Journey of Testing AI Software Systems

UNIST
Mijung Kim

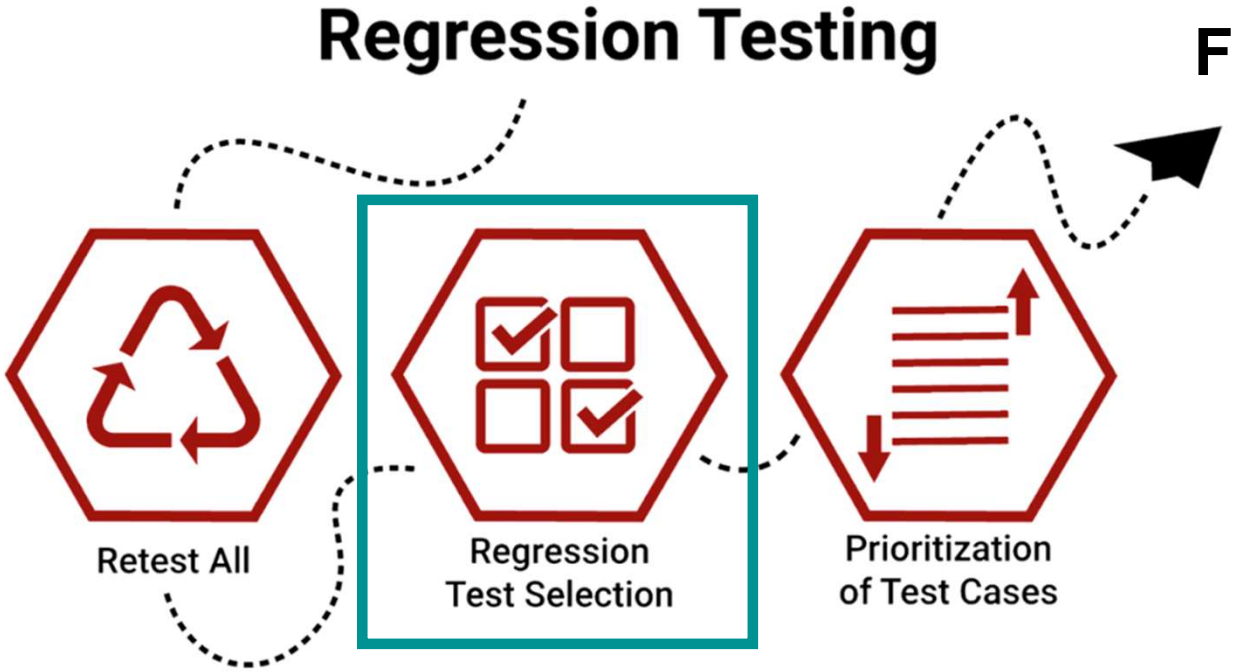
About me: Mijung Kim

Assistant Professor

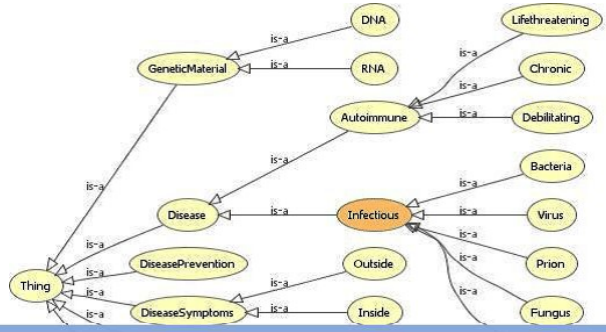
Leads Software Testing and Analysis Research Lab

Working on automated software testing, specifically test generation and fuzzing

My Journey of Software Testing



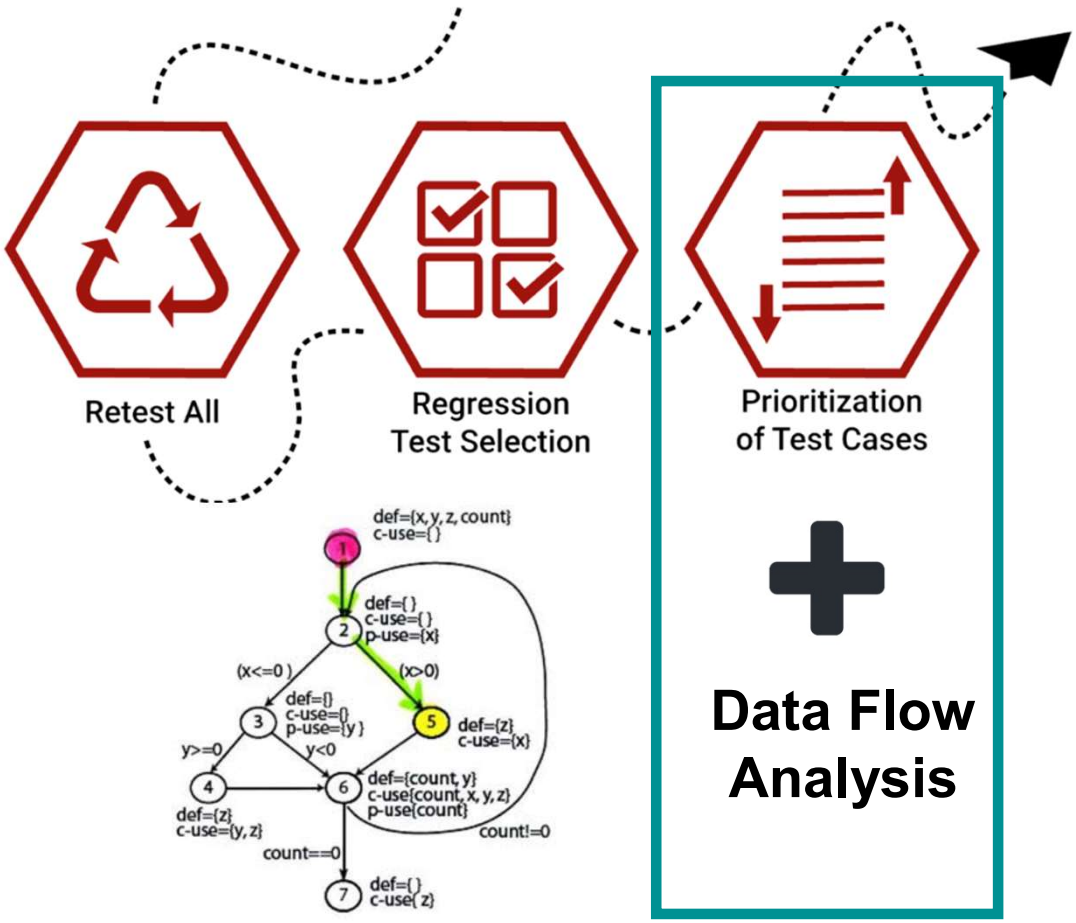
For Clinical Data Ontology



Began to think about practicality of testing research

My Journey of Software Testing

Regression Testing



Data Flow Analysis

Test Generation

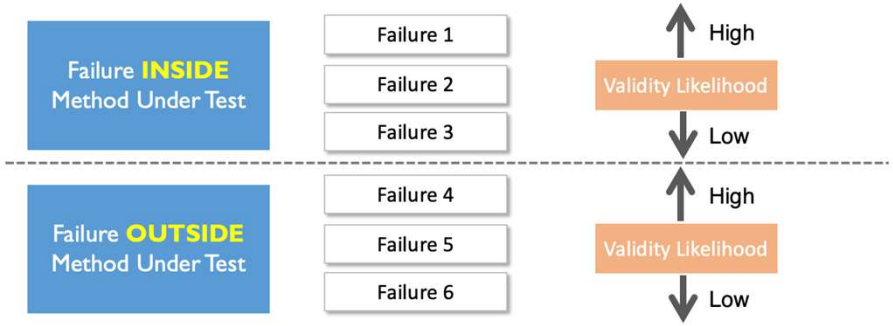
Randoop
Automatic unit test generation

EvoSuite

Many Failing Tests

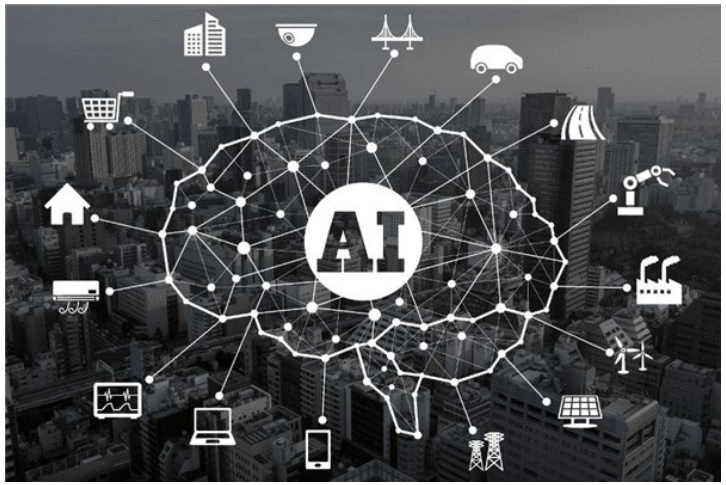
Many False Alarms

Failure Prioritization



My Journey of Software Testing

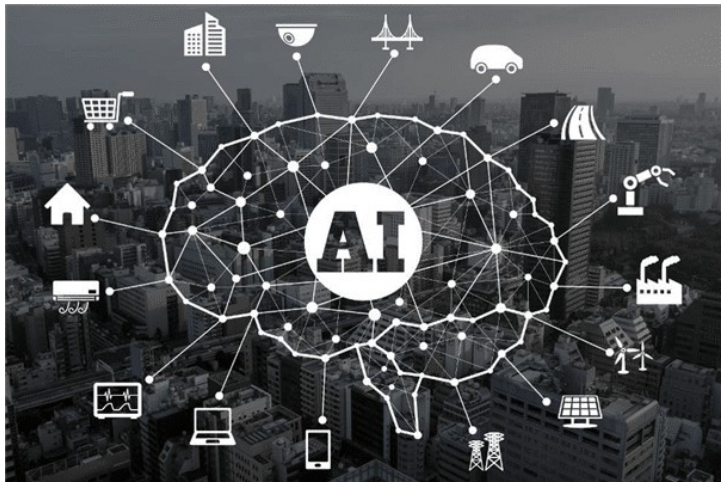
AI Software Systems



Data + Model

My Journey of Software Testing

AI Software Systems

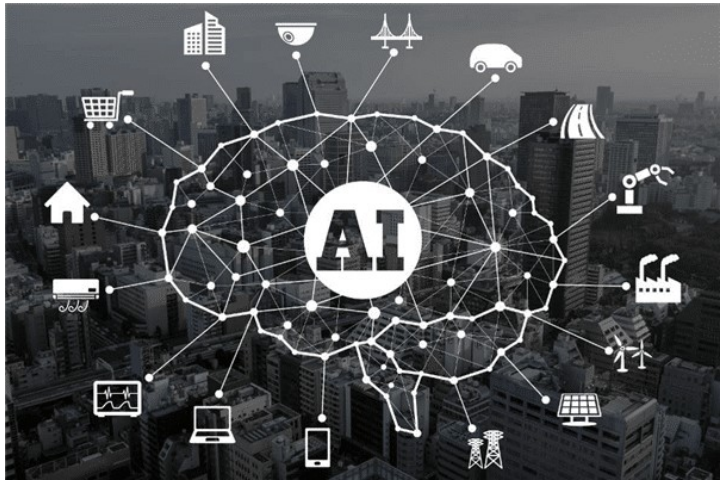


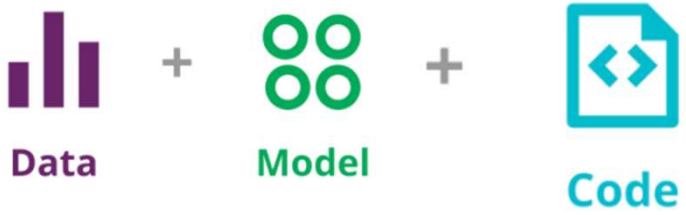
Data + Model + Code

My Journey of Software Testing

Test Generation

For AI Software Systems

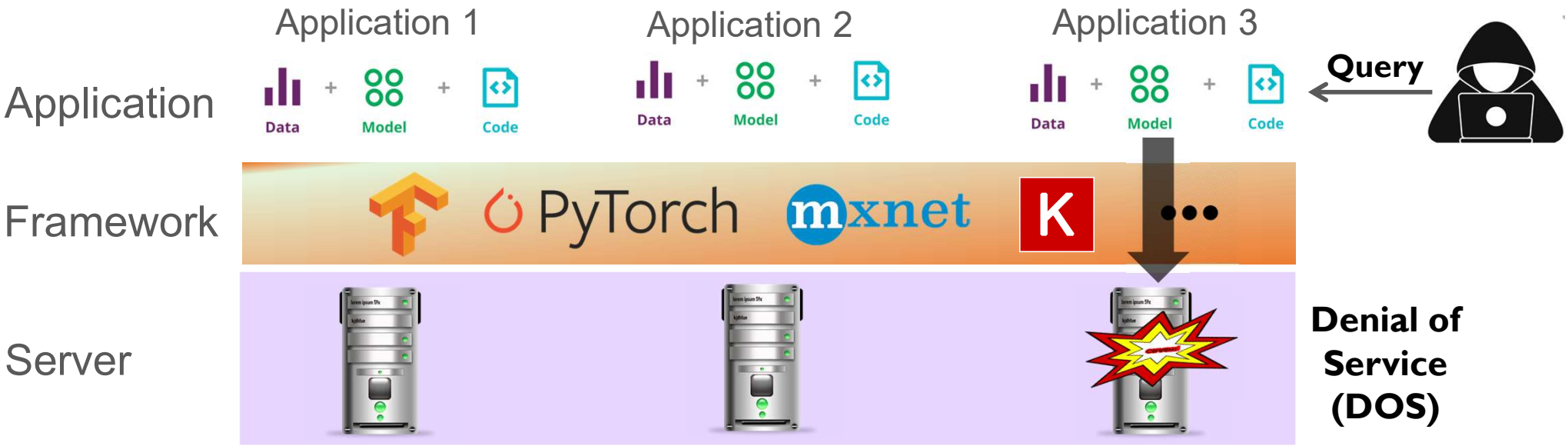




AI Software Systems

Machine Learning (ML) framework contains **bugs**
that may hurt model performance

