Differential Testing of Cross Deep Learning Framework APIs: Revealing Inconsistencies and Vulnerabilities

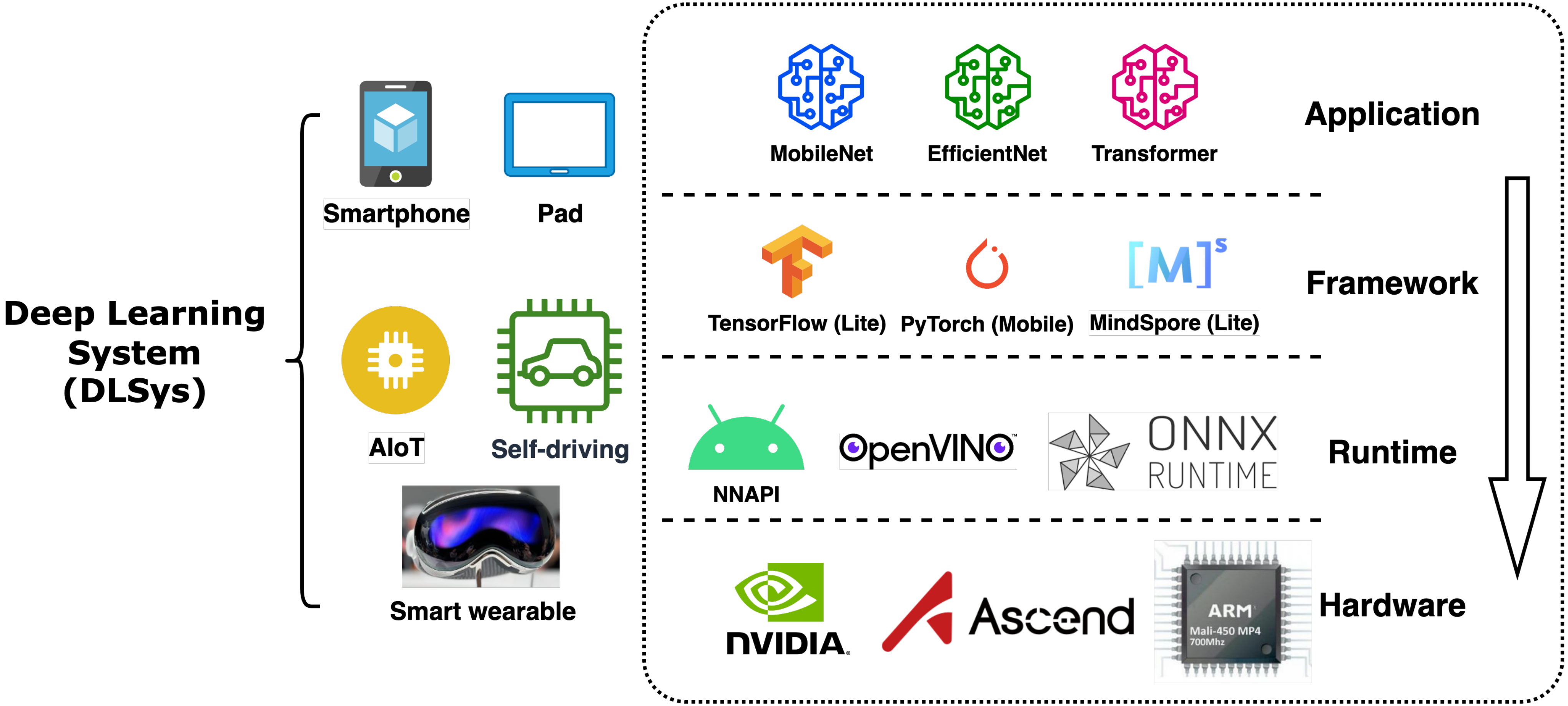
Zizhuang Deng[1,2], Guozhu Meng[1,2,*], Kai Chen[1,2,*], Tong Liu[1,2], Lu Xiang[1,2], Chunyang Chen[3]

[1] SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, China
[3] Monash University, Australia

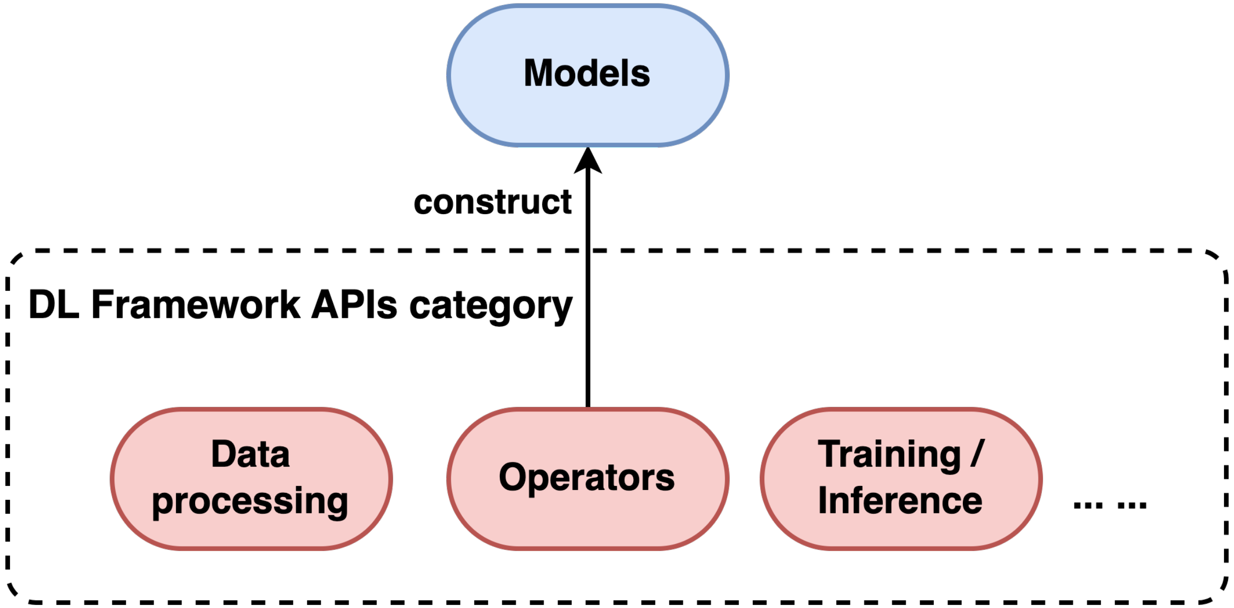
The 32nd USENIX Security Symposium (USENIX Security 2023)

