IvySyn

Automated Vulnerability Discovery in Deep Learning Frameworks

Neophytos Christou[1] Di Jin[1] Vaggelis Atlidakis[1] Baishakhi Ray[2]
Vasileios P. Kemerlis[1]

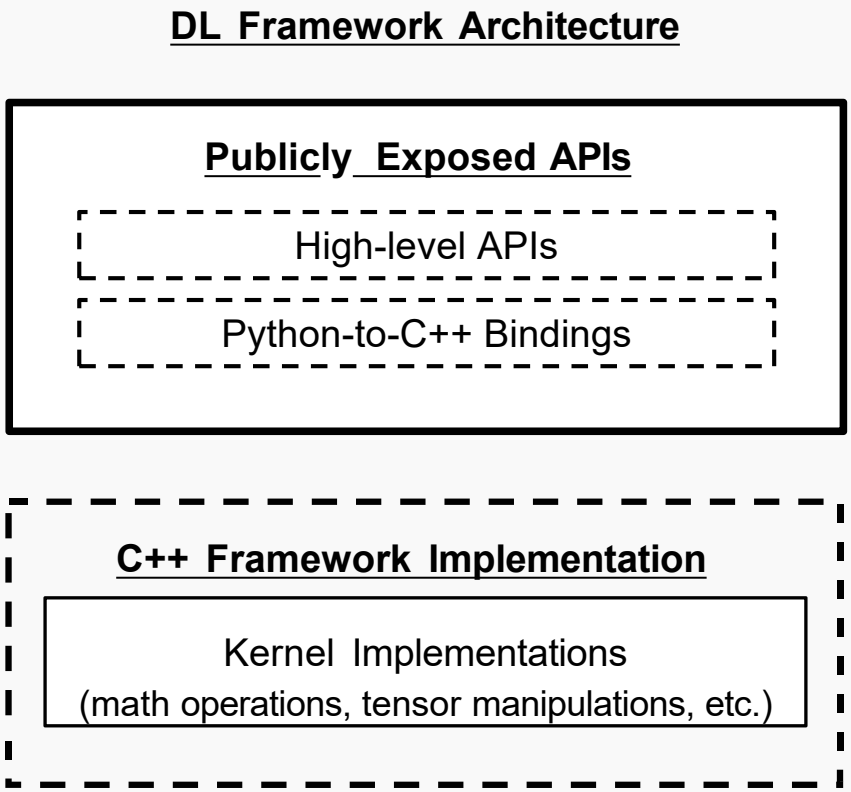
August 9, 2023

[1]Secure Systems Laboratory (SSL)
Department of Computer Science
Brown University

[2]ARiSE Lab
Department of Computer Science
Columbia University

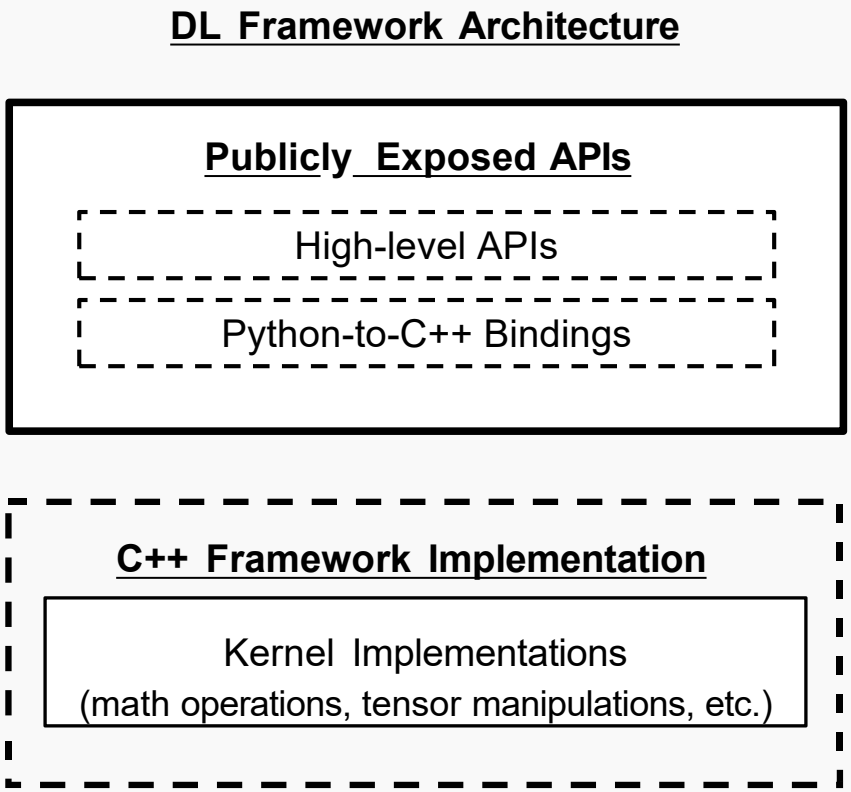
Presented by Giannis Oikonomidis
sdi1600120@di.uoa.gr

Motivation


DL Framework Architecture
Publicly Exposed APIs
High-level APIs
Python-to-C++ Bindings
C++ Framework Implementation
Kernel Implementations
(math operations, tensor manipulations, etc.)

Motivation

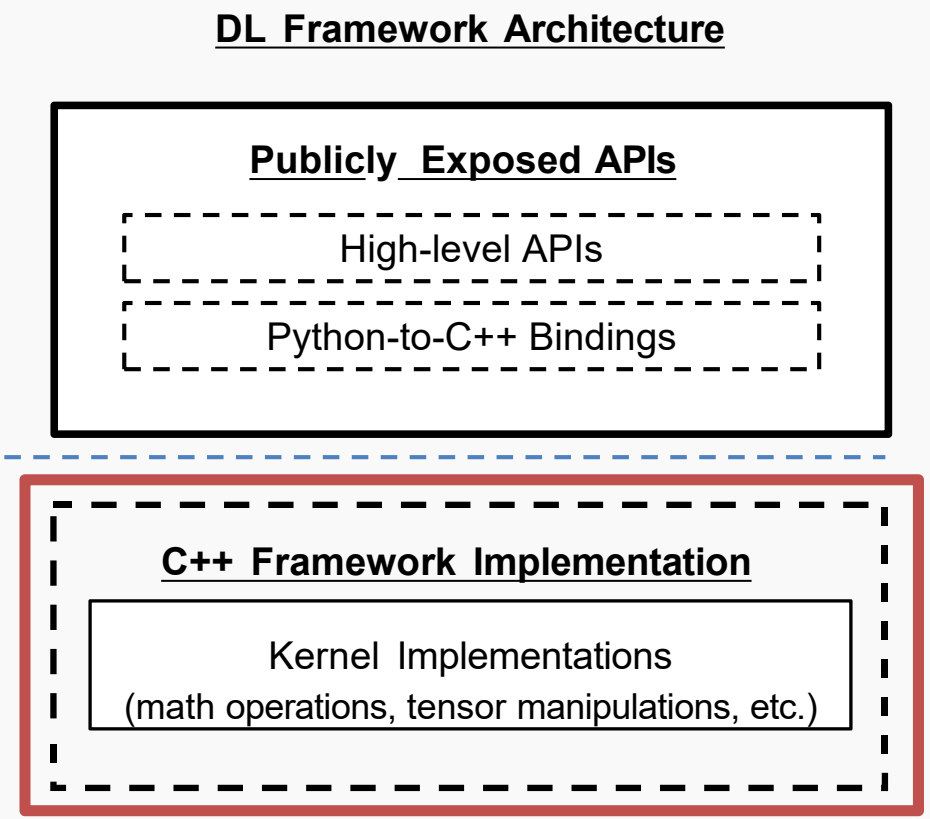
- DL Frameworks offers several high-level APIs for performing DL-specific operations, usually written in managed languages, like Python.
- For performance, the essential parts of such operations(kernel) are implemented in low-level, memory-unsafe languages, like C/C++.



Motivation

- Developer-Accessible
- High-level language (ex. Python)
- Memory-safe

- Memory-unsafe code region (raw memory operations)
- Not accessible directly
- BUT accessible and possibly exploitable through the Bindings and APIs



Motivation

- Core DL framework implementation → memory-unsafe languages
- In 2021–2022 alone, TensorFlow had more than 280 CVE numbers assigned for (potentially-exploitable) vulnerabilities related to memory safety issues.

