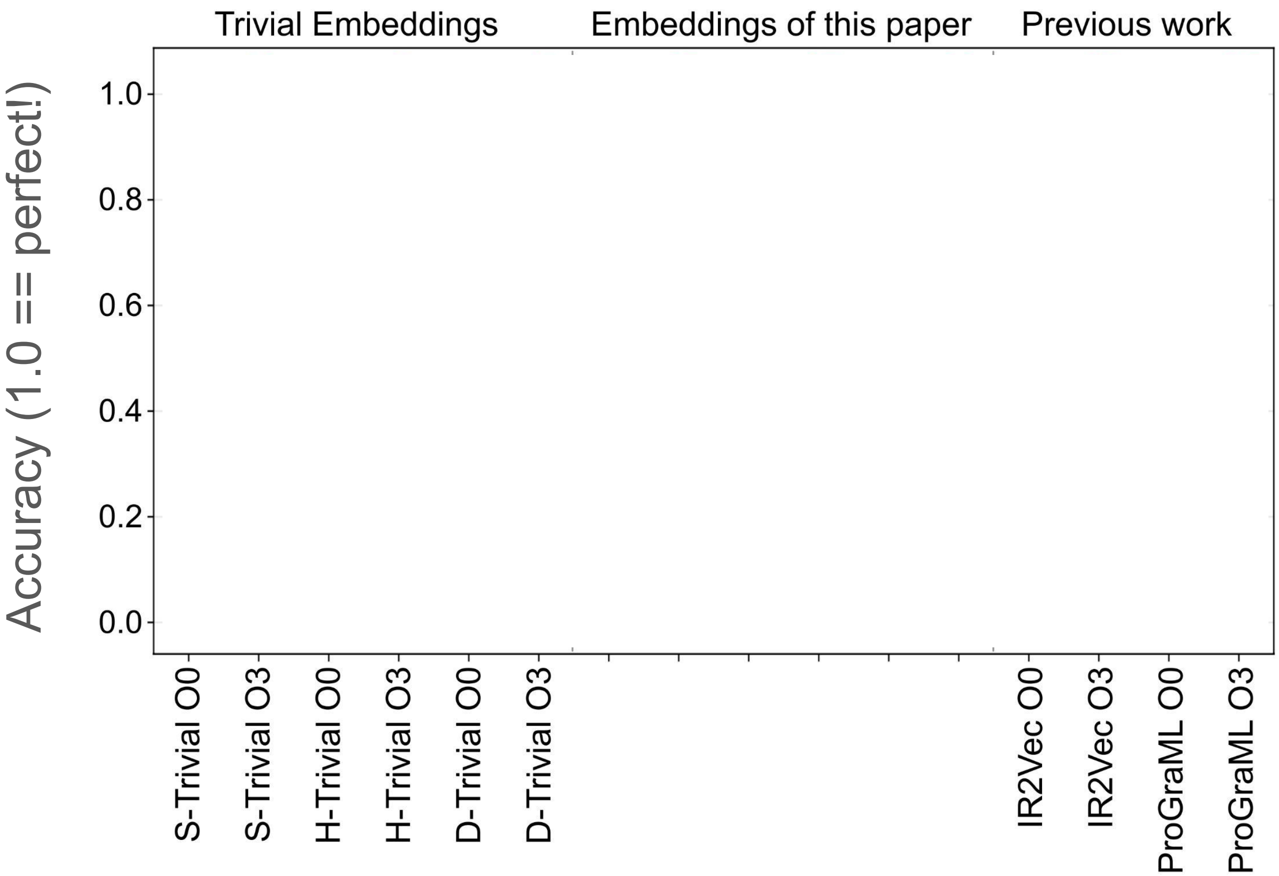


RQ1: The Strawman Hypothesis

Previous work:

beyond histograms

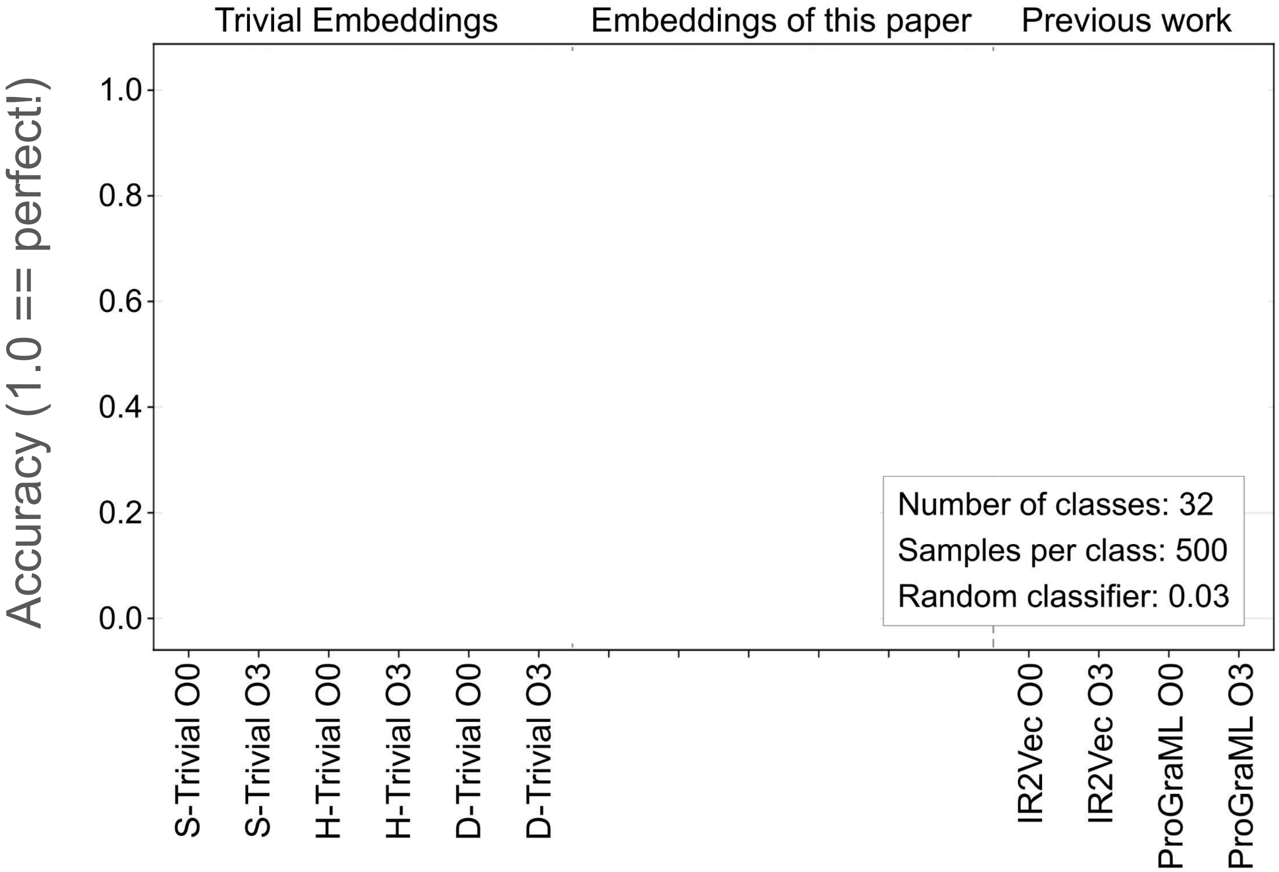
- IR2Vec
- ProGraML



RQ1: The Strawman Hypothesis

Baseline:

