THE UNIVERSITY
OF BRITISH COLUMBIA

MANDOLINE: *Dynamic Slicing of Android Applications with Trace-Based Alias Analysis*

ICST, 2021

Khaled Ahmed

khaledea@ece.ubc.ca

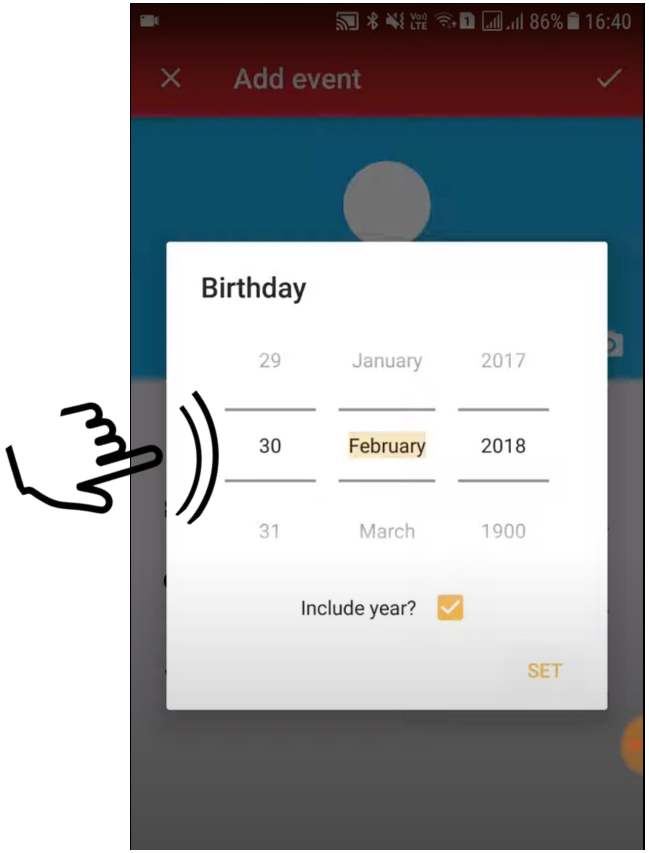
Mieszko Lis

mieszko@ece.ubc.ca

Julia Rubin

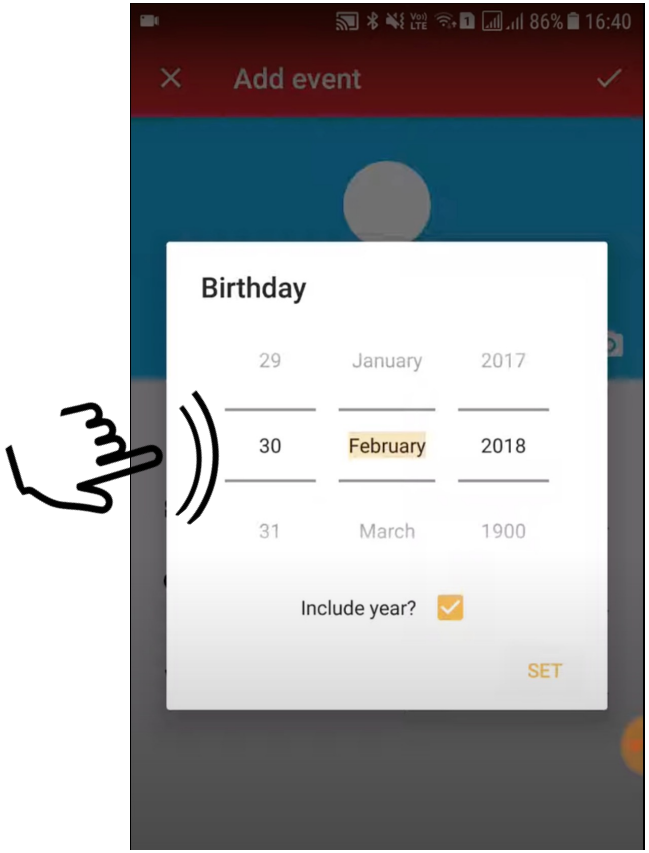
mjulia@ece.ubc.ca

Buggy Android App

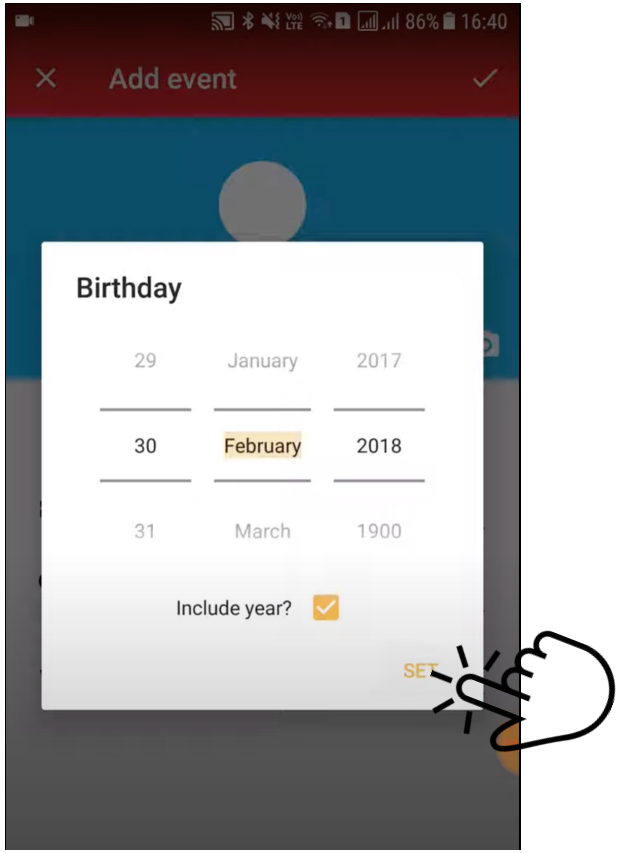


User selects monthly recurring event day

Buggy Android App



User selects monthly recurring event day



Crash! There's no February 30th!

Debugging

```
Date date = new Date(day,month,year);
```

Debugging

```
int day = userInput.d;

Date date = new Date(day,month,year);
```

Debugging

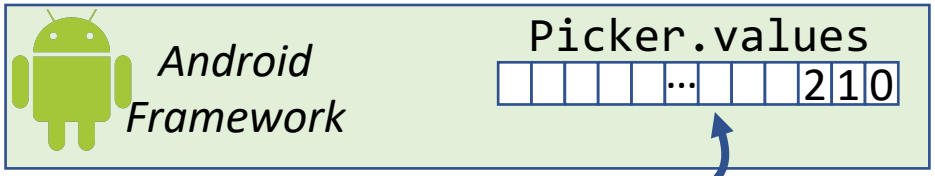
```
bar() {

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```

Debugging

```
bar() {

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```

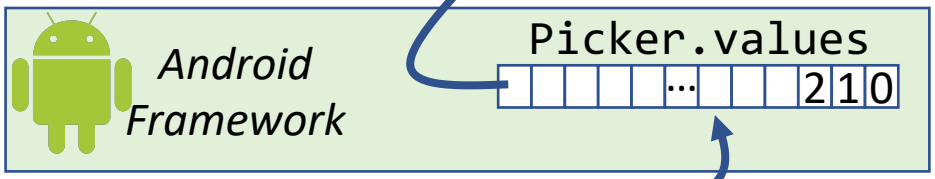
Debugging

```
bar() {

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



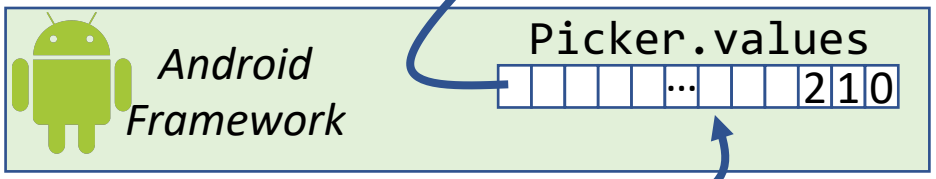
Debugging

```
bar() {

    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {

    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



Debugging

```
bar() {
    picker.setMaxValue(31);
```

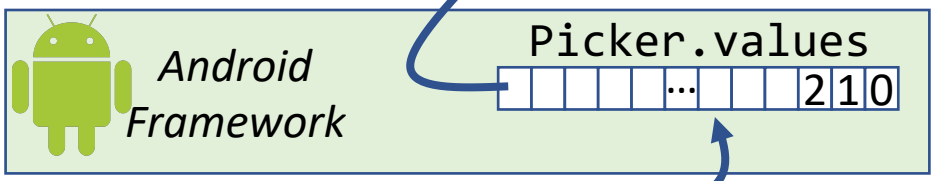



```
    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```

Automated by slicing
slice << code size

Debugging with Slicing

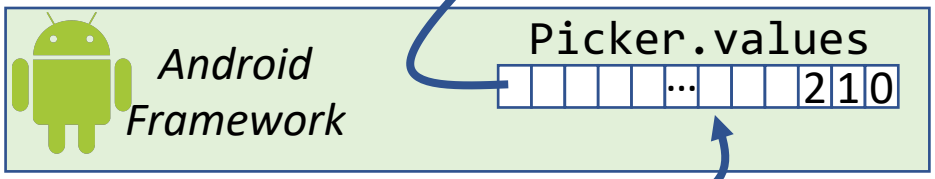
```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



= Data-flow

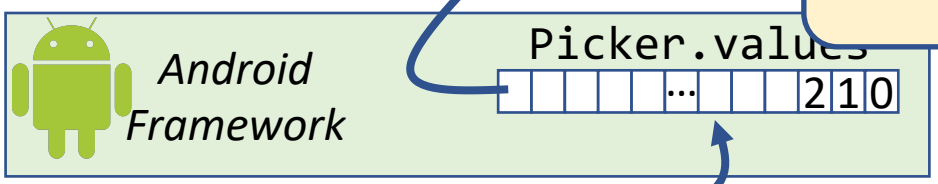
Challenge1: Fields

```
bar() {

    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {

    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



Challenge1: Fields

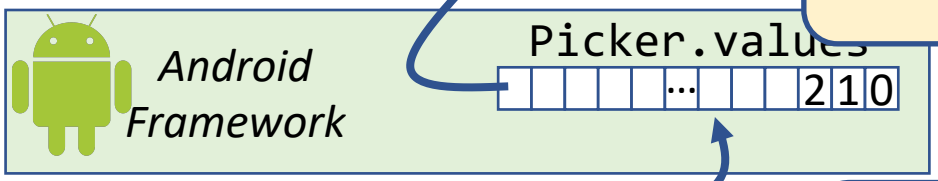
```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