TFSA-2021-126	Use after free in boosted trees creation
TFSA-2021-125	Heap buffer overflow in `FractionalAvgPoolGrad`
TFSA-2021-124	Segfault and heap buffer overflow in `{Experimental,}DatasetToTFRecord`
TFSA-2021-123	Null pointer dereference in `UncompressElement`
TFSA-2021-122	Incorrect validation of `SaveV2` inputs
TFSA-2021-121	Null pointer dereference in `SparseTensorSliceDataset`
TFSA-2021-120	Bad alloc in `StringNGrams` caused by integer conversion

Goals and Past Approaches

IvySyn's Goals

Goals and Past Approaches

IvySyn's Goals

- Automatically uncover *memory safety* and *fatal runtime* errors in DL frameworks

Goals and Past Approaches

IvySyn's Goals

- Automatically uncover *memory safety* and *fatal runtime* errors in DL frameworks
- Help framework developers *identify* and *fix* the uncovered bugs

Goals and Past Approaches

IvySyn's Goals

- Automatically uncover *memory safety* and *fatal runtime* errors in DL frameworks
- Help framework developers *identify* and *fix* the uncovered bugs

Past Approaches

Goals and Past Approaches

IvySyn's Goals

- Automatically uncover *memory safety* and *fatal runtime* errors in DL frameworks
- Help framework developers *identify* and *fix* the uncovered bugs

Past Approaches

- Are not aimed at finding memory safety errors

Goals and Past Approaches

IvySyn's Goals

- Automatically uncover *memory safety* and *fatal runtime* errors in DL frameworks
- Help framework developers *identify* and *fix* the uncovered bugs

Past Approaches

- Are not aimed at finding memory safety errors
- Are not *fully automated*
  - Custom fuzzing drivers, domain-expert annotations, …

IvySyn Overview

The general approach is based on these questions:

- Are there any offending inputs to low-level APIs that can trigger memory errors in native DL kernels?
- Are there any high-level APIs that can propagate the above offending inputs to low-level kernel code?

IvySyn Overview

The general approach is based on these questions:

- Are there any offending inputs to low-level APIs that can trigger memory errors in native DL kernels?
- Are there any high-level APIs that can propagate the above offending inputs to low-level kernel code?

Bottom-up approach for testing DL framework implementations

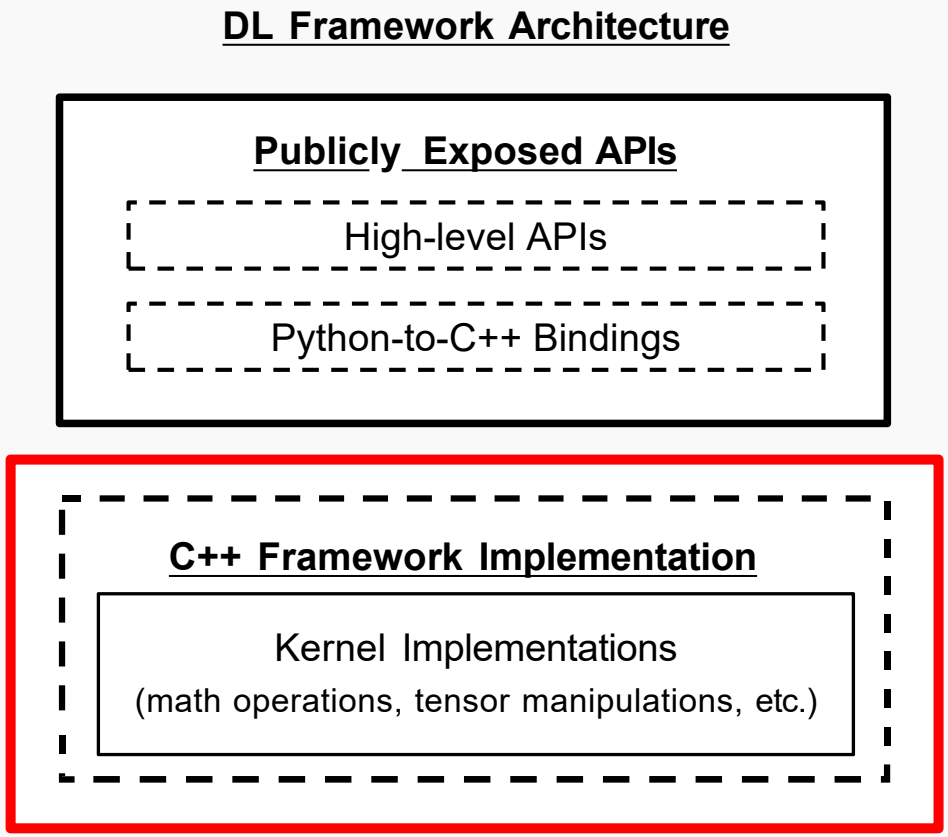
IvySyn Overview

< IvySyn's Approach

IvySyn Overview

< IvySyn's Approach

- Fuzz the native implementation of DL frameworks



IvySyn Overview

< IvySyn's Approach

- Fuzz the native implementation of DL frameworks
- Automatically synthesize *Proof-of-Vulnerability (PoV)* snippets

PoV synthesized by IvySyn

```
import tensorflow as tf

indices = tf.constant([], shape=[2,0],
                      dtype=tf.int64)
values = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
dense_shape = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
default_value = tf.constant(0, shape=[],
                      dtype=tf.int64)
tf.raw_ops.SparseFillEmptyRows(
                indices=indices,
                values=values,
                dense_shape=dense_shape,
                default_value=default_value)
```

PoV triggers crash!

```
$ python3 pov.py
segmentation fault (core dumped)
```

IvySyn Overview

< IvySyn's Approach

- Fuzz the native implementation of DL frameworks
- Automatically synthesize *Proof-of-Vulnerability (PoV)* snippets

Achievements

PoV synthesized by IvySyn

```
import tensorflow as tf

indices = tf.constant([], shape=[2,0],
                      dtype=tf.int64)
values = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
dense_shape = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
default_value = tf.constant(0, shape=[],
                      dtype=tf.int64)
tf.raw_ops.SparseFillEmptyRows(
                indices=indices,
                values=values,
                dense_shape=dense_shape,
                default_value=default_value)
```

PoV triggers crash!

```
$ python3 pov.py
segmentation fault (core dumped)
```

IvySyn Overview

< IvySyn's Approach

- Fuzz the native implementation of DL frameworks
- Automatically synthesize *Proof-of-Vulnerability (PoV)* snippets

Achievements

- Uncovered 61 previously-unknown vulnerabilities

PoV synthesized by IvySyn

```
import tensorflow as tf

indices = tf.constant([], shape=[2,0],
                      dtype=tf.int64)
values = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
dense_shape = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
default_value = tf.constant(0, shape=[],
                      dtype=tf.int64)
tf.raw_ops.SparseFillEmptyRows(
                indices=indices,
                values=values,
                dense_shape=dense_shape,
                default_value=default_value)
```

PoV triggers crash!

```
$ python3 pov.py
segmentation fault (core dumped)
```

IvySyn Overview

< IvySyn's Approach

- Fuzz the native implementation of DL frameworks
- Automatically synthesize *Proof-of-Vulnerability (PoV)* snippets

Achievements

- Uncovered 61 previously-unknown vulnerabilities
- Assigned with 39 unique CVEs

PoV synthesized by IvySyn

```
import tensorflow as tf

indices = tf.constant([], shape=[2,0],
                      dtype=tf.int64)
values = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
dense_shape = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
default_value = tf.constant(0, shape=[],
                      dtype=tf.int64)
tf.raw_ops.SparseFillEmptyRows(
                indices=indices,
                values=values,
                dense_shape=dense_shape,
                default_value=default_value)
```