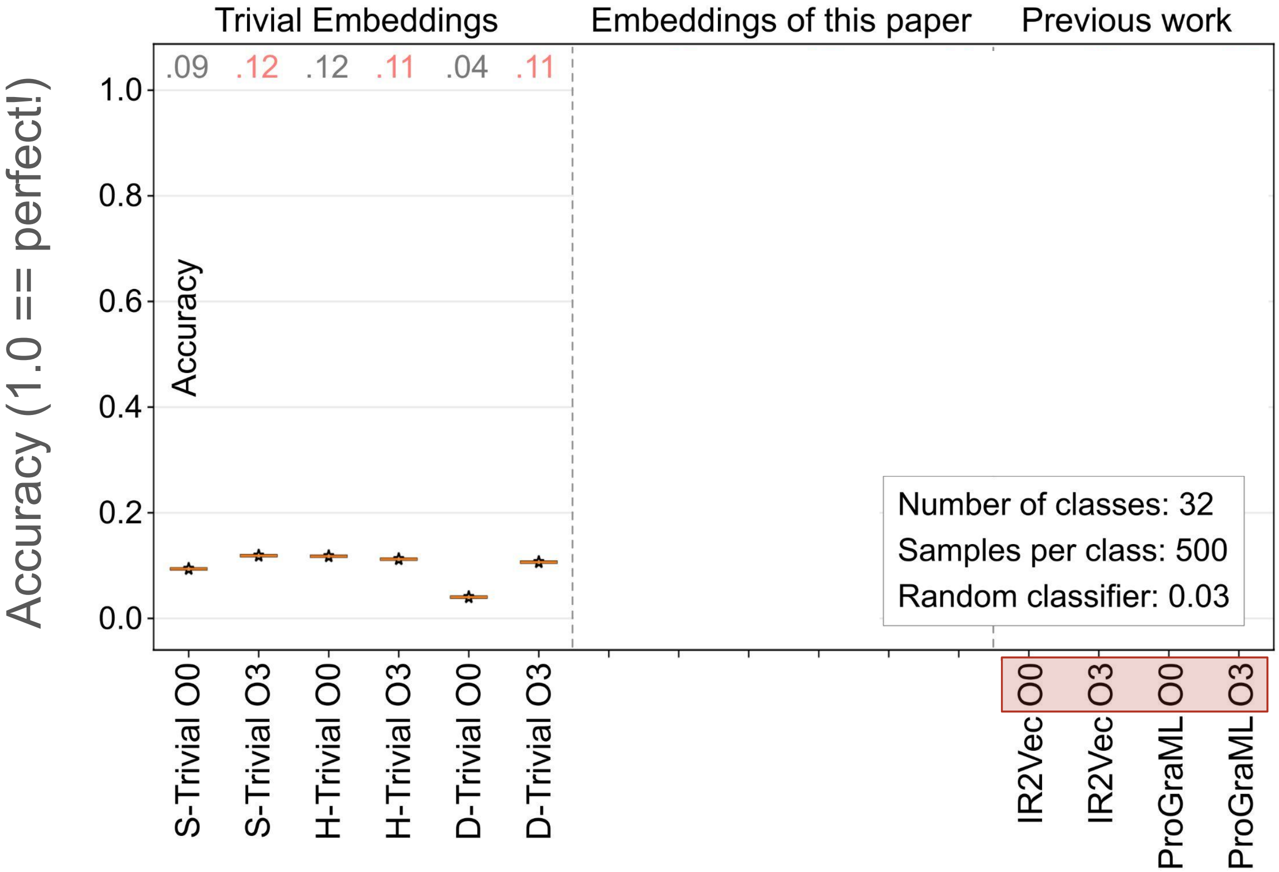
- 32 Problems
- 500 solutions



RQ1: The Strawman Hypothesis

Trivial classifiers

- 10-15%



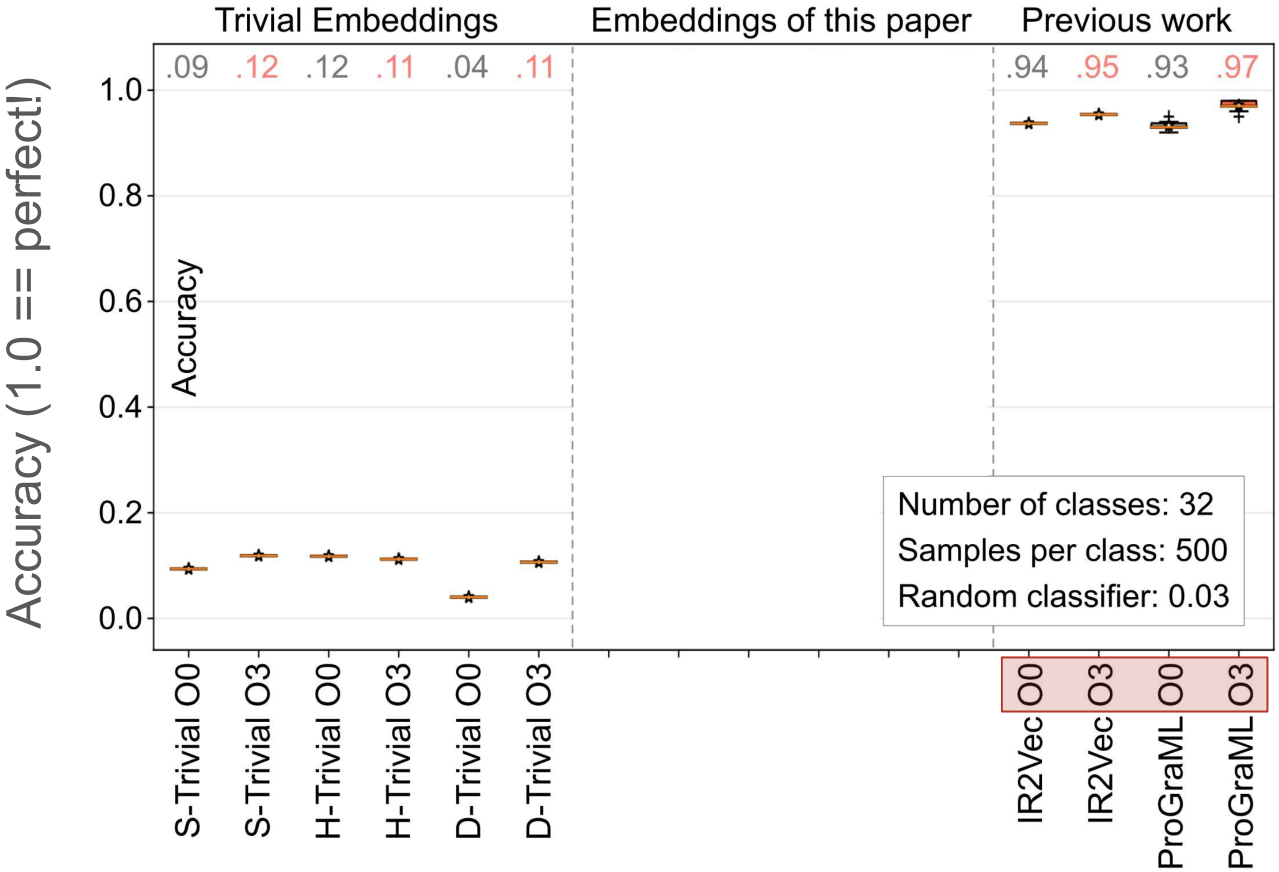
RQ1: The Strawman Hypothesis

Trivial classifiers

- 10-15%

Previous work

- +- 95%



RQ1: The Strawman Hypothesis

Trivial classifiers

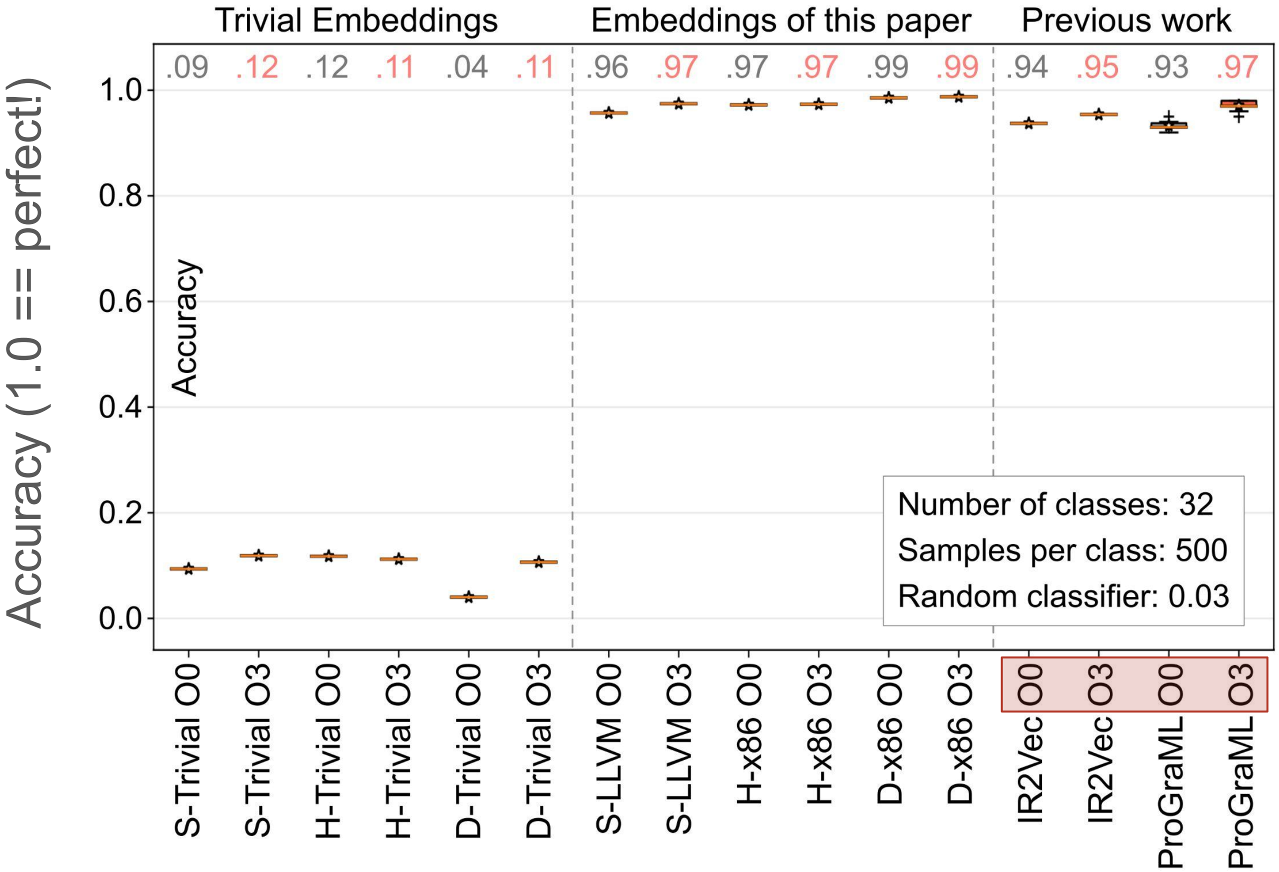
- 10-15%

Previous work

- +- 95%

This paper

- > 95%



Precision of different histogram-based classifiers

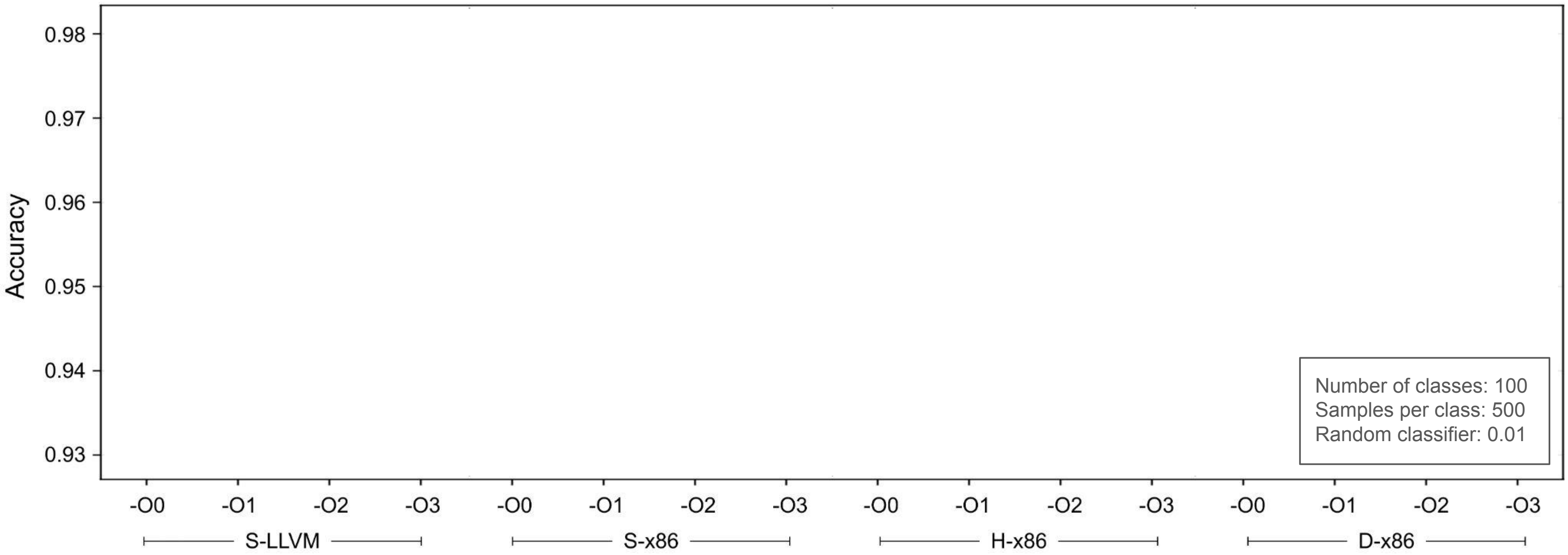
Does the optimization level matter?

Precision of different histogram-based classifiers

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Does the optimization level matter?

Precision of different histogram-based classifiers



Number of classes: 100
Samples per class: 500
Random classifier: 0.01

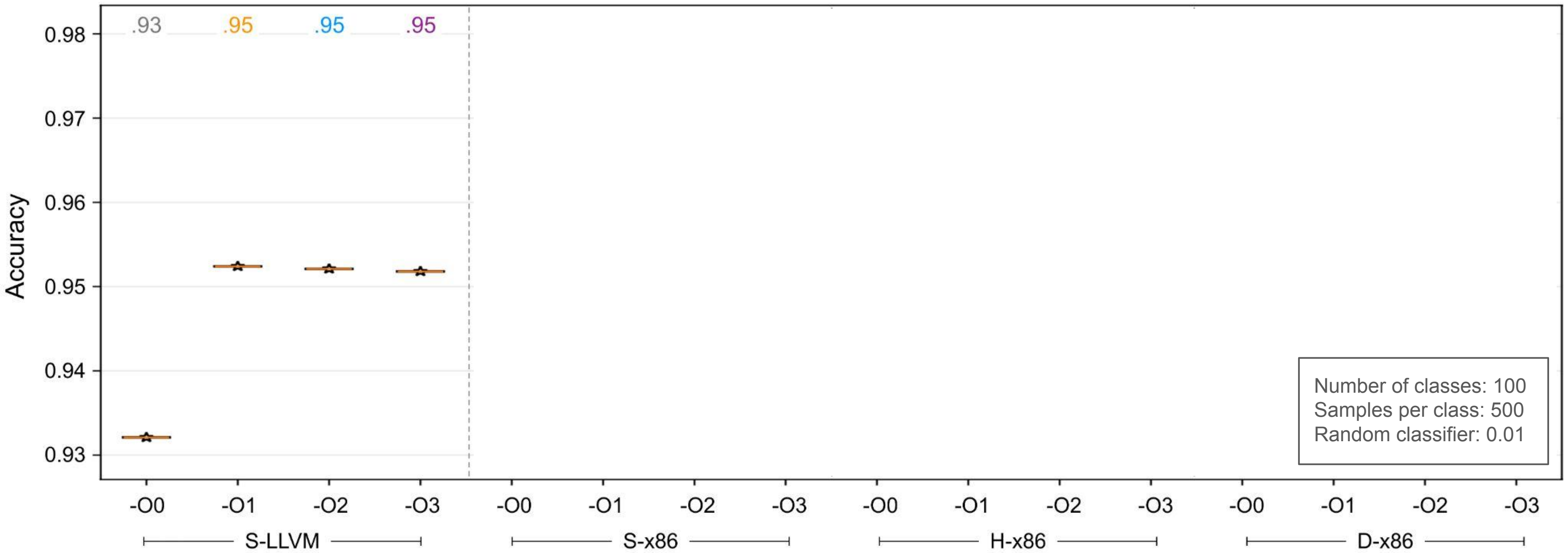
Static
(LLVM IR)

Static
(x86)

Hybrid
(x86)

Dynamic
(x86)

Precision of different histogram-based classifiers



Number of classes: 100
Samples per class: 500
Random classifier: 0.01

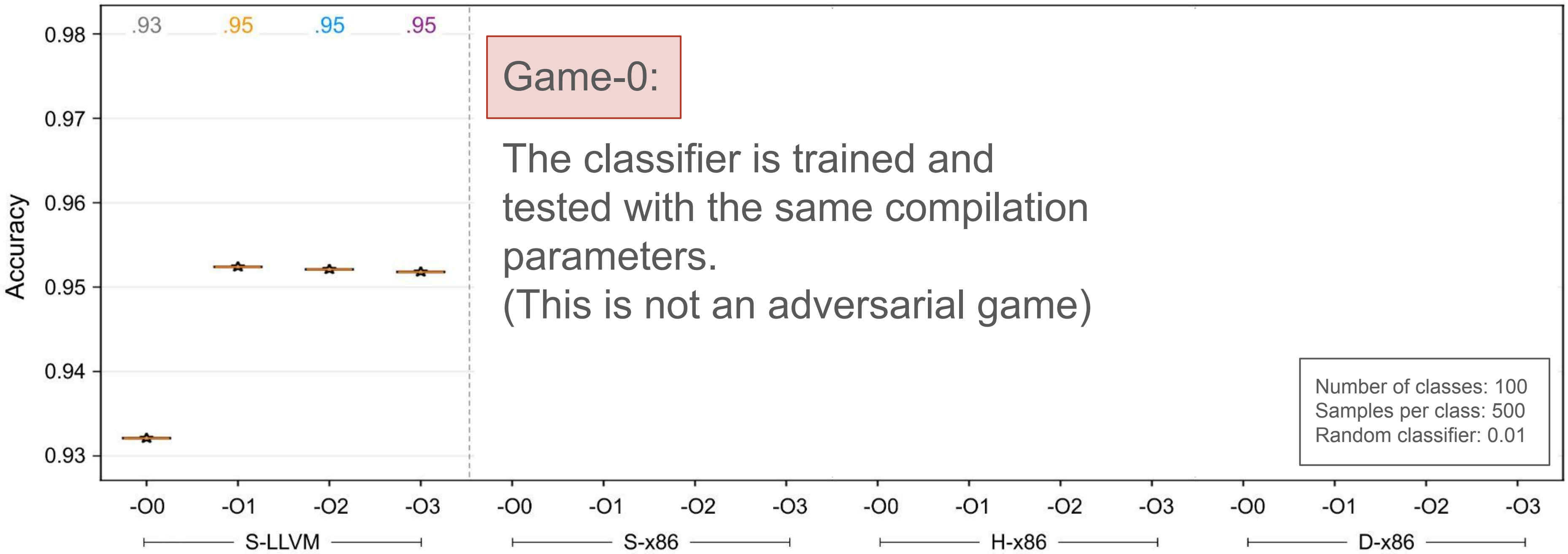
Static
(LLVM IR)