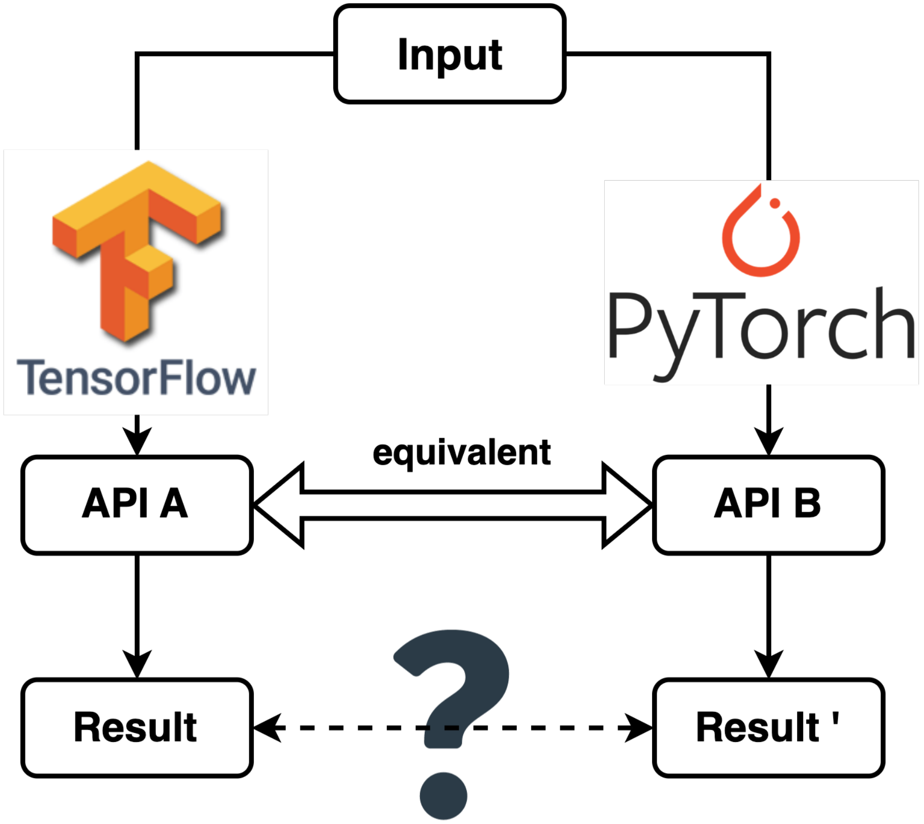
Background

- **Deep learning systems are rapidly evolving**



Cross Deep learning Framework APIs

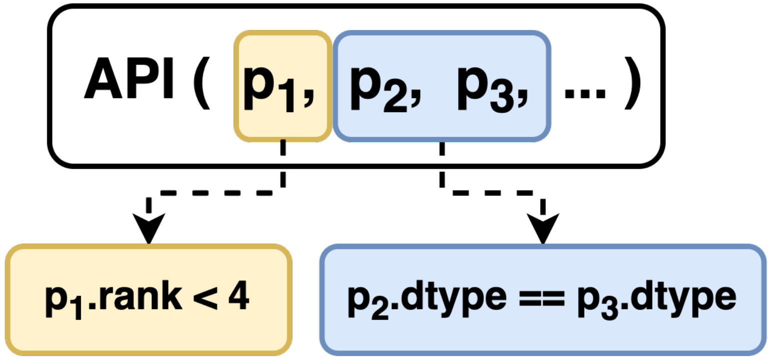




Using differential testing for cross deep learning framework testing.