PoV triggers crash!

```
$ python3 pov.py
segmentation fault (core dumped)
```

Outline

IvySyn Architecture - Instrumentation

Evaluation

Conclusion

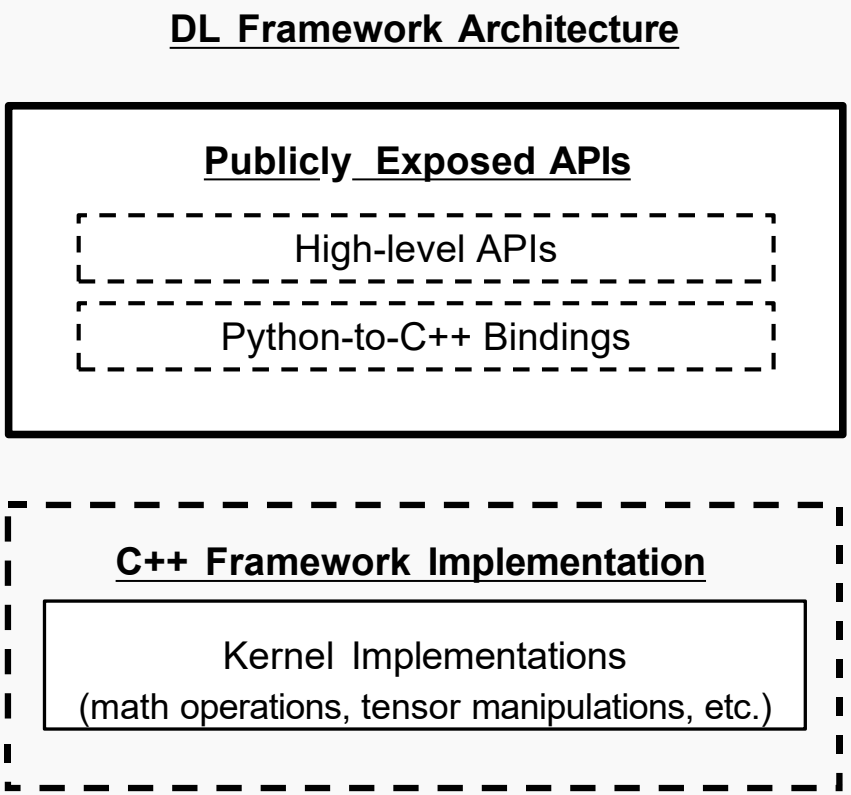
Future Plans

IvySyn Design Principles

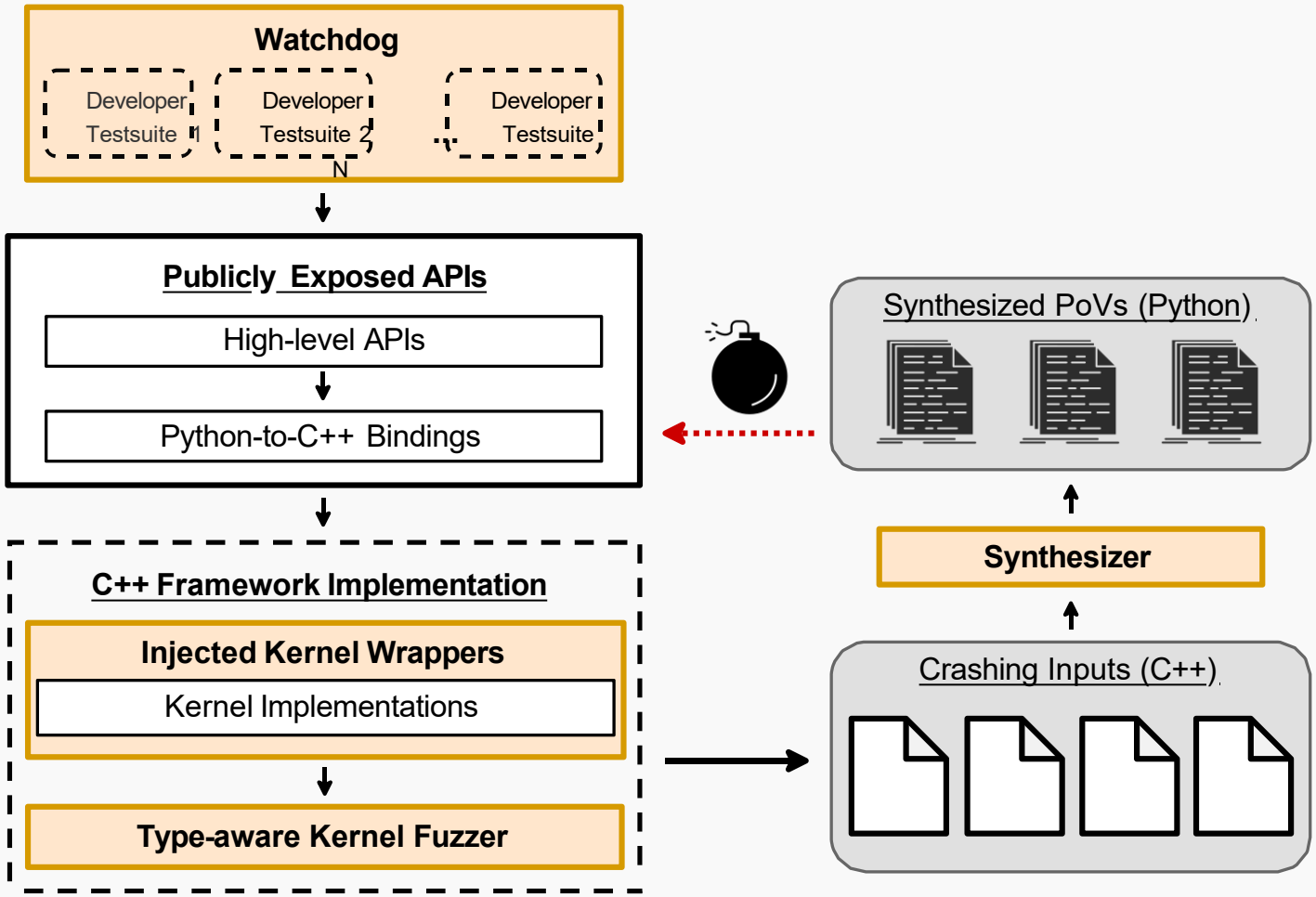
IvySyn:

- Focuses on two famous DL frameworks: TensorFlow and Pytorch.
- Assumes its target kernels are strongly-typed and avoid shared state.
- Assumes the existence of developer-provided unit tests.
- Leverages the DL framework mappings between high-level and low-level APIs.

IvySyn Architecture Overview



IvySyn Architecture Overview



IvySyn Architecture → Extracting Kernel Implementations

In DL frameworks like PyTorch and TensorFlow every operation (Conv2D, ReLU, MatMul, etc.) must be registered with the framework.
Example (Tensor-flow style):

```
REGISTER_KERNEL_BUILDER(
     Name("Conv2D").Device(DEVICE_CPU),
     Conv2DOp);
```

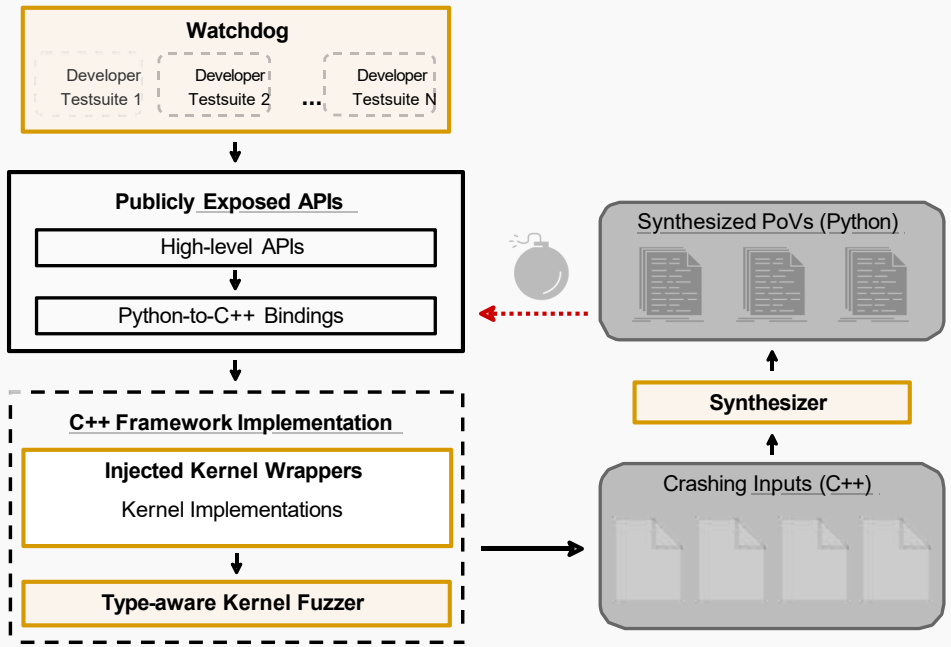
This gives IvySyn structured metadata for fuzzing:

```
Kernel: Conv2D
Arguments:
    Tensor input
    Tensor weight
    int stride
    int padding
Device: CPU
```

This is enough to construct type-aware mutations and start the fuzzing sessions.

IvySyn Architecture → Kernel Instrumentation

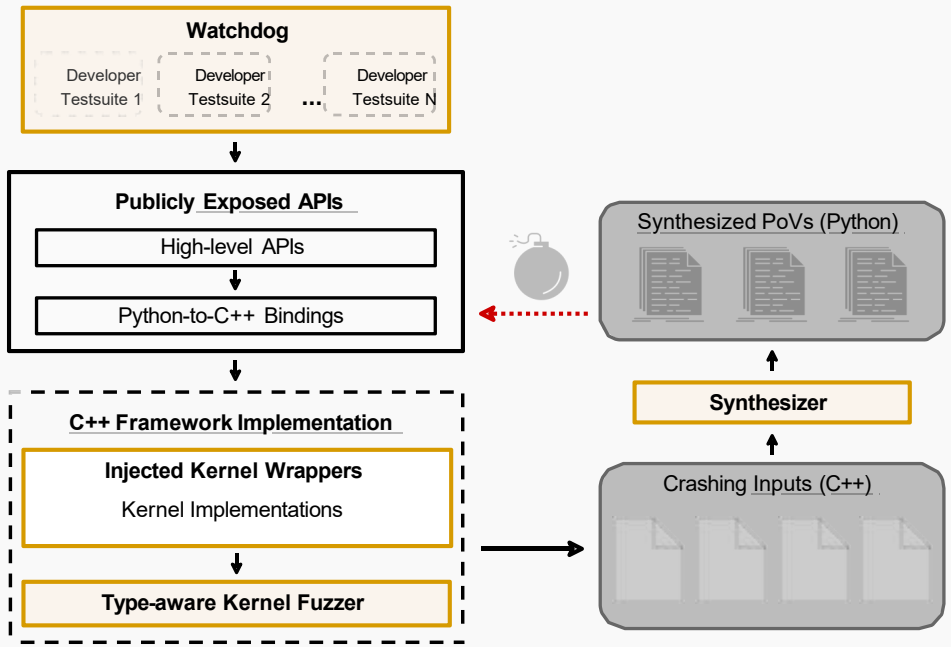
- IvySyn automatically wraps each framework's *kernels* with fuzzing drivers