Static
(x86)

Hybrid
(x86)

Dynamic
(x86)

Precision of different histogram-based classifiers



Game-0:

The classifier is trained and tested with the same compilation parameters.
(This is not an adversarial game)

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Static
(LLVM IR)

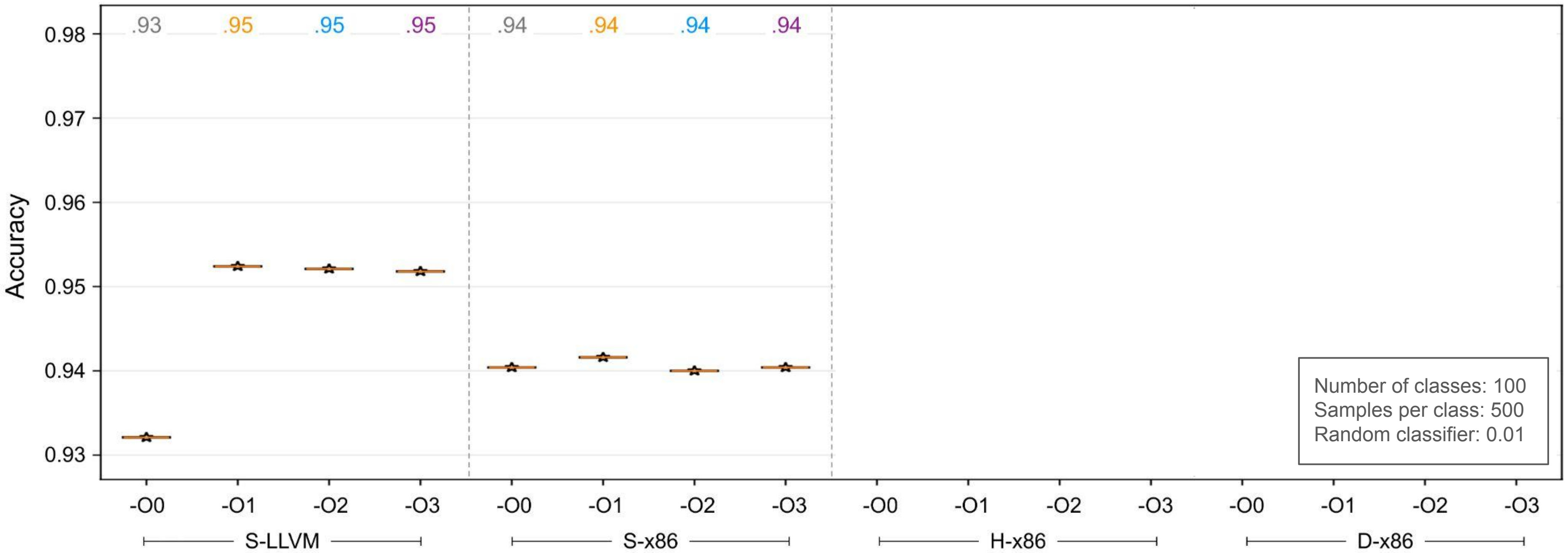
Static
(x86)

Hybrid
(x86)

Dynamic
(x86)

Precision of different histogram-based classifiers

DCC



Number of classes: 100
Samples per class: 500
Random classifier: 0.01

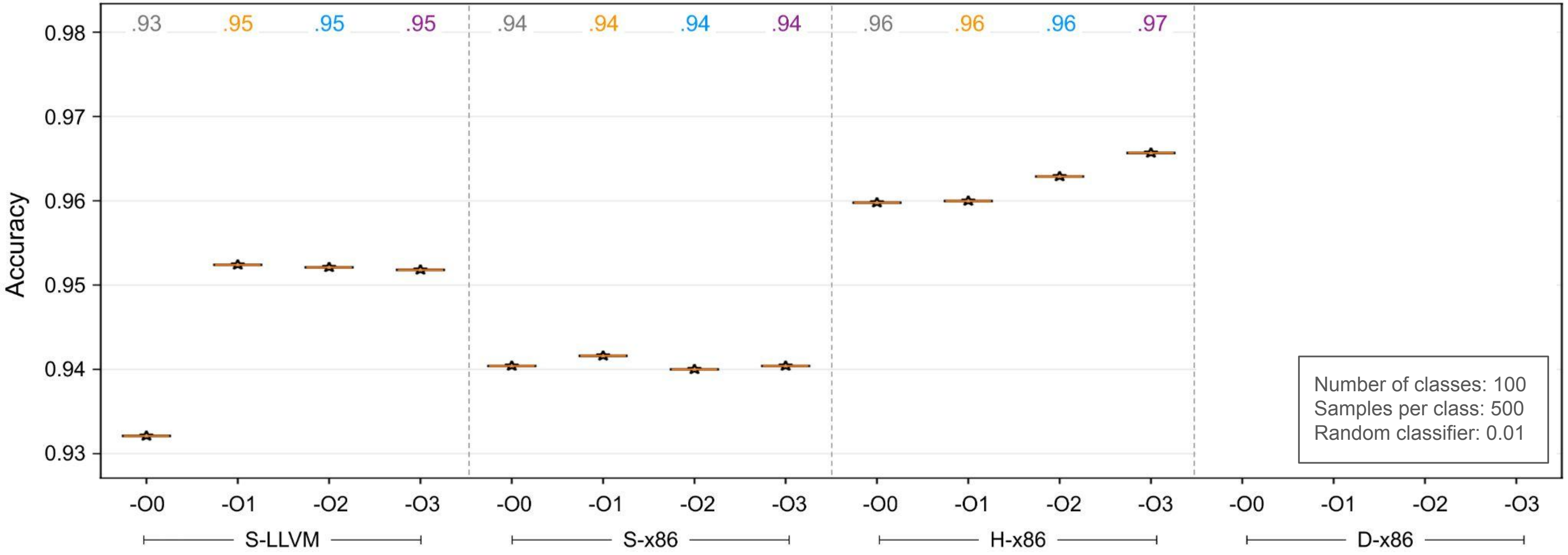
Static
(LLVM IR)

Static
(x86)

Hybrid
(x86)

Dynamic
(x86)

Precision of different histogram-based classifiers



Number of classes: 100
Samples per class: 500
Random classifier: 0.01

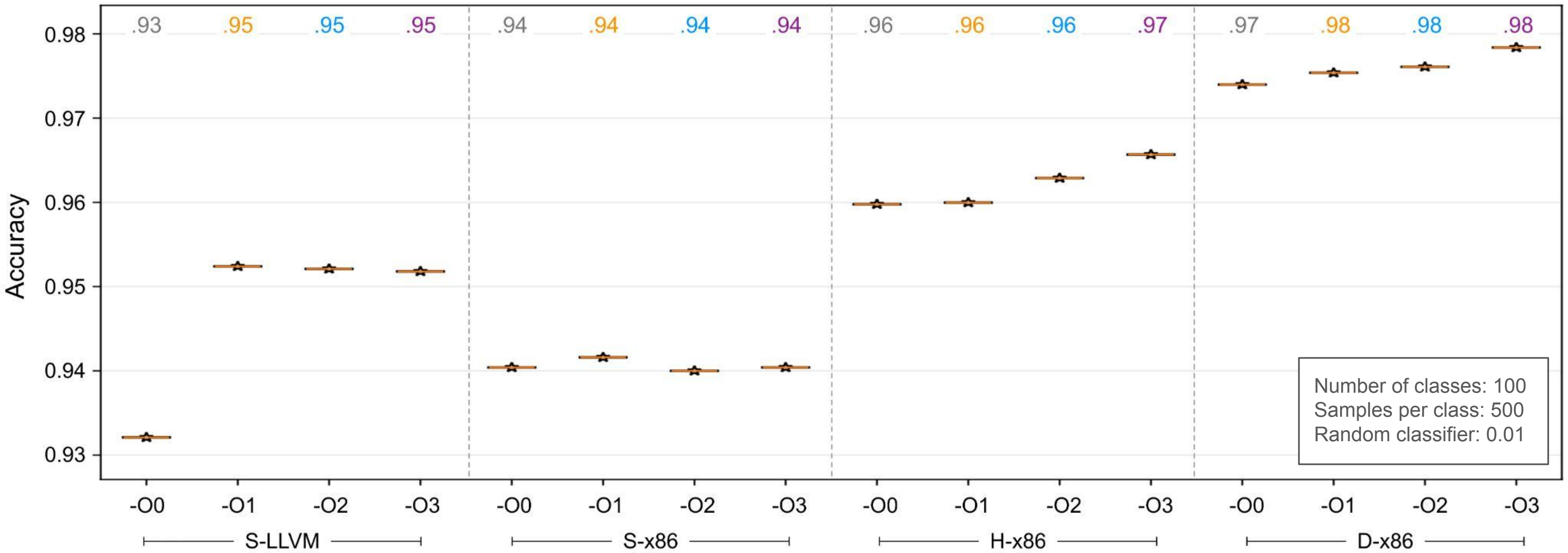
Static (LLVM IR)

Static (x86)

Hybrid (x86)

Dynamic (x86)

Precision of different histogram-based classifiers



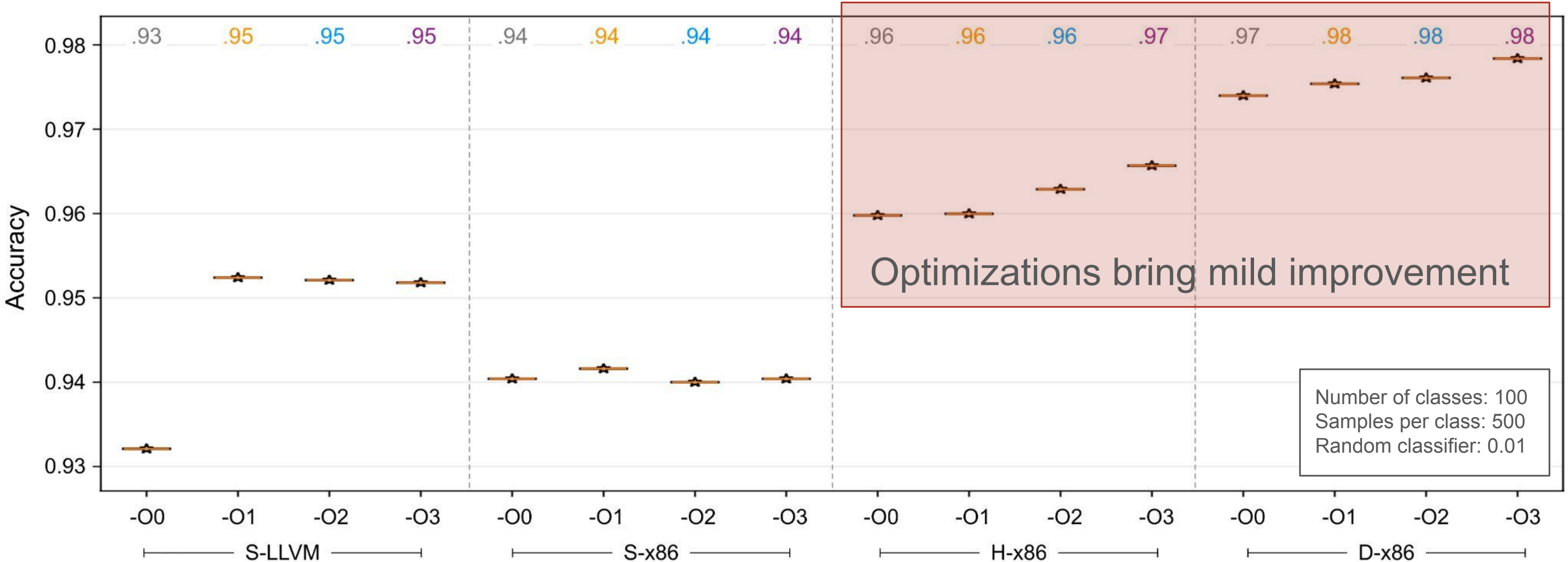
Static (LLVM IR)

Static (x86)

Hybrid (x86)

Dynamic (x86)