Same field, different names

Challenge1: Fields

```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```

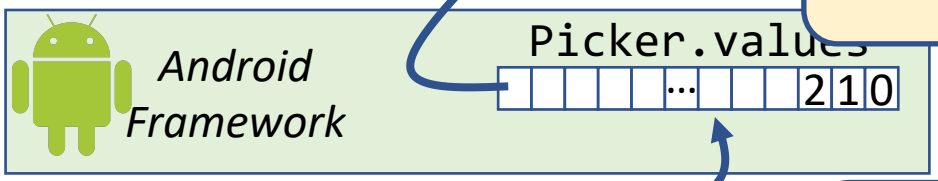


Same field, different names

Simple solution:
record memory address
= slow for Android

Challenge1: Fields

```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



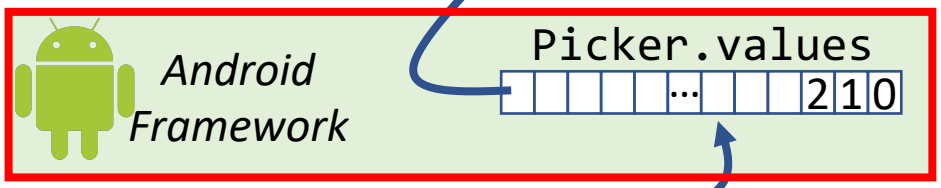
Same field, different names

Simple solution:
record memory address
= slow for Android

Android kills slow apps

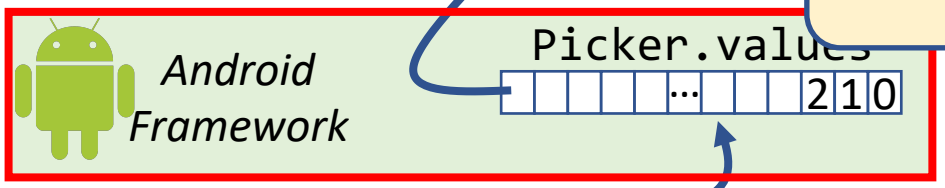
Challenge2: Framework

```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



Challenge2: Framework

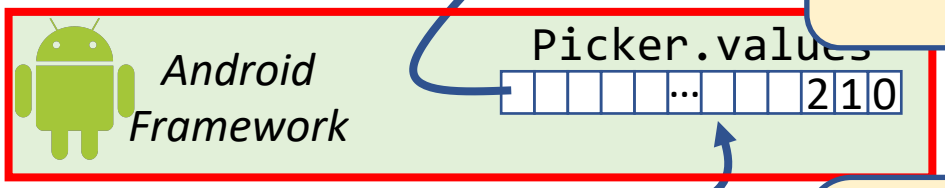
```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



Rely on framework methods

Challenge2: Framework

```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```

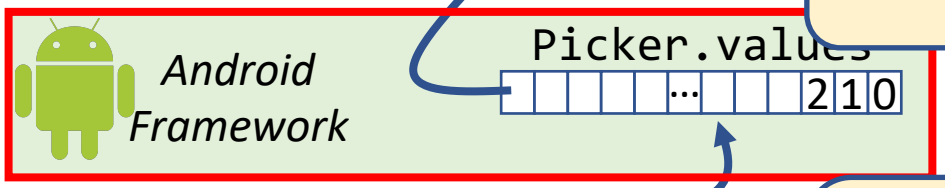


Rely on framework methods

Simple solution:
Instrument entire framework

Challenge2: Framework

```
bar() {
    picker.setMaxValue(31);

    selected.d = picker.getValue();

    foo(selected);
}
foo(Object userInput) {
    int day = userInput.d;

    Date date = new Date(day,month,year);
}
```



Rely on framework methods

Simple solution:
Instrument entire framework

Not practical

Our Insight

Our Insight

`selected.d`

↑

`userInput.d`

→

Light-weight Instrumentation on-device

+

field data-flow analysis on the trace

Our Insight

`selected.d`

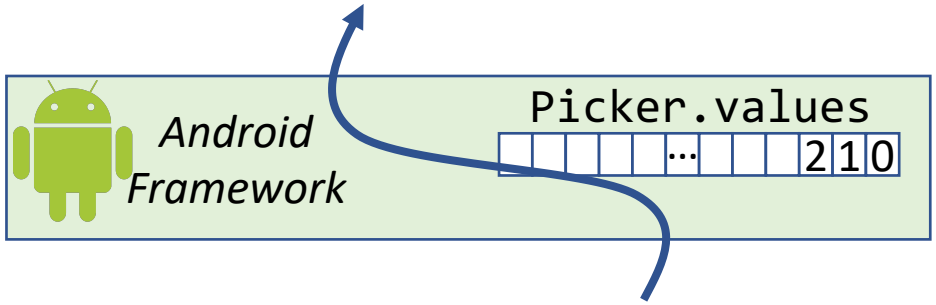
↑

`userInput.d`

→ Light-weight Instrumentation on-device

+

field data-flow analysis on the trace



→ framework modeling

Trace-Based Field Analysis

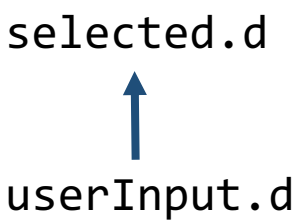
Goal: Field data-flows w/o object addresses

```
selected.d
     ↑
userInput.d
```

Trace-Based Field Analysis

Goal: Field data-flows w/o object addresses

- How to produce a trace with light-weight instrumentation?



Trace-Based Field Analysis

Goal: Field data-flows w/o object addresses

```
selected.d
    ↑
userInput.d
```

- How to produce a trace with light-weight instrumentation?
- What information to use in the trace instead of object addresses?

Light-Weight Instrumentation

```
jane.age = 15
john     = jane
john.age = 25
res = jane.age
```

Light-Weight Instrumentation

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

Record executed statement IDs

Field Data-Flow Analysis

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

Field Data-Flow Analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

use

Field Data-Flow Analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

use

Field Data-Flow Analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

definition?

use

Field Data-Flow Analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

definition?

use

Field Data-Flow Analysis

```
     john aliases jane
john.age aliases jane.age
```

```
age = 15
```

definition?

```
2  john      = jane

3  john.age = 25

4  res = jane.age
```

use

Field Data-Flow Analysis

```
     john aliases jane
john.age aliases jane.age
```

```
age = 15
```

definition?

```
2  john      = jane
```

definition

```
3  john.age = 25
```

```
4  res = jane.age
```

use

Field Alias Analysis

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

Field Alias Analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

use

Field Alias Analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

use

Field Alias Analysis

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

use

Field Alias Analysis

Search for `john.age` too

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

use

Field Alias Analysis

Search for `john.age` too

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

use

Field Alias Analysis

Search for `john.age` too

```
1   jane.age = 15
2   john     = jane
3   john.age = 25
4   res = jane.age
```

use

Field Alias Analysis

Search for `john.age` too

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

definition

use

Field Alias Analysis

Search for `john.age` too

```
1  jane.age = 15
2  john     = jane
3  john.age = 25
4  res = jane.age
```

definition

use

Backward + forward analysis on trace

Field Alias Analysis

```
Input: ICDG, t, v
Output: T'
begin
    V ← {v}                                          ▹ Aliases of the variable v
    S ← BackwardAnalysis(ICDG, V, t)                 ▹ Alias statements
    return LastDefined(S, t)                         ▹ Definitions of v and its fields

Procedure BackwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▹ Traverse the ICDG in inverse execution order
        foreach t' s.t. (t', t) ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v and LHS(t') have a common prefix then
                ▹ t' is a definition of a variable in V
                S ← S ∪ {t'}                         ▹ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↑, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S

Procedure ForwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▹ Traverse the ICDG in execution order
        foreach t' s.t. (t, t') ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v is a prefix of LHS(t') then
                ▹ t' is a definition of a field of a variable in V
                S ← S ∪ {t'}                         ▹ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↓, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S

Procedure AliasAnalysis(d, V, t')
    ▹ Initialize with the original set of variables
    if d =↑ then
        V_b ← V; V_f ← ∅
    else if d =↓ then
        V_b ← ∅; V_f ← V
    foreach v ∈ V s.t. v and RHS(t') have a common prefix do
        ▹ LHS(t') is a new alias for v
        V_f ← V_f ∪ ExtendFields(LHS(t'))            ▹ Follow it forward
    foreach v ∈ V s.t. v is a prefix of LHS(t') do
        ▹ t' is a re-definition of a field of v
        ▹ Follow the assigned variable both backward and forward
        V_b ← V_b ∪ RHS(t')
        V_f ← V_f ∪ RHS(t')
    foreach v ∈ V s.t. LHS(t') is a prefix or equal to v do
        ▹ t' is a full re-definition of v
        if d =↑ then
            V_b ← V_b \ {v}                          ▹ Do not search before the definition
            ▹ Follow the assigned variable both backward and forward
            V_b ← V_b ∪ ExtendFields(RHS(t'))
            V_f ← V_f ∪ ExtendFields(RHS(t'))
        else if d =↓ then
            V_f ← V_f \ {v}                          ▹ Do not search for new variables
    return <V_b,V_f>
```

Field Alias Analysis

```
Input: ICDG, t, v
Output: T'
begin
    V ← {v}                                          ▹ Aliases of the variable v
    S ← BackwardAnalysis(ICDG, V, t)                 ▹ Alias statements
    return LastDefined(S, t)                         ▹ Definitions of v and its fields
Procedure BackwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▹ Traverse the ICDG in inverse execution order
        foreach t' s.t. (t', t) ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v and LHS(t') have a common prefix then
                ▹ t' is a definition of a variable in V
                S ← S ∪ {t'}                         ▹ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↑, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S
Procedure ForwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▹ Traverse the ICDG in execution order
        foreach t' s.t. (t, t') ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v is a prefix of LHS(t') then
                ▹ t' is a definition of a field of a variable in V
                S ← S ∪ {t'}                         ▹ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↓, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S
Procedure AliasAnalysis(d, V, t')
    ▹ Initialize with the original set of variables
    if d =↑ then
        V_b ← V; V_f ← ∅
    else if d =↓ then
        V_b ← ∅; V_f ← V
    foreach v ∈ V s.t. v and RHS(t') have a common prefix do
        ▹ LHS(t') is a new alias for v
        V_f ← V_f ∪ ExtendFields(LHS(t'))            ▹ Follow it forward
    foreach v ∈ V s.t. v is a prefix of LHS(t') do
        ▹ t' is a re-definition of a field of v
        ▹ Follow the assigned variable both backward and forward
        V_b ← V_b ∪ RHS(t')
        V_f ← V_f ∪ RHS(t')
    foreach v ∈ V s.t. LHS(t') is a prefix or equal to v do
        ▹ t' is a full re-definition of v
        if d =↑ then
            V_b ← V_b \ {v}                          ▹ Do not search before the definition
            ▹ Follow the assigned variable both backward and forward
            V_b ← V_b ∪ ExtendFields(RHS(t'))
            V_f ← V_f ∪ ExtendFields(RHS(t'))
        else if d =↓ then
            V_f ← V_f \ {v}                          ▹ Do not search for new variables
    return <V_b,V_f>
```

Search for field definitions <u>backward</u>

Field Alias Analysis