IvySyn Architecture → Kernel Instrumentation

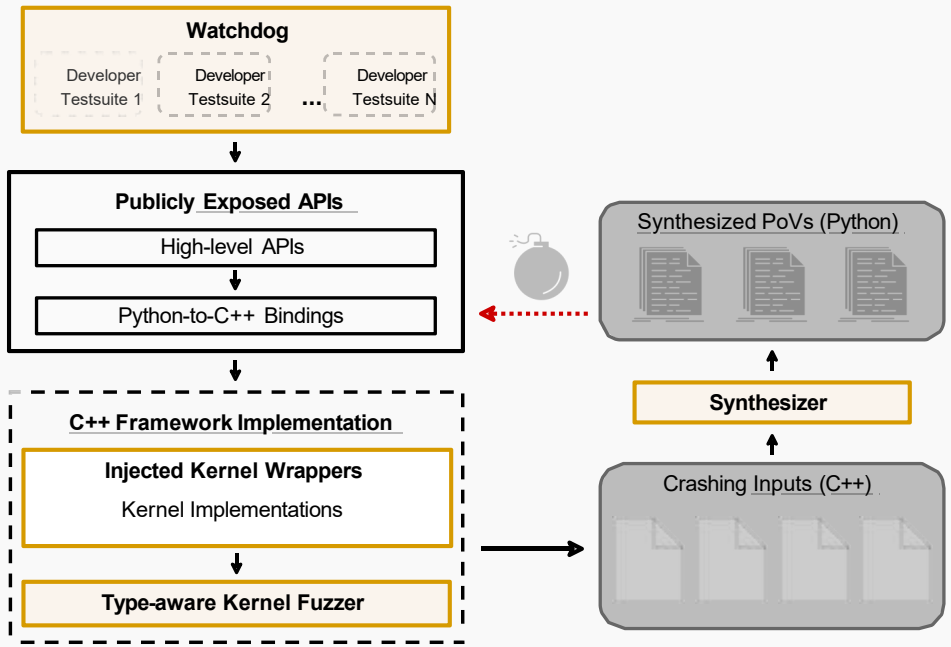
- IvySyn automatically wraps each framework's *kernels* with fuzzing drivers

How?



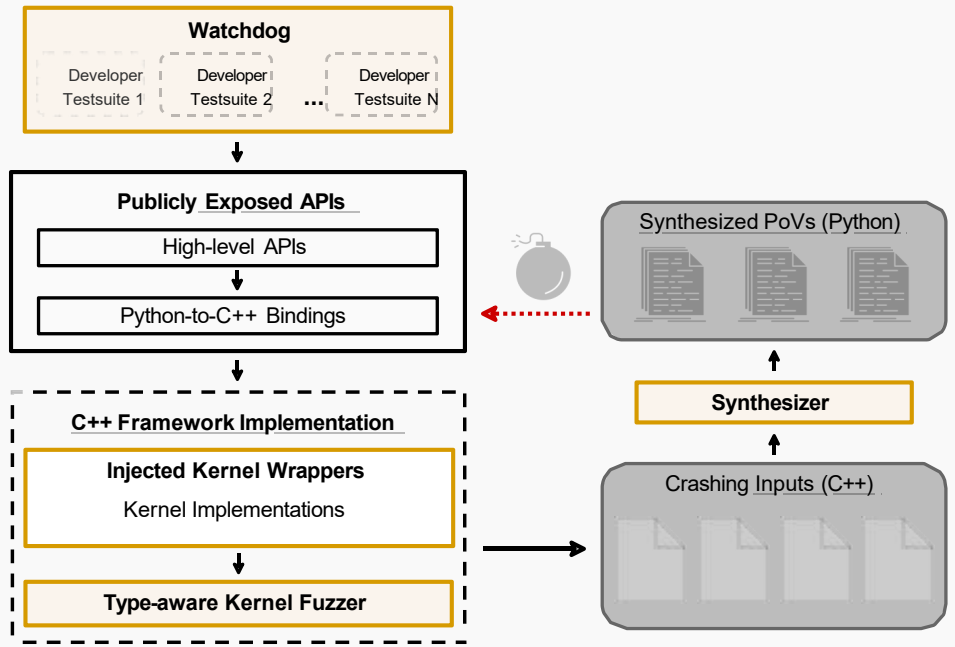
IvySyn Architecture → Kernel Instrumentation

- IvySyn automatically wraps each framework's *kernels* with fuzzing drivers
- Original kernel foo() ⇢ do_foo()
- Creates instrumented kernel named foo()



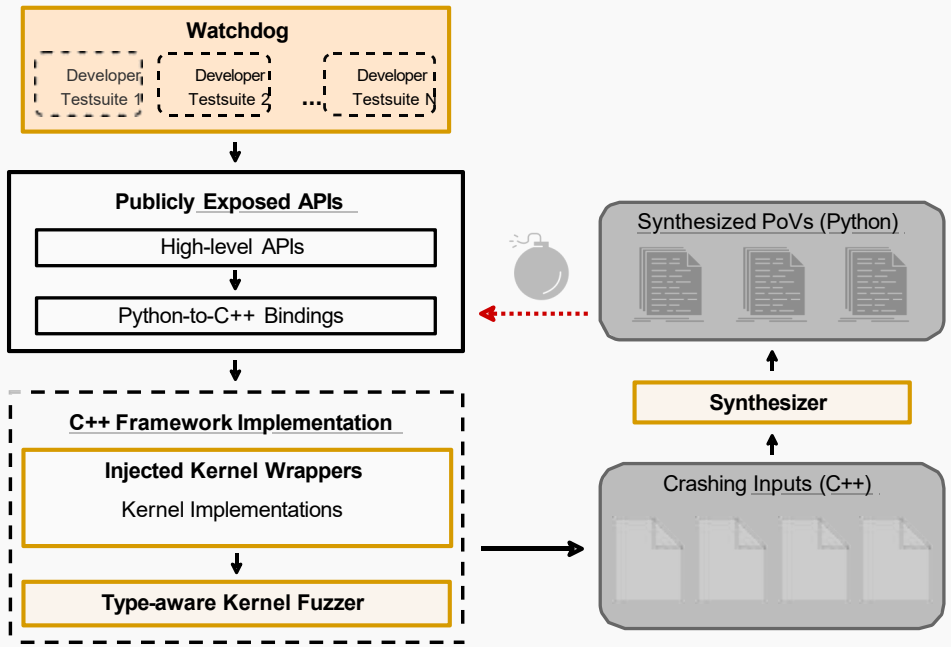
IvySyn Architecture → Kernel Instrumentation

```cpp
Tensor cosine_similarity(const Tensor& x1,
    const Tensor& x2, int64_t dim, double eps){

    if (fuzzing::already_fuzzing ||
        fuzzing::was_fuzzed("cosine_similarity"))
        // Return the original function result
        return do_cosine_similarity(x1, x2,
            dim, eps);
    fuzzing::already_fuzzing = true;

    Tensor retval, x1_fuzz, x2_fuzz;
    int64_t dim_fuzz; double eps_fuzz;
    std::vector<std::string> types =
      {"Tensor&", "Tensor&",
       "int64_t", "double"};
    std::vector<void *> args{};
    args.push_back((void *) &x1);
    ... // Repeat for x2, dim, and eps
    // Initialize fuzzer with the original
    // args and types
    fuzzing::Fuzzer fuzzer =
      fuzzing::Fuzzer("cosine_similarity",
        types, args);

    while (fuzzer.has_more_mutations(true)){
        // Get the next combination of inputs
        x1_fuzz = fuzzer.get_next_mut_tensor();
        ... // Repeat for x2, dim, and eps
        try {
            // Invoke the original function
            fuzzer.mut_start_time();
            do_cosine_similarity(x1_fuzz,
                x2_fuzz, dim_fuzz, eps_fuzz);
            fuzzer.mut_end_time(false);
        } catch (...) { ... }
    }
    fuzzing::already_fuzzing = false;

    // Return the original function result
    return do_cosine_similarity(x1, x2,
        dim, eps);
}
```



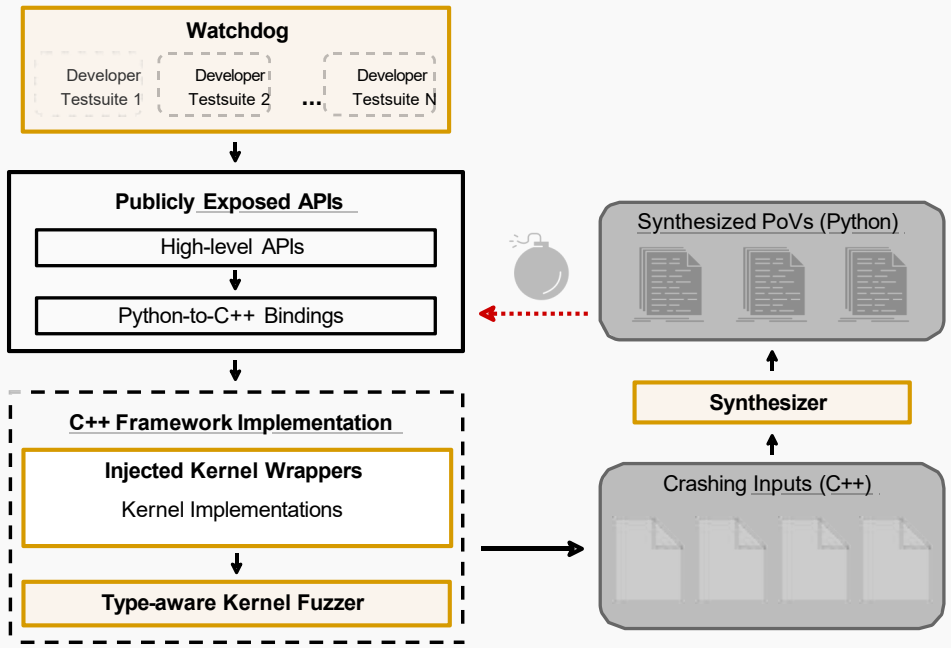
IvySyn Architecture → Force-executing Instrumented Kernels