Precision of different histogram-based classifiers



Optimizations bring mild improvement

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

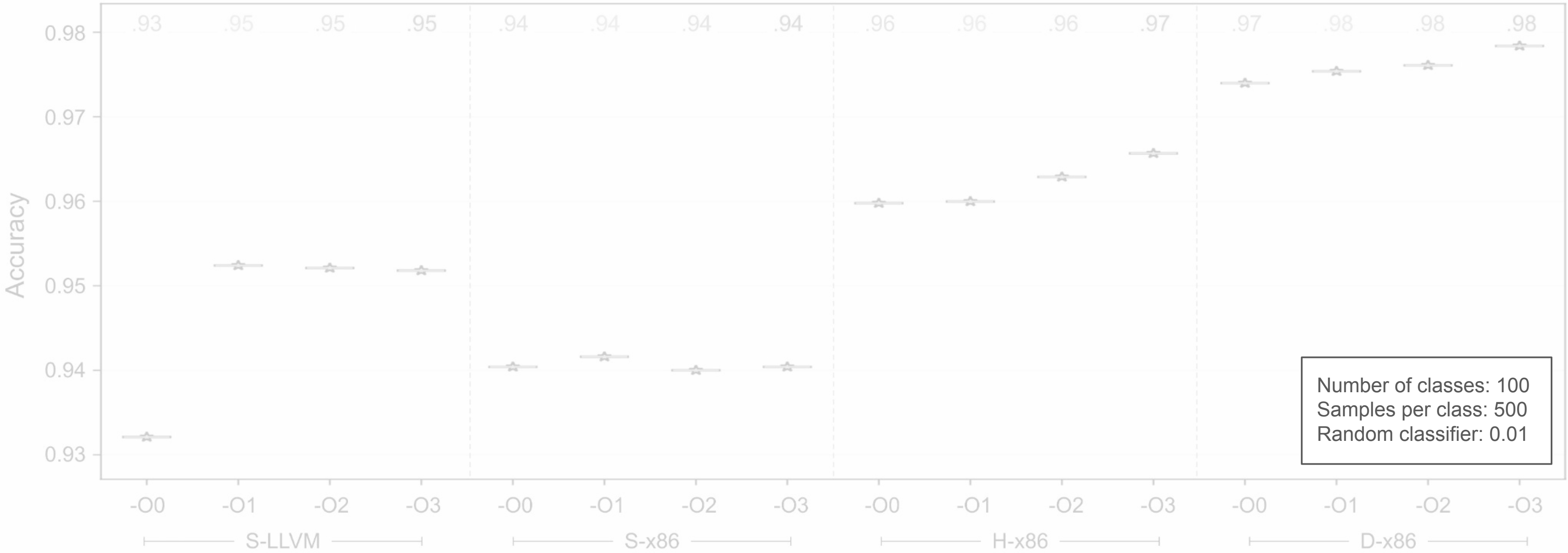
Static
(LLVM IR)

Static
(x86)

Hybrid
(x86)

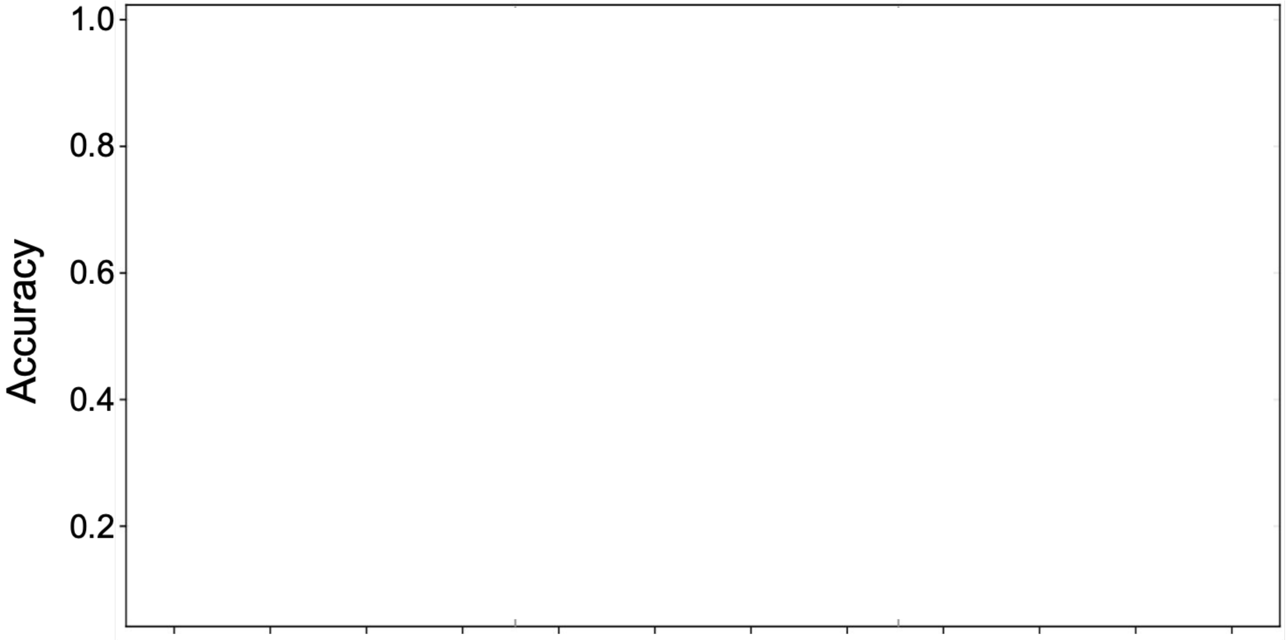
Dynamic
(x86)

Game-1: Adversary obfuscates programs



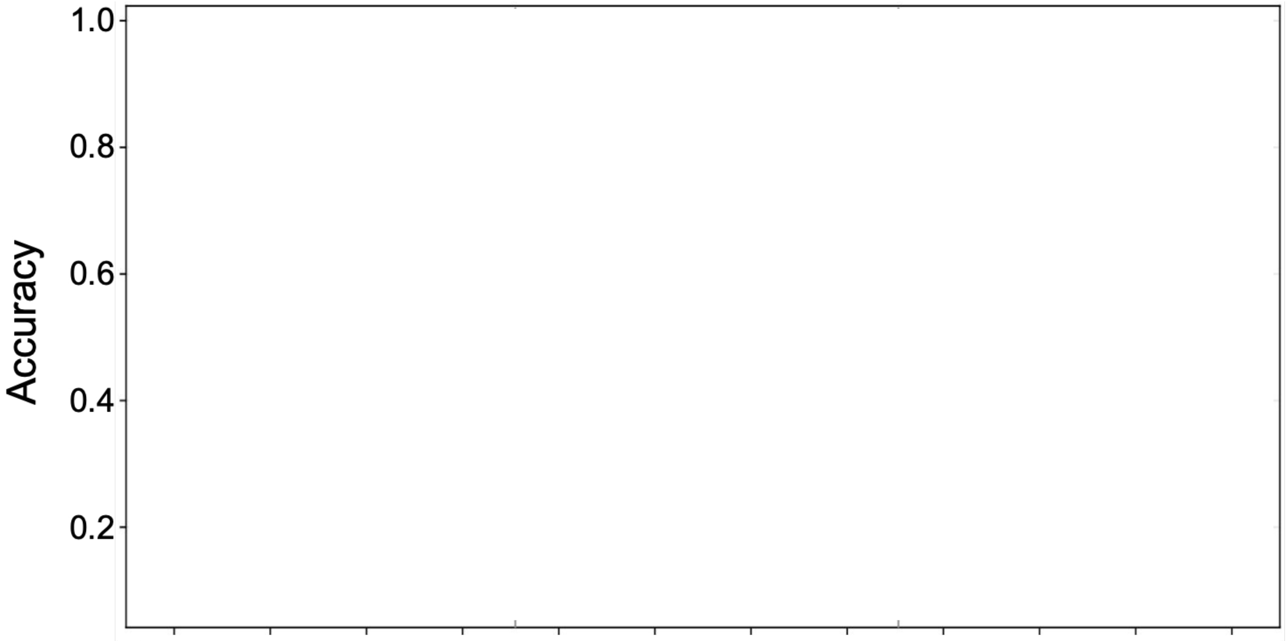
Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs



Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs



bcf: bogus control flow

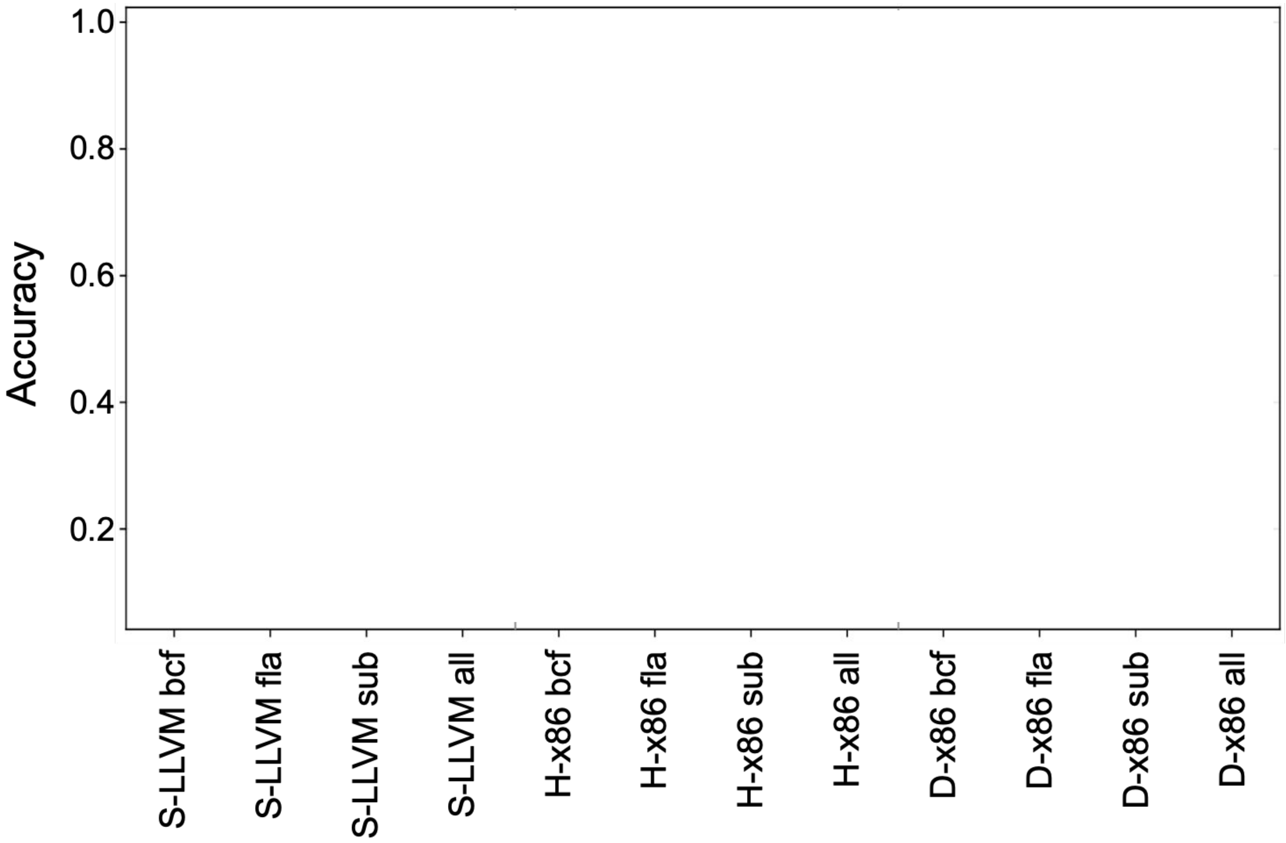
fla: control-flow flattening

sub: instruction substitution

all: bcf + fla + sub

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs



bcf: bogus control flow

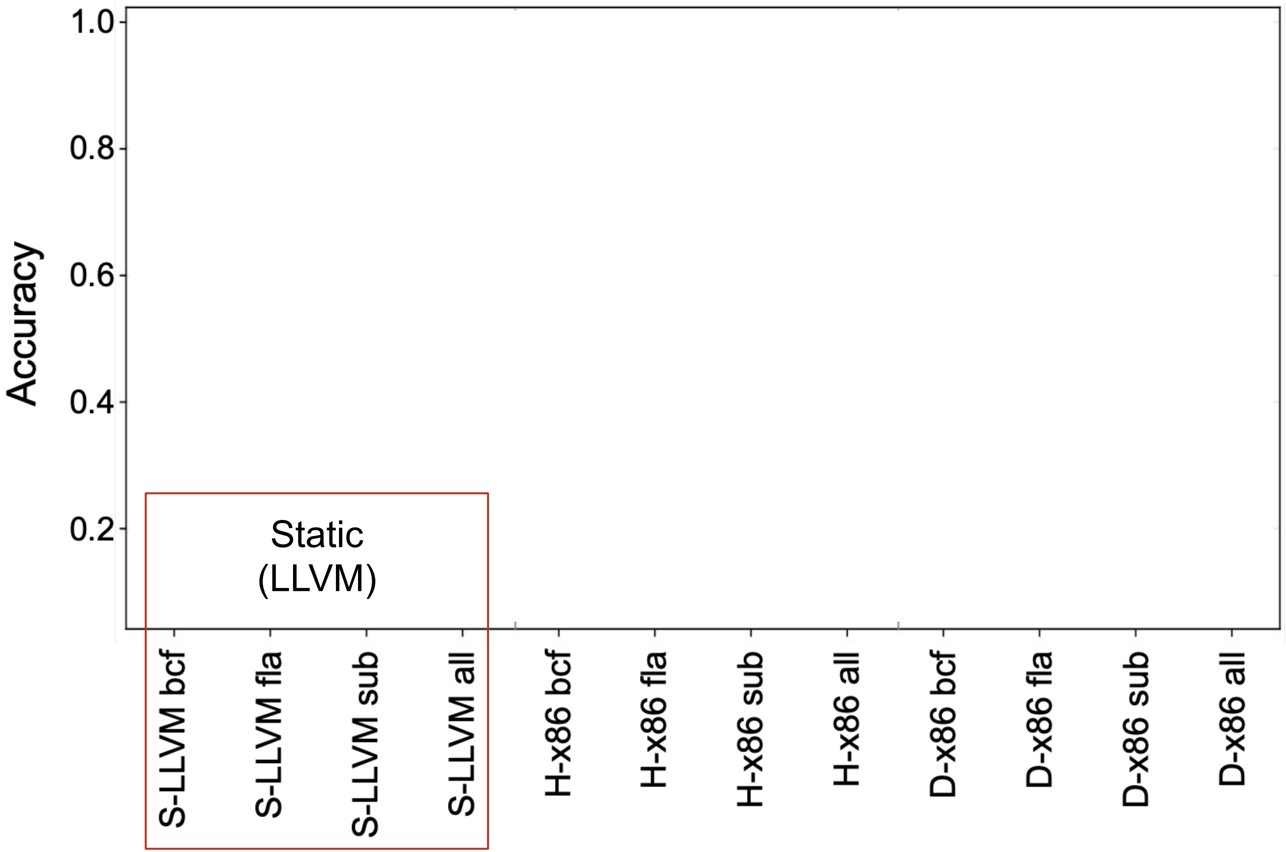
fla: control-flow flattening

sub: instruction substitution

all: bcf + fla + sub

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs



bcf: bogus control flow

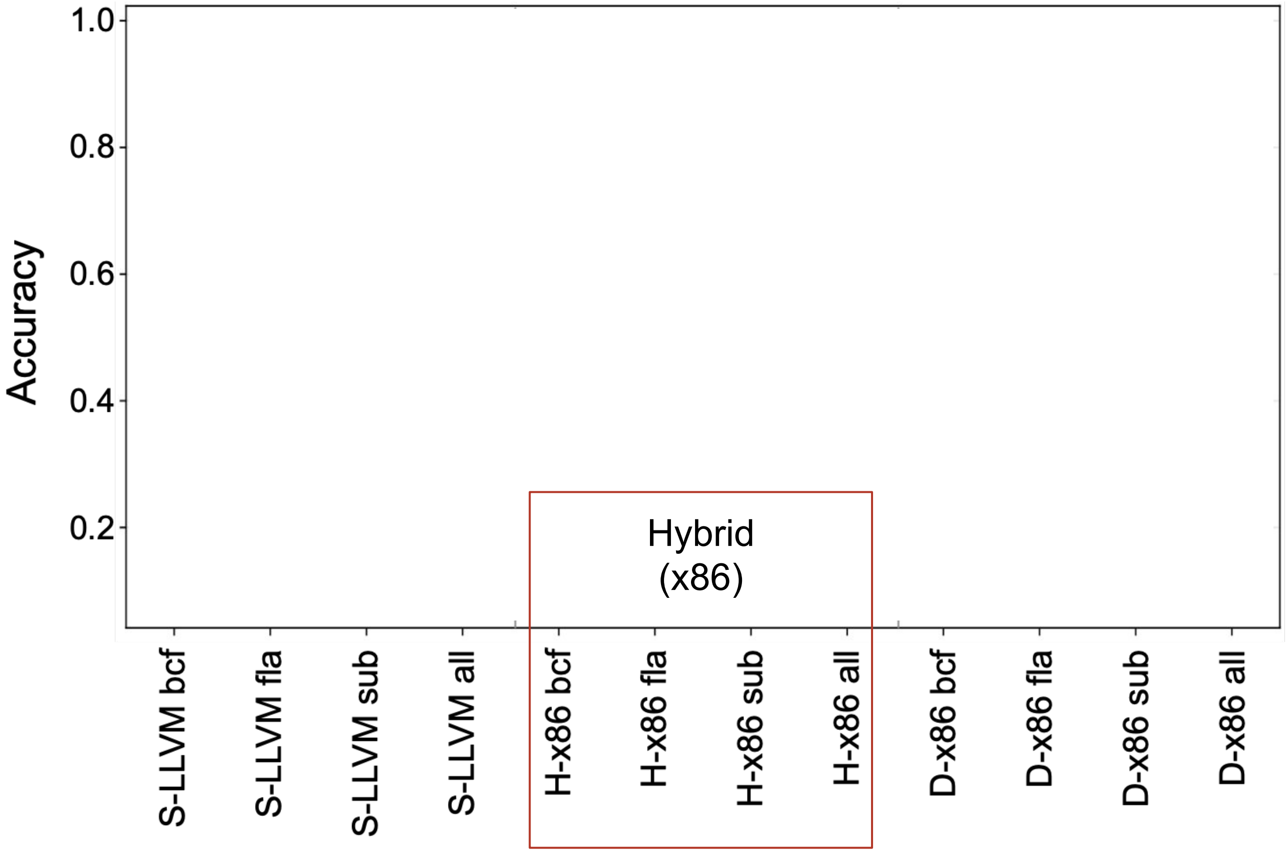
fla: control-flow flattening

sub: instruction substitution

all: bcf + fla + sub

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs



bcf: bogus control flow

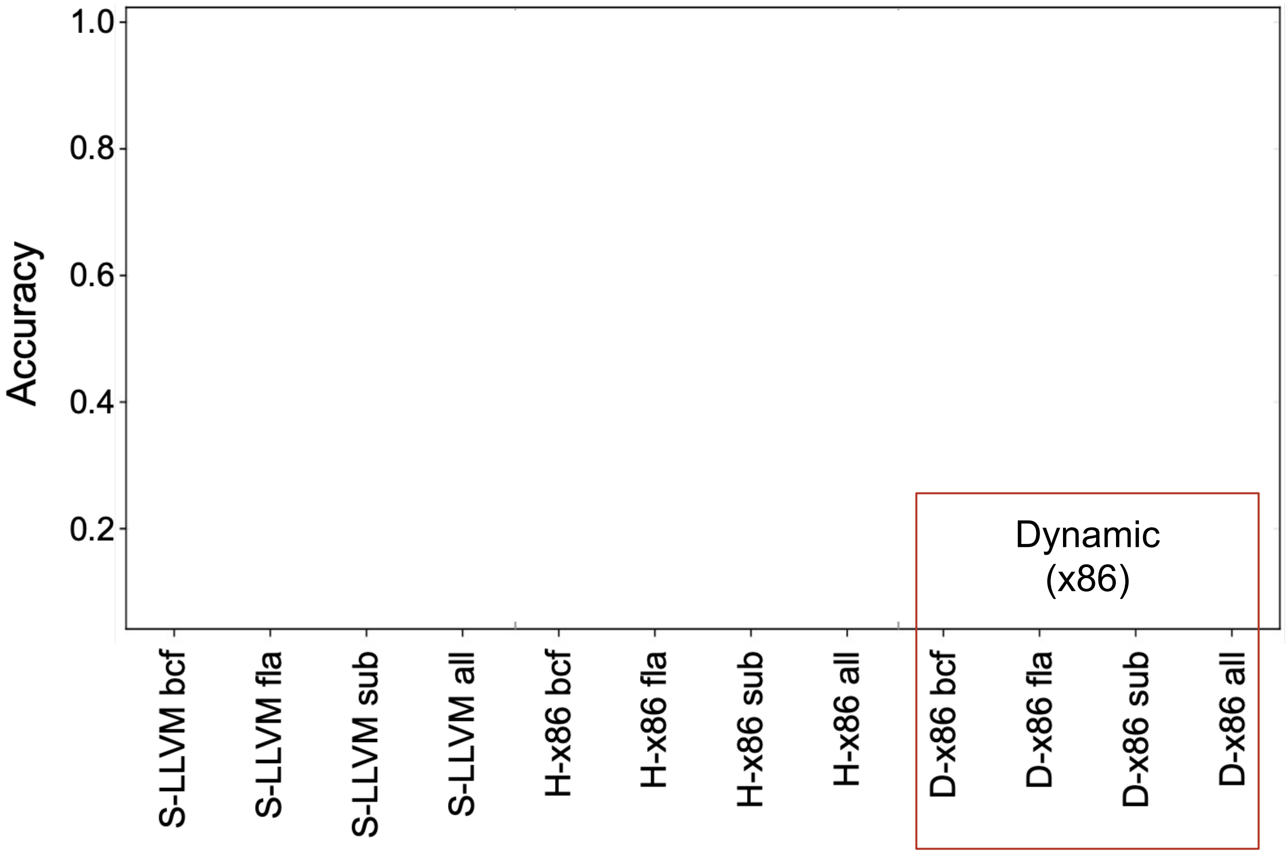
fla: control-flow flattening

sub: instruction substitution

all: bcf + fla + sub

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs



bcf: bogus control flow