AI Software Systems

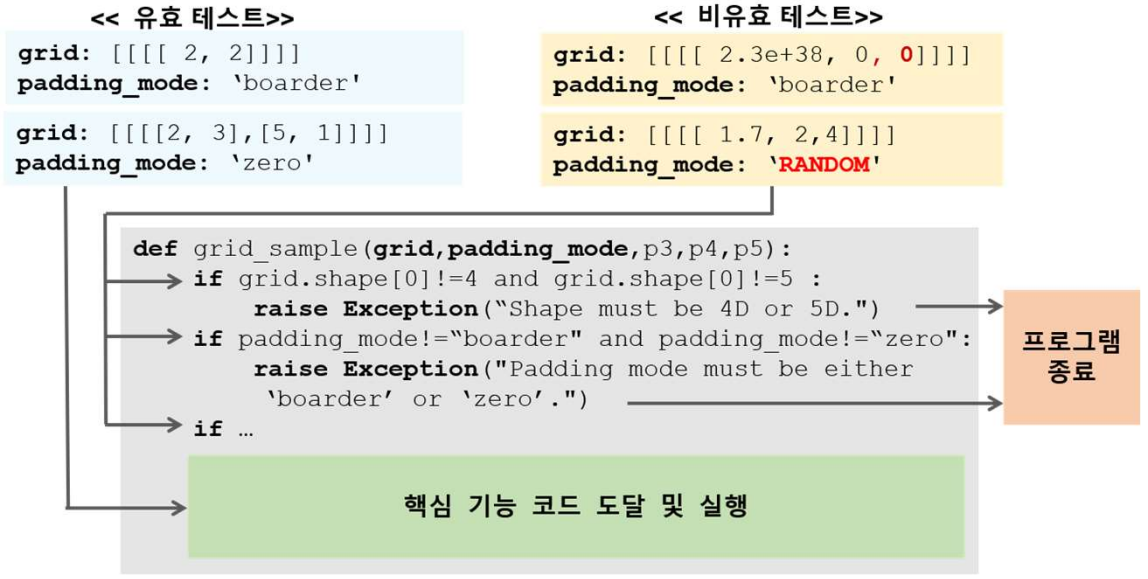


ML framework contains **vulnerabilities** that may lead to DOS

My AI Software Testing Journey Begins

How is  tested?

- Fuzz drivers are available only for 19 C++ APIs
- Generates inputs **in format of byte arrays**, which are hard to mutate to generate ***valid*** tensor objects
- Requires machine learning specific **input constraints**,



Our work, DocTor

- Motivated by necessity of **input validity**
- **Document-guided Test Generation tool for Deep Learning Frameworks**
  - Leverages documents to extract API constraints
  - Guides test generation based on constraints

- Document is a good source of **constraints**

`torch.nn.functional.grid_sample(input, grid, mode='bilinear', padding_mode='zeros',`

Parameters

- **input** (*Tensor*) – input of shape $(\mathrm{N}, \mathrm{C}, \mathrm{H}_{\mathrm{in}}, \mathrm{W}_{\mathrm{in}})$ (4-D case) or $(\mathrm{N}, \mathrm{C}, \mathrm{D}_{\mathrm{in}}, \mathrm{H}_{\mathrm{in}}, \mathrm{W}_{\mathrm{in}})$ (5-D case)
- **grid** (*Tensor*) – flow-field of shape $(\mathrm{N}, \mathrm{H}_{\mathrm{out}}, \mathrm{W}_{\mathrm{out}}, 2)$ (4-D case) or $(\mathrm{N}, \mathrm{D}_{\mathrm{out}}, \mathrm{H}_{\mathrm{out}}, \mathrm{W}_{\mathrm{out}}, 3)$ (5-D case)
- **padding_mode** (*str*) – padding mode for outside grid values `'zeros'` | `'border'` | `'reflection'`.

How to extract constraints

`torch.nn.functional.grid_sample(input, grid, mode='bilinear', pad(`
`align_corners=None)`

Parameters

- **input** (*Tensor*) – input of shape (N, C, H_{in}, W_{in}) (4-D case) or (N case)
- **grid** (*Tensor*) – flow-field of shape $(N, H_{out}, W_{out}, 2)$ (4-D case) o $(N, D_{out}, H_{out}, W_{out}, 3)$ (5-D case)
- **padding_mode** (*str*) – padding mode for outside grid values `'zeros'` | `'border'` | `'reflection'`

`torch.nn.functional.affine_grid(theta, size, align_corners=None)`

Parameters

- **theta** (*Tensor*) – input batch of affine matrices with shape $(N \times 2 \times 3)$ for 2D or $(N \times 3 \times 4)$ for 3D
- **size** (*torch.Size*) – the target output image size. $(N \times C \times H \times W$ for 2D or $N \times C \times D \times H \times W$ for 3D) Example: torch.Size((32, 3, 24, 24))

`torch.nn.functional.conv2d(input, weight, bias=None, stri`
`groups=1)` → Tensor

Parameters

- **input** – input tensor of shape $(minibatch, in\_chann$
- **weight** – filters of shape $(out\_channels, \frac{in\_channels}{groups},$
- **bias** – optional bias tensor of shape $(out\_channels)$.
- **stride** – the stride of the convolving kernel. Can be a sing

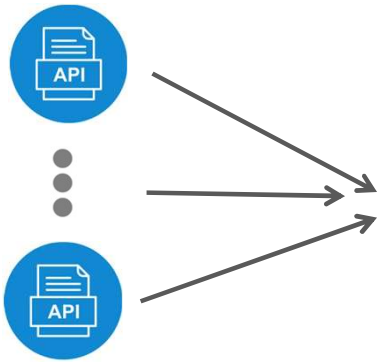
`torch.nn.functional.conv1d(input, weight, bias=None, stride=1, padding=0, dilation=1,`
`groups=1)` → Tensor

Parameters

- **input** – input tensor of shape $(minibatch, in\_channels, iW)$
- **weight** – filters of shape $(out\_channels, \frac{in\_channels}{groups}, kW)$
- **bias** – optional bias of shape $(out\_channels)$. Default: `None`
- **stride** – the stride of the convolving kernel. Can be a single number or a one-element tuple (*sW,*). Default: 1

How to extract constraints

<< API Documents >>



<< Frequent subtrees >>

	Subtree
1	D_TYPE of type
2	D_STRUCTURE CONSTAINT_NUM d
3	D_STRUCTURE CONSTAINT_NUM d
...	...

<< Constraints for each API >>


```
doc_dtype:
- Tensor
ndim:
- '4'
- '5'
shape:
- '[n,c,d_in,h_in,w_in]'
- '[n,c,h_in,w_in]'
```

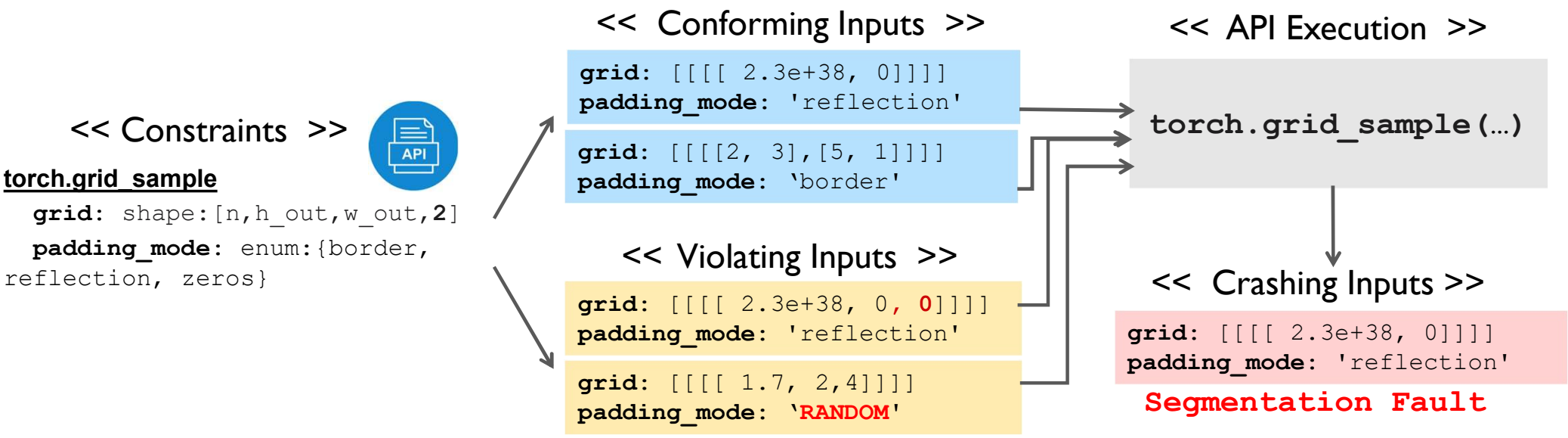

```
dtype:
- string
enum:
- border
- reflection
- zeros
```

<< Construct Rules>>

$$p = noise\_shape$$

$$IR(p) = p_T \in \{D\_TYPE\} \wedge p_S \in \{D\_STRUCTURE\} \wedge p_D \in \{CONSTANT\_NUM\}$$

How to generate test inputs



This crashing test input **detected new bug** fixed by PyTorch's developers

```
-   return minimum(Vec(max_val), maximum(in, Vec(0)));
+   // ... in order to clamp Nans to zero
+   return clamp_max(Vec(max_val), clamp_min(Vec(0), in));
```

Results – Quality of Constraint Extraction

Ran DocTer for  PYTORCH mxnet

2,415 out of 2,558 APIs with constraints extracted

1,338 rules automatically constructed

16,035 constraints extracted in total

603 parameters manually evaluated

85.4% precision achieved

Results – Bug Detection

2,000 inputs

2,000 inputs

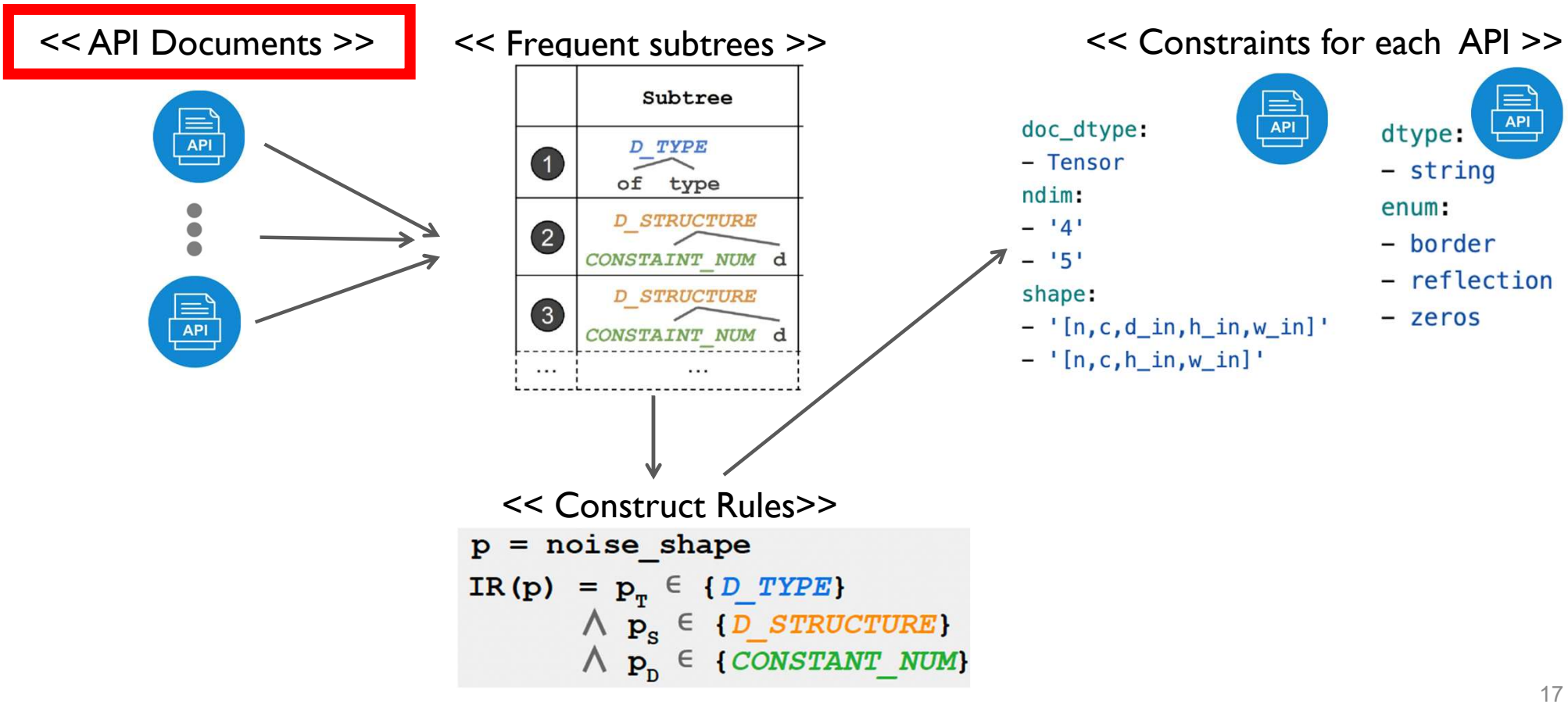
1,000 inputs

1,000 inputs

	TensorFlow	PyTorch	MXNet	Total
Baseline	41	7	11	59
DocTer	61	18	15	**94**
DocTer - CI	47	14	14	75
DocTer - VI	51	18	12	81

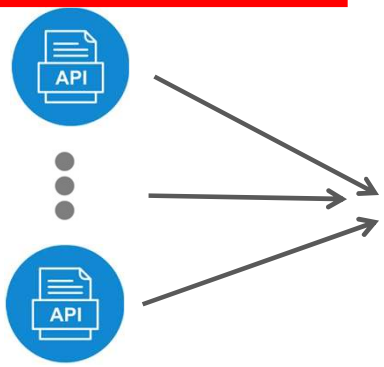
Baseline: no constraint provided
CI: conforming Inputs
VI: violating Inputs

Big Limitation of DocTor



Big Limitation of DocTor

Manually Annotated API Documents



<< Frequent subtrees >>

	Subtree
1	D_TYPE of type
2	D_STRUCTURE CONSTAINT_NUM d
3	D_STRUCTURE CONSTAINT_NUM d
...	...