- IvySyn runs DL frameworks' developer-provided unit tests



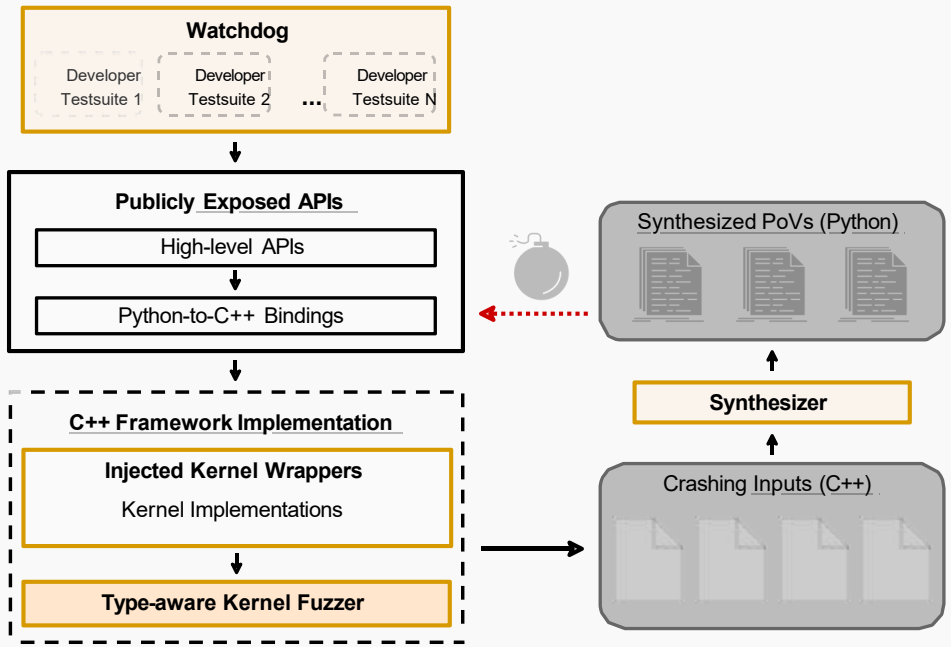
IvySyn Architecture → Force-executing Instrumented Kernels

- IvySyn runs DL frameworks' developer-provided unit tests
- Execution reaches the *instrumented* kernels …



IvySyn Architecture → Force-executing Instrumented Kernels

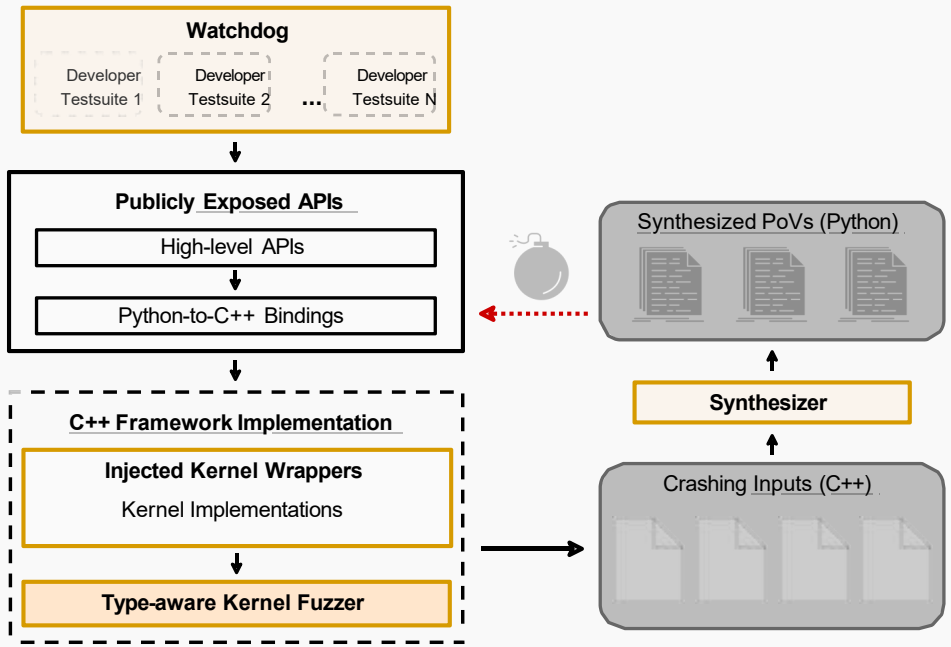
- IvySyn runs DL frameworks' developer-provided unit tests
- Execution reaches the *instrumented* kernels …
- …and bootstraps a fuzzing session



IvySyn Architecture → Force-executing Instrumented Kernels

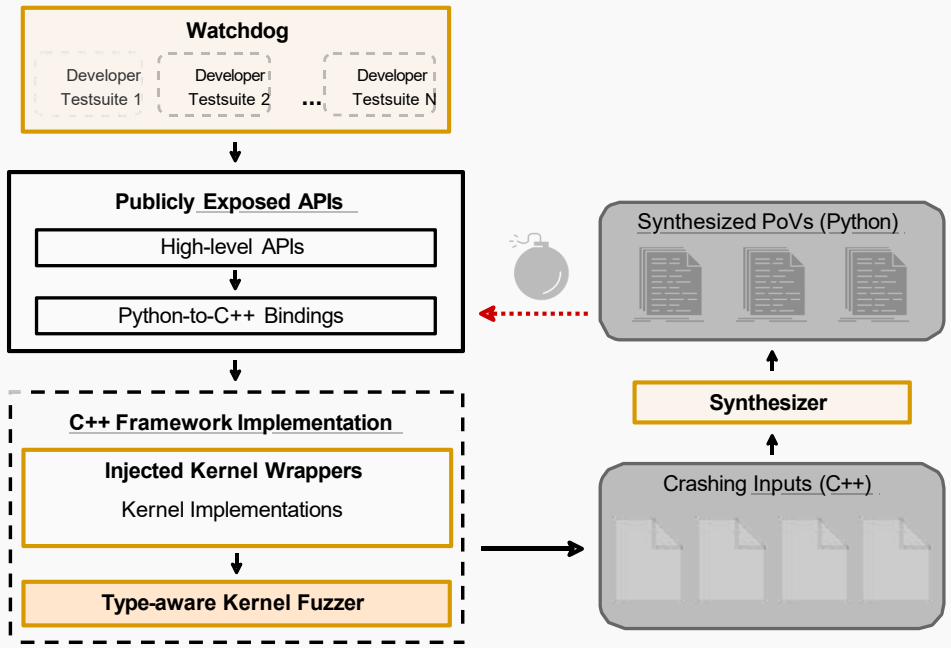
- IvySyn runs DL frameworks' developer-provided unit tests
- Execution reaches the *instrumented* kernels …
- …and bootstraps a fuzzing session

Why does IvySyn need the unit tests?