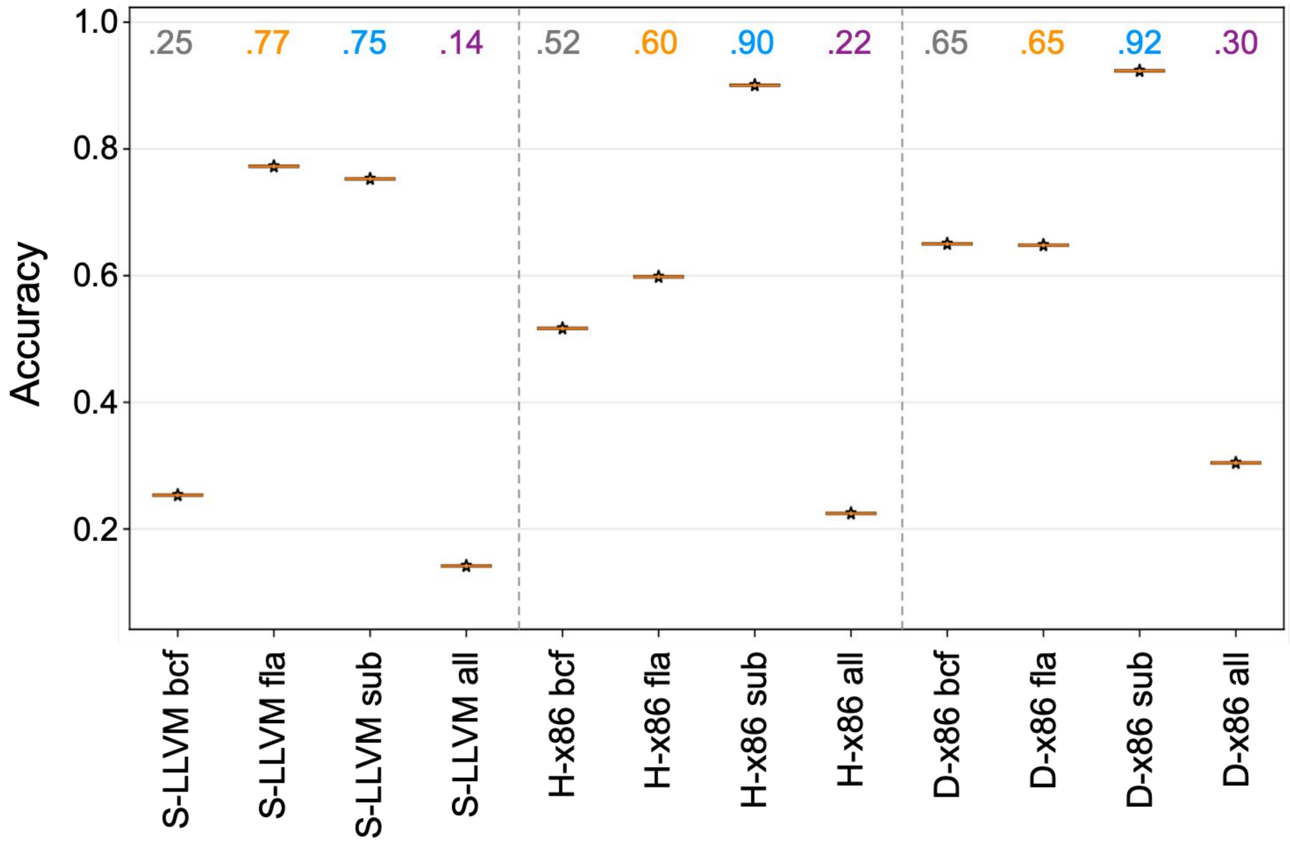
fla: control-flow flattening

sub: instruction substitution

all: bcf + fla + sub

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs



bcf: bogus control flow

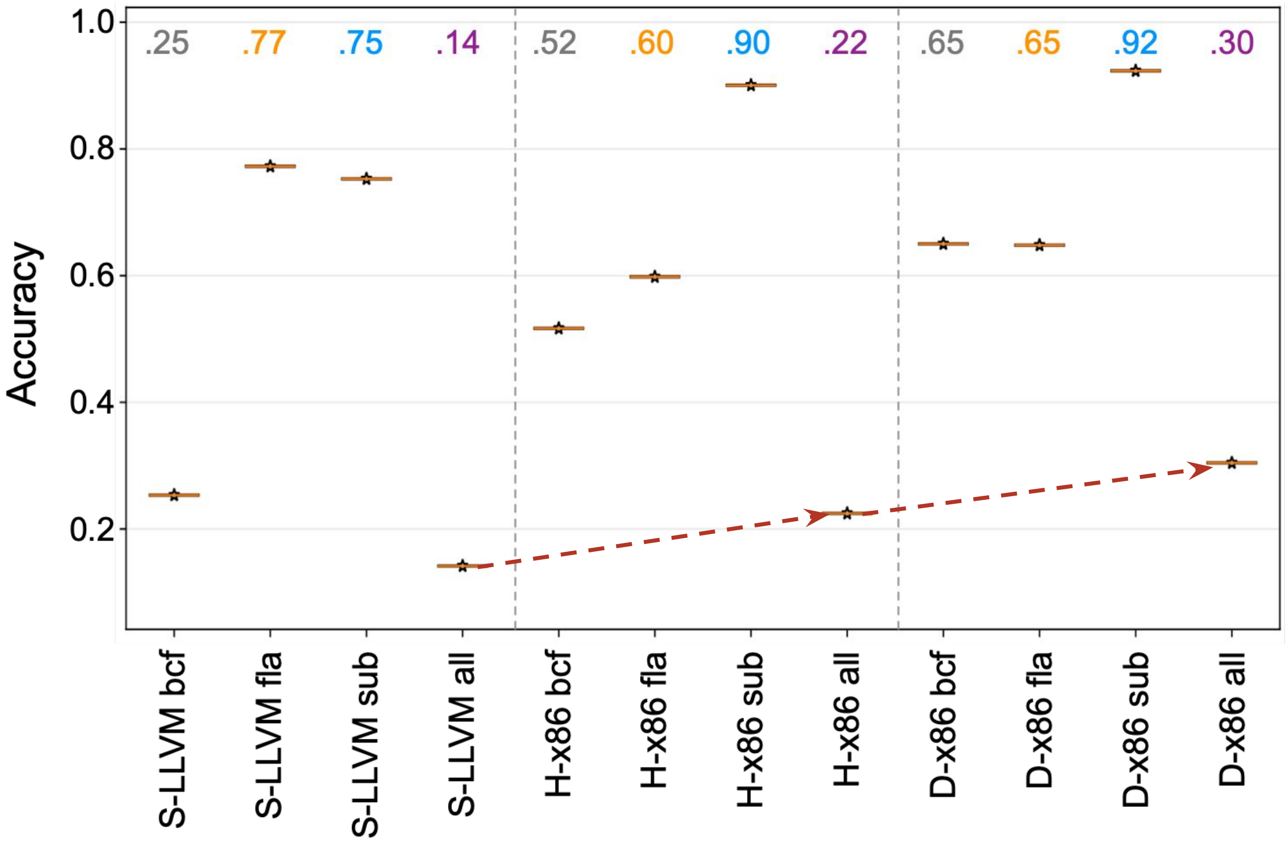
fla: control-flow flattening

sub: instruction substitution

all: bcf + fla + sub

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

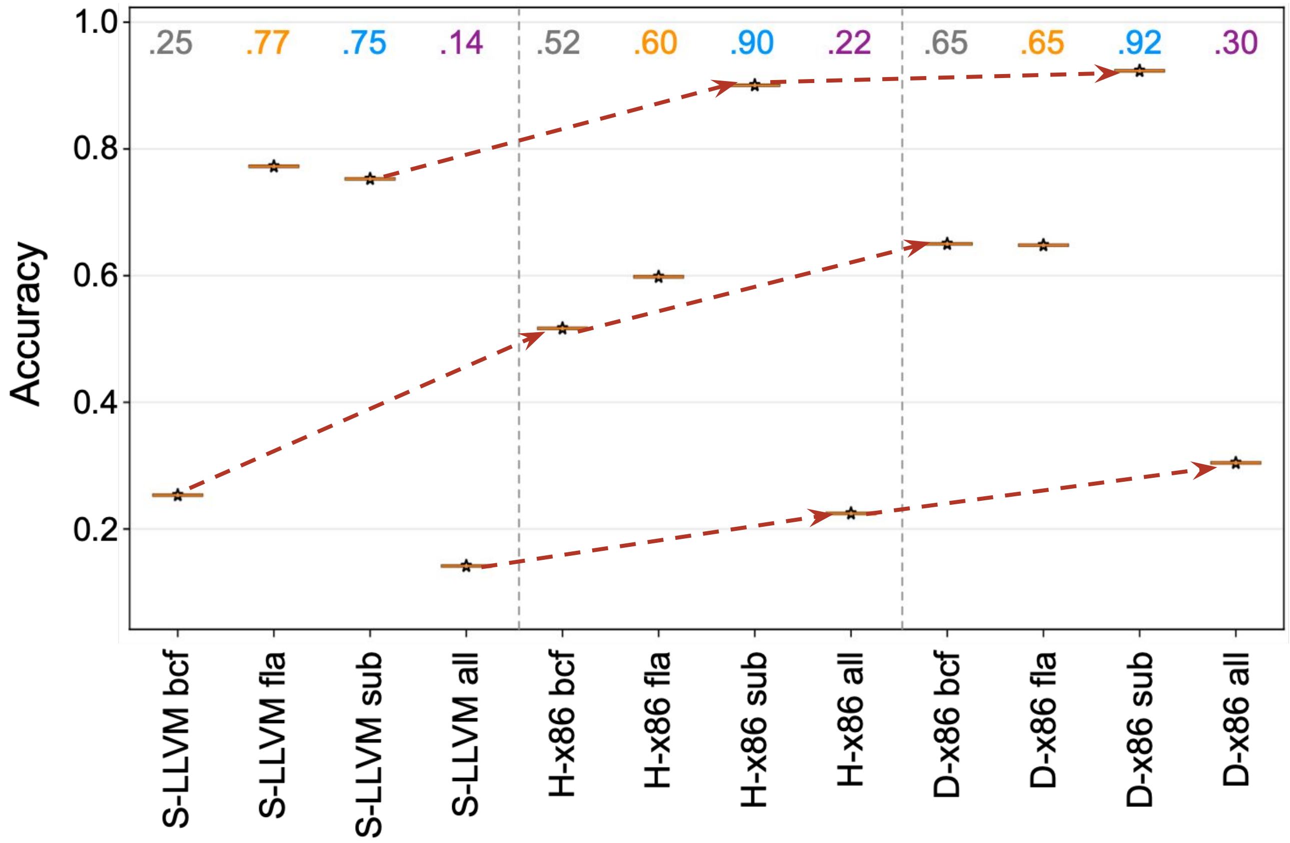
Game-1: Adversary obfuscates programs



1. Static
2. Hybrid
3. Dynamic

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs

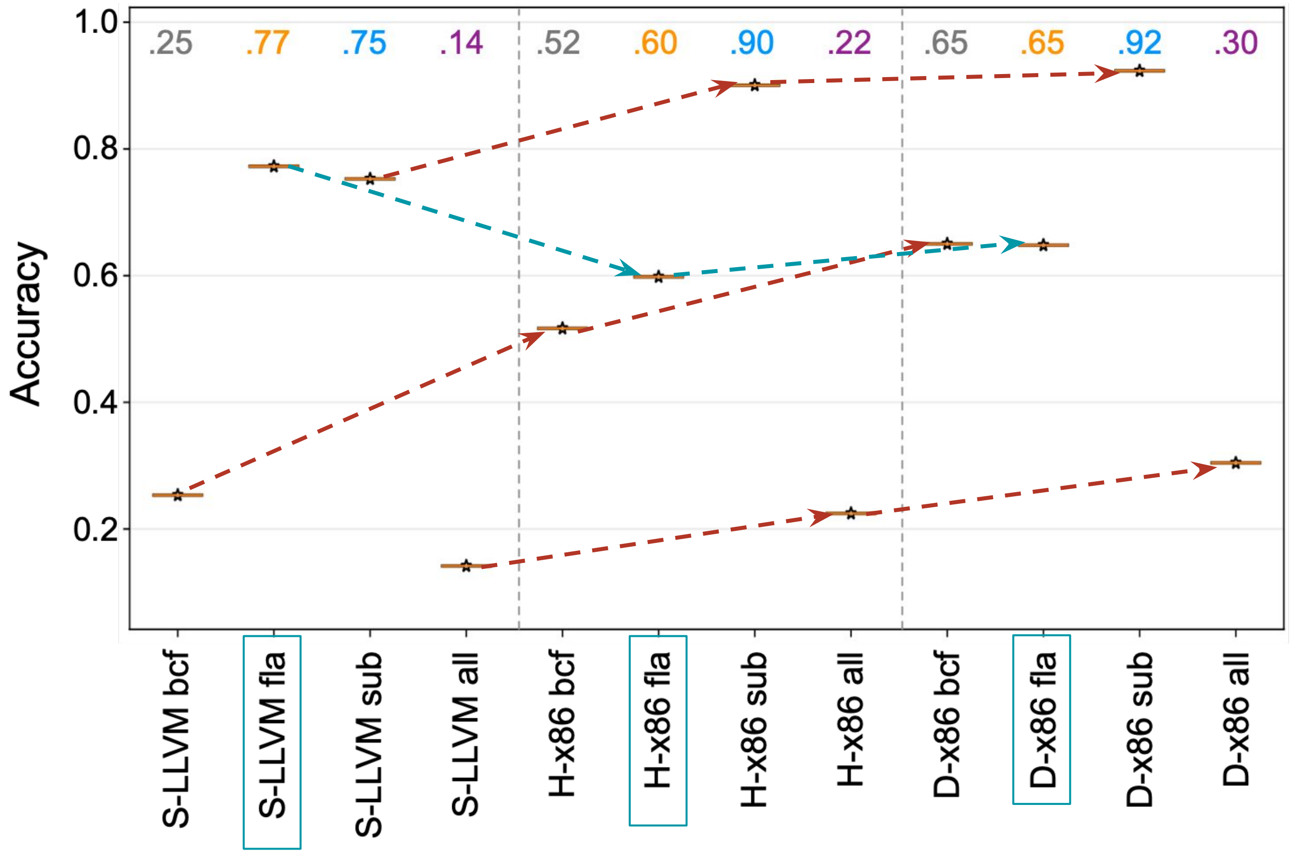


1. Static
2. Hybrid
3. Dynamic

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

Game-1: Adversary obfuscates programs

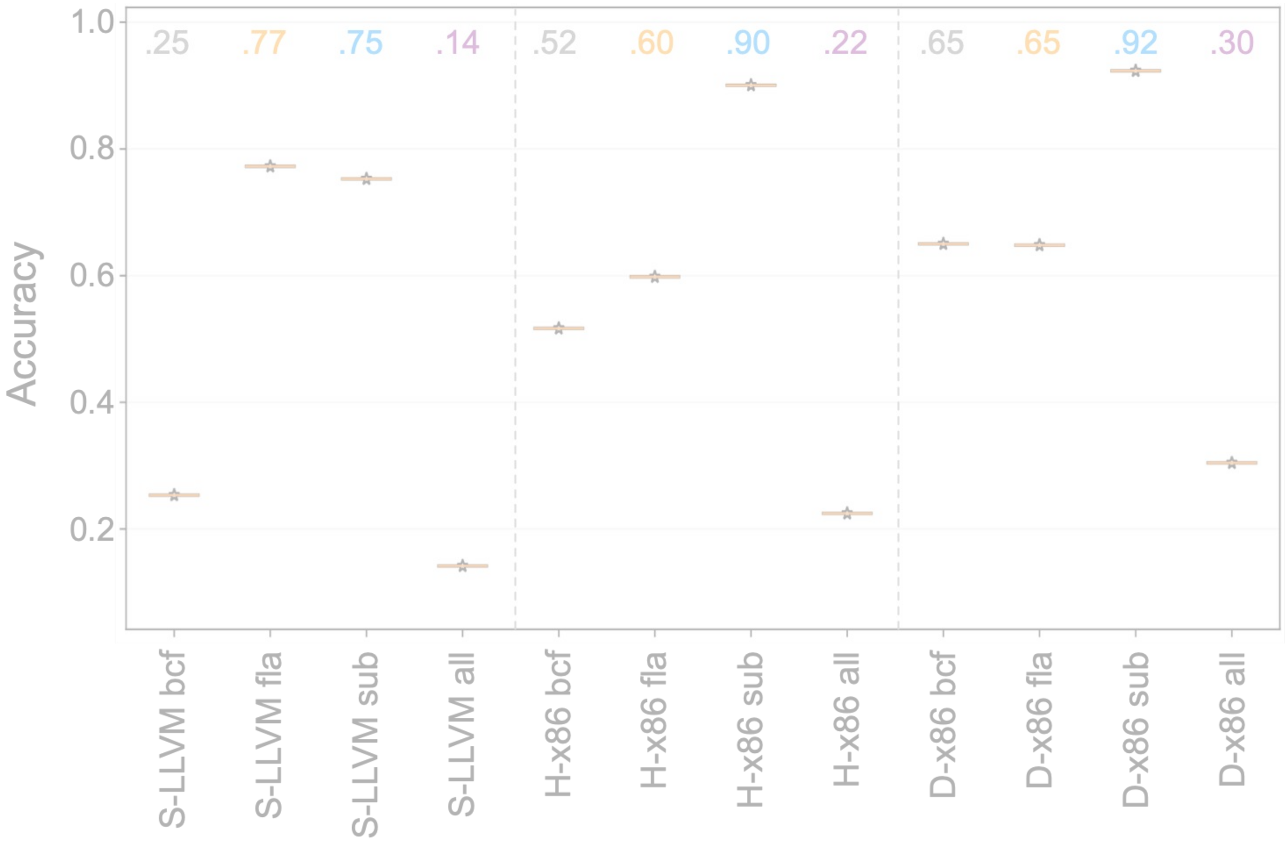
DCC



1. Static
2. Hybrid
3. Dynamic

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

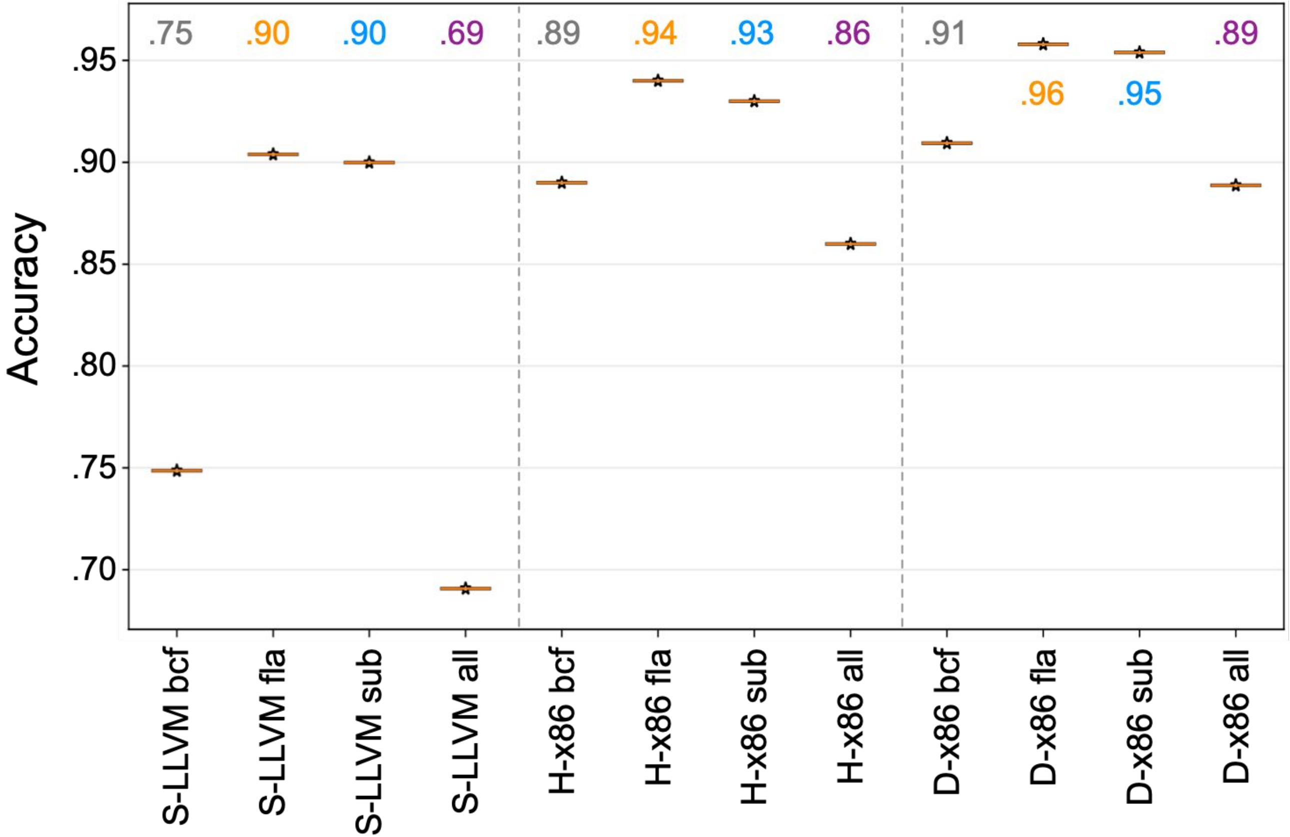