```
Input: ICDG, t, v
Output: T'
begin
    V ← {v}                                          ▹ Aliases of the variable v
    S ← BackwardAnalysis(ICDG, V, t)                 ▹ Alias statements
    return LastDefined(S, t)                         ▹ Definitions of v and its fields

Procedure BackwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▹ Traverse the ICDG in inverse execution order
        foreach t' s.t. (t', t) ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v and LHS(t') have a common prefix then
                ▹ t' is a definition of a variable in V
                S ← S ∪ {t'}                         ▹ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↑, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S

Procedure ForwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▹ Traverse the ICDG in execution order
        foreach t' s.t. (t, t') ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v is a prefix of LHS(t') then
                ▹ t' is a definition of a field of a variable in V
                S ← S ∪ {t'}                         ▹ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↓, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S

Procedure AliasAnalysis(d, V, t')
    ▹ Initialize with the original set of variables
    if d =↑ then
        V_b ← V; V_f ← ∅
    else if d =↓ then
        V_b ← ∅; V_f ← V
    foreach v ∈ V s.t. v and RHS(t') have a common prefix do
        ▹ LHS(t') is a new alias for v
        V_f ← V_f ∪ ExtendFields(LHS(t'))                ▹ Follow it forward
    foreach v ∈ V s.t. v is a prefix of LHS(t') do
        ▹ t' is a re-definition of a field of v
        ▹ Follow the assigned variable both backward and forward
        V_b ← V_b ∪ RHS(t')
        V_f ← V_f ∪ RHS(t')
    foreach v ∈ V s.t. LHS(t') is a prefix or equal to v do
        ▹ t' is a full re-definition of v
        if d =↑ then
            V_b ← V_b \ {v}                          ▹ Do not search before the definition
            ▹ Follow the assigned variable both backward and forward
            V_b ← V_b ∪ ExtendFields(RHS(t'))
            V_f ← V_f ∪ ExtendFields(RHS(t'))
        else if d =↓ then
            V_f ← V_f \ {v}                          ▹ Do not search for new variables
    return <V_b,V_f>
```

Search for field definitions <u>forward</u>

Field Alias Analysis

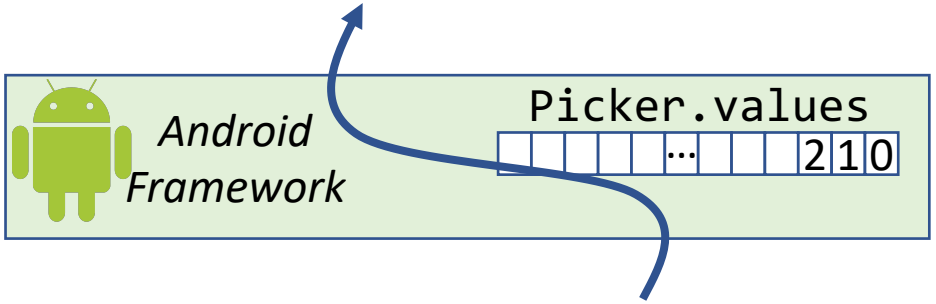
```
Input: ICDG, t, v
Output: T'
begin
    V ← {v}                                          ▷ Aliases of the variable v
    S ← BackwardAnalysis(ICDG, V, t)                 ▷ Alias statements
    return LastDefined(S, t)                         ▷ Definitions of v and its fields

Procedure BackwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▷ Traverse the ICDG in inverse execution order
        foreach t' s.t. (t', t) ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v and LHS(t') have a common prefix then
                ▷ t' is a definition of a variable in V
                S ← S ∪ {t'}                         ▷ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↑, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S

Procedure ForwardAnalysis(ICDG, V, t)
    begin
        S ← ∅
        ▷ Traverse the ICDG in execution order
        foreach t' s.t. (t, t') ∈ ICDG do
            if method(t) ≠ method(t') then
                V ← ChangeScope(V, method(t), method(t'))
            if ∃v ∈ V s.t. v is a prefix of LHS(t') then
                ▷ t' is a definition of a field of a variable in V
                S ← S ∪ {t'}                         ▷ Add the definition statement
            <V_b,V_f> ← AliasAnalysis(↓, V, t')
            S ← S ∪ BackwardAnalysis(ICDG, V_b, t')
            S ← S ∪ ForwardAnalysis(ICDG, V_f, t')
        return S
```

```
Procedure AliasAnalysis(d, V, t')
    ▷ Initialize with the original set of variables
    if d =↑ then
        V_b ← V; V_f ← ∅
    else if d =↓ then
        V_b ← ∅; V_f ← V
    foreach v ∈ V s.t. v and RHS(t') have a common prefix do
        ▷ LHS(t') is a new alias for v
        V_f ← V_f ∪ ExtendFields(LHS(t'))            ▷ Follow it forward
    foreach v ∈ V s.t. v is a prefix of LHS(t') do
        ▷ t' is a re-definition of a field of v
        ▷ Follow the assigned variable both backward and forward
        V_b ← V_b ∪ RHS(t')
        V_f ← V_f ∪ RHS(t')
    foreach v ∈ V s.t. LHS(t') is a prefix or equal to v do
        ▷ t' is a full re-definition of v
        if d =↑ then
            V_b ← V_b \ {v}                          ▷ Do not search before the definition
            ▷ Follow the assigned variable both backward and forward
            V_b ← V_b ∪ ExtendFields(RHS(t'))
            V_f ← V_f ∪ ExtendFields(RHS(t'))
        else if d =↓ then
            V_f ← V_f \ {v}                          ▷ Do not search for new variables
    return <V_b,V_f>
```

Decide on variables to search for and the direction

Framework Data-Flows

Goal: Data-flows inside the Framework



How to find data-flows inside framework w/o instrumentation?

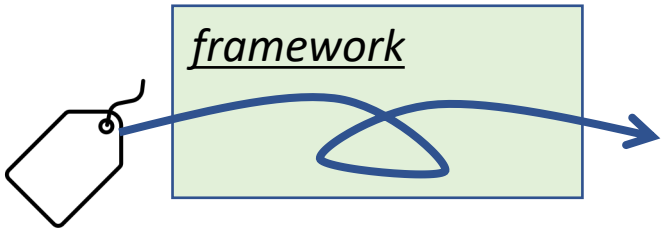
Taint-Propagation Models

StubDroid [1] & FlowDroid [2]

[1] Arzt, Steven, and Eric Bodden. "StubDroid: automatic inference of precise data-flow summaries for the android framework." ICSE, 2016
[2] Arzt, Steven, et al. "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." PLDI, 2014

Taint-Propagation Models

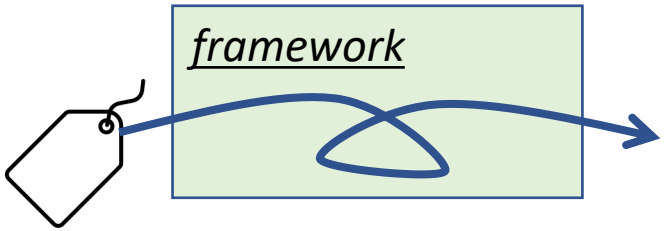
StubDroid [1] & FlowDroid [2]



[1] Arzt, Steven, and Eric Bodden. "StubDroid: automatic inference of precise data-flow summaries for the android framework." ICSE, 2016
[2] Arzt, Steven, et al. "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." PLDI, 2014

Taint-Propagation Models

StubDroid [1] & FlowDroid [2]

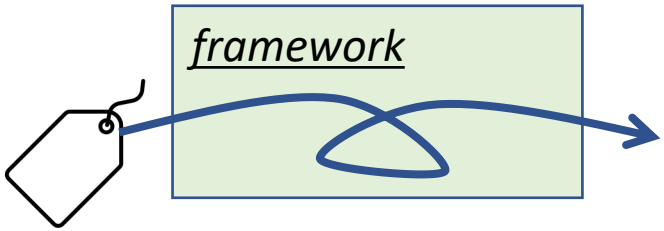


Express how taint propagates

[1] Arzt, Steven, and Eric Bodden. "StubDroid: automatic inference of precise data-flow summaries for the android framework." ICSE, 2016
[2] Arzt, Steven, et al. "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." PLDI, 2014

Taint-Propagation Models

StubDroid [1] & FlowDroid [2]



Express how taint propagates

From:

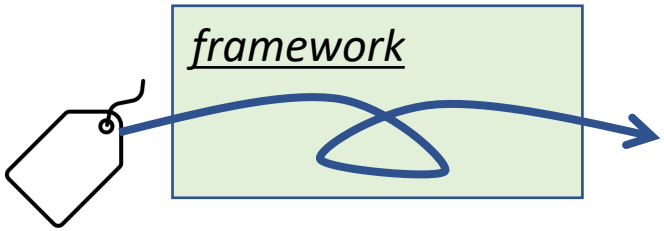
- Parameters
- Receiver

} + Fields

[1] Arzt, Steven, and Eric Bodden. "StubDroid: automatic inference of precise data-flow summaries for the android framework." ICSE, 2016
[2] Arzt, Steven, et al. "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." PLDI, 2014

Taint-Propagation Models

StubDroid [1] & FlowDroid [2]



Express how taint propagates

From:

- Parameters
- Receiver

} + Fields

To:

- Parameters
- Receiver
- Return

} + Fields

[1] Arzt, Steven, and Eric Bodden. "StubDroid: automatic inference of precise data-flow summaries for the android framework." ICSE, 2016
[2] Arzt, Steven, et al. "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." PLDI, 2014

Taint-Propagation Models

```
Array::set(index, value) {
    this.inA[index] = value;
}
```

Taint-Propagation Models

```
Array::set(index, value) {
  this.inA[index] = value;
}
```

Taint-propagation:

`value` → `this.inA`

Taint-Propagation Models

```
Array::set(index, value) {
  this.inA[index] = value;
}
```

Taint-propagation:

`value` → `this.inA`

`index` ✗

`this` ✗

Taint-Propagation Models

```
Array::set(index, value) {
    this.inA[index] = value;
}
```

```
Log::info(value) {
    print(value);
}
```

Taint-propagation:

value → this.inA

index ✗→

this ✗→

Taint-Propagation Models

```
Array::set(index, value) {
  this.inA[index] = value;
}
```

```
Log::info(value) {
  print(value);
}
```

Taint-propagation:

`value` → `this.inA`

`index` ✗

`this` ✗

`value` ✗

Taint-Propagation Models

```
Array::set(index, value) {
  this.inA[index] = value;
}
```

Taint-propagation:

value → this.inA

index ✗

this ✗