<< Construct Rules>>

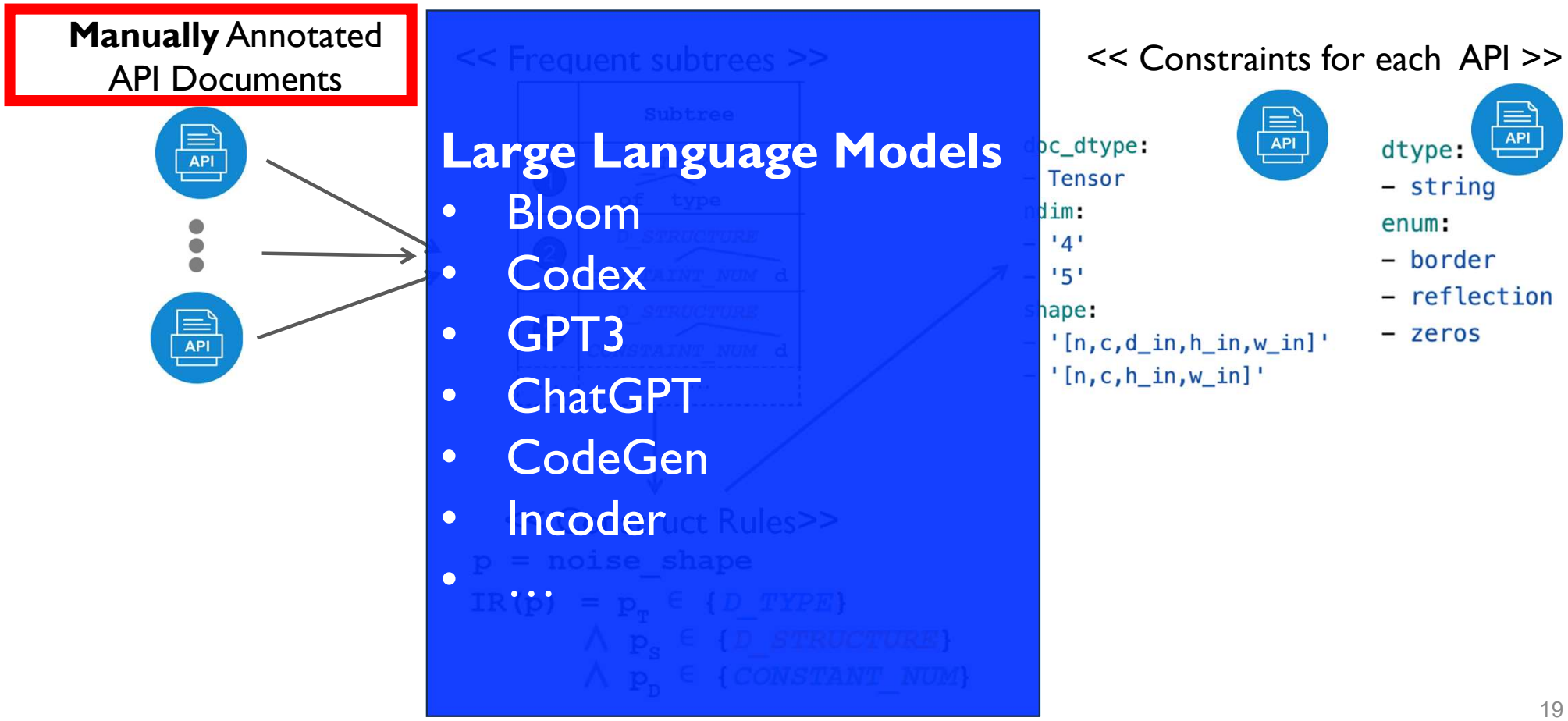
$$p = noise\_shape$$
$$IR(p) = p_T \in \{D\_TYPE\} \wedge p_S \in \{D\_STRUCTURE\} \wedge p_D \in \{CONSTANT\_NUM\}$$

<< Constraints for each API >>


```
doc_dtype:
- Tensor
ndim:
- '4'
- '5'
shape:
- '[n,c,d_in,h_in,w_in]'
- '[n,c,h_in,w_in]'
```


```
dtype:
- string
enum:
- border
- reflection
- zeros
```

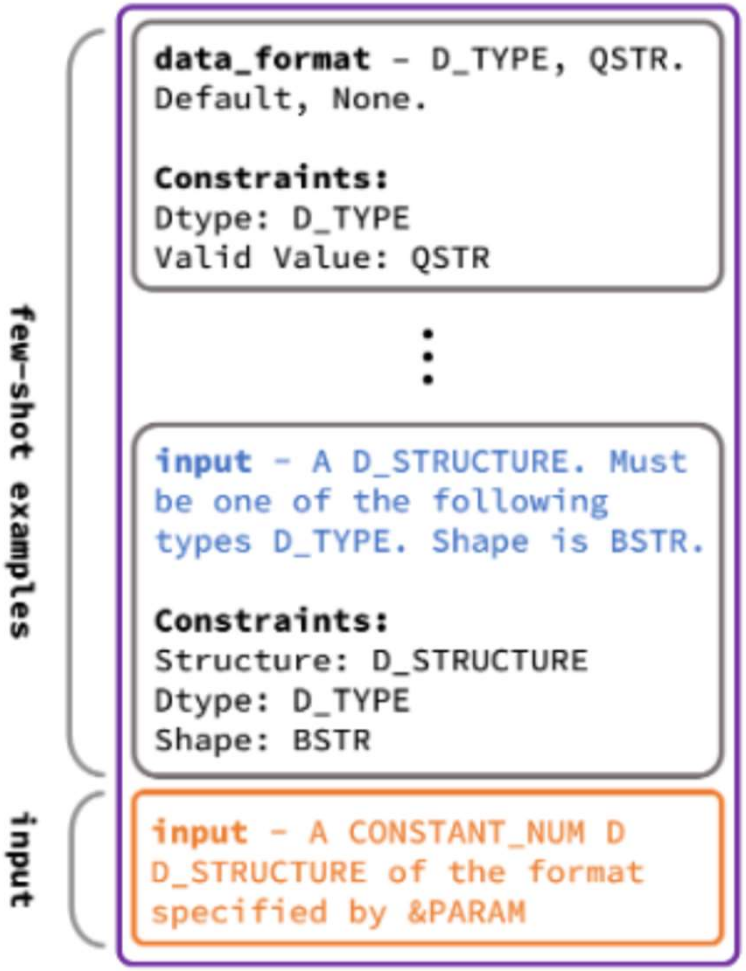
Ongoing: using LLM for generating constraints



Prompt Construction

```
API parameter description: <EX_1-DESCP>
Constraints: <EX_1-CONSTR>
...
API parameter description: <EX_k-DESCP>
Constraints: <EX_k-CONSTR>

API parameter description: <INP-DESCP>
Constraints: <OUT-CONSTR>
```



DocLang's Results with BLOOM

Within Same Project

Subject	Approach	# Train	Total *P/R/F1* (%)
PyTorch	**DocLang**	80	87.8/81.2/**84.4**
		40	83.5/71.6/77.1
		20	83.0/68.5/75.1
	DocTer	80	86.7/31.3/46.0
		40	96.5/14.4/25.1
		20	96.4/9.3/24.7
TensorFlow	**DocLang**	80	81.5/77.3/**79.4**
		40	77.5/70.9/74.0
		20	76.0/71.4/73.6
	DocTer	80	76.3/42.6/54.7
		40	79.5/24.9/37.9
		20	35.5/6.18/10.5
MXNet	**DocLang**	80	85.2/75.0/**79.8**
		40	80.2/71.8/75.8
		20	78.8/71.1/74.8
	DocTer	80	83.3/37.4/51.6
		40	88.0/36.2/51.3
		20	91.6/16.0/27.3

Cross Projects

Subject	Approach	Target	Total *P/R/F1* (%)
PyTorch	**DocLang**	TensorFlow	70.0/48.4/**57.2**
		MXNet	82.9/64.7/**72.7**
	DocTer	Tensorflow	70.7/8.3/14.9
		MXNet	89.9/23.7/37.5
TensorFlow	**DocLang**	PyTorch	62.8/66.5/**64.6**
		MXNet	74.6/66.9/**70.6**
	DocTer	PyTorch	44.6/4.2/7.7
		MXNet	68.6/10.8/18.7
MXNet	**DocLang**	PyTorch	64.2/56.6/**60.1**
		TensorFlow	58.9/43.2/**49.8**
	DocTer	PyTorch	40.5/5.4/9.6
		TensorFlow	41.8/6.4/11.1

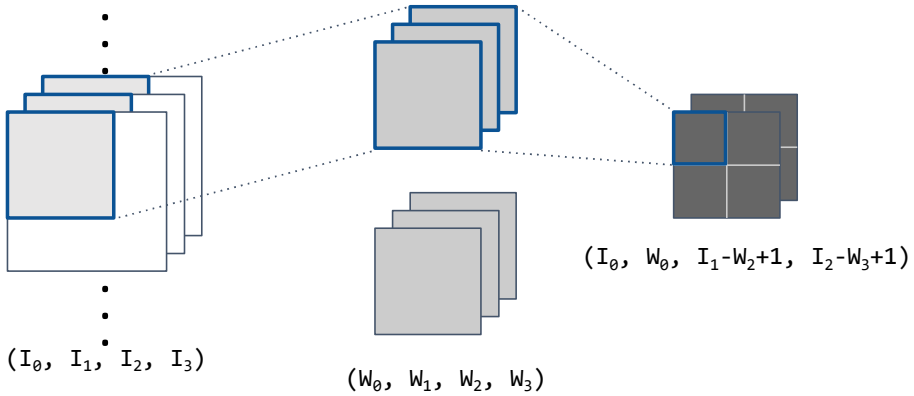
More results

Approach/ Model		Overall F1 (%)				Avg. time (s) per request	Cost K=10 ($)
		K=10	20	40	60		
Codex	davinci	86.0	87.4	87.6	**88.2**	2.2	0
GPT-3	davinci	84.6	86.5	-	-	2.1	26.2
	curie	76.9	-	-	-	0.7	2.62
GPT-3.5	turbo	81.3	83.7	-	-	0.8	2.62
BLOOM		78.1	-	-	-	12.6	0
CodeGen (Multi)	16B	80.4	-	-	-	7.5	0
	6B	78.5	-	-	-	3.9	0
	2B	75.1	-	-	-	3.3	0
	350M	74.6	-	-	-	1.1	0
Incoder	6B	76.4	-	-	-	2.1	0
	1B	78.7	-	-	-	0.6	0

More limitations

- Still, it's hard to generate ***valid*** test input.

`conv2d(input, weight, s, p, d, g):`



TORCH.NN.FUNCTIONAL.CONV2D

torch.nn.functional.conv2d(*input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1*) → Tensor

Parameters:

- **input** – input tensor of shape $(\text{minibatch}, \text{in\_channels}, iH, iW)$
- **weight** – filters of shape $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, kH, kW)$
- **bias** – optional bias tensor of shape (out_channels). Default: `None`
- **stride** – the stride of the convolving kernel. Can be a single number or a tuple (*sH, sW*). Default: 1
- **padding** –

  implicit paddings on both sides of the input. Can be a string {'valid', 'same'}, single number or a tuple (*padH, padW*). Default: 0 `padding='valid'` is the same as no padding. `padding='same'` pads the input so the output has the same shape as the input. However, this mode doesn't support any stride values other than 1.

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

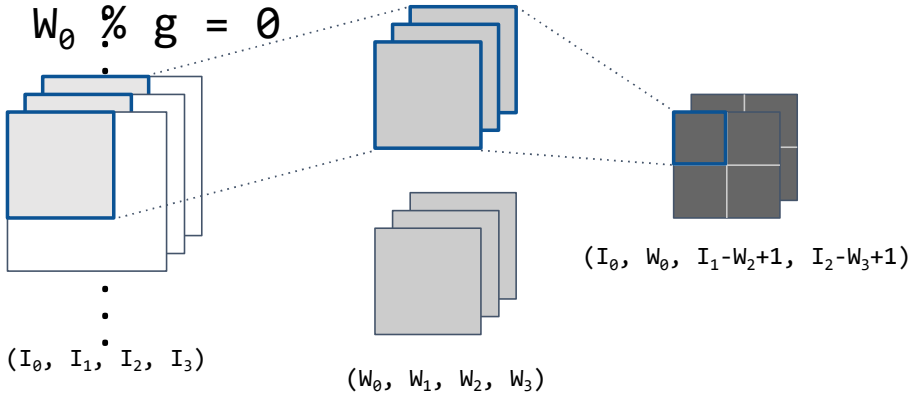
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

More limitations

- Still, it's hard to generate ***valid*** test input.

```
conv2d(input, weight, s, p, d, g):
 I₁ = W₁×g ∧
  I₂ + 2×p₀ - d₀ × (W₂-1) ≥ 1 ∧
  I₃ + 2×p₁ - d₁ × (W₃-1) ≥ 1 ∧
  s₀ ≥ 1 ∧ s₁ ≥ 1 ∧
  W₀ % g = 0
```



TORCH.NN.FUNCTIONAL.CONV2D

torch.nn.functional.conv2d(*input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1*) → Tensor

Parameters:

- **input** – input tensor of shape $(\text{minibatch}, \text{in\_channels}, iH, iW)$
- **weight** – filters of shape $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, kH, kW)$
- **bias** – optional bias tensor of shape (out_channels). Default: `None`
- **stride** – the stride of the convolving kernel. Can be a single number or a tuple (*sH, sW*). Default: 1
- **padding** –

  implicit paddings on both sides of the input. Can be a string {'valid', 'same'}, single number or a tuple (*padH, padW*). Default: 0 `padding='valid'` is the same as no padding. `padding='same'` pads the input so the output has the same shape as the input. However, this mode doesn't support any stride values other than 1.