Game-2: What if the Classifier "knows" the Obfuscator?



1. Static
2. Hybrid
3. Dynamic

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

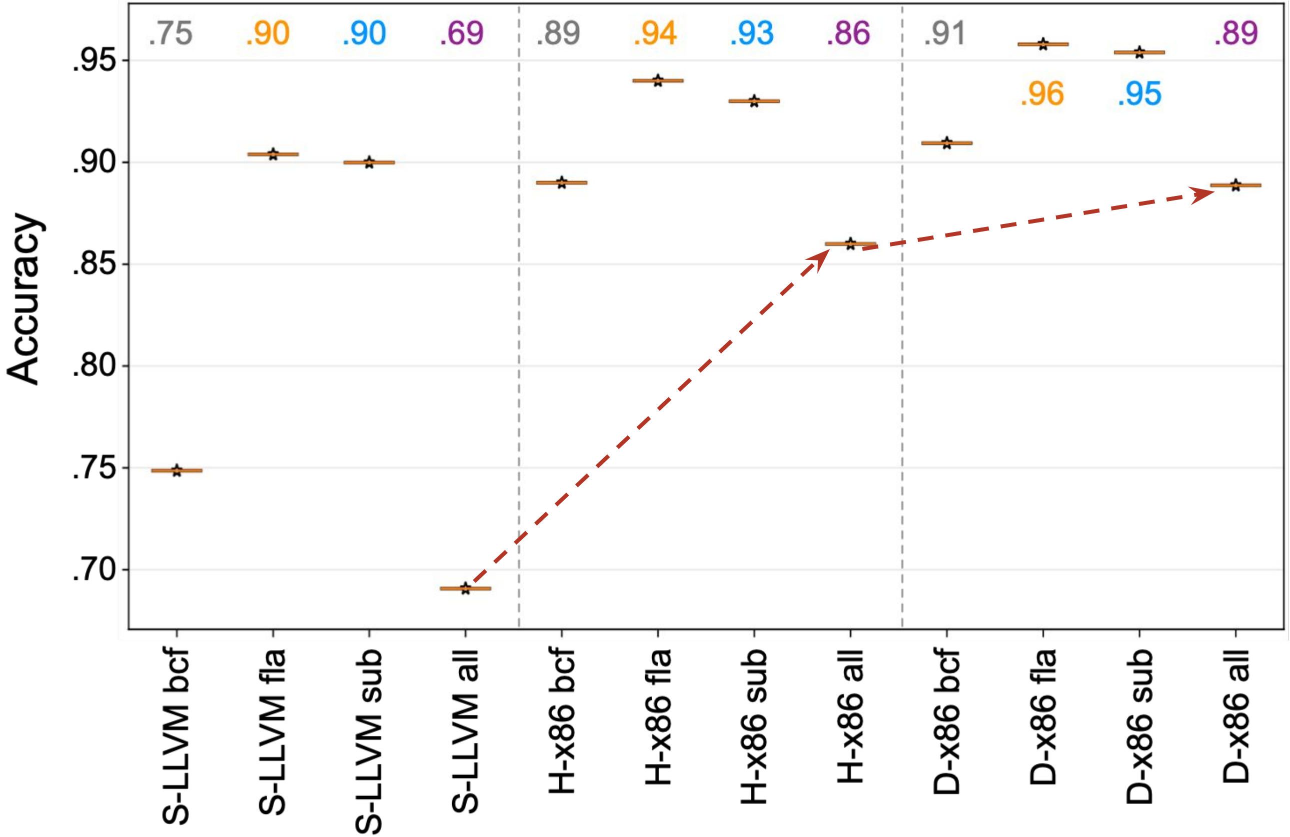
Game-2: What if the Classifier "knows" the Obfuscator?



Classifiers recover "part" of Game-0's accuracy

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

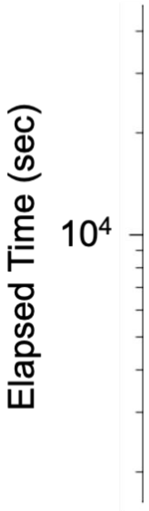
Game-2: What if the Classifier "knows" the Obfuscator?



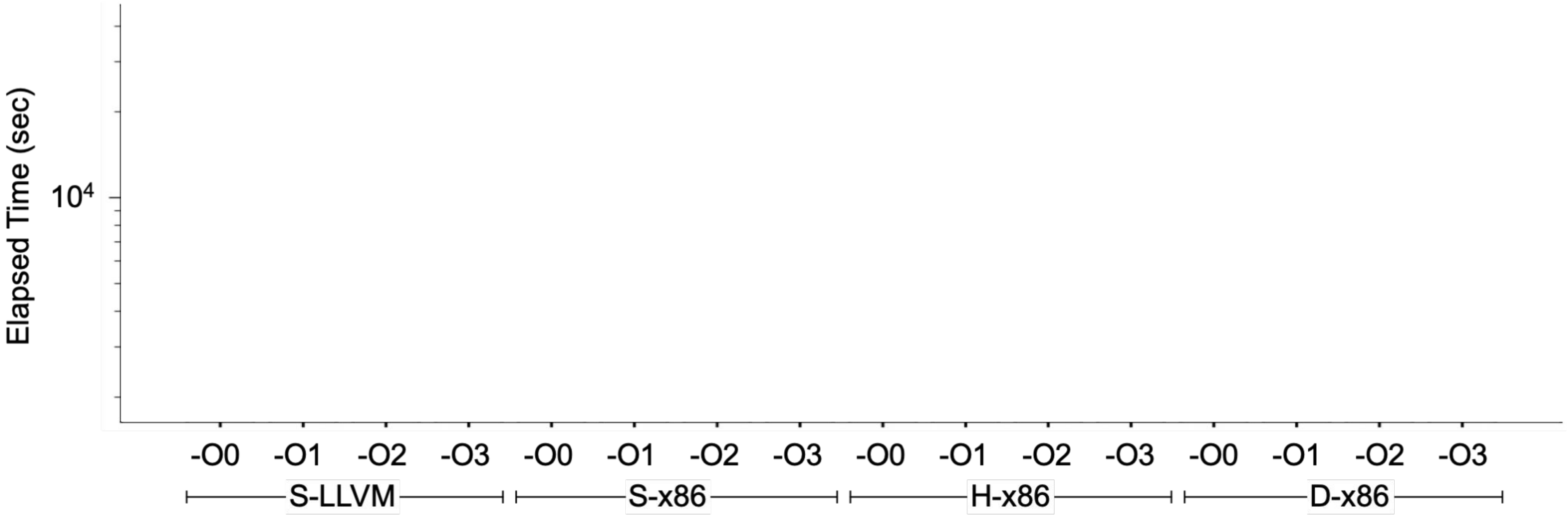
Classifiers recover "part" of Game-0's accuracy

Number of classes: 100
Samples per class: 500
Random classifier: 0.01

What about Speed?



What about Speed?



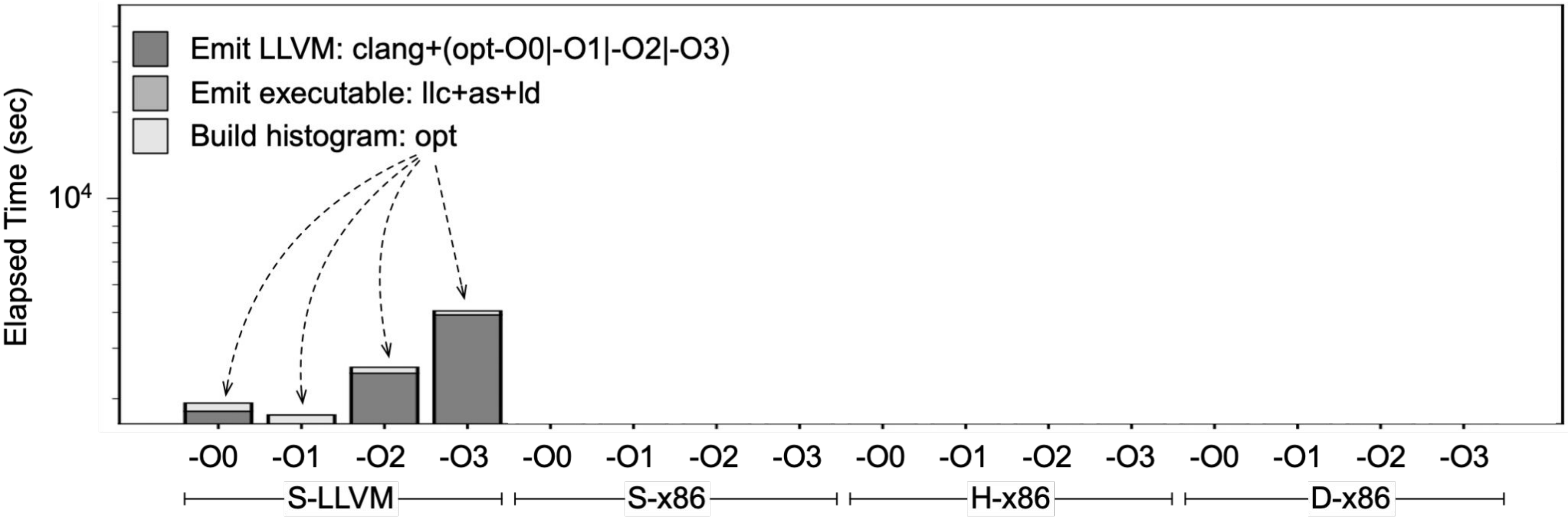
Static
(LLVM IR)

Static
(x86)

Hybrid
(x86)

Dynamic
(x86)

What about Speed?



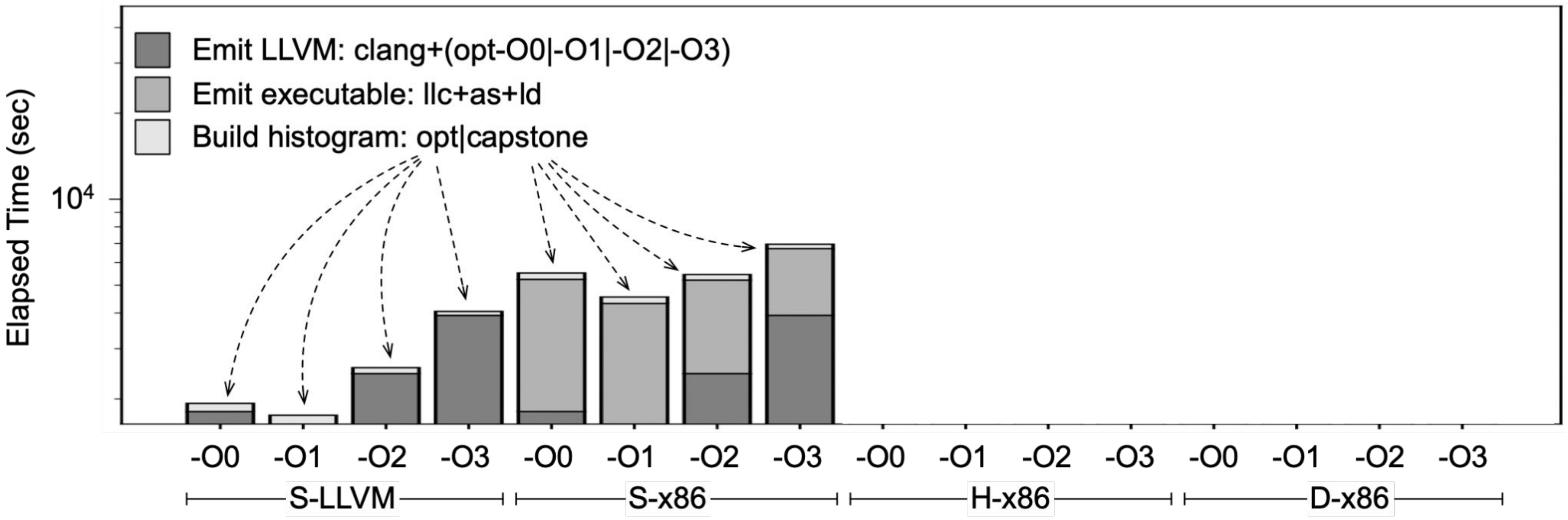
Static (LLVM IR)

Static (x86)

Hybrid (x86)

Dynamic (x86)

What about Speed?