```
Log::info(value) {
  print(value);
}
```

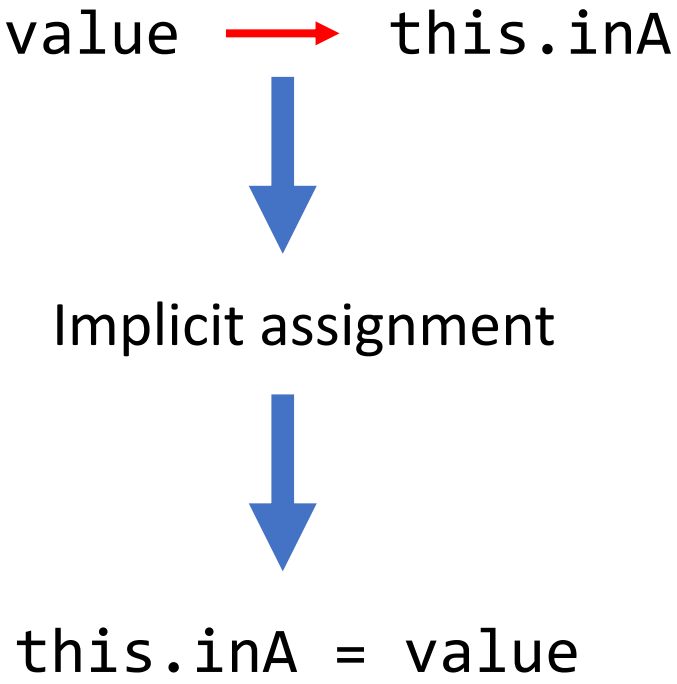
value ✗

Accurate taint propagation rules for Java and Android methods

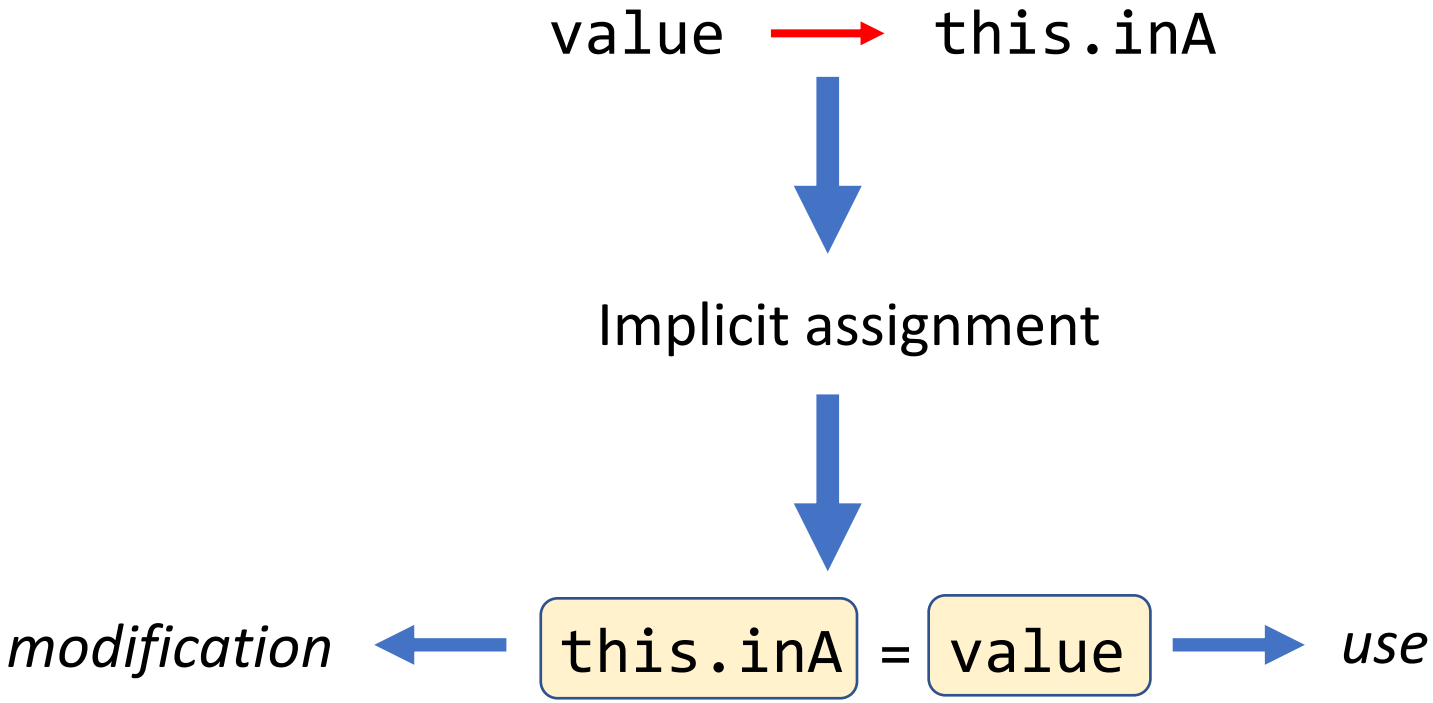
Modeling w/ Implicit Assignemnts

`value` → `this.inA`

Modeling w/ Implicit Assignemnts



Modeling w/ Implicit Assignemnts



Framework Modeling: Example

Code	Model
`x = 5`	
`array.set(1, x)`	
`y = array.get(1)`	`y = array.inA`

Framework Modeling: Example

Code

```
x = 5

array.set(1, x)