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

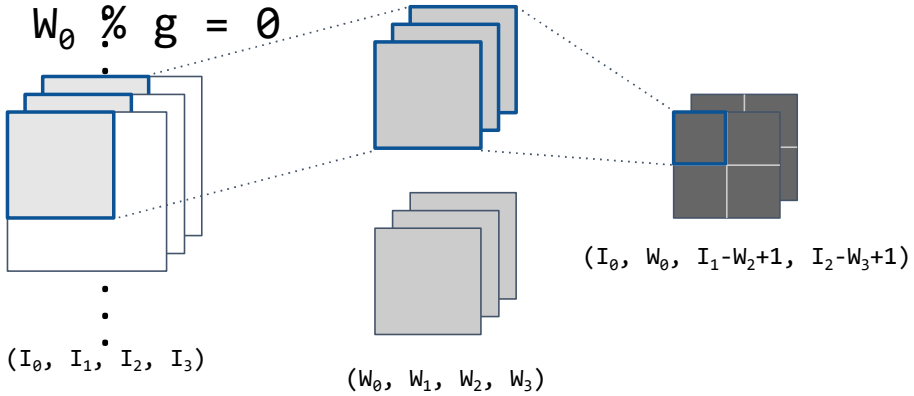
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

More limitations

- Still, it's hard to generate ***valid*** test input.

```
conv2d(input, weight, s, p, d, g):
 I1 = W1×g ∧
  I2 + 2×p0 - d0 × (W2-1) ≥ 1 ∧
  I3 + 2×p1 - d1 × (W3-1) ≥ 1 ∧
  s0 ≥ 1 ∧ s1 ≥ 1 ∧
  W0 % g = 0
```



DocTer:
Relational constraints are hard to be extracted from document.

FreeFuzz:
Mutation on individual argument is prone to violate relational constraints.

TORCH.N

torch.nn.functio ... ps=1) →

Parameters:

- **input** – input tensor of shape $(\text{minibatch}, \text{in\_channels}, iH, iW)$
- **bias**
- **stride** – the st
- **padding** – implicit paddi ... le (*padH*, *padW*). Defau ... the output has the same shape as the input. However, this mode doesn't support any stride values other than 1.

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

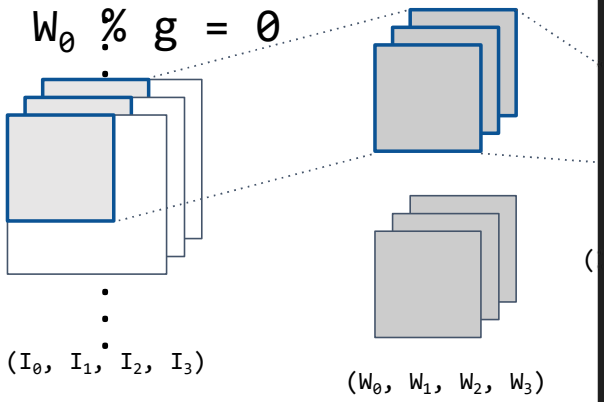
More limitations

- It's hard to test ***diverse*** code region.

{DocTer, FreeFuzz}'s **black-box** fuzzing is not good for finding unexplored codes.

conv2d(input, weight, s, p, d, g):

$$
\begin{aligned}
& I_1 = W_1 \times g \ \wedge \\
& I_2 + 2 \times p_0 - d_0 \times (W_2 - 1) \\
& I_3 + 2 \times p_1 - d_1 \times (W_3 - 1) \\
& s_0 \geq 1 \ \wedge \ s_1 \geq 1 \ \wedge \\
& W_0 \ \% \ g = 0
\end{aligned}
$$




```
auto ConvParams::use_nnpack(const at::Tensor& input, const at::Tensor& weight) const -> bool {
#if AT_NNPACK_ENABLED()
  return at::_nnpack_available() &&
         input.device().is_cpu() &&
         input.scalar_type() == kFloat && // only on CPU Float Tensors
         !is_dilated() && // or dilation
         !transposed &&   // or transposed tensors
         input.ndimension() == 4 && // must be in NCHW format
         weight.ndimension() == 4 &&
         (weight.size(2) < 17) && (weight.size(3) < 17) // NNPACK only supports kernels up to 16x16
#if !defined(C10_MOBILE)
         && input.size(0) >= 16 // ensure large enough batch size to ensure perf, tuneable
#endif
     ;
#endif
  return false;
}
```

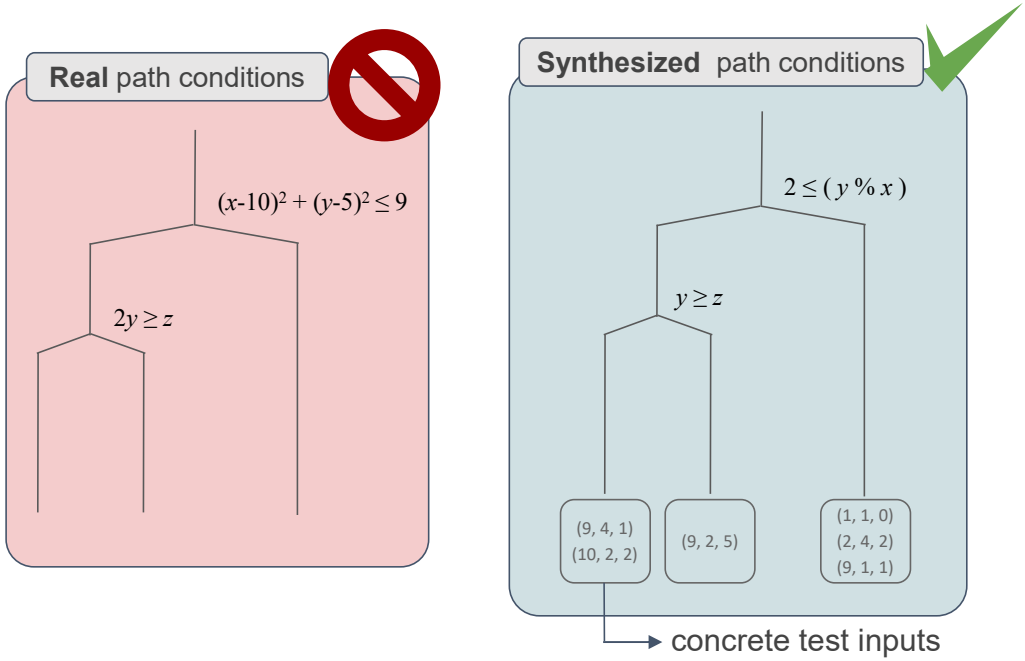
subset of conv2d API which needs narrower constraints

Traditional fuzzing technique can’t be a solution

- Challenge: Input ***validity*** and ***diversity***
- Grey-box fuzzing
  - Byte level mutation apt to break input validity
- Dynamic Symbolic Execution (a.k.a White-box fuzzing)
  - It is not suitable for DLL which
    - has huge code size
    - has complex path conditions
    - uses lots of external libraries

Ongoing: fuzzing via path condition synthesis

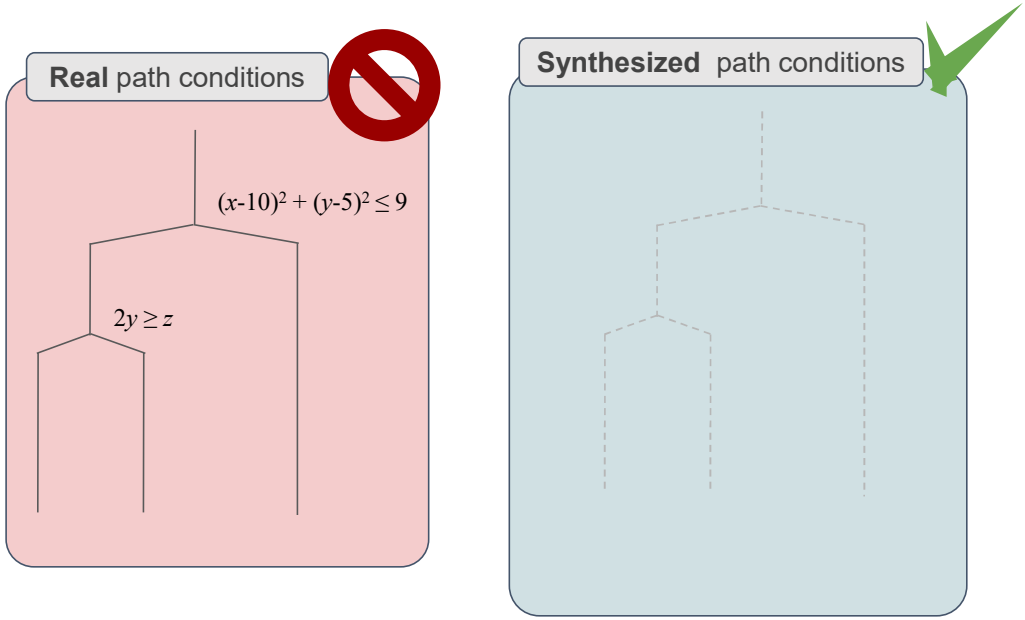
- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.



[2]We use DUET[7] as a path condition synthesizer.

Ongoing: fuzzing via path condition synthesis

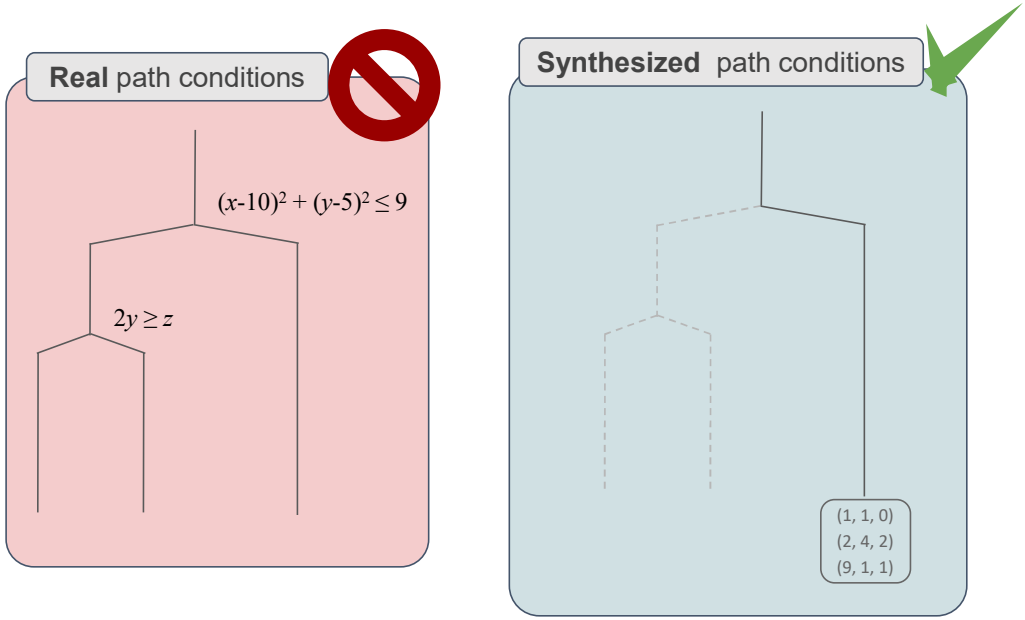
- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.



[2]We use DUET[7] as a path condition synthesizer.

Ongoing: fuzzing via path condition synthesis

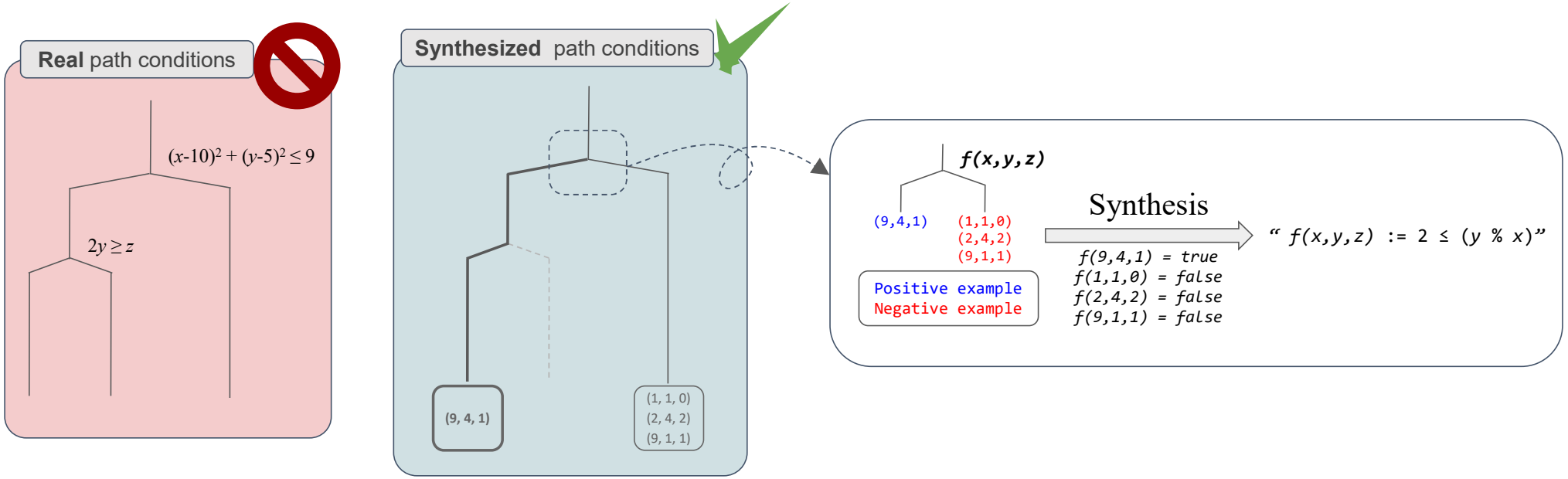
- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.