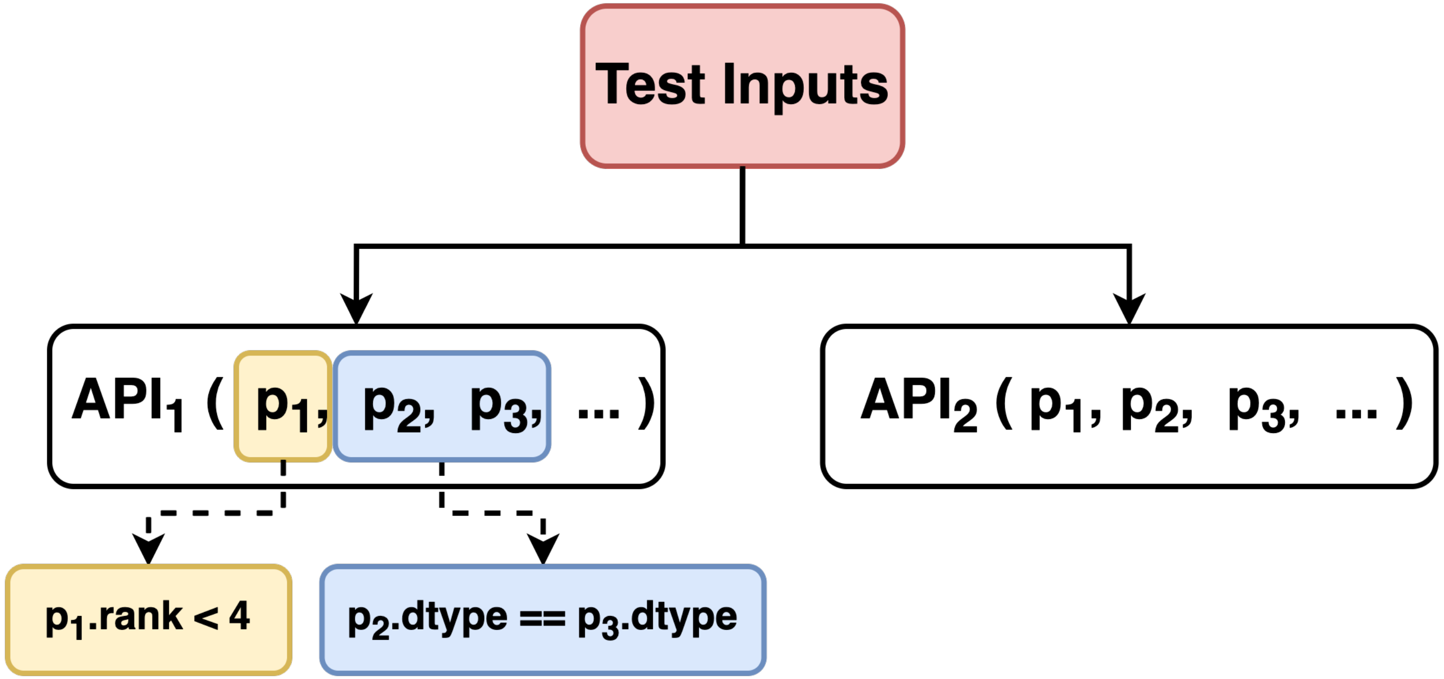
Differential Testing is challenging

- **Extract the constraints of API parameters and their implicit dependencies**



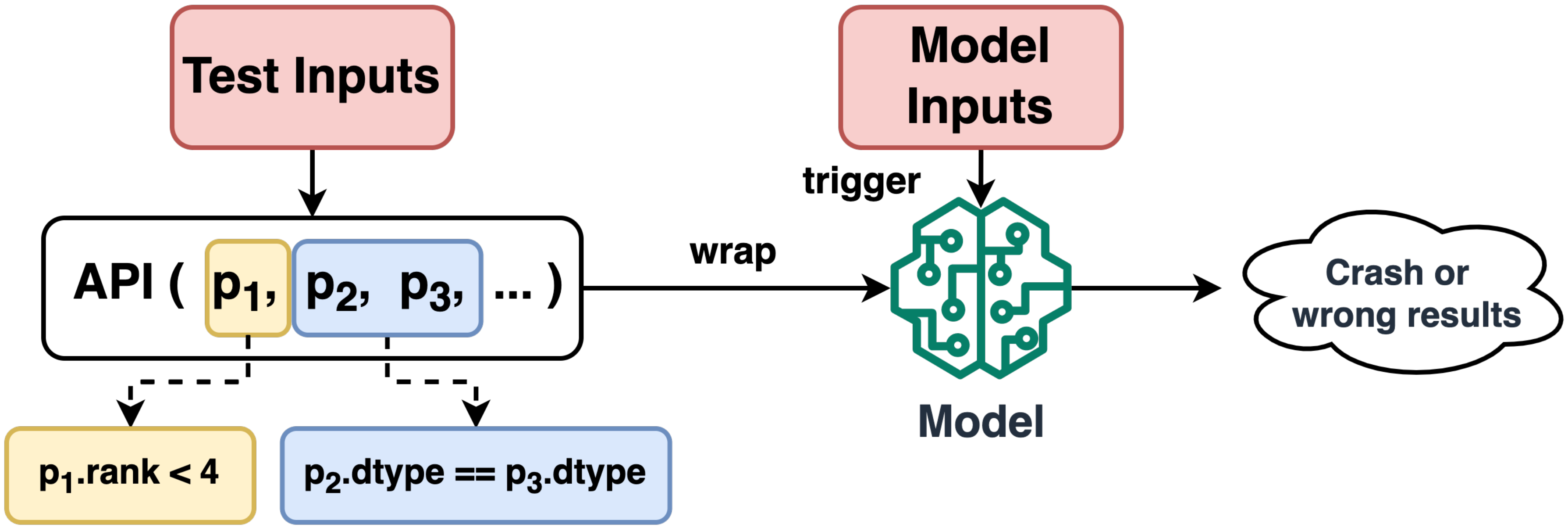
Differential Testing is challenging

- **Extract constraints**
- **Generate representative test cases**



It’s hard to evaluate these bugs

- APIs in Real world models
- Craft model input to trigger the buggy API



Counterpart APIs

- **For an API $\boldsymbol{f}$, $\boldsymbol{counterpart(f)} = \{\boldsymbol{f_1}, \ldots, \boldsymbol{f_n}\}, (\boldsymbol{n} \geq \mathbf{1})$**
  - ❑ **Semantic equivalence**

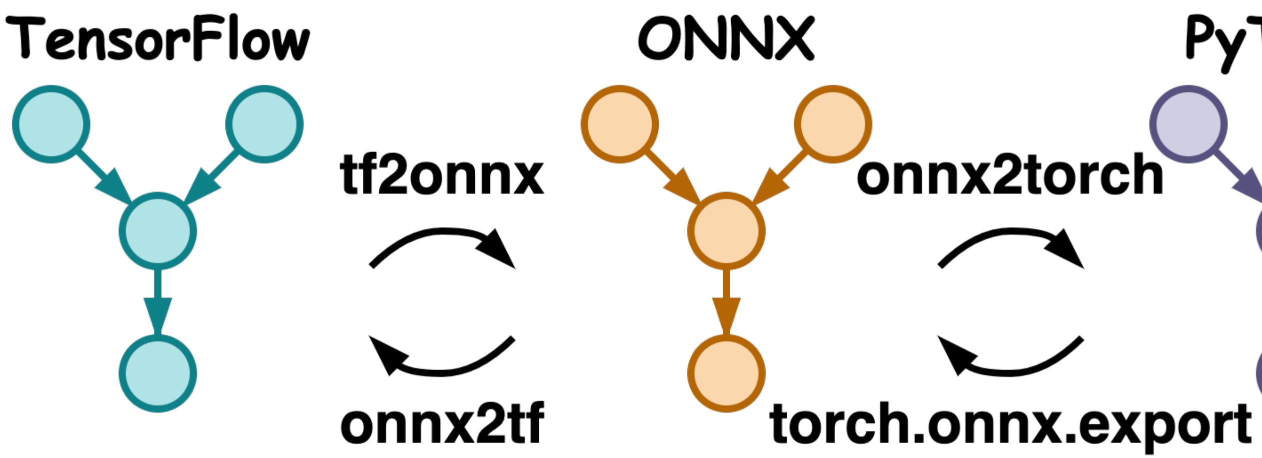
$$\|f(x) - (f_1 \circ \ldots \circ f_n)(x)\|_p \leq \varepsilon$$

  - ❑ **Sequentiality**
    - **AdjustContrastv2(x0, x1) = Add(Mul(x1, Sub(x0, ReduceMean(x0))), ReduceMean(x0))**

How to extract counterpart APIs across DL frameworks?