y = array.get(1)
```

Model

```
y = array.inA
```

Framework Modeling: Example

Code		Model
`x = 5`		
`array.set(1, x)`	→	`array.inA = x`
`y = array.get(1)`	→	`y = array.inA`

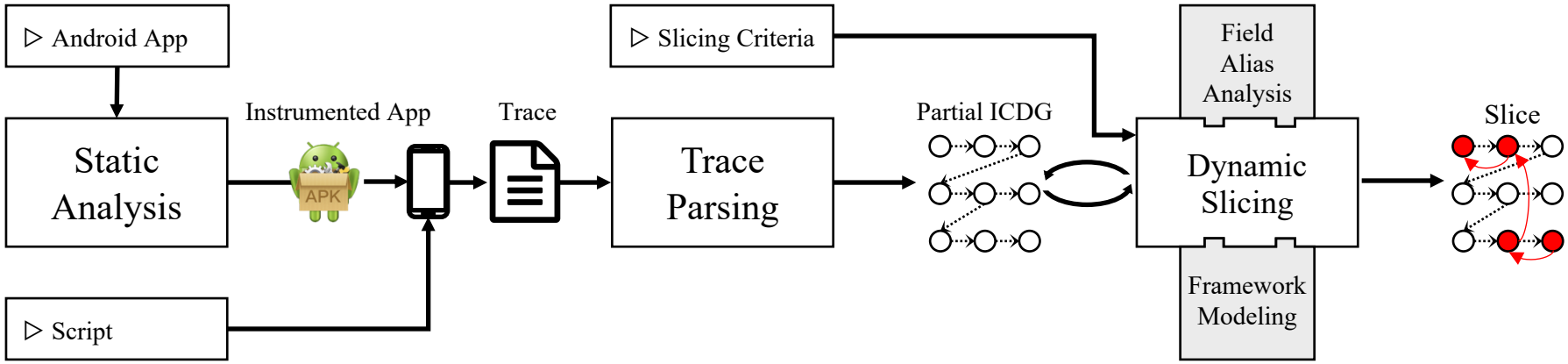
Framework Modeling: Example

Code		Model
`x = 5`		
`array.set(1, x)`	→	`array.inA = x`
`y = array.get(1)`	→	`y = array.inA`

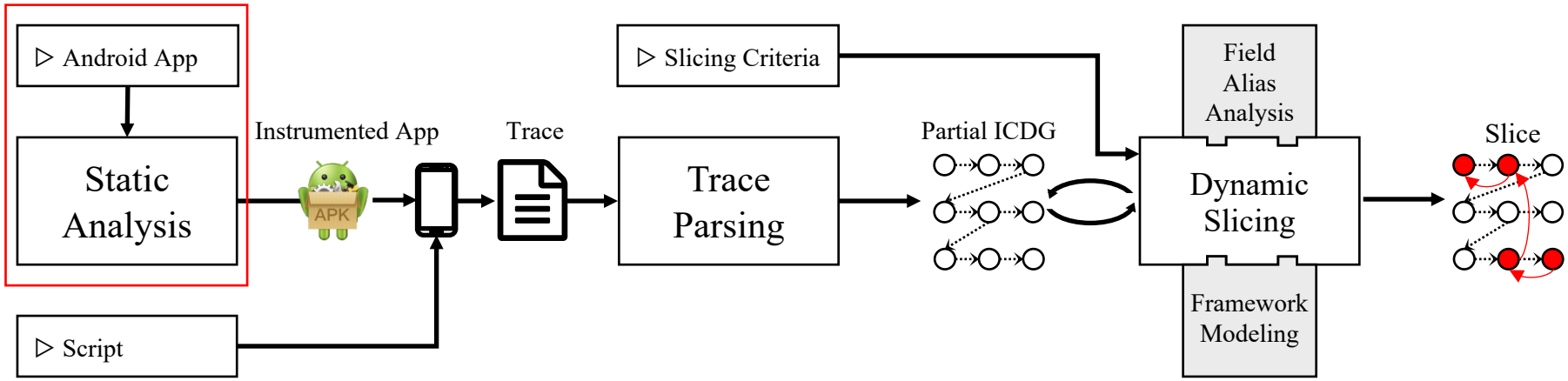
Framework Modeling: Example

Code		Model
`x = 5`		
`array.set(1, x)`	→	`array.inA = x`
`y = array.get(1)`	→	`y = array.inA`

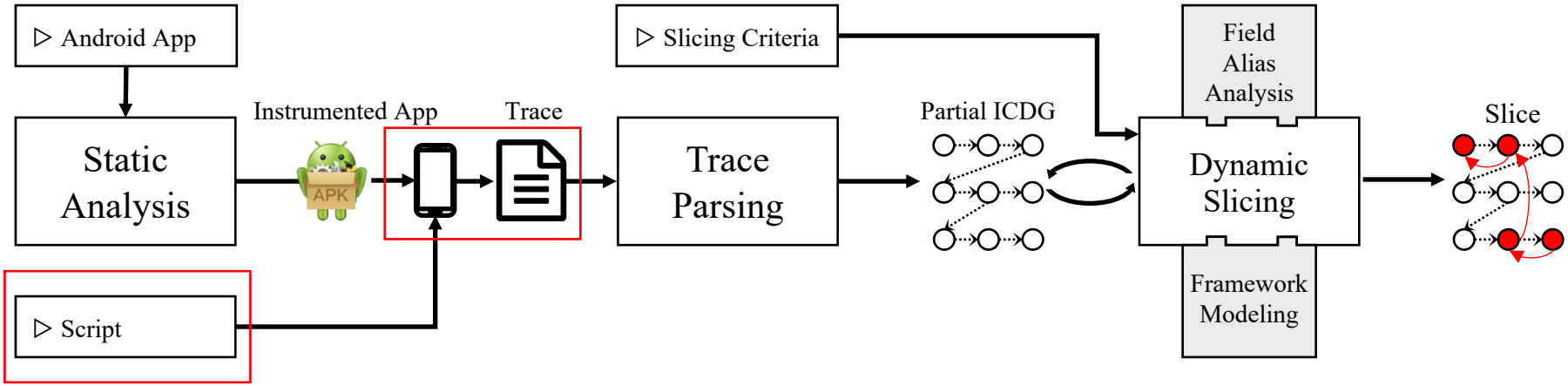
Our Solution: MANDOLINE



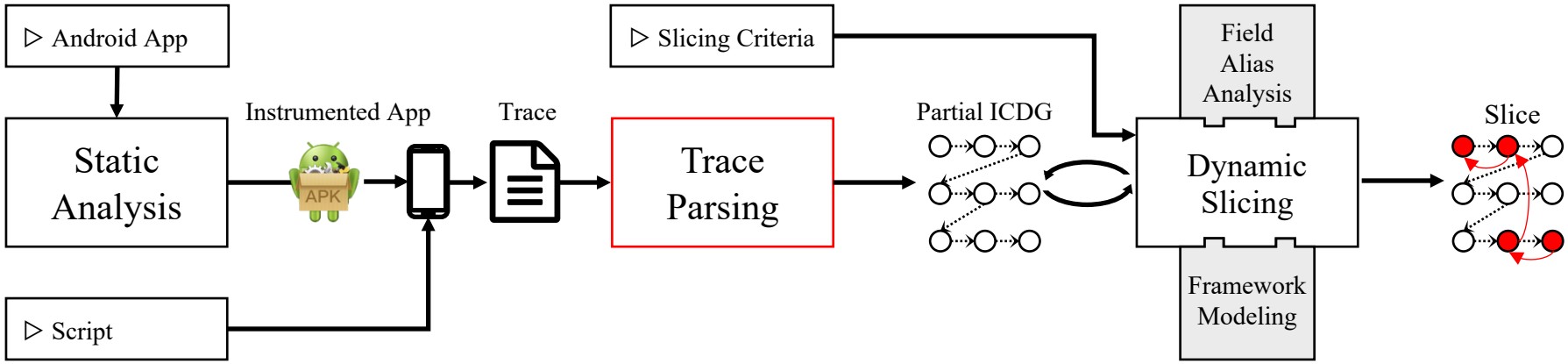
Our Solution: MANDOLINE



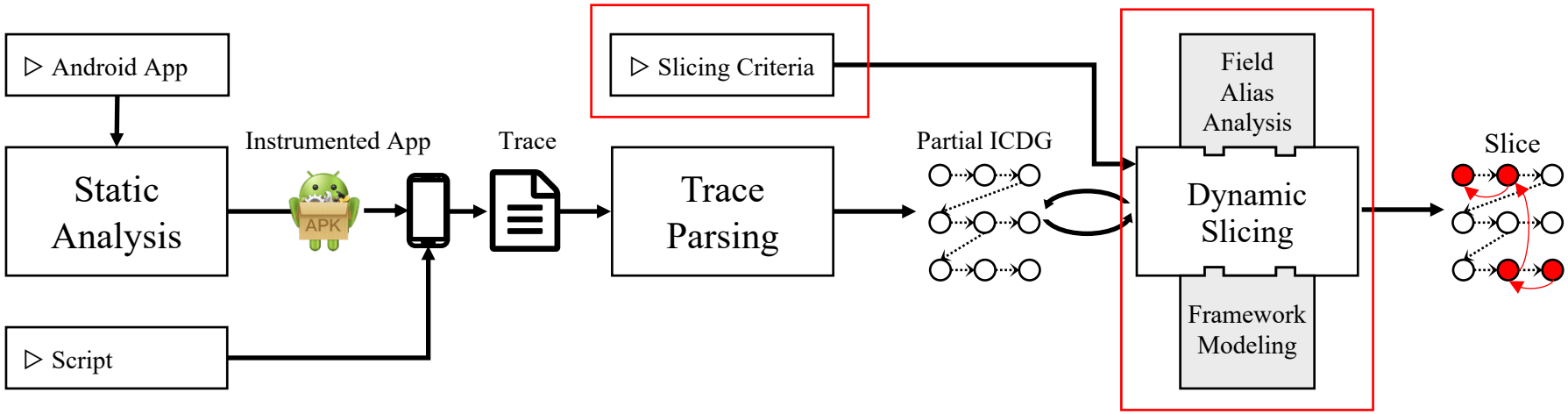
Our Solution: MANDOLINE



Our Solution: MANDOLINE



Our Solution: MANDOLINE



Evaluation

RQ1: Accuracy

vs. state-of-the-art

RQ2: Performance

Subjects

12 Apps from DroixBench [3] and ReCDroid [4]