[2]We use DUET[7] as a path condition synthesizer.

Ongoing: fuzzing via path condition synthesis

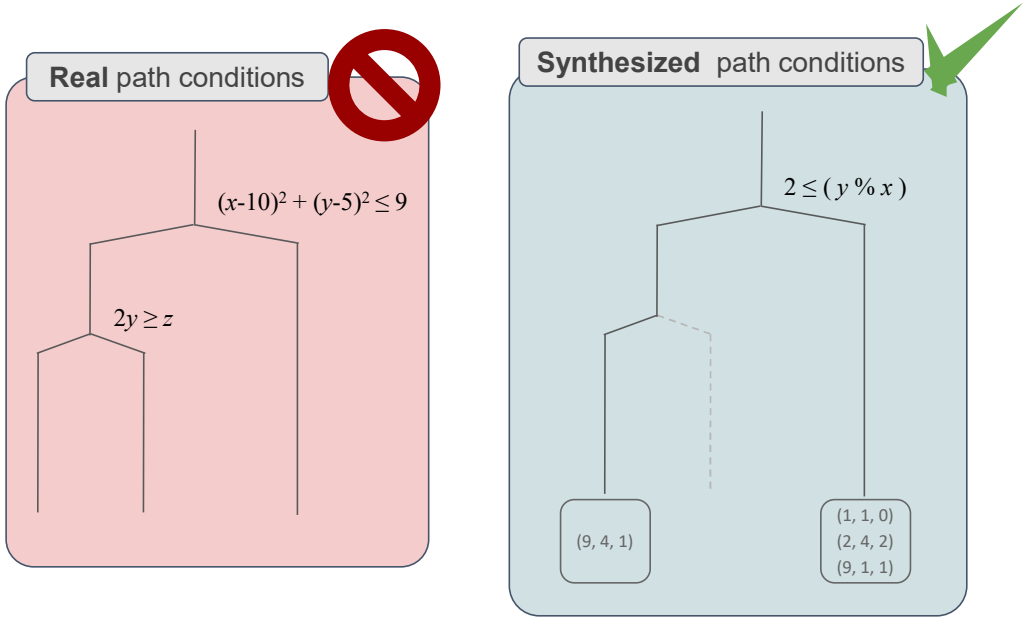
- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.



[2]We use DUET[7] as a path condition synthesizer.

Ongoing: fuzzing via path condition synthesis

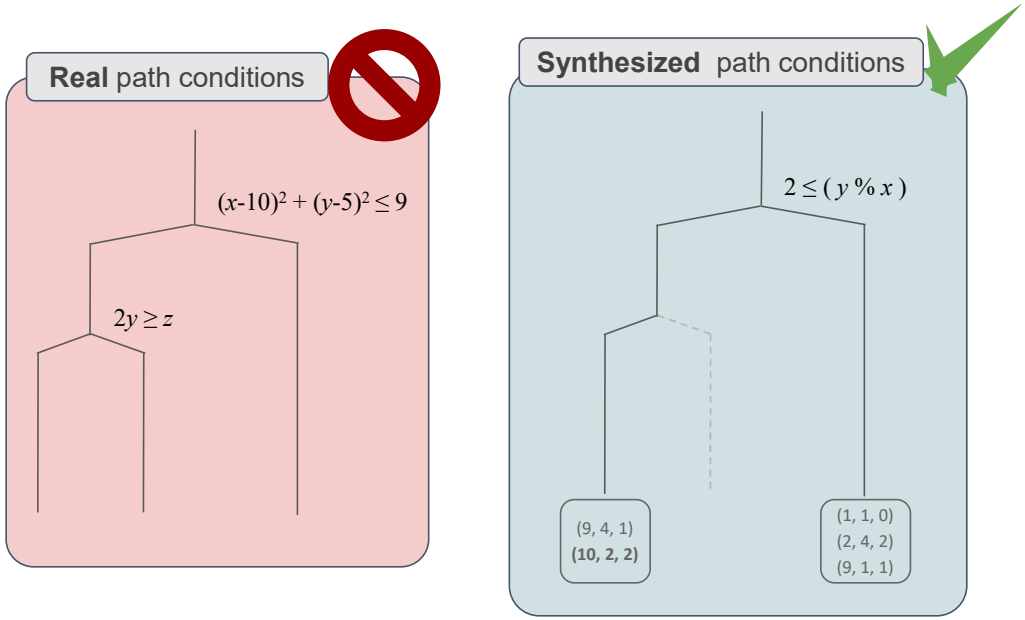
- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.



[2]We use DUET[7] as a path condition synthesizer.

Ongoing: fuzzing via path condition synthesis

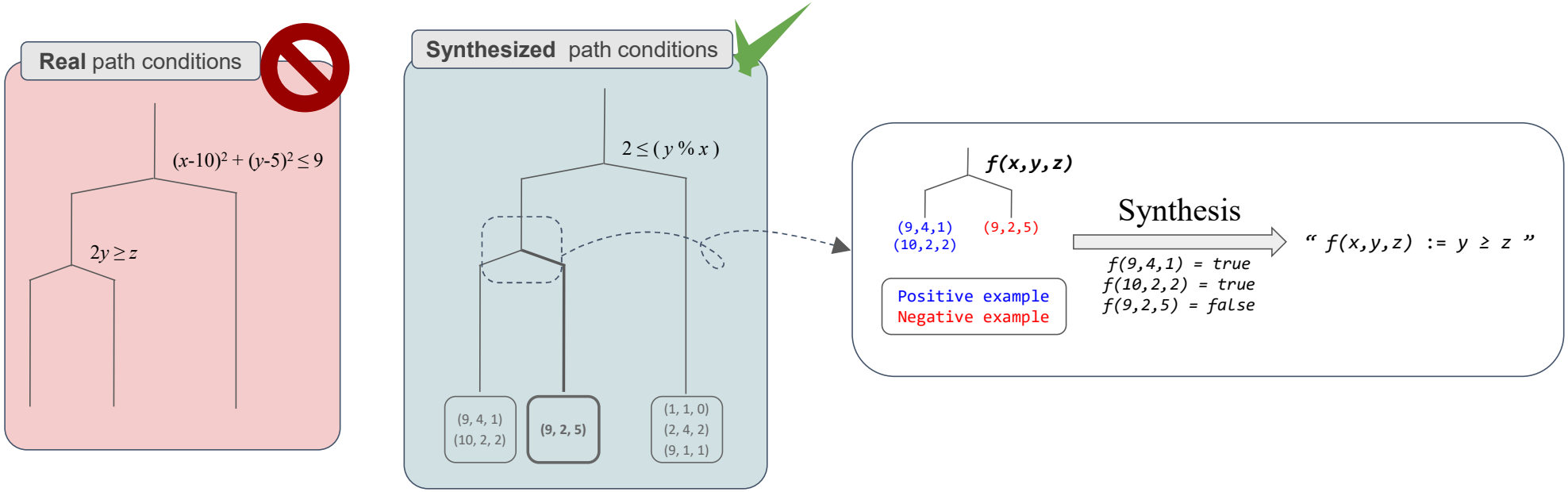
- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.



[2]We use DUET[7] as a path condition synthesizer.

Ongoing: fuzzing via path condition synthesis

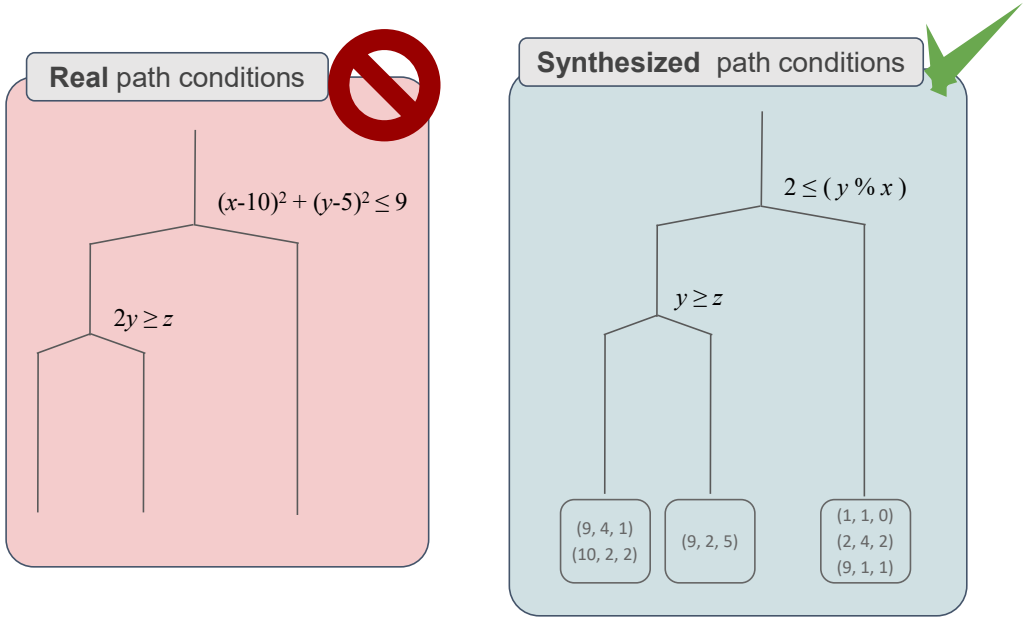
- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.

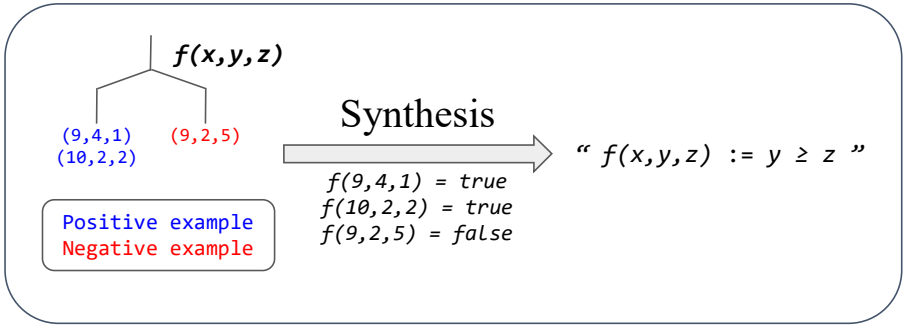


[2]We use DUET[7] as a path condition synthesizer.

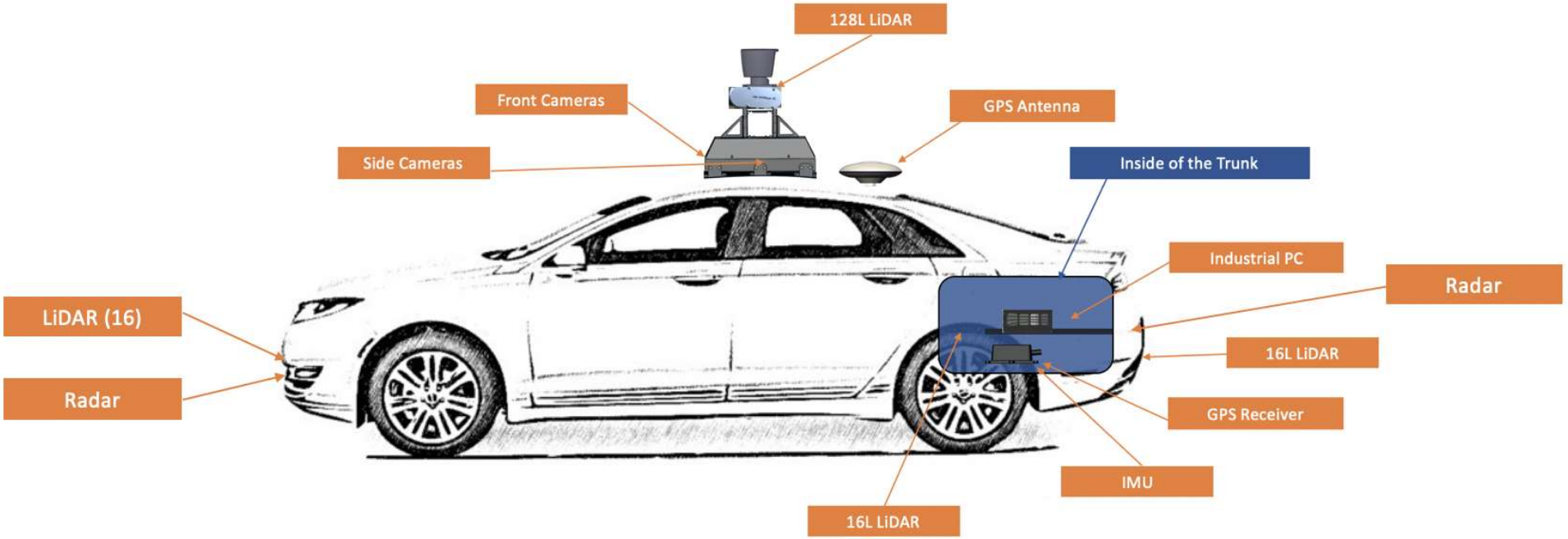
Ongoing: fuzzing via path condition synthesis

- Idea: Use **approximated** path condition with **light weight** instrumentation, instead of **exact** path condition with **heavy** equipment.