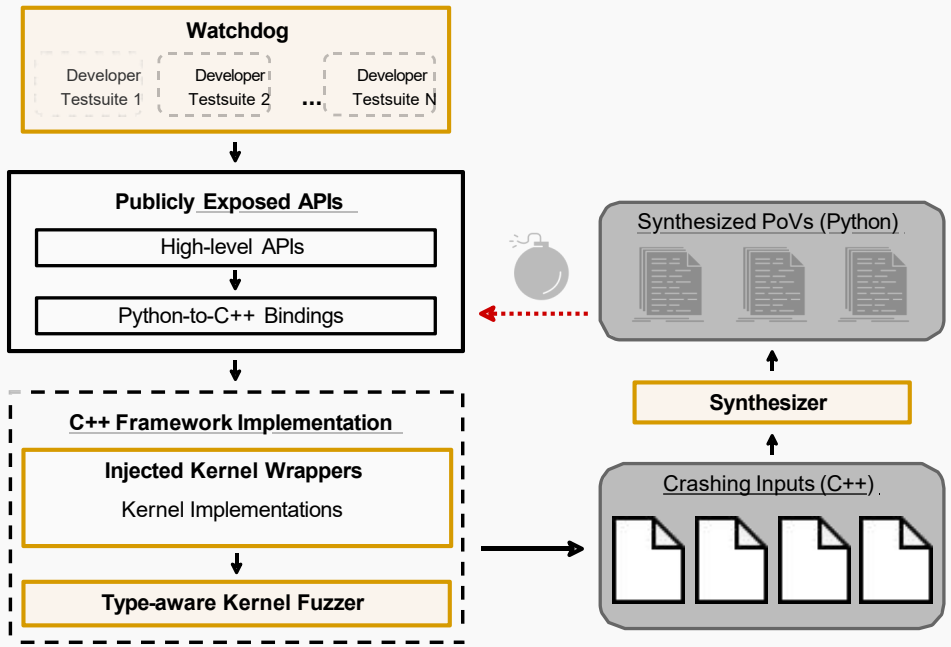
IvySyn Architecture → Type-aware Fuzzer

- Performs type-aware mutations based on the original argument types



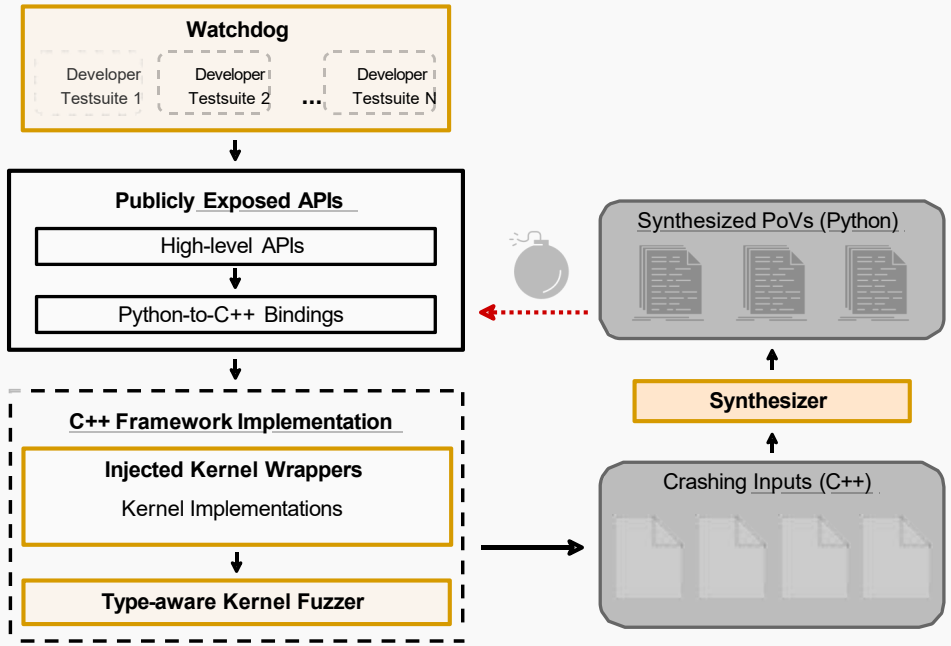
IvySyn Architecture → Type-aware Fuzzer

- Performs type-aware mutations based on the original argument types
- Logs *native crashing inputs* in crash-reports



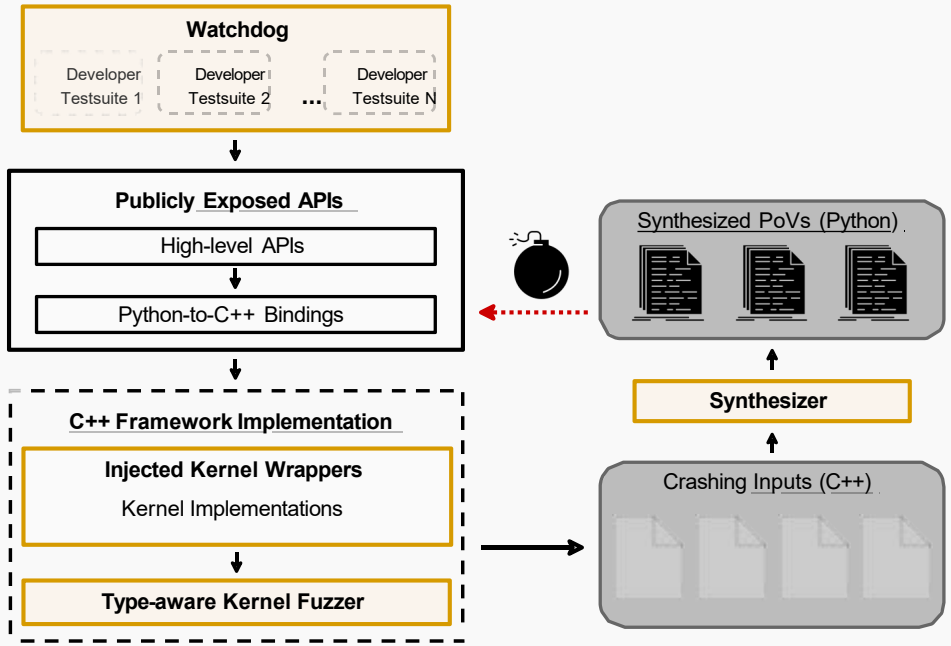
IvySyn Architecture → PoV Synthesis

- Logged crash-reports are fed into IvySyn's synthesizer



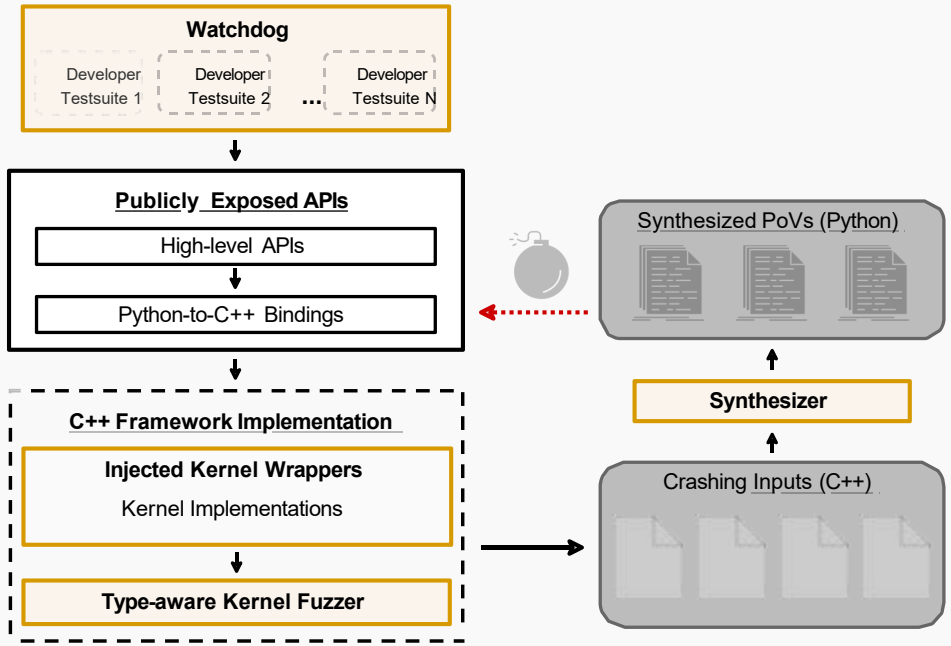
IvySyn Architecture → PoV Synthesis

- Logged crash-reports are fed into IvySyn's synthesizer
- The synthesizer generates *Proof-of-Vulnerability (PoV)* snippets



IvySyn Architecture → PoV Synthesis

- Logged crash-reports are fed into IvySyn's synthesizer
- The synthesizer generates *Proof-of-Vulnerability (PoV)* snippets
- The PoVs trigger the native crashes from publicly exposed Python APIs



IvySyn Architecture → PoV Synthesis (cont'd)

Crash-report produced by IvySyn

```
# SparseFillEmptyRowsOp
Tensor<type: int64 shape: [2,0] values: >
Tensor<type: int64 shape: [3] values: 2 0 1>
Tensor<type: int64 shape: [3] values: 2 0 1>
Tensor<type: int64 shape: [] values: 0>
```

IvySyn Architecture → PoV Synthesis (cont'd)

Crash-report produced by IvySyn

```
# SparseFillEmptyRowsOp
Tensor<type: int64 shape: [2,0] values: >
Tensor<type: int64 shape: [3] values: 2 0 1>
Tensor<type: int64 shape: [3] values: 2 0 1>
Tensor<type: int64 shape: [] values: 0>
```

</> Corresponding PoV synthesized by IvySyn

```
import tensorflow as tf

indices = tf.constant([], shape=[2,0],
                      dtype=tf.int64)
values = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
dense_shape = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
default_value = tf.constant(0, shape=[],
                      dtype=tf.int64)
tf.raw_ops.SparseFillEmptyRows(
                indices=indices,
                values=values,
                dense_shape=dense_shape,
                default_value=default_value)
```

IvySyn Architecture → PoV Synthesis (cont'd)

Crash-report produced by IvySyn

```
# SparseFillEmptyRowsOp
Tensor<type: int64 shape: [2,0] values: >
Tensor<type: int64 shape: [3] values: 2 0 1>
Tensor<type: int64 shape: [3] values: 2 0 1>
Tensor<type: int64 shape: [] values: 0>
```

PoV triggers crash!

```
$ python3 pov.py
segmentation fault (core dumped)
```

</> Corresponding PoV synthesized by IvySyn

```
import tensorflow as tf

indices = tf.constant([], shape=[2,0],
                      dtype=tf.int64)
values = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
dense_shape = tf.constant([2,0,1], shape=[3],
                      dtype=tf.int64)
default_value = tf.constant(0, shape=[],
                      dtype=tf.int64)
tf.raw_ops.SparseFillEmptyRows(
                indices=indices,
                values=values,
                dense_shape=dense_shape,
                default_value=default_value)
```

Outline

IvySyn Architecture - Instrumentation

Evaluation

Conclusion

Future Plans

Evaluation

Evaluation

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

Evaluation

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?
Q2. Is *IvySyn* effective in leveraging crashing inputs to synthesize PoVs?