- Faulty apps w/ crashes
- Reproducible crashes
- Crashes = slicing criteria

[3] Tan, Shin Hwei, et al. "Repairing crashes in android apps." ICSE, 2018.
[4] Zhao, Yu, et al. "Recdroid: automatically reproducing android application crashes from bug reports." ICSE, 2019

Baseline

AndroidSlicer [5] (state-of-the-art)

- First slicer for Android apps
- Inter-callback data-flows
- Misses data-flows in fields and framework

AndroidSlicer++:

- Fixed implementation issues
- Improved instrumentation
- Allows fair comparison

[5] Azim, Tanzirul, et al. "Dynamic slicing for android." ICSE, 2019

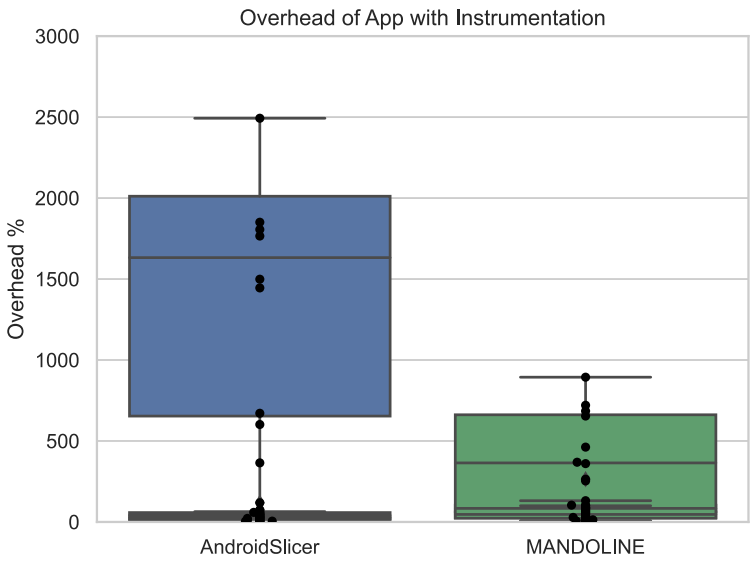
Benchmark

Manually produced slices

- Cross validated by two researchers
- First Android slices benchmark
- State-of-the-art [5] only compared slice to trace sizes

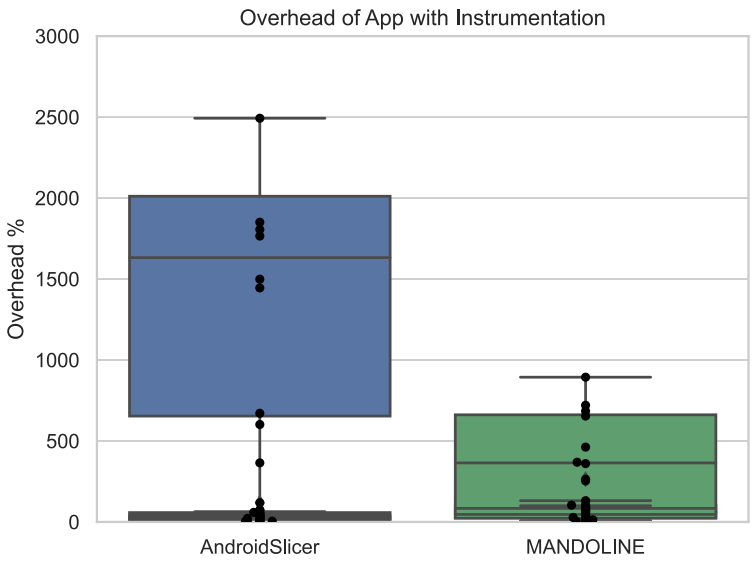
[5] Azim, Tanzirul, et al. "Dynamic slicing for android." ICSE, 2019