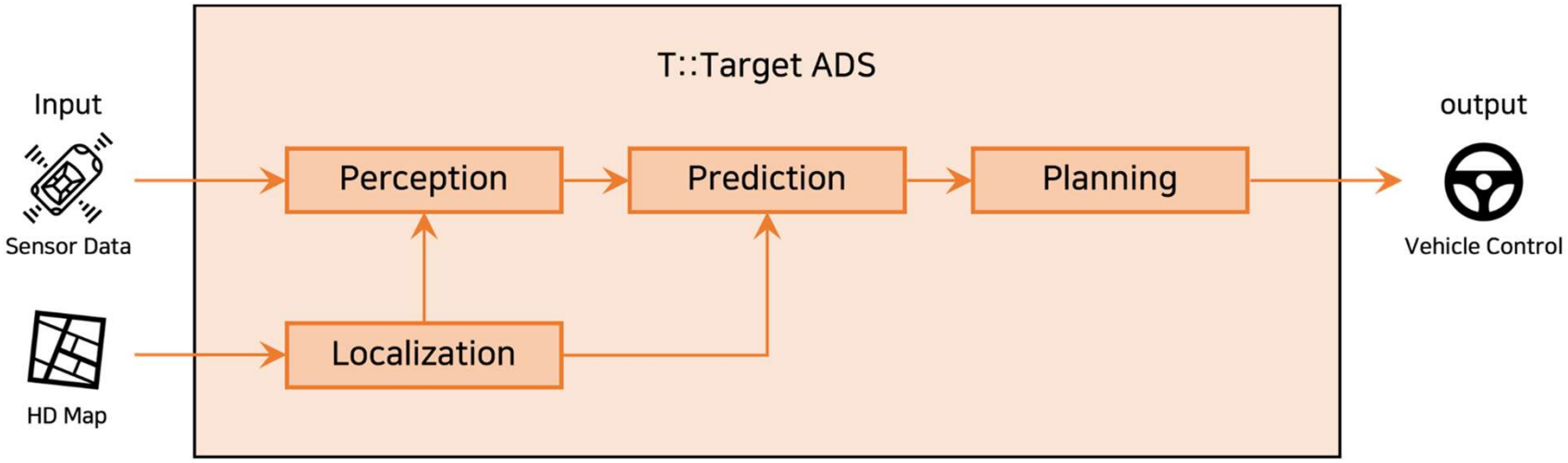
[2]We use DUET[7] as a path condition synthesizer.



- Sensing Data is **the input of ADSs**
- Collect all of sensor **data in real-time**
- e.g. camera (light info), radar(RF signal), LiDAR(high-power laser signal)

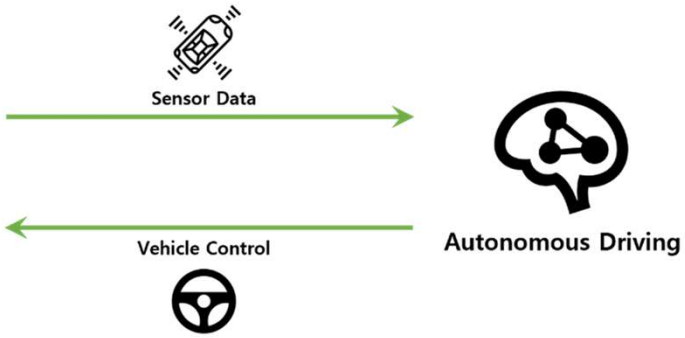
Autonomous Driving Systems (ADSs)

Software Overview of ADSs



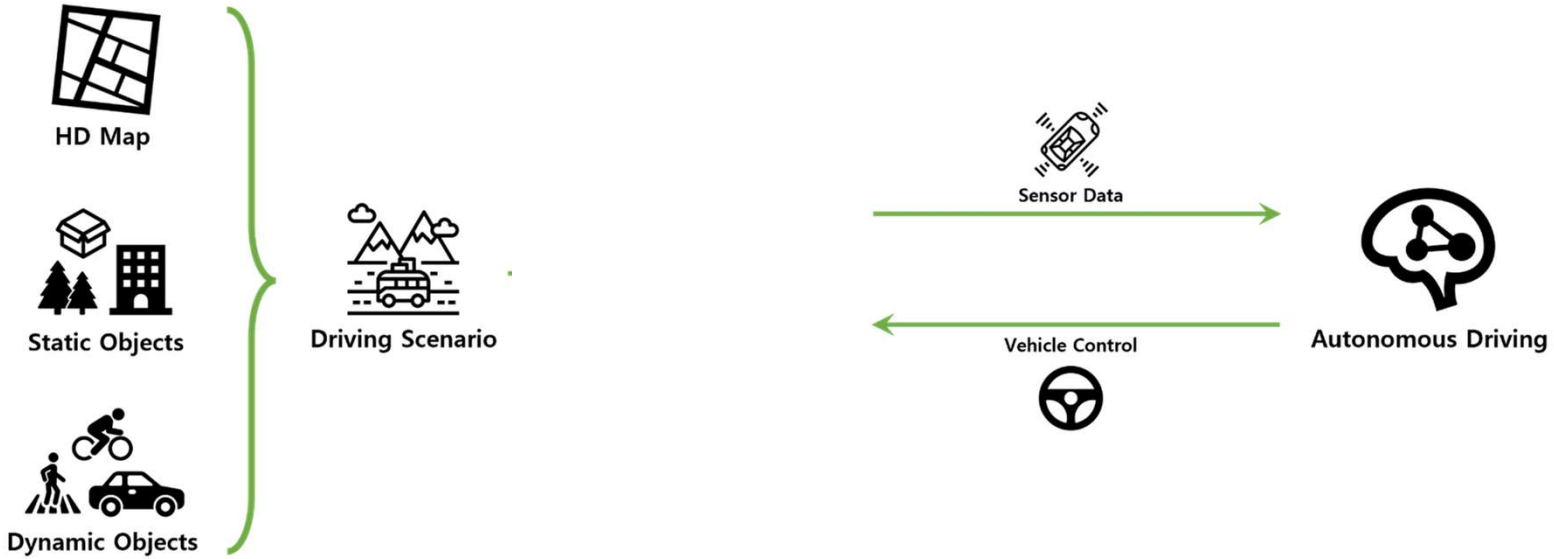
- Perception : object detection, object classification and tracking (dynamic object, traffic light)
- Prediction : object trajectory, object behavior, object maneuver, risk
- Planning : global planning, local planning(obstacle avoidance), maneuver decision

Autonomous Driving Simulation



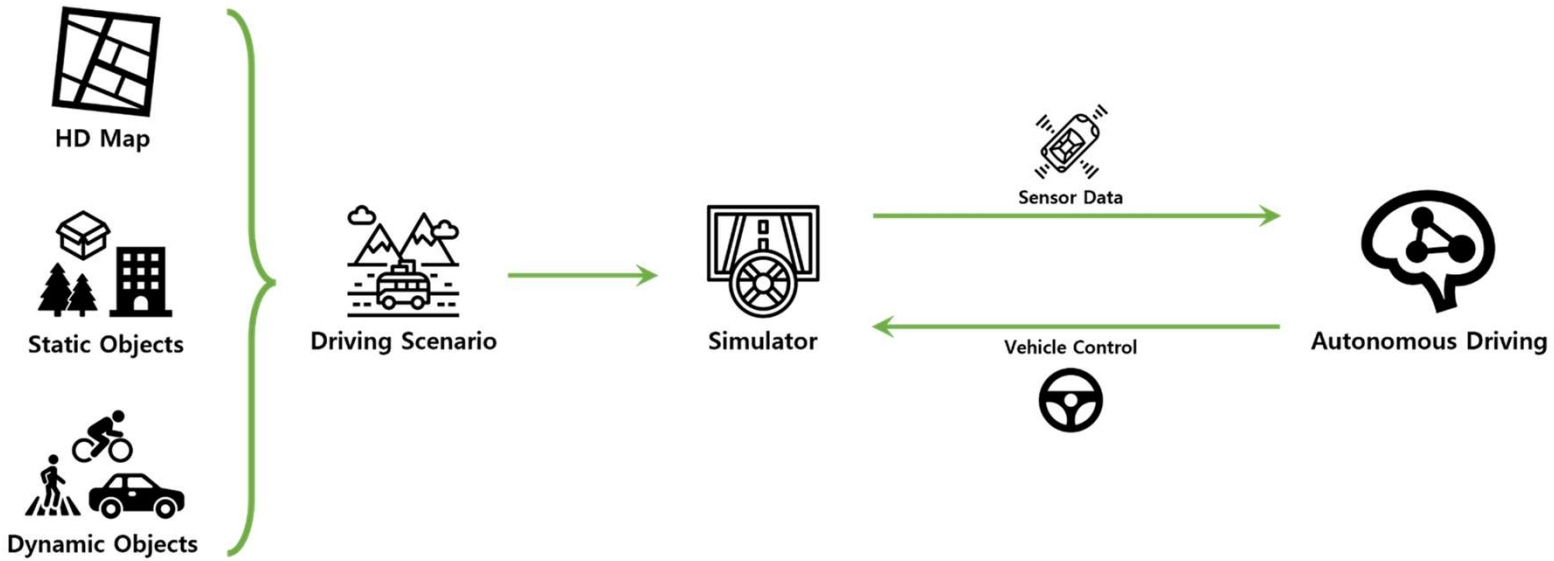
- Utilize 3D game engines to construct photo-realistic virtual driving scenes
- Provide various sensors, such as LiDAR, camera, IMU, and GPS
- Provide a communication bridge that exchanges messages

Autonomous Driving Simulation



- Utilize 3D game engines to construct photo-realistic virtual driving scenes
- Provide various sensors, such as LiDAR, camera, IMU, and GPS
- Provide a communication bridge that exchanges messages

Autonomous Driving Simulation



- Utilize 3D game engines to construct photo-realistic virtual driving scenes
- Provide various sensors, such as LiDAR, camera, IMU, and GPS
- Provide a communication bridge that exchanges messages

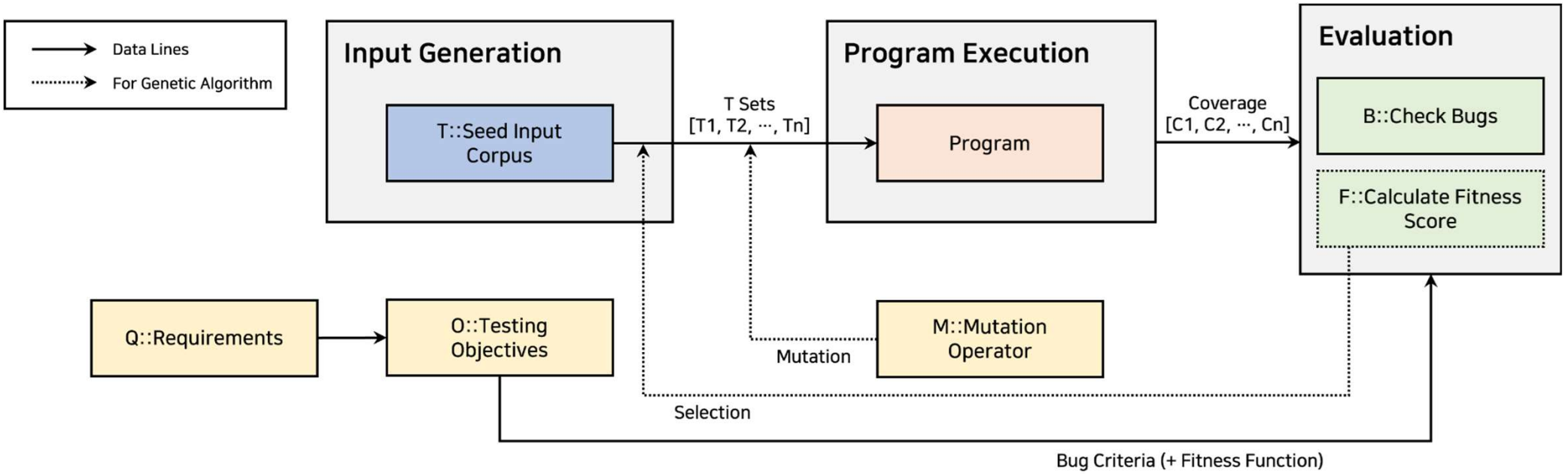
What is a driving scenario?

Components	Description
HD-Map	ASAM OpenDRIVE High-definition map
Environments	Weather
	Friction of Road
	Static Object
	Traffic Light
Ego-vehicle Route	In case of the vehicle that under the end-to-end ADS, we don't need each trajectory by time.
NPC Actors	Vehicle
	Pedestrian
	Bicycle

```
num_of_vehicle = 2
num_of_pedestrian = 1
vehicle_x_0 = 120
vehicle_y_0 = 70
vehicle_yaw_0 = 60
vehicle_speed_0 = 7
vehicle_x_1 = 190
…
pedestrian_x_0 = 210
…
```