Evaluation

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?
Q2. Is *IvySyn* effective in leveraging crashing inputs to synthesize PoVs?
Q3. Which *IvySyn* mutations are the most successful in uncovering memory errors?

Evaluation

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?
Q2. Is *IvySyn* effective in leveraging crashing inputs to synthesize PoVs?
Q3. Which *IvySyn* mutations are the most successful in uncovering memory errors?
Q4. What are the security ramifications of the PoVs synthesized with *IvySyn*?

Evaluation → IvySyn vs Atheris

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

- Compared IvySyn's *efficiency* at uncovering crashes against Atheris[†]

[†] *Atheris: A Coverage-Guided, Native Python Fuzzer*. Google.

Evaluation → IvySyn vs Atheris

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

- Compared IvySyn's *efficiency* at uncovering crashes against Atheris[†]
- Leveraged IvySyn's argument logging functionality and synthesizer to generate *fuzzing drivers* for Atheris

[†] *Atheris: A Coverage-Guided, Native Python Fuzzer*. Google.

Evaluation → IvySyn vs Atheris

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

- Compared IvySyn's *efficiency* at uncovering crashes against Atheris[†]
- Leveraged IvySyn's argument logging functionality and synthesizer to generate *fuzzing drivers* for Atheris
- Generated two different variants of drivers

[†] *Atheris: A Coverage-Guided, Native Python Fuzzer*. Google.

Evaluation → IvySyn vs Atheris

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

- Compared IvySyn's *efficiency* at uncovering crashes against Atheris[†]
- Leveraged IvySyn's argument logging functionality and synthesizer to generate *fuzzing drivers* for Atheris
- Generated two different variants of drivers

Atheris[+]

- Drivers without type awareness
- Atheris randomly chooses argument types

[†] *Atheris: A Coverage-Guided, Native Python Fuzzer*. Google.

Evaluation → IvySyn vs Atheris

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

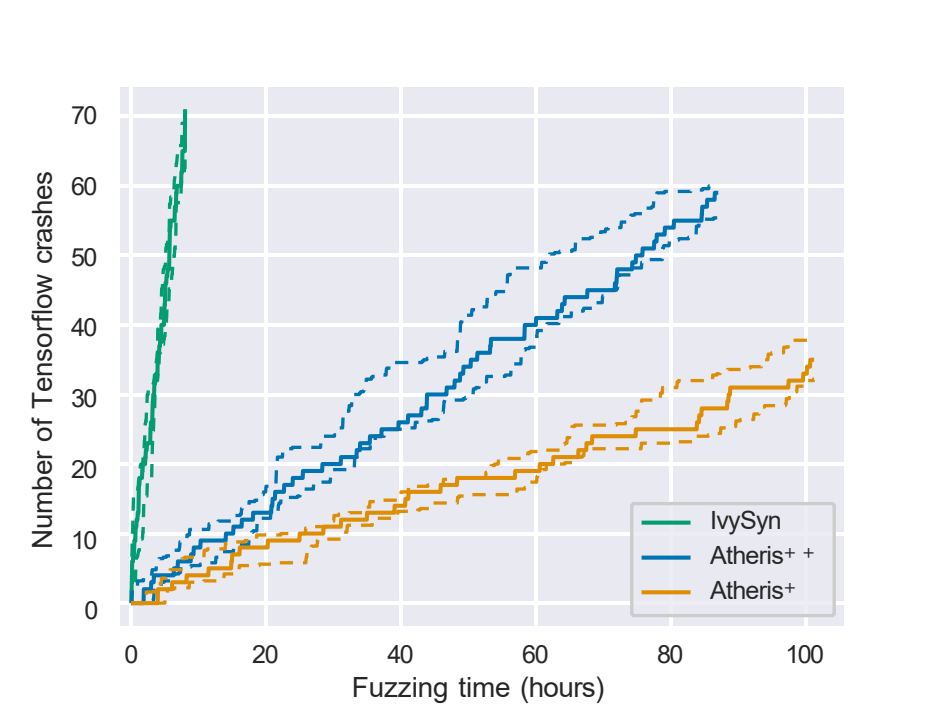
- Compared IvySyn's *efficiency* at uncovering crashes against Atheris[†]
- Leveraged IvySyn's argument logging functionality and synthesizer to generate *fuzzing drivers* for Atheris
- Generated two different variants of drivers

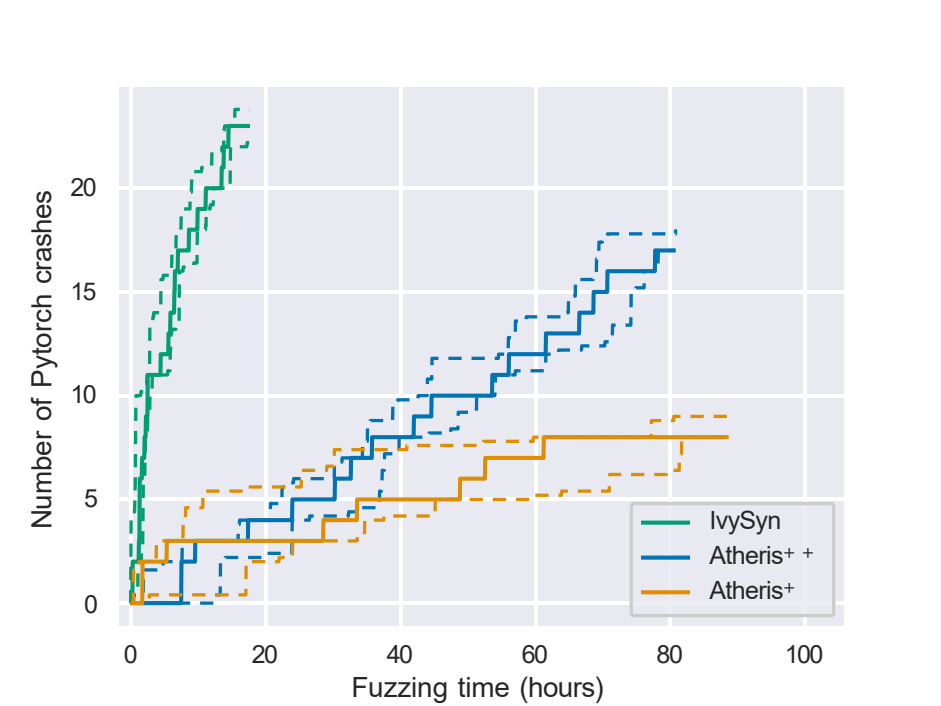
Atheris⁺	Atheris⁺⁺
• Drivers without type awareness • Atheris randomly chooses argument types	• Drivers with type awareness • The drivers provide Atheris with the proper argument types

[†]*Atheris: A Coverage-Guided, Native Python Fuzzer*. Google.

Evaluation → IvySyn vs Atheris (cont'd)

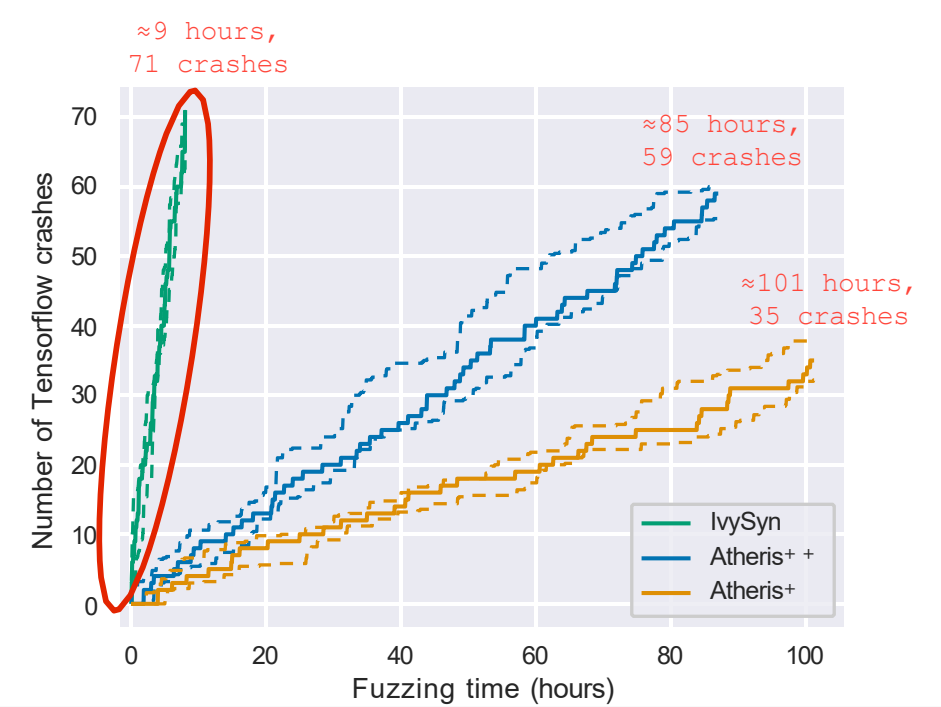
Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