Accuracy and Peformance



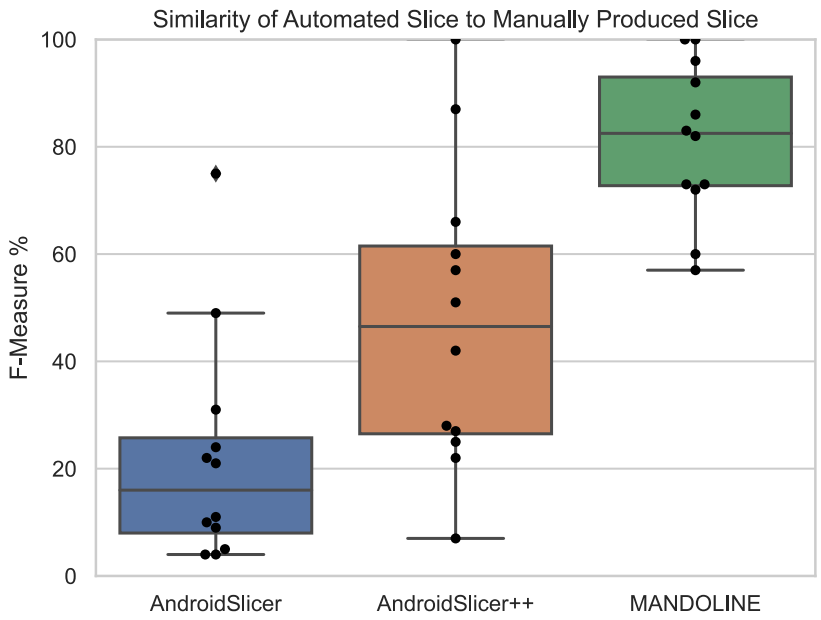
Accuracy and Peformance

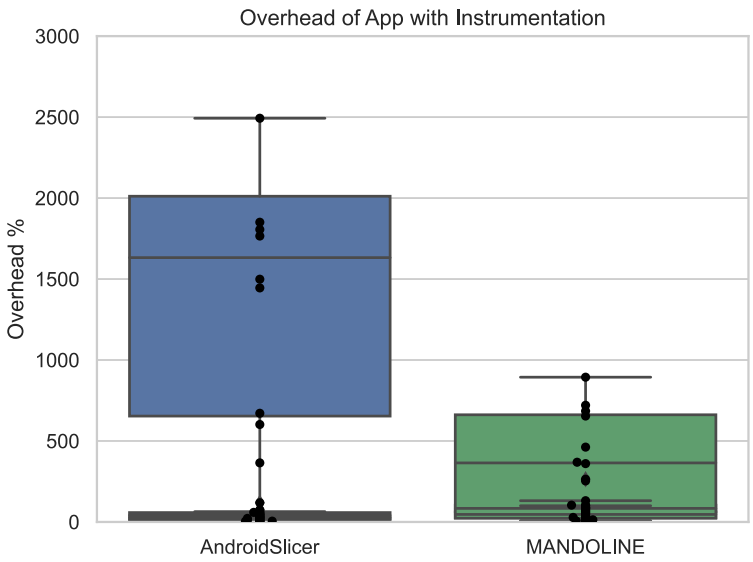
RQ1



Accuracy and Peformance

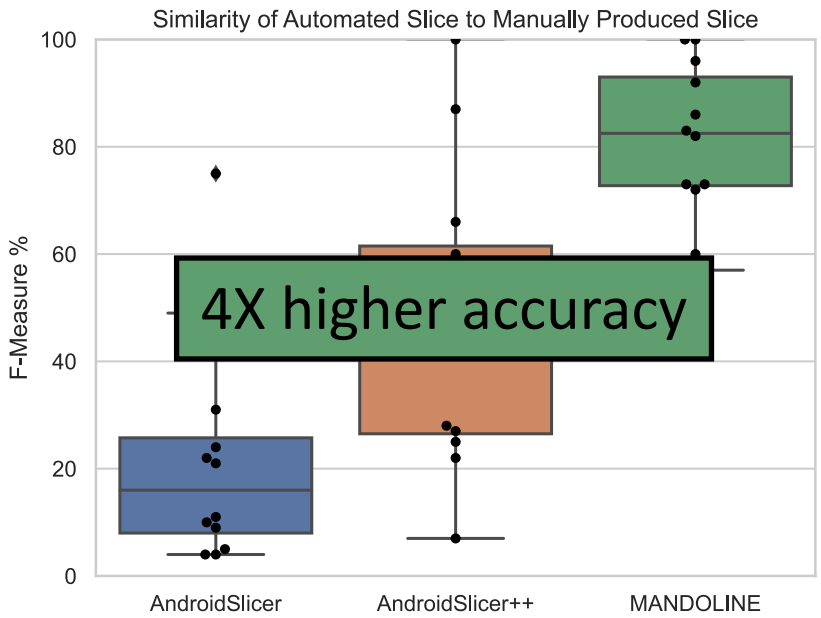
RQ1

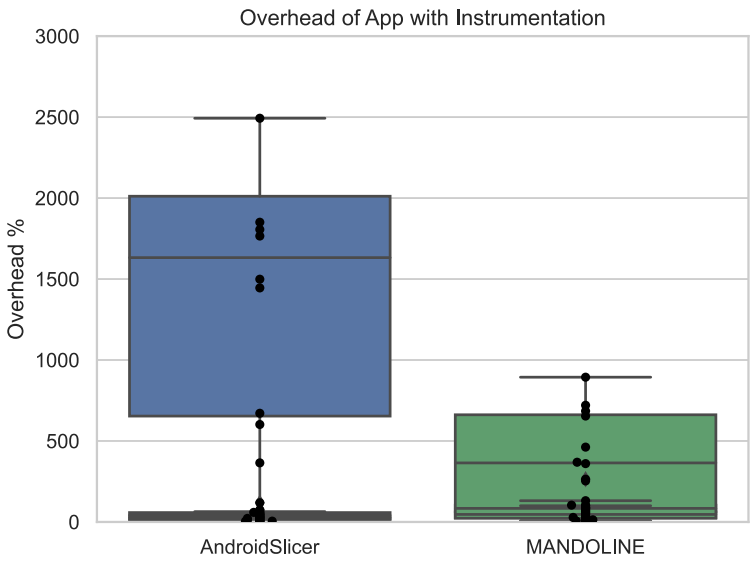




Accuracy and Peformance

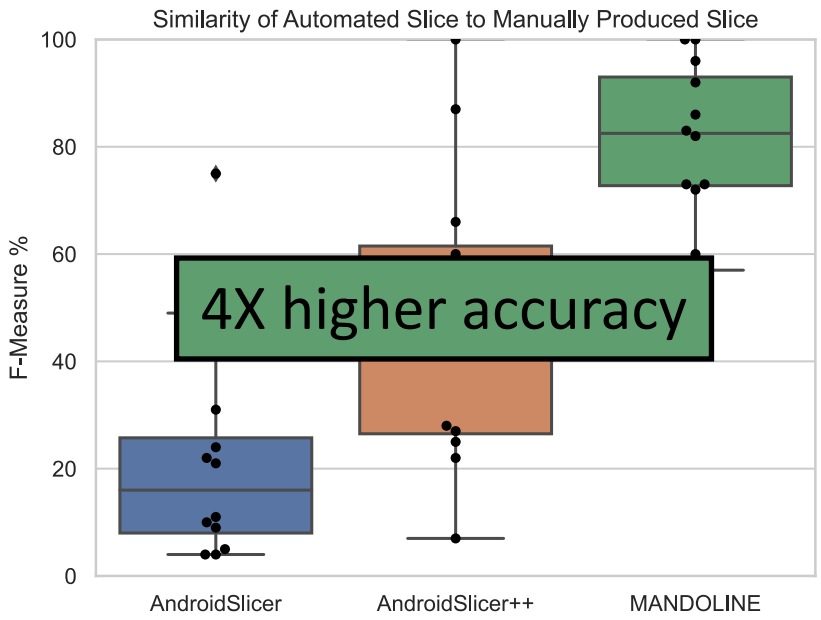
RQ1



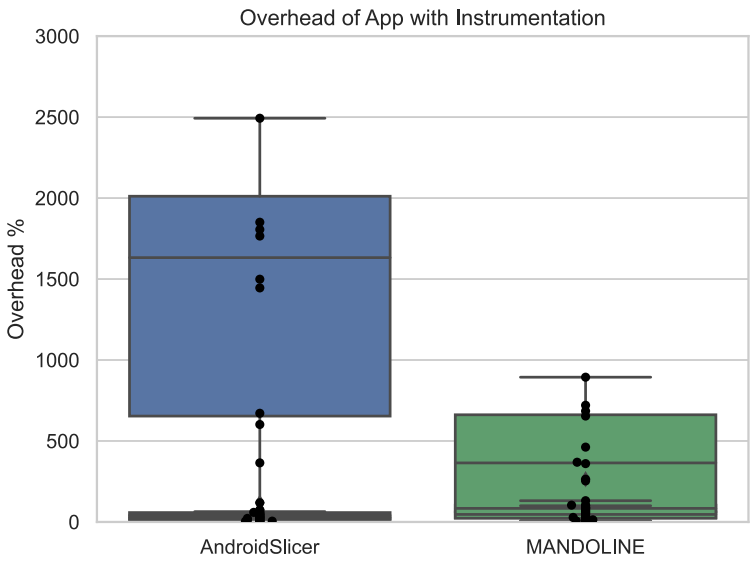


Accuracy and Peformance

RQ1

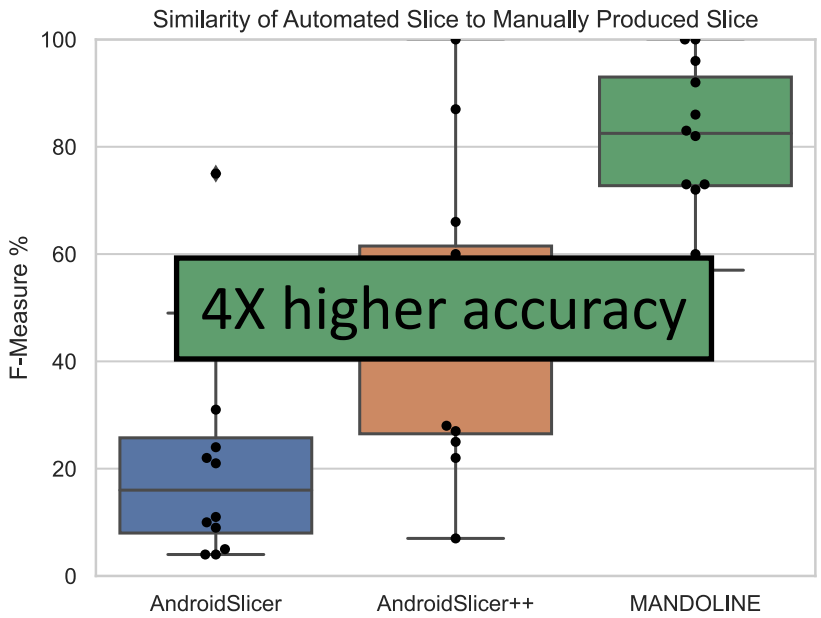


RQ2

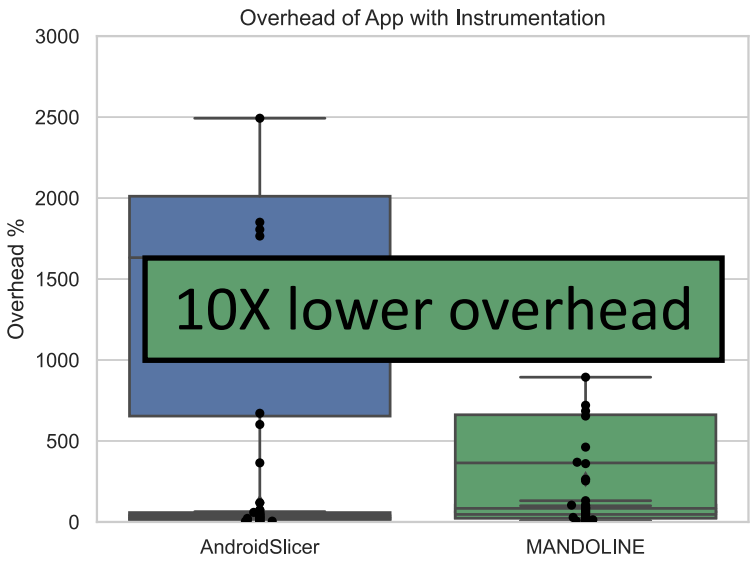


Accuracy and Peformance

RQ1

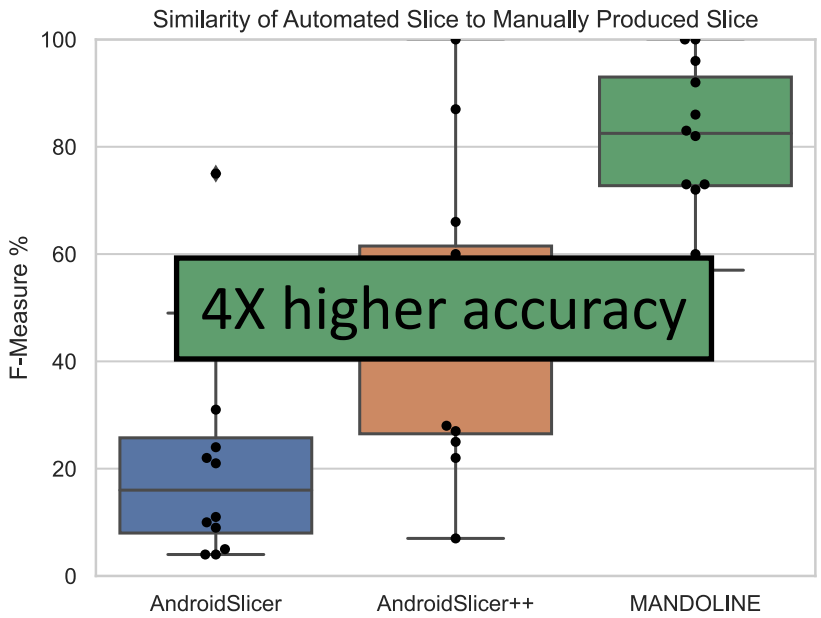


RQ2

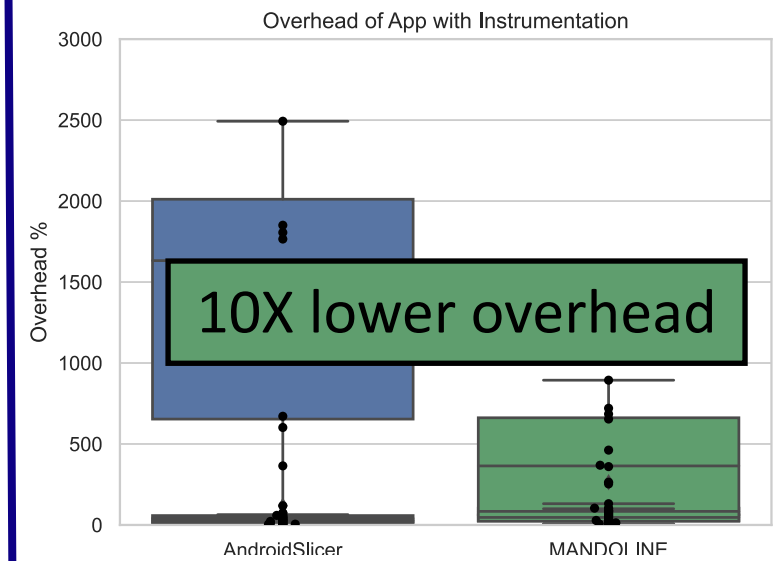


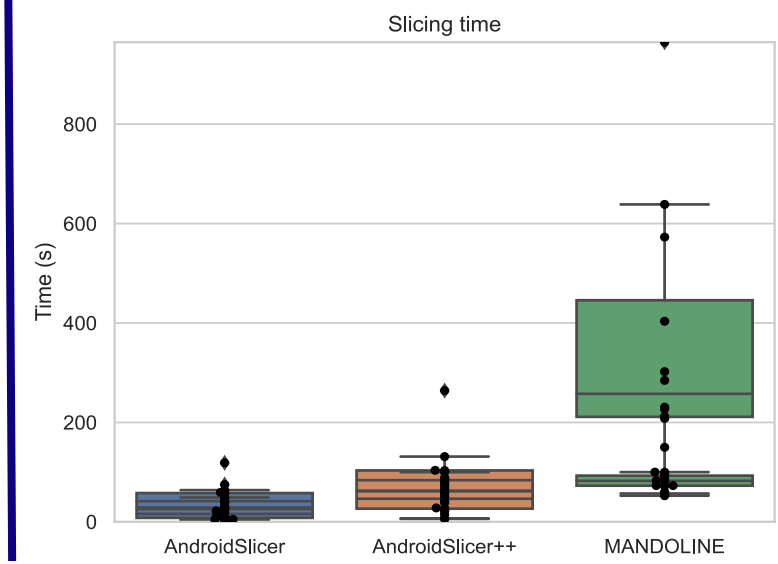
Accuracy and Peformance

RQ1



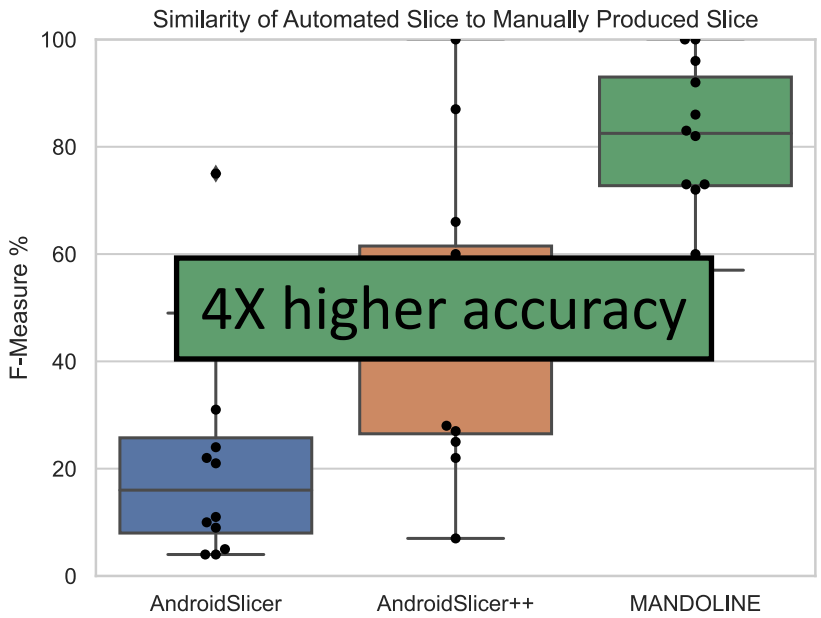
RQ2



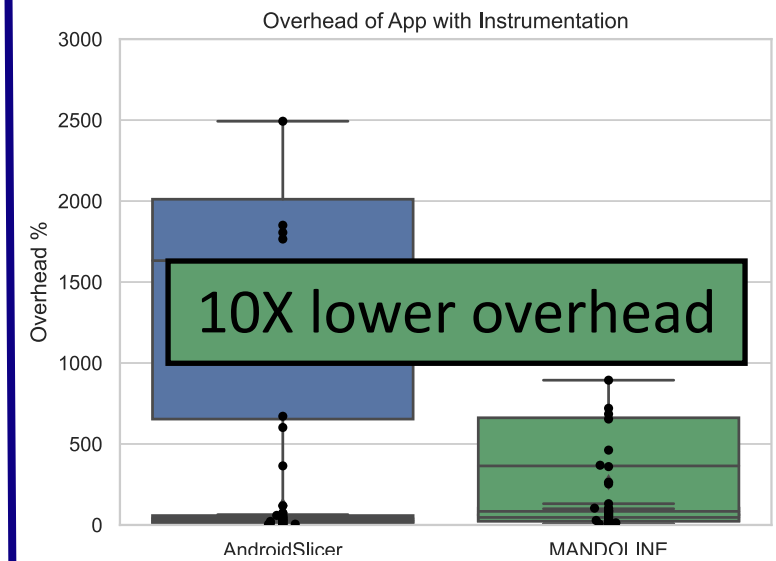


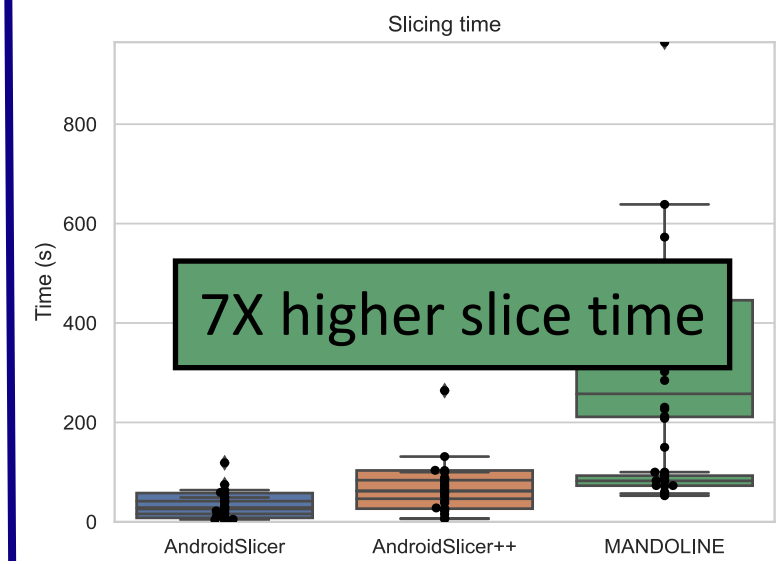
Accuracy and Peformance

RQ1