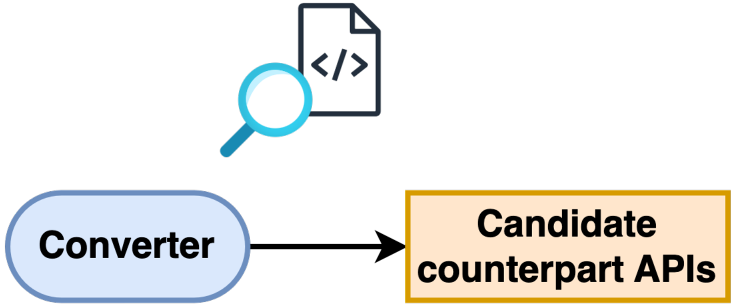
Counterparts Extraction

- **Model converter**



Counterparts Extraction

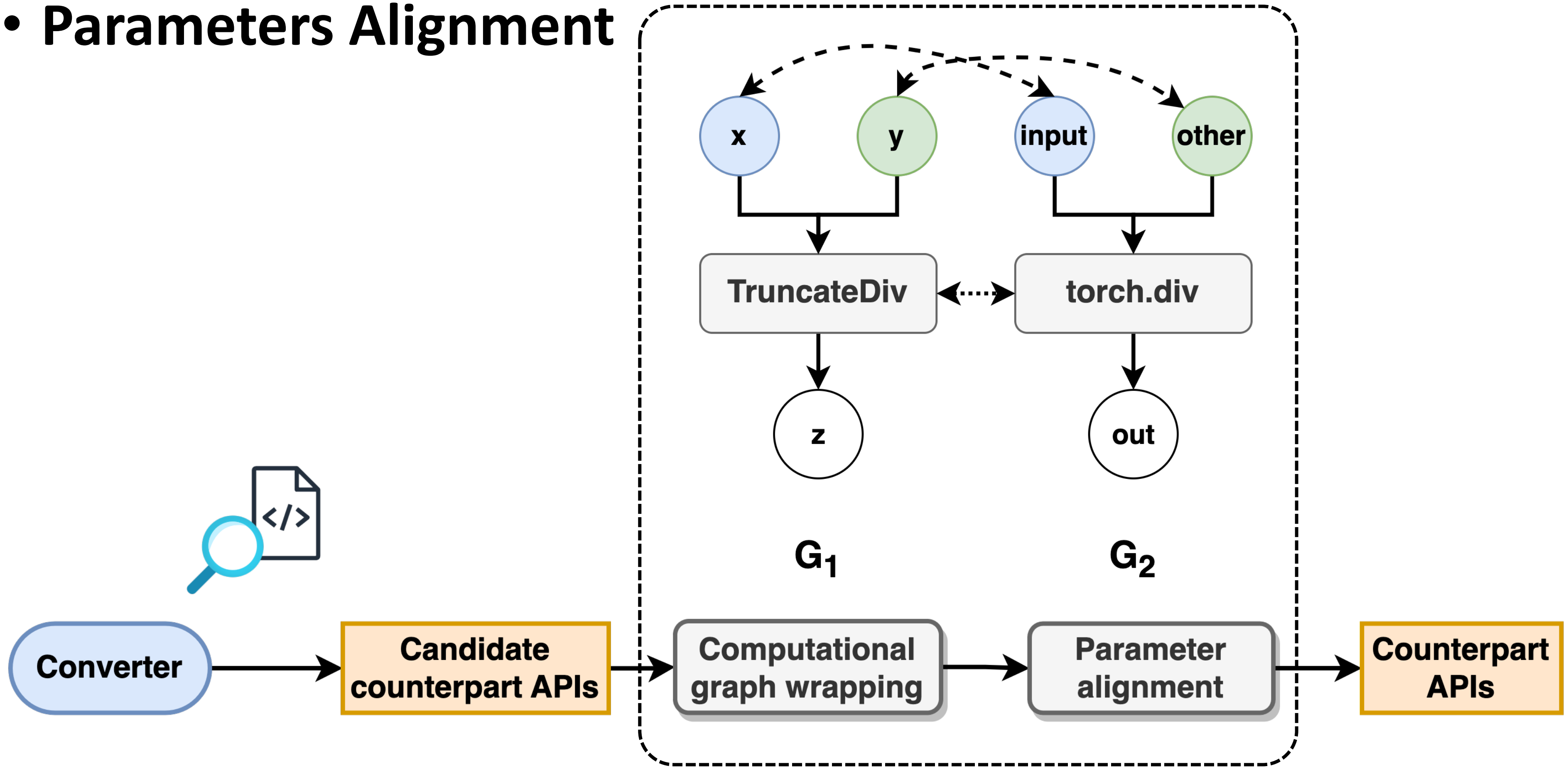
- **Static analysis on coverter code**




```
registry: Dict[str, handler] =
{"onnx::AveragePool": PoolMapper,
 "onnx::MaxPool" : PoolMapper, ...}
class PoolMapper(ONNXToMindSporeMapper):
  def _operation_name_in_ms(*args, **kwargs):
    if kwargs['op_name']=='onnx::AveragePool‘:
      op_name = “nn.AvgPool{}d”
    else:
      op_name = “nn.MaxPool{}d”
    dim = len(kwargs['params']['strides‘])
    if dim == 3:
      return "P.MaxPool3D"
    return op_name.format(dim)
```

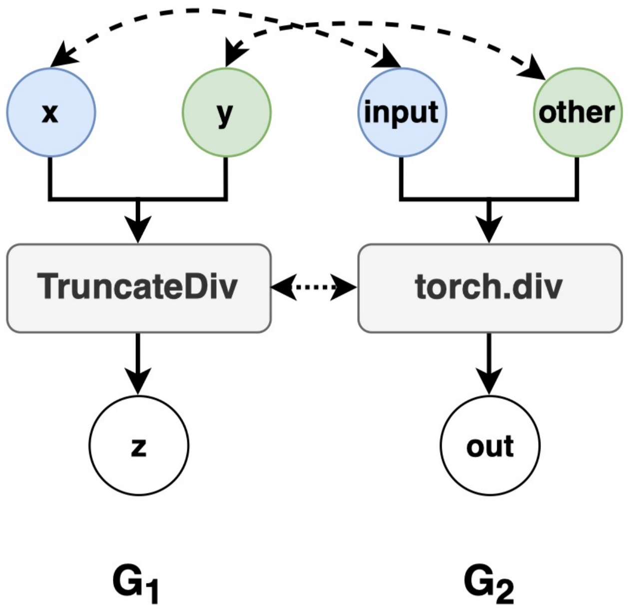
Counterparts Extraction

- **Parameters Alignment**



Constraint Extraction

- **Constraints of API parameters**
  - **On five attributes: ① type ② shape ③ data type ④ rank ⑤ data value**
- **API profiles**