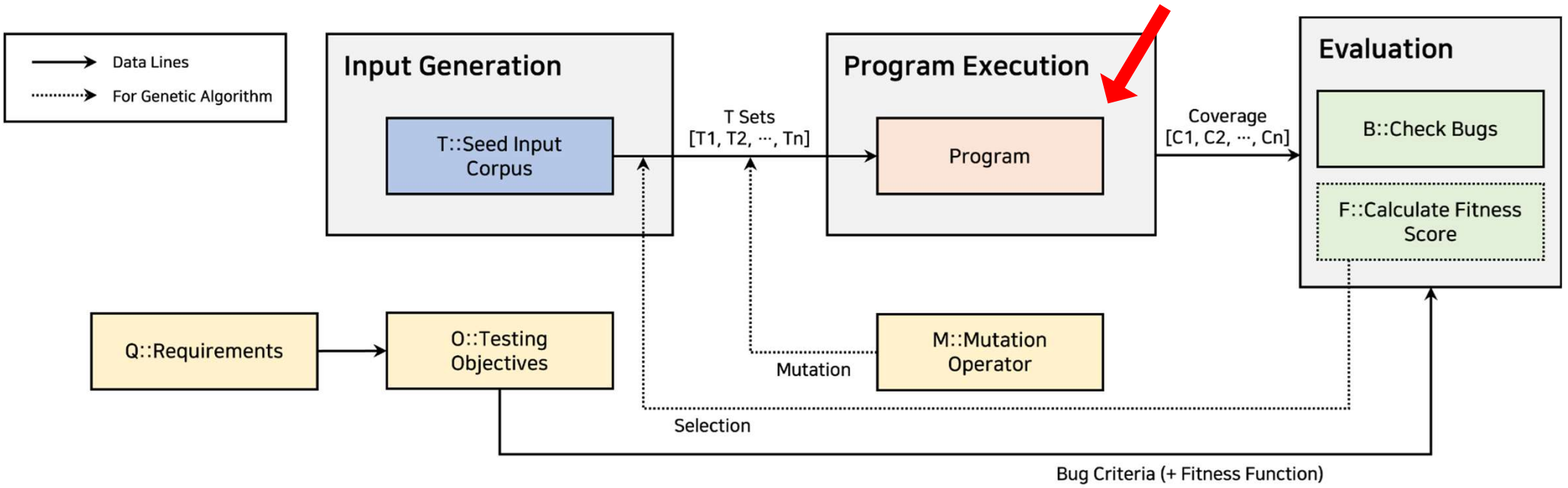
Scenario Example

Fuzzing Autonomous Driving System

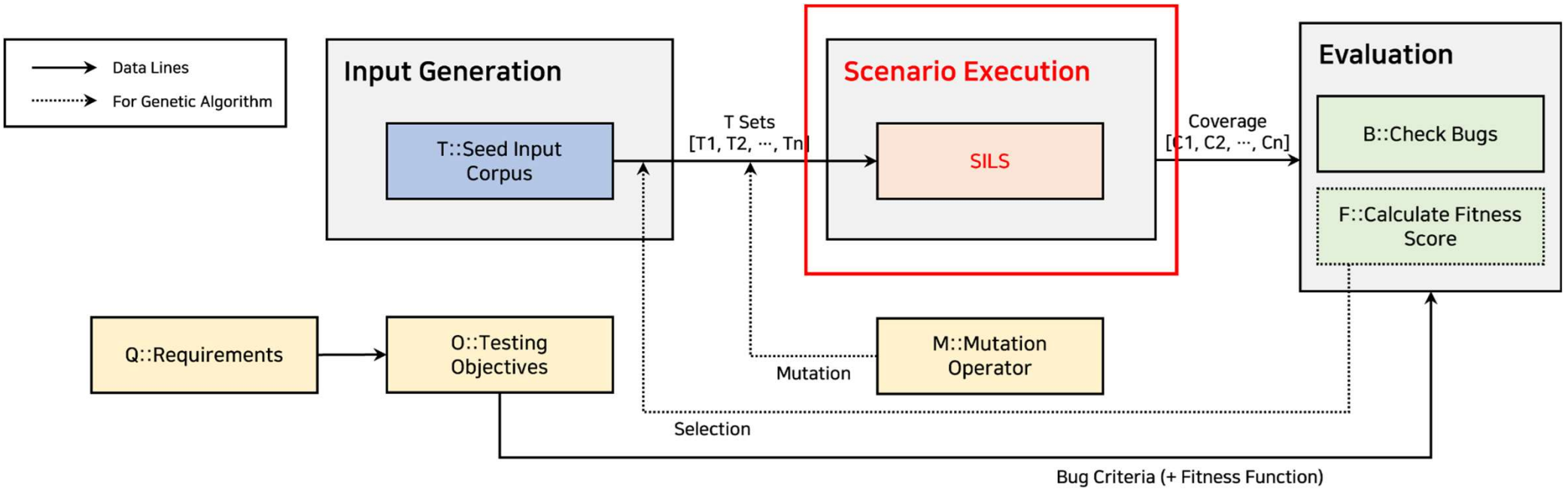


Normal Evolutionary Fuzzing Overview

Fuzzing Autonomous Driving System

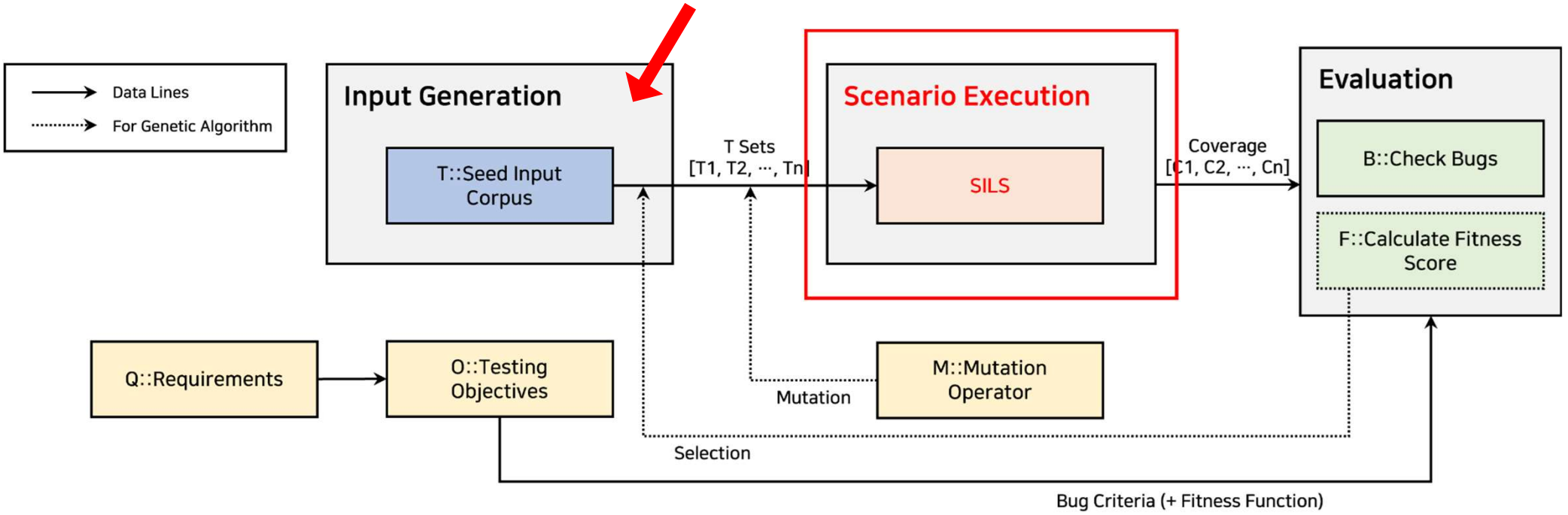


Fuzzing Autonomous Driving System

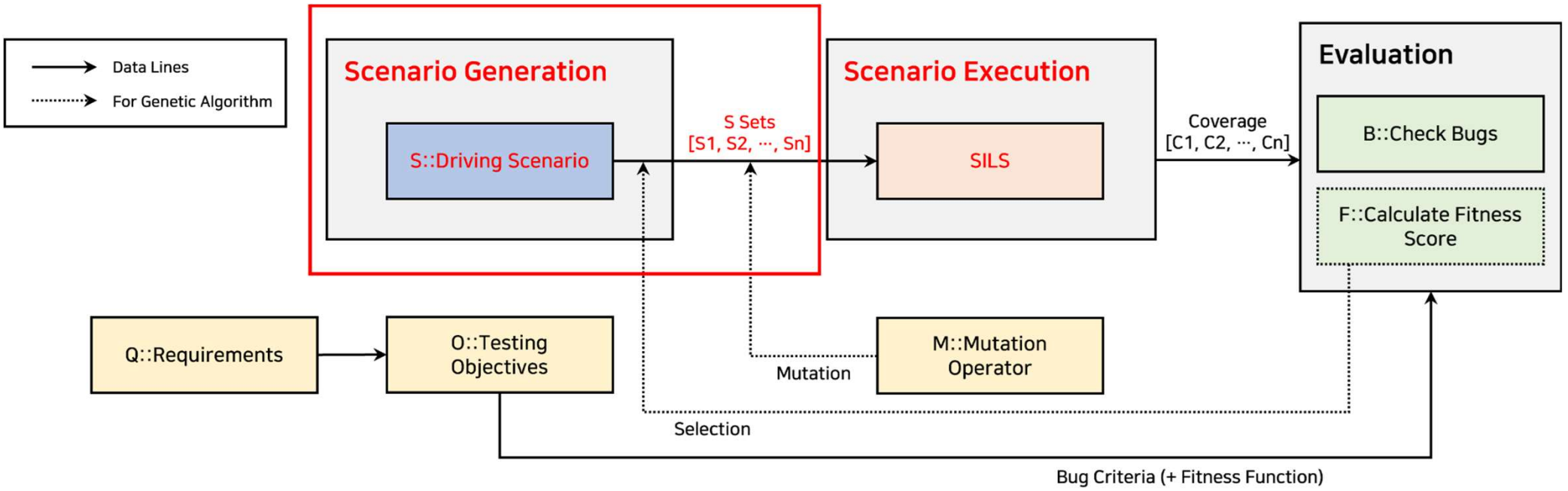


Program Execution >> Scenario Execution

Fuzzing Autonomous Driving System

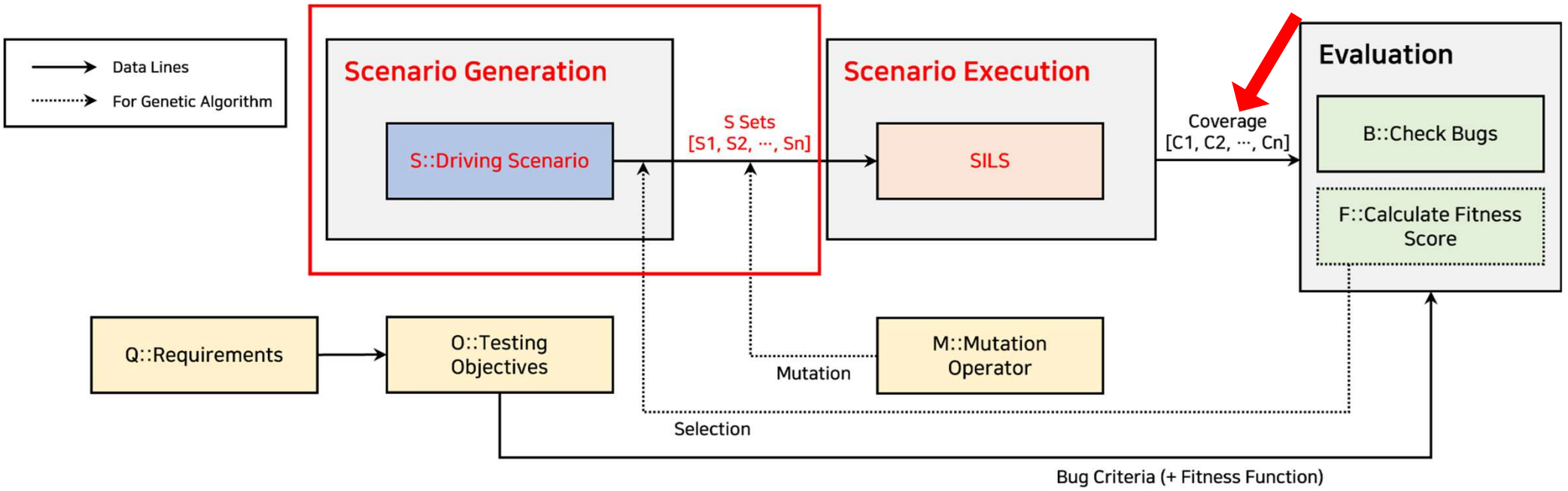


Fuzzing Autonomous Driving System

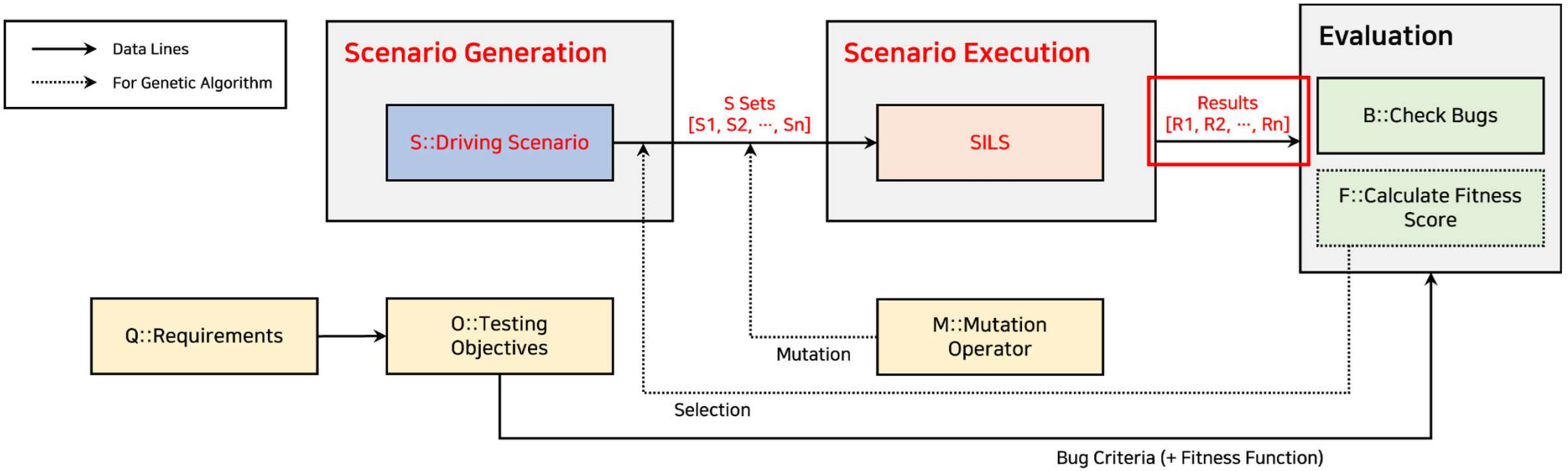


Seed Input Corpus >> S::Driving Scenario

Fuzzing Autonomous Driving System

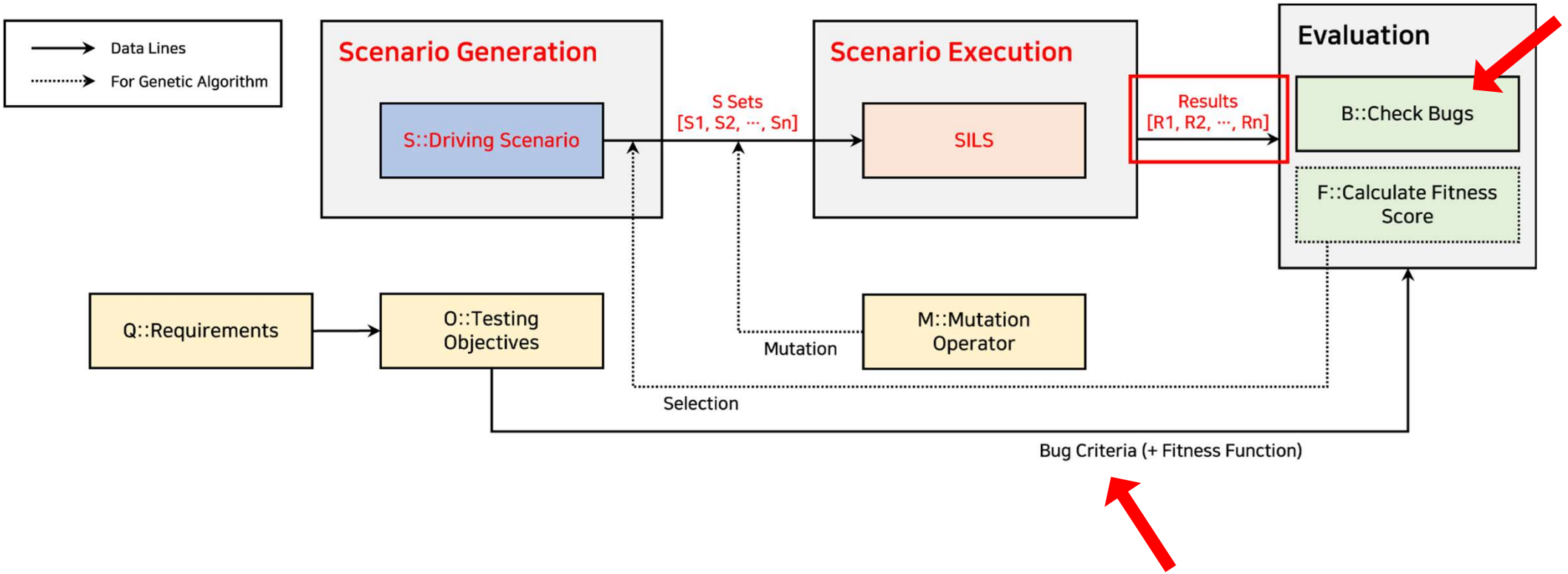


Fuzzing Autonomous Driving System

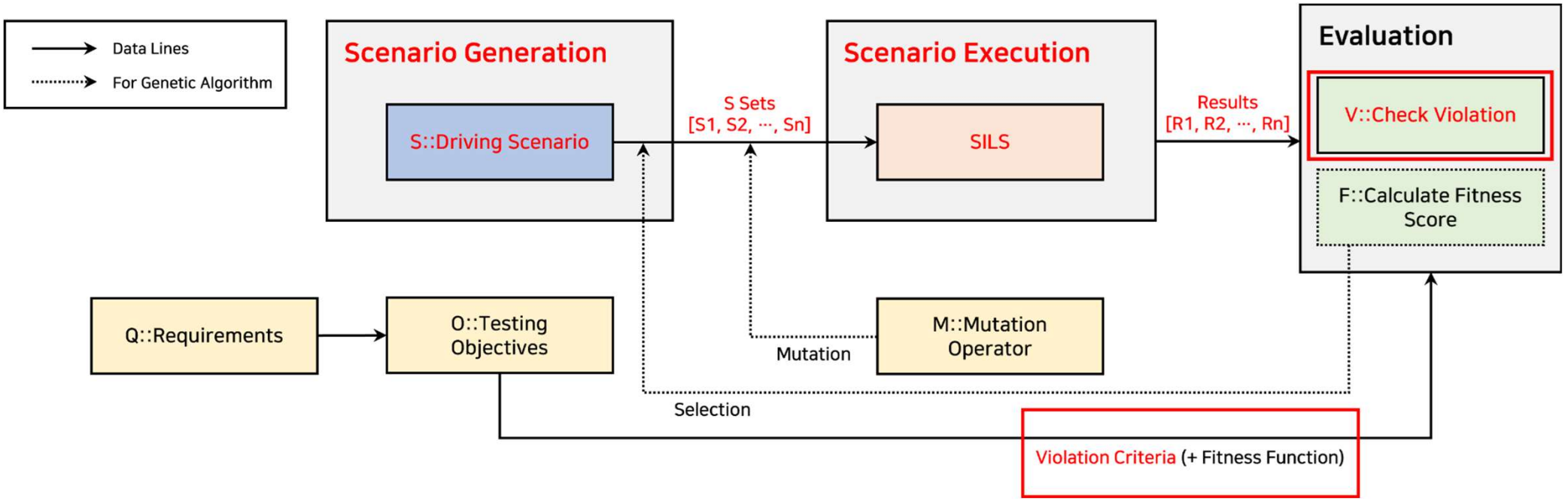


Coverage >> Simulation Results

Fuzzing Autonomous Driving System



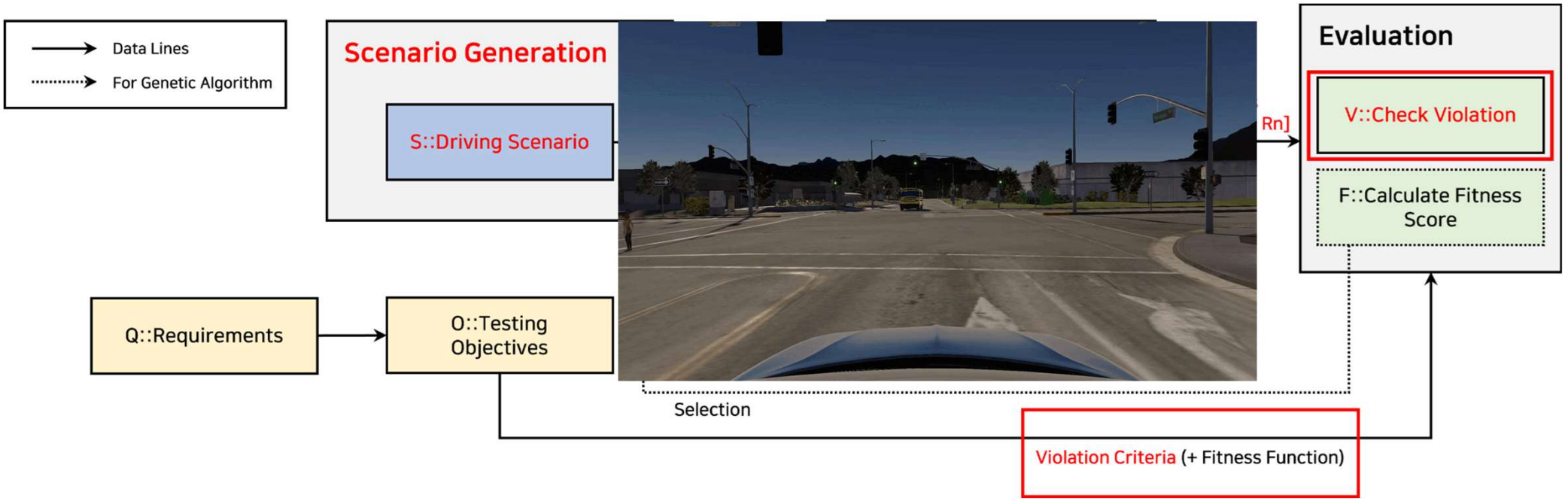
Fuzzing Autonomous Driving System



Bug Criteria >> Violation Criteria

(Targeted Bug) >> (Targeted Safety Violation - e.g. collision, red light violation)

Fuzzing Autonomous Driving System



Bug Criteria >> Violation Criteria