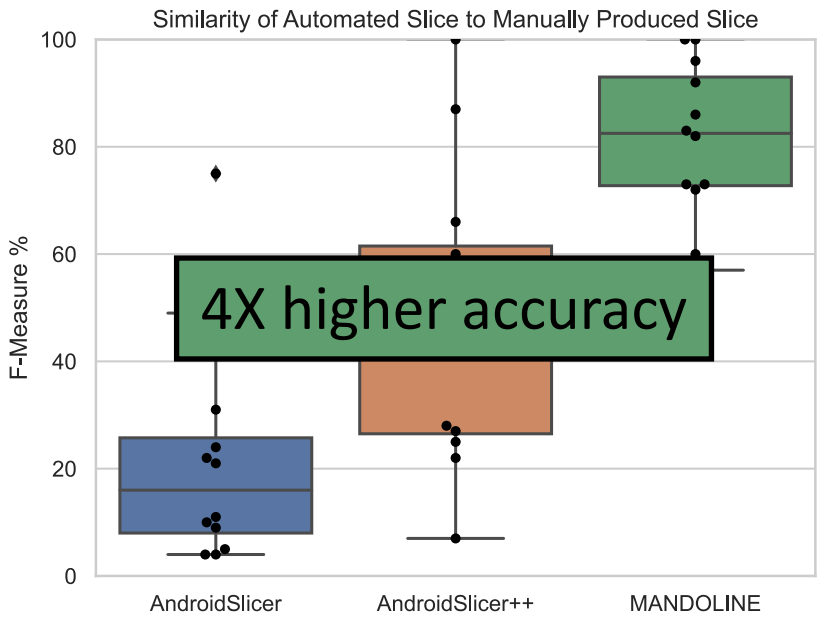
RQ2





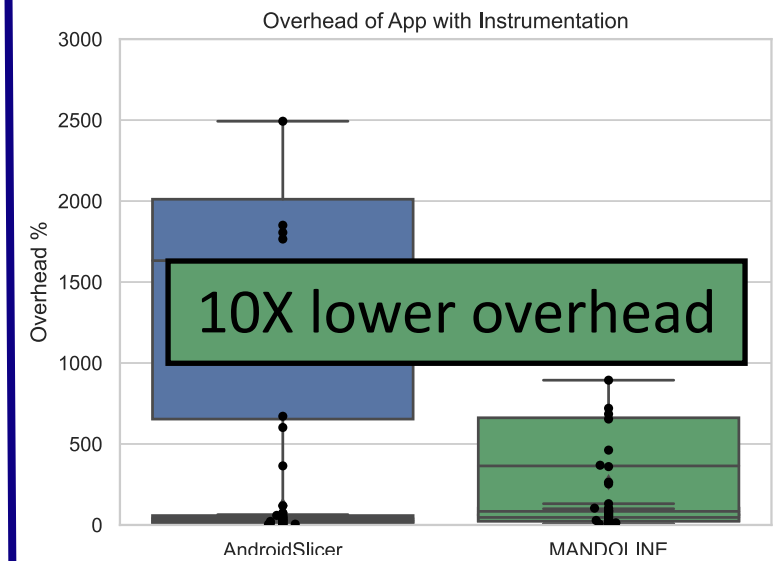
Accuracy and Peformance

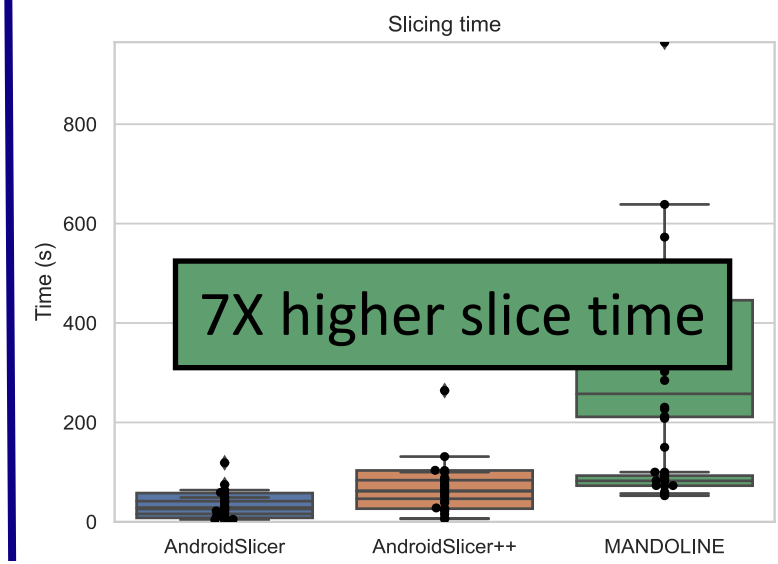
RQ1



Acceptable slice time given huge accuracy improvement

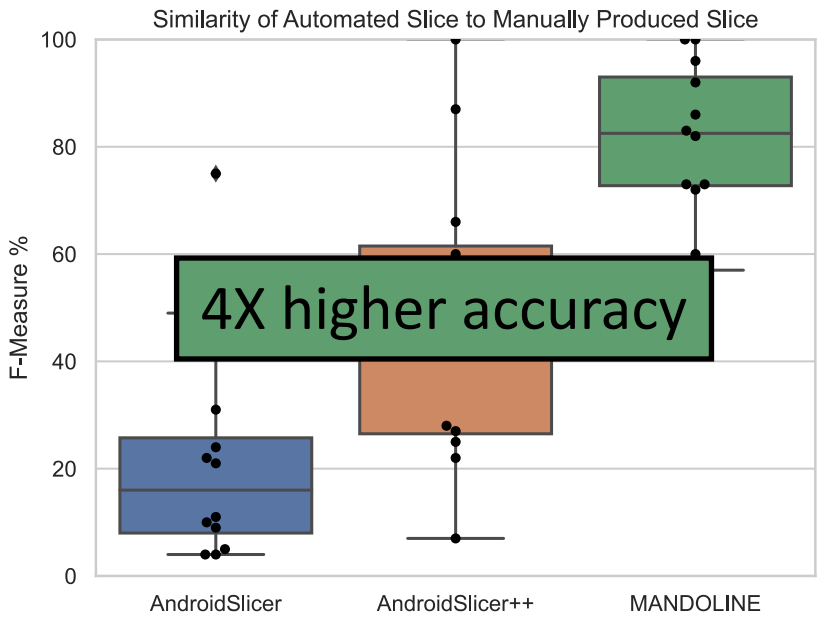
RQ2





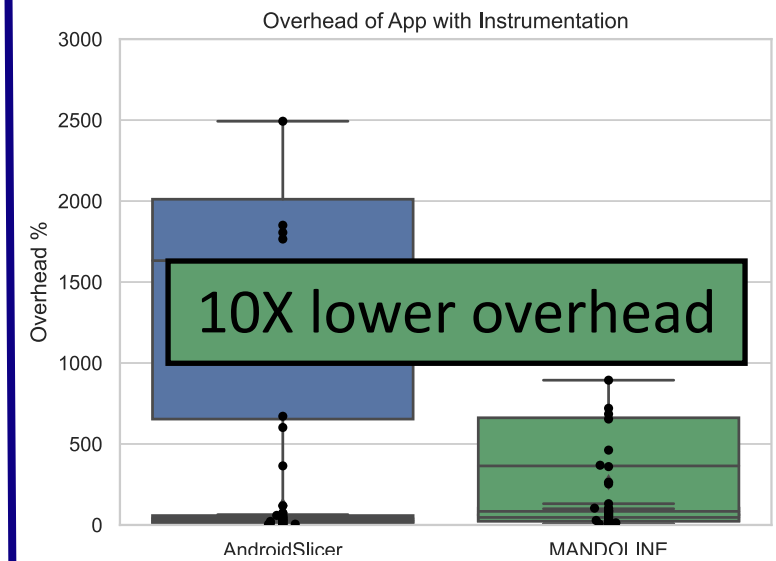
Accuracy and Peformance

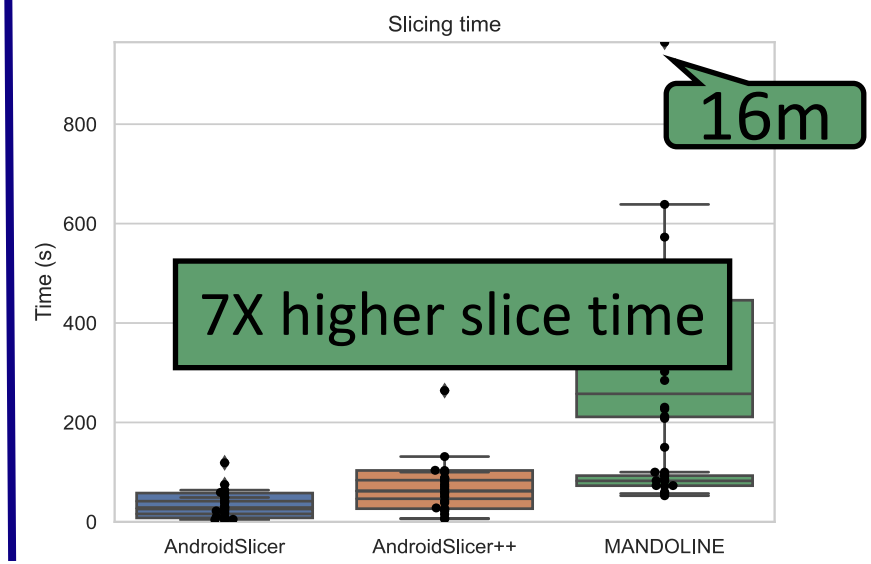
RQ1



Acceptable slice time given huge accuracy improvement

RQ2



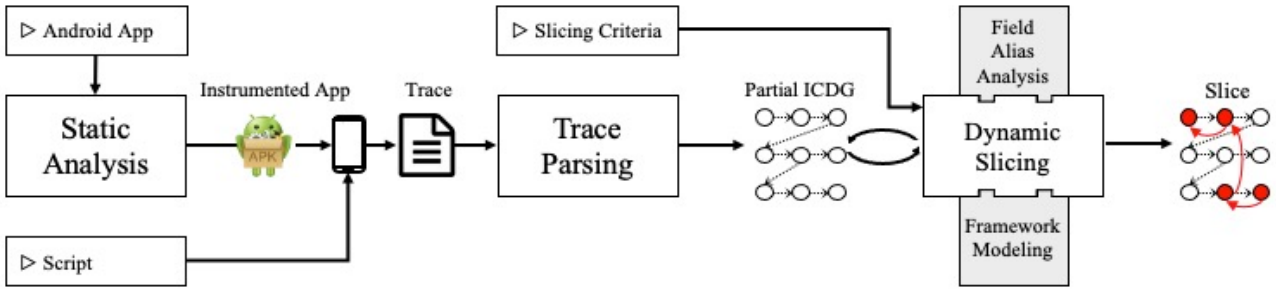


Summary

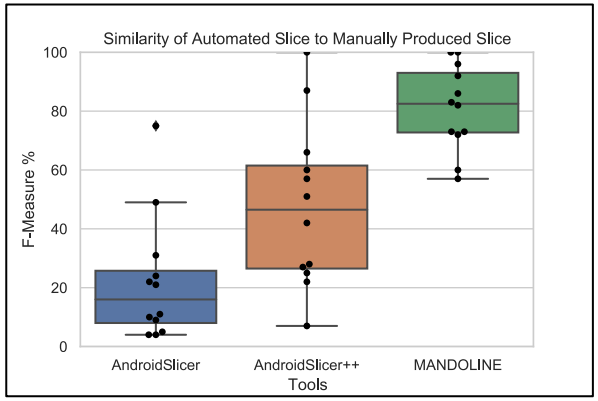
Insights

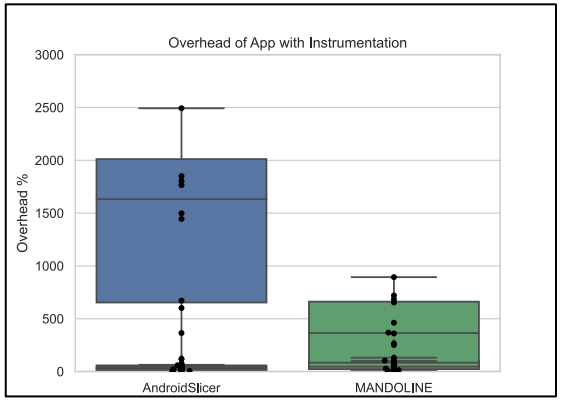
- Extract field data flows from the trace using alias analysis
- Use framework models from static analysis

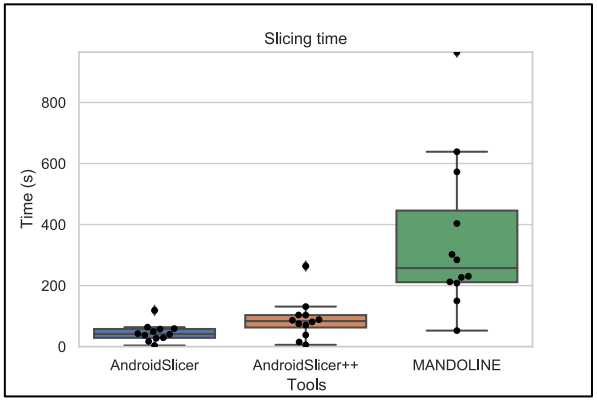
Approach



Results







Code + Benchmark suite: https://resess.github.io/PaperAppendices/Mandoline/

Contact: Khaled Ahmed, UBC, khaledea@ece.ubc.ca