Tensorflow

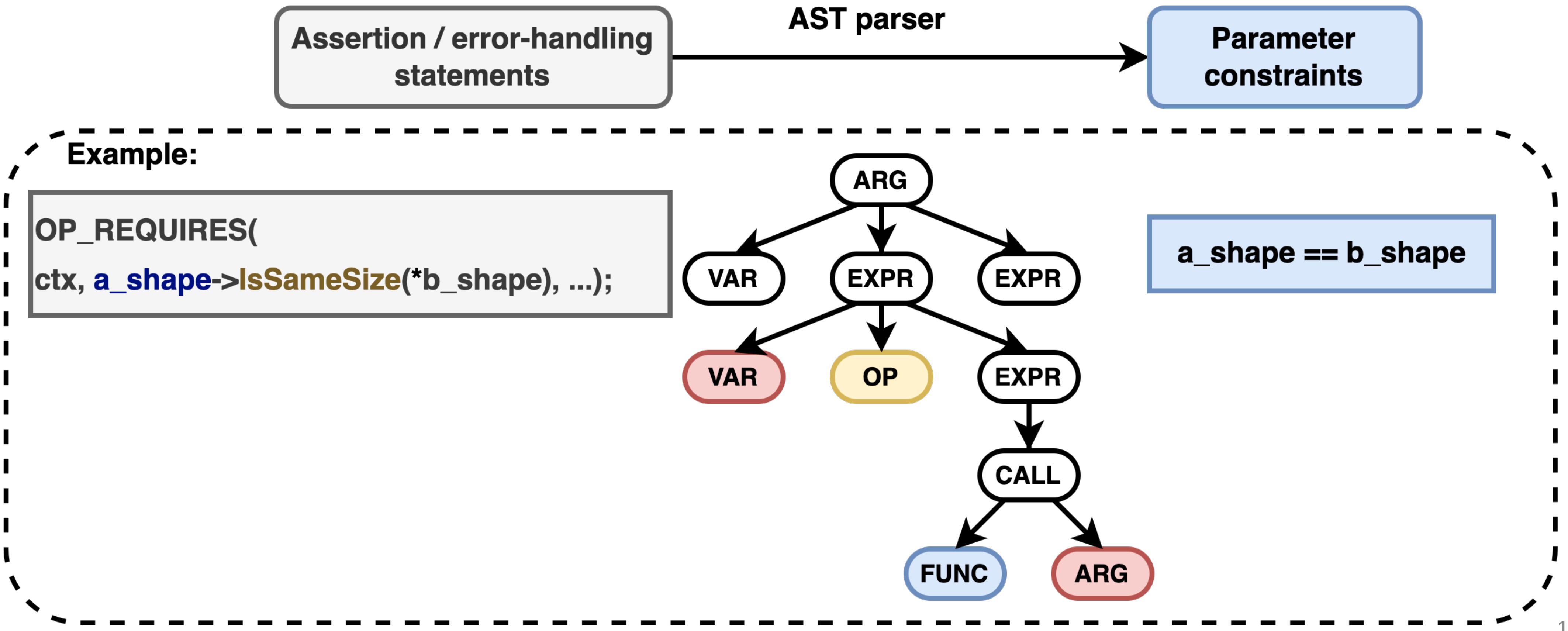
```
op {
  name: "TruncateDiv"
  input_arg {
    name: "x"
    type_attr: "T"}
  input_arg {
    name: "y"
    type_attr: "T"}
  output_arg {
    name: "z"
    type_attr: "T"}
}
attr {
  name: "T"
  type: "type"
  allowed_values {
    list {
      type: DT_HALF
      type: DT_FLOAT
      type: DT_DOUBLE
      type: DT_UINT8
      ... }
  }
}
```

PyTorch

```
op {
  name: "torch.div"
  input_arg {
    name: "input"
    type_attr: "T"}
  input_arg {
    name: "other"
    type_attr: "T"}
  output_arg {
    name: "out"
    type_attr: "T"}
}
attr {
  name: "T"
  type: "type"
  allowed_values {
    list {
      type: Tensor
      type: Number}}}
attr {
  name: "rounding_mode"
  type: "str"
  default_value {
    b: None } ... }
```