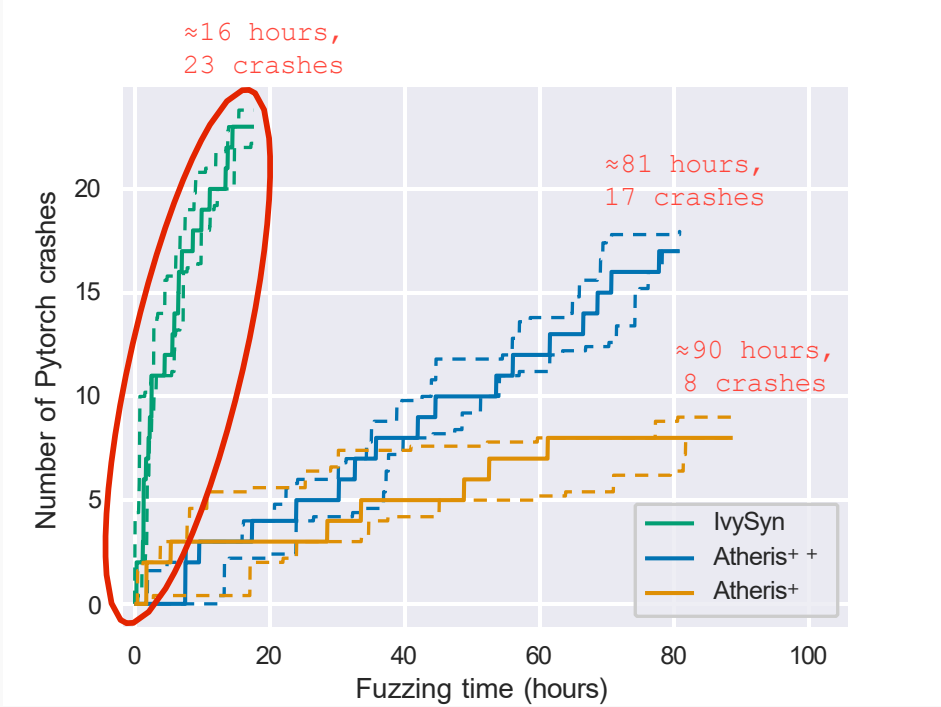




Evaluation → IvySyn vs Atheris (cont'd)

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?





- IvySyn uncovers more crashes than Atheris, and does so faster

Evaluation → IvySyn vs Atheris (cont'd)

Q1. Is *IvySyn* efficient in uncovering crashing inputs over time?

Number of crashes found by IvySyn vs Atheris (over 5 iterations)

	Fuzzer	TensorFlow	PyTorch
Total Crashes	Atheris+	47	9
	Atheris++	64	18
	IvySyn	80	25
Union	All	87	30

- IvySyn uncovers more crashes than Atheris, and does so faster

Evaluation → IvySyn vs DocTer

Q2. Is *IvySyn* effective in leveraging crashing inputs to synthesize PoVs?

- Compared IvySyn's *effectiveness* at synthesizing PoVs against the *semi-automated* DocTer[‡] tool

[‡] *DocTer: Documentation-Guided Fuzzing for Testing Deep Learning API Functions* Xie *et al.*

Evaluation → IvySyn vs DocTer

Q2. Is *IvySyn* effective in leveraging crashing inputs to synthesize PoVs?

- Compared IvySyn's *effectiveness* at synthesizing PoVs against the *semi-automated* DocTer‡ tool

Synthesized PoVs (IvySyn vs DocTer) (over 10 iterations)

Fuzzer	Total		Median		Median Running Time (mins)	
	TensorFlow	PyTorch	TensorFlow	PyTorch	TensorFlow	PyTorch
DocTer	16	9	12	7	199	736
IvySyn	19	14	15	11	184	569

- IvySyn synthesizes more PoVs than DocTer, *without* manual effort

‡*DocTer: Documentation-Guided Fuzzing for Testing Deep Learning API Functions*. Xie *et al.*

Evaluation → Overall Results

Q2. Is *IvySyn* effective in leveraging crashing inputs to synthesize PoVs?

Accumulated `IvySyn` results

Framework	Fuzzed Kernels	Unique Crashes	Synthesized PoVs
TensorFlow	412	103	86 / 103 (83%)
PyTorch	747	81	49 / 81 (60%)
All	1159	184	135 / 184 (73%)

- `IvySyn` synthesized 135 PoVs and was attributed with 39 CVEs

Evaluation ➜ Effectiveness per Mutation Type

Q3. Which *IvySyn* mutations are the most successful in uncovering memory errors?

Number of PoVs per mutation type

IvySyn Type-aware Mutation Type	Total PoVs
Tensors with random dimension sizes	46
Tensors with extreme values	25
Permutations of original arguments	19
Zero values	12
Lists with extreme values	13
Tensors with empty shape	8
Extreme values in primitive types	6
Empty lists	3
Deep tensors	3

Evaluation → Effectiveness per Mutation Type

Q3. Which *IvySyn* mutations are the most successful in uncovering memory errors?

Number of PoVs per mutation type

IvySyn Type-aware Mutation Type	Total PoVs
Tensors with random dimension sizes	46
Tensors with extreme values	25
Permutations of original arguments	19
Zero values	12
Lists with extreme values	13
Tensors with empty shape	8
Extreme values in primitive types	6
Empty lists	3
Deep tensors	3

- DL-specific (*e.g.*, tensor) mutations are especially effective

Evaluation ➜ Security Analysis

Q4. What are the security ramifications of the PoVs synthesized with *IvySyn*?

- IvySyn successfully synthesized PoVs for 135 of the total 184 TensorFlow and PyTorch crashes found.

Evaluation → Security Analysis

Q4. What are the security ramifications of the PoVs synthesized with *IvySyn*?

- IvySyn successfully synthesized PoVs for 135 of the total 184 TensorFlow and PyTorch crashes found.
- To what extent these PoVs could be abused to corrupt memory locations, leak program contents, or cause the respective runtime environment to crash?

Evaluation → Security Analysis

Q4. What are the security ramifications of the PoVs synthesized with *IvySyn*?

- IvySyn successfully synthesized PoVs for 135 of the total 184 TensorFlow and PyTorch crashes found.
- To what extent these PoVs could be abused to corrupt memory locations, leak program contents, or cause the respective runtime environment to crash?

Framework	SIGABRT	SIGFPE	SIGSEGV
TensorFlow	56	4	26
PyTorch	-	16	33

Evaluation → Security Analysis

Q4. What are the security ramifications of the PoVs synthesized with *IvySyn*?

- IvySyn successfully synthesized PoVs for 135 of the total 184 TensorFlow and PyTorch crashes found.
- To what extent these PoVs could be abused to corrupt memory locations, leak program contents, or cause the respective runtime environment to crash?

Framework	SIGABRT	SIGFPE	SIGSEGV
TensorFlow	56	4	26
PyTorch	-	16	33

- Most crashes do not cause memory corruption(floating-point exceptions, aborts), but there are still security-relevant threats(possible DoS).

Evaluation → Security Analysis

Q4. What are the security ramifications of the PoVs synthesized with *IvySyn*?

- IvySyn successfully synthesized PoVs for 135 of the total 184 TensorFlow and PyTorch crashes found.
- To what extent these PoVs could be abused to corrupt memory locations, leak program contents, or cause the respective runtime environment to crash?

Framework	SIGABRT	SIGFPE	SIGSEGV
TensorFlow	56	4	26
PyTorch	-	16	33

- Most crashes do not cause memory corruption(floating-point exceptions, aborts), but there are still security-relevant threats(possible DoS).
- Majority of segfaults(memory read and write operations at controlled addresses) are security critical:
  By controlling the addresses of these operations, an attacker could leverage them to gain arbitrary read/write primitives.

Outline

IvySyn Architecture - Instrumentation

Evaluation

Conclusion

Future Plans

Conclusion

- Fully-automated framework
  - Perform type-aware, DL-specific mutations

Conclusion

- Fully-automated framework
  - Perform type-aware, DL-specific mutations
  - Fuzz *native implementation* of DL frameworks

Conclusion

- Fully-automated framework
  - Perform type-aware, DL-specific mutations
  - Fuzz *native implementation* of DL frameworks
  - Synthesize PoVs to trigger detected crashes from Python

Conclusion

- Fully-automated framework
  - Perform type-aware, DL-specific mutations
  - Fuzz *native implementation* of DL frameworks
  - Synthesize PoVs to trigger detected crashes from Python
- Identified 61 previously-unknown vulnerabilities

Conclusion

- Fully-automated framework
  - Perform type-aware, DL-specific mutations
  - Fuzz *native implementation* of DL frameworks
  - Synthesize PoVs to trigger detected crashes from Python
- Identified 61 previously-unknown vulnerabilities
- Assigned with 39 unique CVEs

Outline

IvySyn Architecture - Instrumentation

Evaluation

Conclusion

Future Plans

Pushing IvySyn Further: Limitations and Research Opportunities

Scope Expansion

- Extend to additional DL frameworks and hardware backends.
- Explore applicability to non-DL codebases.

Pushing IvySyn Further: Limitations and Research Opportunities

Scope Expansion

- Extend to additional DL frameworks and hardware backends.
- Explore applicability to non-DL codebases.

Technical Advancements

- Support stateful kernels or kernels with non-{tensor, array, scalar} arguments types.
- Coverage-guided, type-aware kernel fuzzing (combine IvySyn's kernel hooks with Atheris-style feedback).
- GPU-Aware and Race-Condition Fuzzing.

THANK YOU