Static
(LLVM IR)

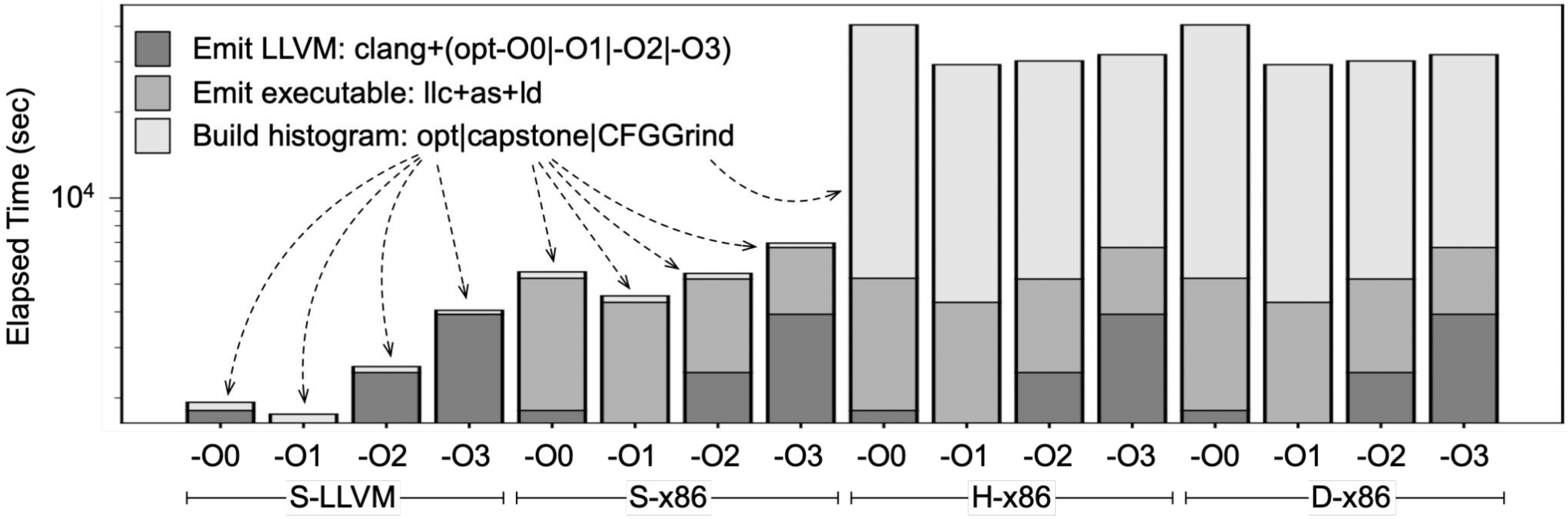
Static
(x86)

Hybrid
(x86)

Dynamic
(x86)

What about Speed?

DCC



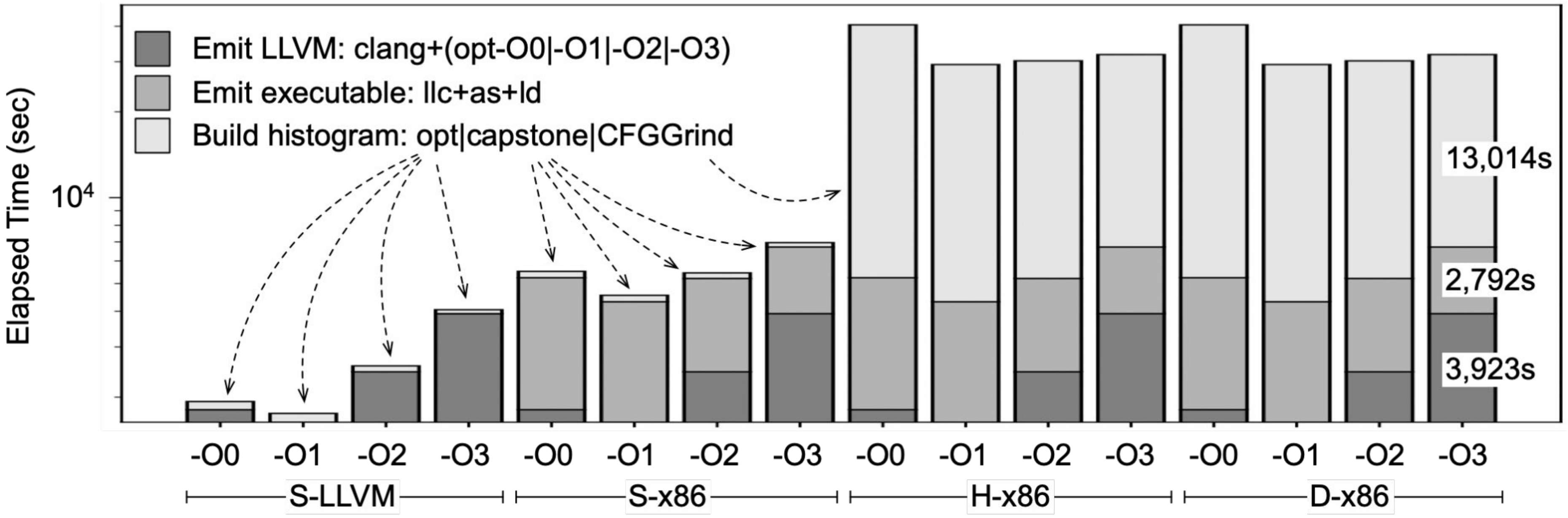
Static
(LLVM IR)

Static
(x86)

Hybrid
(x86)

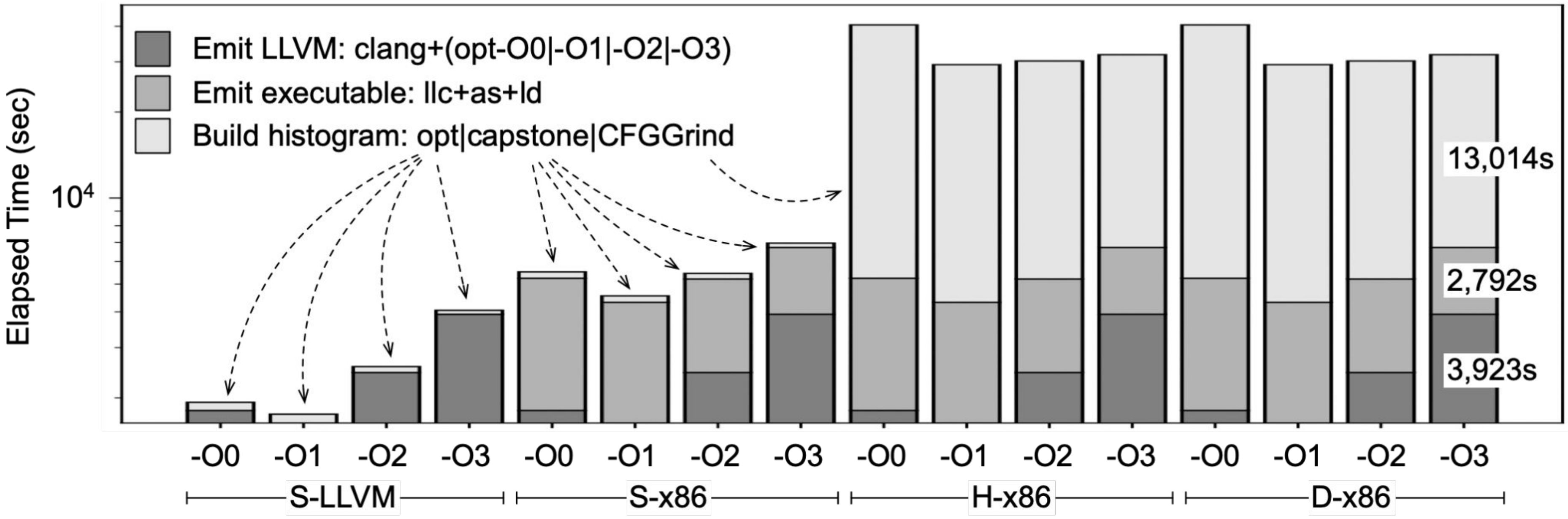
Dynamic
(x86)

What about Speed?



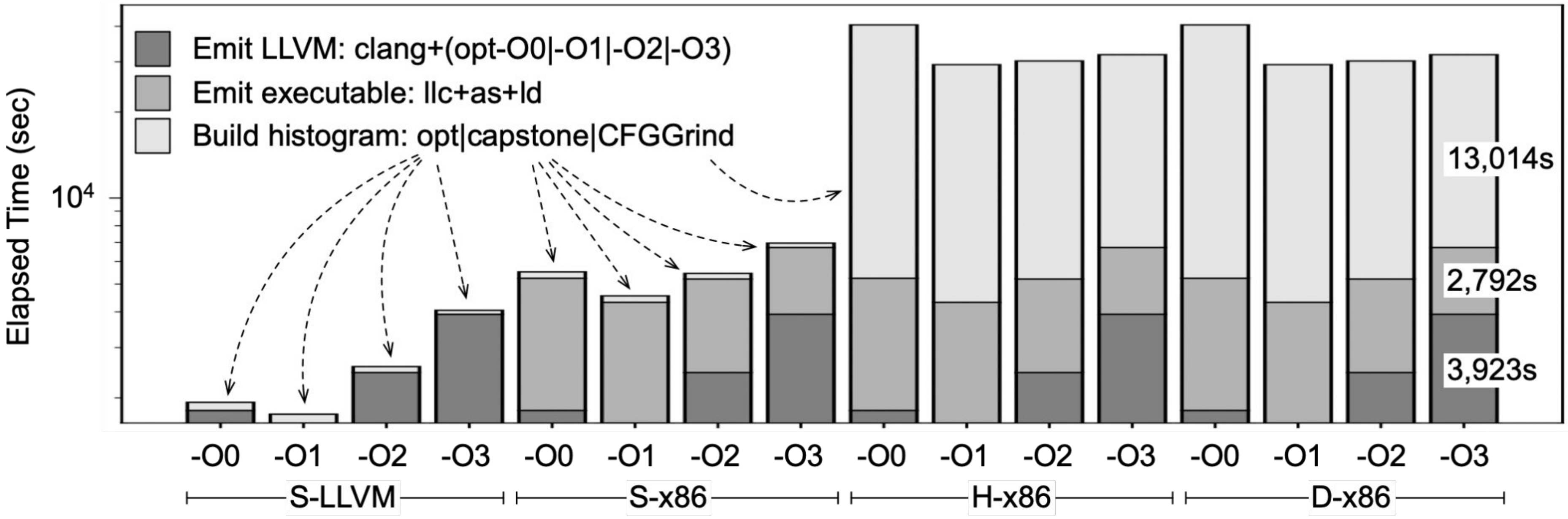
CFGGrind slows down programs by **85x**

What about Speed?



If we account to everything, S-LLVM is **8.8x** faster than D-x86

What about Speed?



If we account to everything, S-86 is **4.0x** faster than D-x86

Conclusions

- Dynamic classification is 4.0-9.0x slower than static classification

Conclusions

- Dynamic classification is 4.0-9.0x slower than static classification
- Dynamic classification is slightly more precise without an adversary (Game-0)

Conclusions

- Dynamic classification is 4.0-9.0x slower than static classification
- Dynamic classification is slightly more precise without an adversary (Game-0)
- Dynamic classification is twice as precise with an adversary (Game-1)

Conclusions

- Dynamic classification is 4.0-9.0x slower than static classification
- Dynamic classification is slightly more precise without an adversary (Game-0)
- Dynamic classification is twice as precise with an adversary (Game-1)

Open question:

By how much can we reduce the overhead of dynamic classification, without losing too much precision?

References & Contact

Fernando Pereira

fernando@dcc.ufmg.br

http://lac.dcc.ufmg.br

References & Contact

Fernando Pereira

fernando@dcc.ufmg.br

http://lac.dcc.ufmg.br

Artifacts Available
acm
V1.1
Artifacts Evaluated
acm
Functional V1.1
Results Reproduced
acm
V1.1

https://github.com/ComputerSystemsLaboratory/Rouxinol

References & Contact

Fernando Pereira

fernando@dcc.ufmg.br

http://lac.dcc.ufmg.br

Artifacts Available
acm
V1.1
Artifacts Evaluated
acm
Functional V1.1
Results Reproduced
acm
V1.1

https://github.com/ComputerSystemsLaboratory/Rouxinol

Reference

A Comparative Study on the Accuracy and the Speed of Static and Dynamic Program Classifiers

https://homepages.dcc.ufmg.br/~fernando/publications/papers/CC25_Faustino.pdf