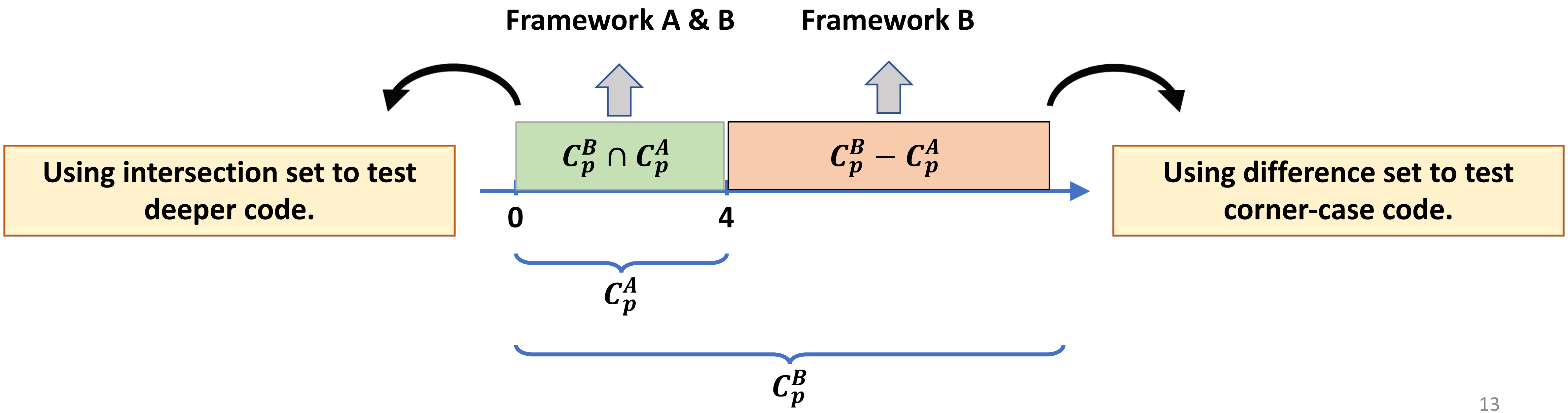
Constraint Extraction

- **API Implementation**



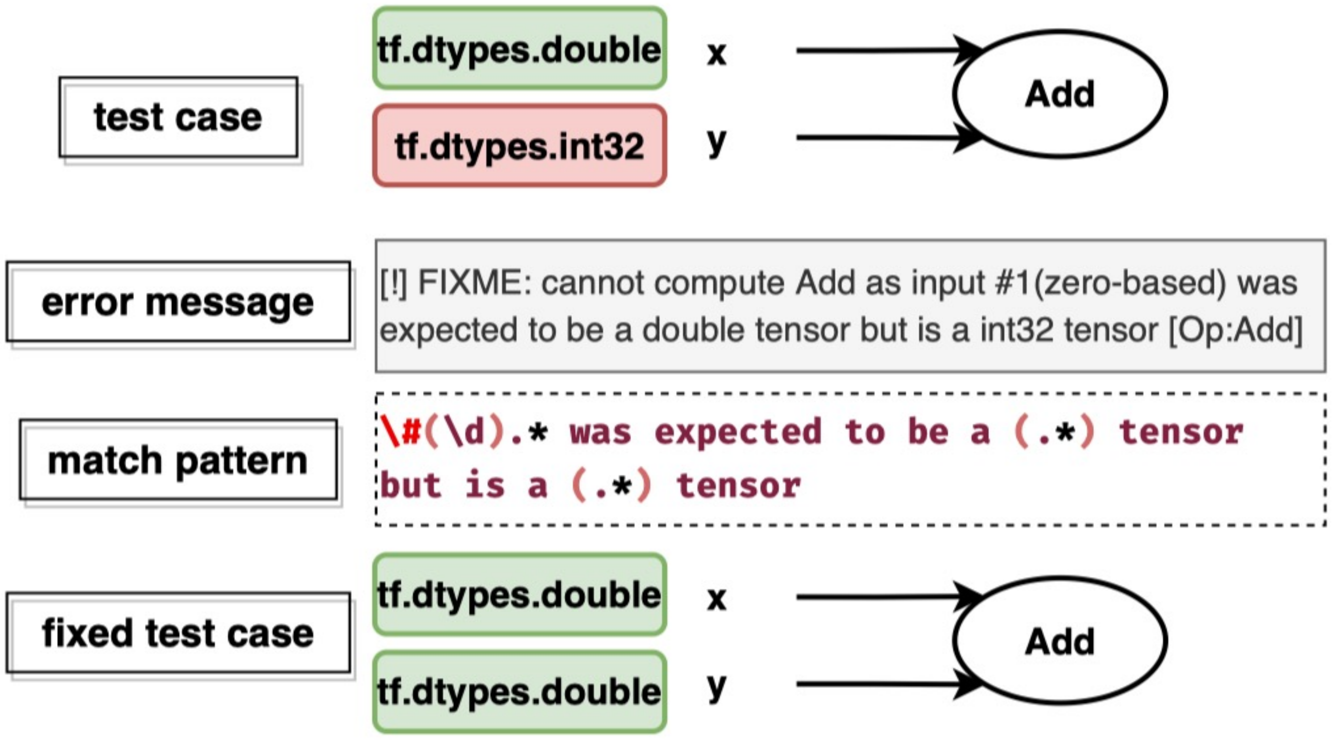
Test Case Generation

- **Joint Constraints Analysis**
  - **For the rank attribute of parameter**
    - ❑ C_p^A / C_p^B **represents constraints in framework A/B**



Testing Optimization

- **Error-guided Test Case Fixing**

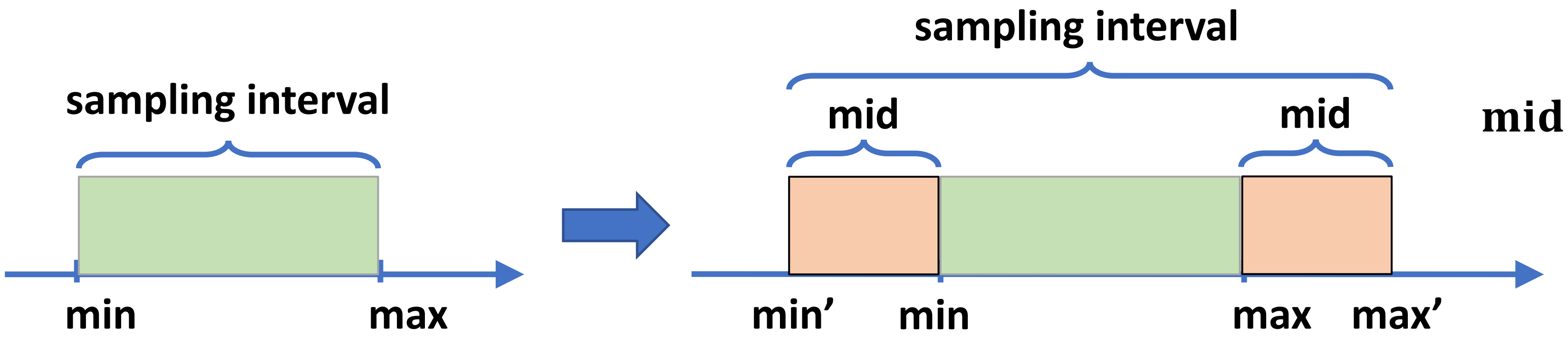


This optimization is to test deeper code.

Testing Optimization

- **Range Extension**

Value:



$$\mathbf{mid} = \frac{\mathbf{max} - \mathbf{min}}{\mathbf{2}}$$

Shape: **(2, 3)** → **(2, 3, 1)**

❑ **Special values, differential parameters**

This optimization is to test corner-case code.

Evaluation on bug finding

- **Statistics of crash bugs(177) & non-crash bugs(80)**
  - **230 ones are newly found**

	Version	#Bug	Segv	FPE	Abort
TF	2.11	26	13	0	13
TFL	2.11	0	0	0	0
ORT	1.12.1	0	0	0	0
MS	1.9.0 & nightly	100	90	8	2
Paddle	develop	23	15	6	2
Pytorch	1.10.0 & 1.12.1	28	27	0	1
Total		**177**	**145**	**14**	**18**

(a) Crash bugs

	TF	TFL	ORT	MS	PyTorch	Paddle
TF	2	2	1	-	0	-
TFL	0	0	0	-	0	-
ORT	10	0	0	-	3	-
MS	0	0	5	3	3	-
PyTorch	24	0	14	-	1	-
Paddle	3	0	6	-	3	0

(b) Inconsistent (non-crash) bugs

We found a total of 257 bugs on 1,658 APIs of 6 DL frameworks.

Evaluation on bug finding

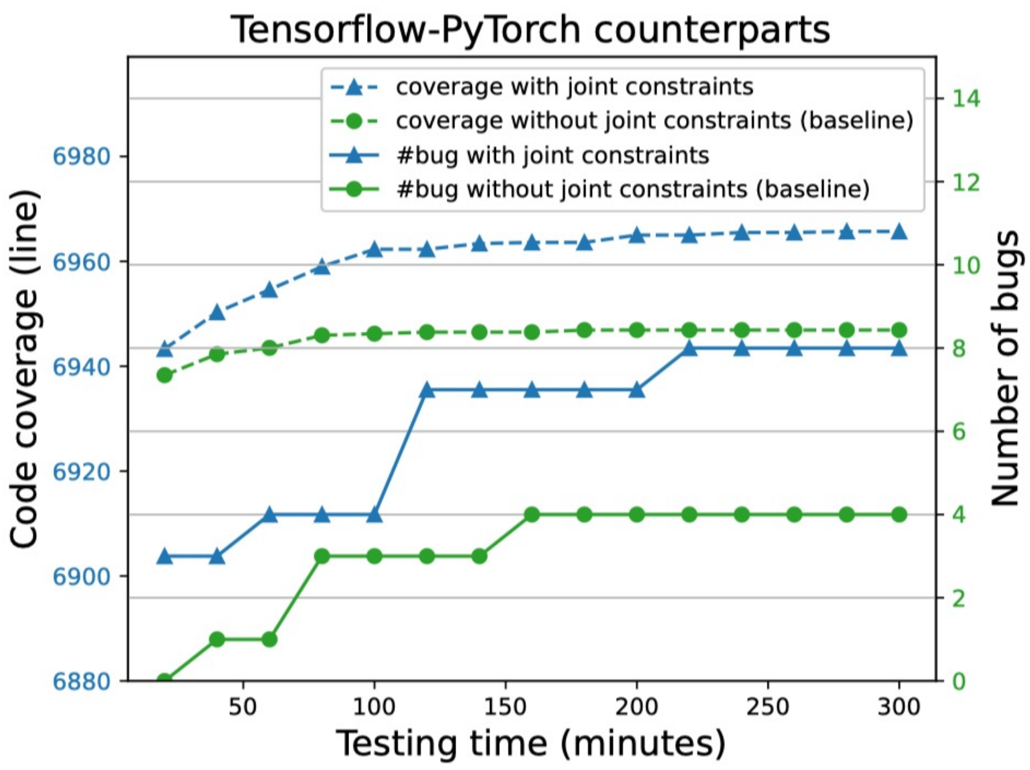
- **8 CVEs**
- **$1,100+ bounty!**

ID	CVSS	Framework	Type	Symptom	Description
CVE-2022-35935	7.5	TensorFlow	missing validation	`CHECK` failure	given a **nonscalar** `num_results` value
CVE-2022-41883	7.5	TensorFlow	missing validation	OOB segfault	`indices` list shorter than the `data` list
CVE-2022-41899	7.5	TensorFlow	missing validation	segfault	given wrong shape tensors
CVE-2022-41891	7.5	TensorFlow	missing validation	segfault	element_shape=[]
CVE-2022-41897	7.5	TensorFlow	missing validation	Heap OOB	outsize inputs
CVE-2022-45907	9.8	PyTorch	code injection	arbitrary code execution	using dangerous `eval`
CVE-2022-45908	9.8	Paddle	code injection	arbitrary code execution	using dangerous `eval`
CVE-2022-46742	9.8	Paddle	code injection	arbitrary code execution	using dangerous `eval`

Evaluation on code coverage

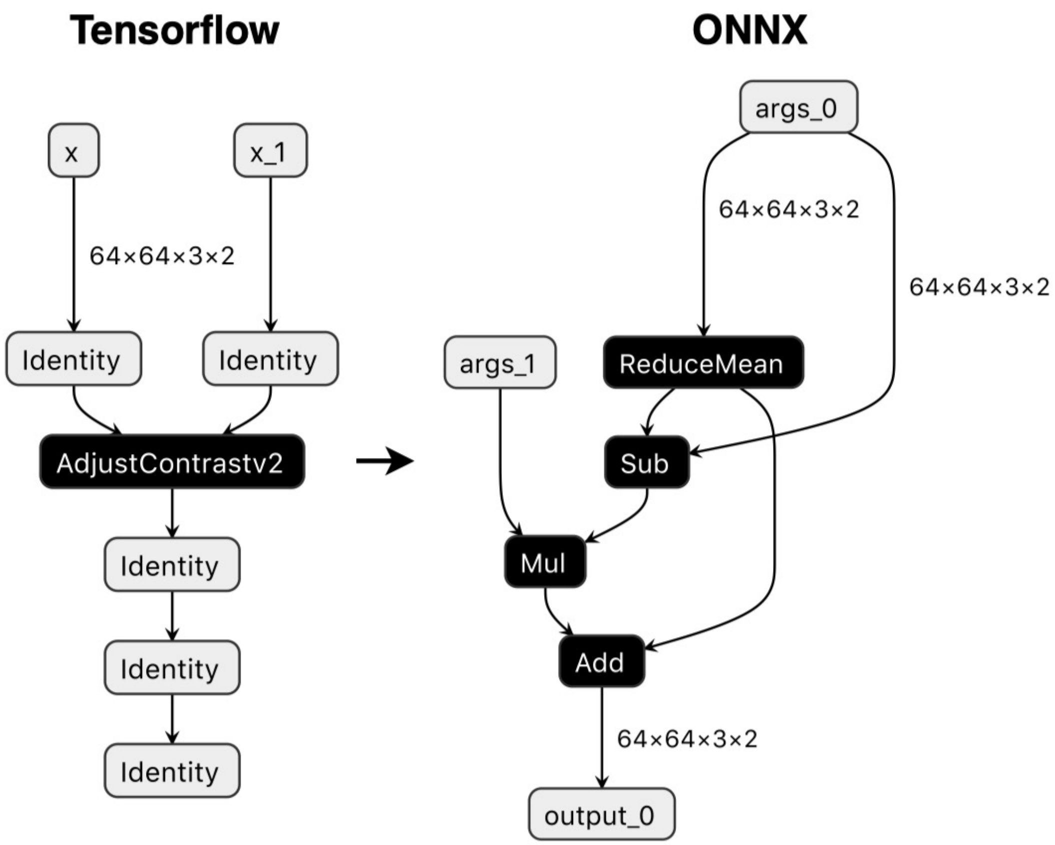
- **Comparative experiments**
  - **Compare with previous work**
  - **Ablation study**

	Tensorflow		Pytorch		Total	
	Cov.	**#Bug**	**Cov.**	**#Bug**	**Cov.**	**#Bug**
Atheris	32213	7	38502	4	70715	11
FreeFuzz	40702	13	44183	19	84885	32
DocTer	42877	18	45035	15	87912	33
TENSORSCOPE	**55362**	**24**	**43905**	**21**	**99267**	**45**



It is observed that our tool (TensorScope) detects the most number of bugs and the highest code coverage. These results indicate that joint constraints can effectively guide the testing process.

Case study of converter bugs



AdjustContrastv2(
Images=tf.random.uniform([64,64,3,2],dtype=tf.dtypes.float32,maxval=255),
contrast_factor=tf.random.uniform([],dtype=tf.dtypes.float32,maxval=1),
)

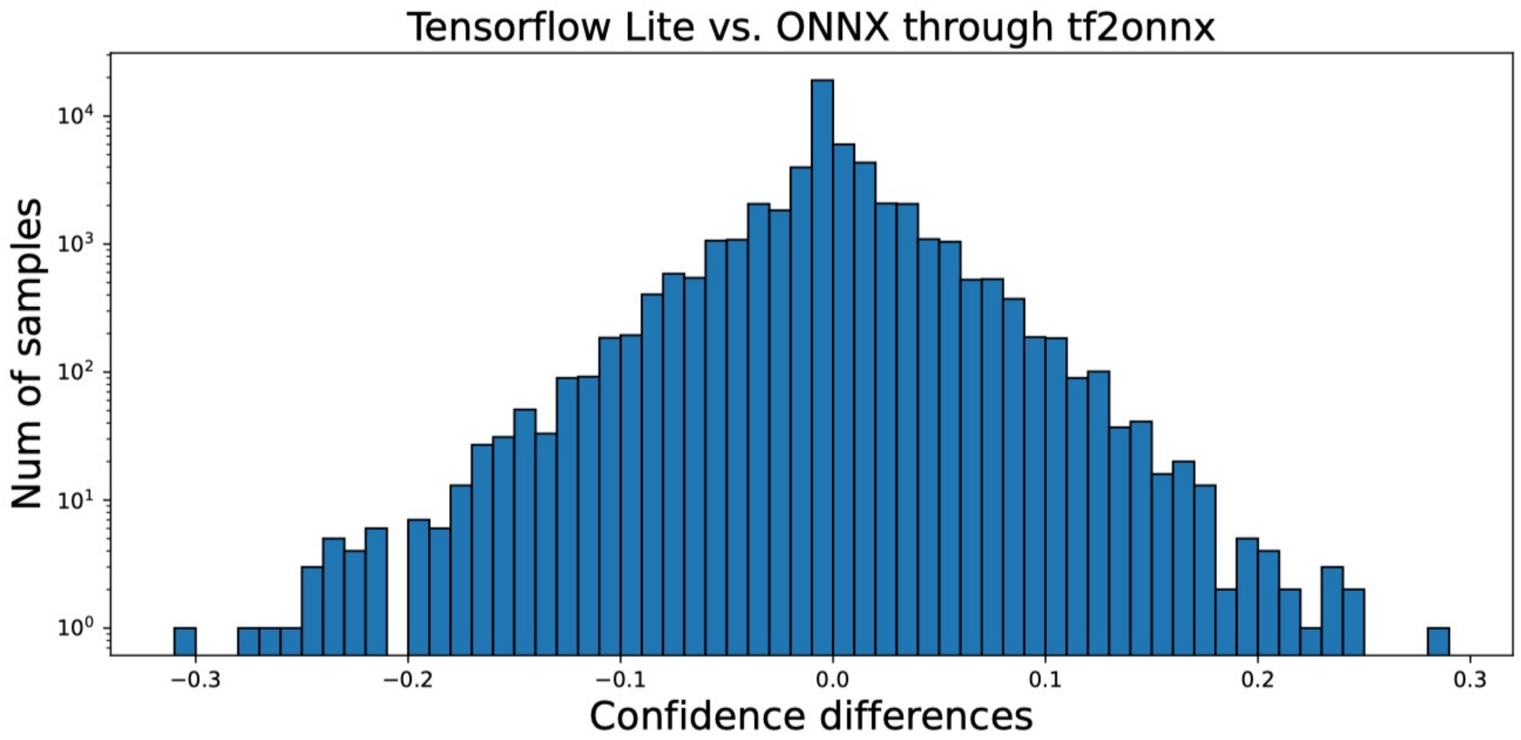
Here is the inconsistent results:

tf_res : [[[[154.93831 131.07579], [141.15346 162.48589], [123.68347 143.64304]], ...]]

onnx_res : [[[[153.70927 , 126.60315], [139.92442 , 158.01324], [122.45444 , 139.1704]], ...]]

Hazard analysis

- **MobileNet model**
  - **Top-1 accuracy 72.3% (TensorFlow) -> 68.8% (ONNX)**
- **Classify “Snail” to “bubble”**



Thanks!

Q&A

Contact: dengzizhuang@iie.ac.cn