(Targeted Bug) >> (Targeted Safety Violation - e.g. collision, red light violation)

Existing Research on Fuzzing AD Systems

<<Input Generation>>

- **Mutation-based Fuzzing**
  - Mutate **pre-defined** scenarios : NSGA2-SM[ASE'16], AV-Fuzzer[ISSRE'20], EMOOD[ASE'21]
- **Generation-based Random Fuzzing**
  - Random under **pre-defined** constraints : DriveFuzz [CCS'22] ,ScenoRITA[arXiv'22], DoppelTest[ICSE'23]

Existing Research on Fuzzing AD Systems

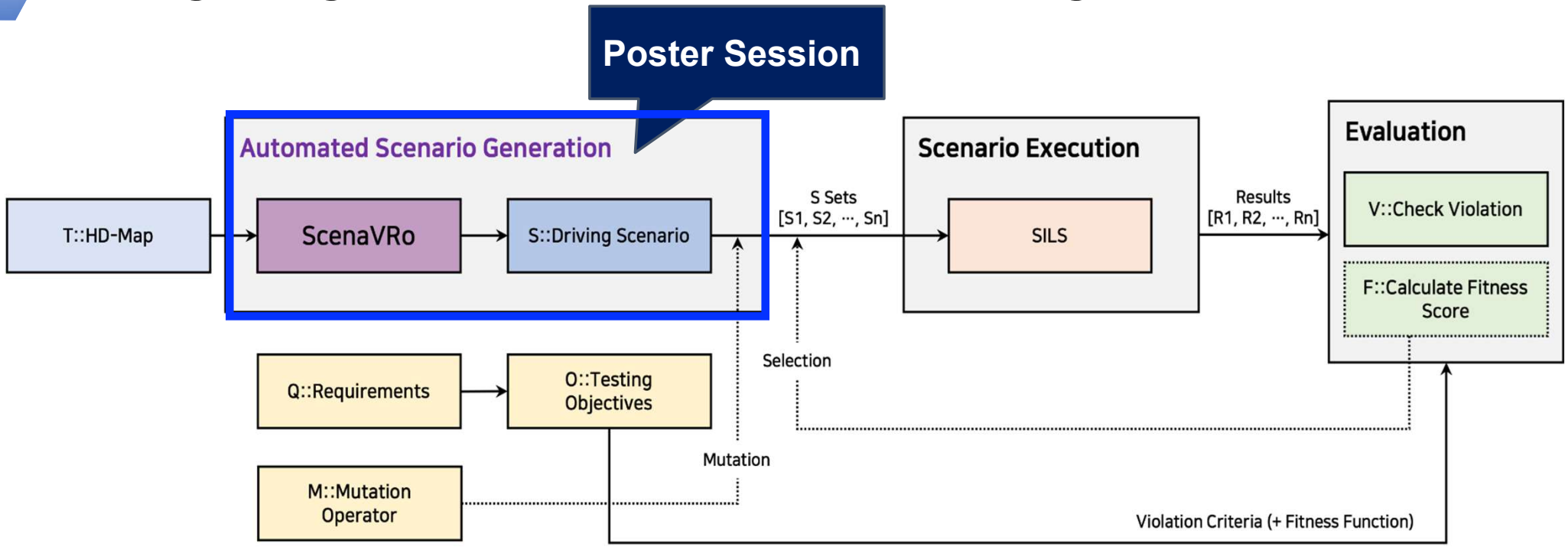
<<Oracles>>

- **Collision :** Carlo et al.[ICST'20]
- **Immobility** : PlanFuzz[NDSS'22]
- **Trajectory coverage :** Tang et al.[ICRA'21], ASF[arXiv'21]
- **Specific traffic laws :** LawBreaker[ASE'22]
- **Multi-objectives**

  Speed::S, Invaliding lanes::L, Running red light::R, Collision::C, Safe Distance::D, Over-acceleration::O, Running on stop sign::N, Immobility::I

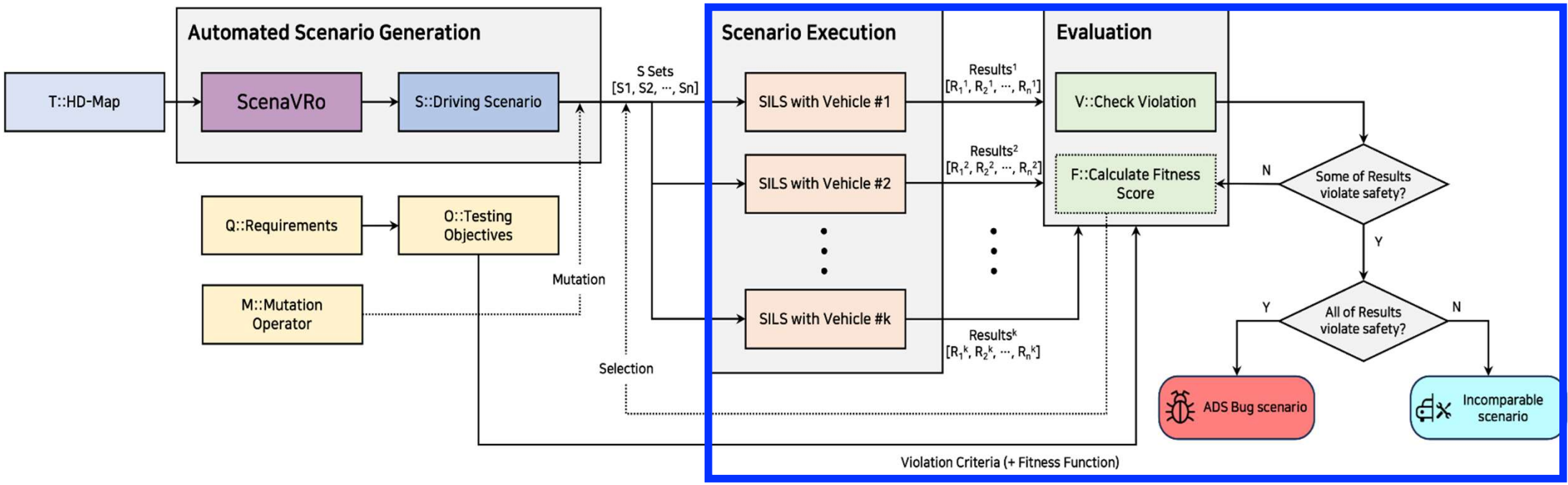
  - FITEST[ASE'18] : SRCD, EMOOD[ASE'21] : SLRDO, DriveFuzz[CCS'22] : SIRC, MOSAT[FSE'22] : SLRC

Ongoing: contribution on input generation



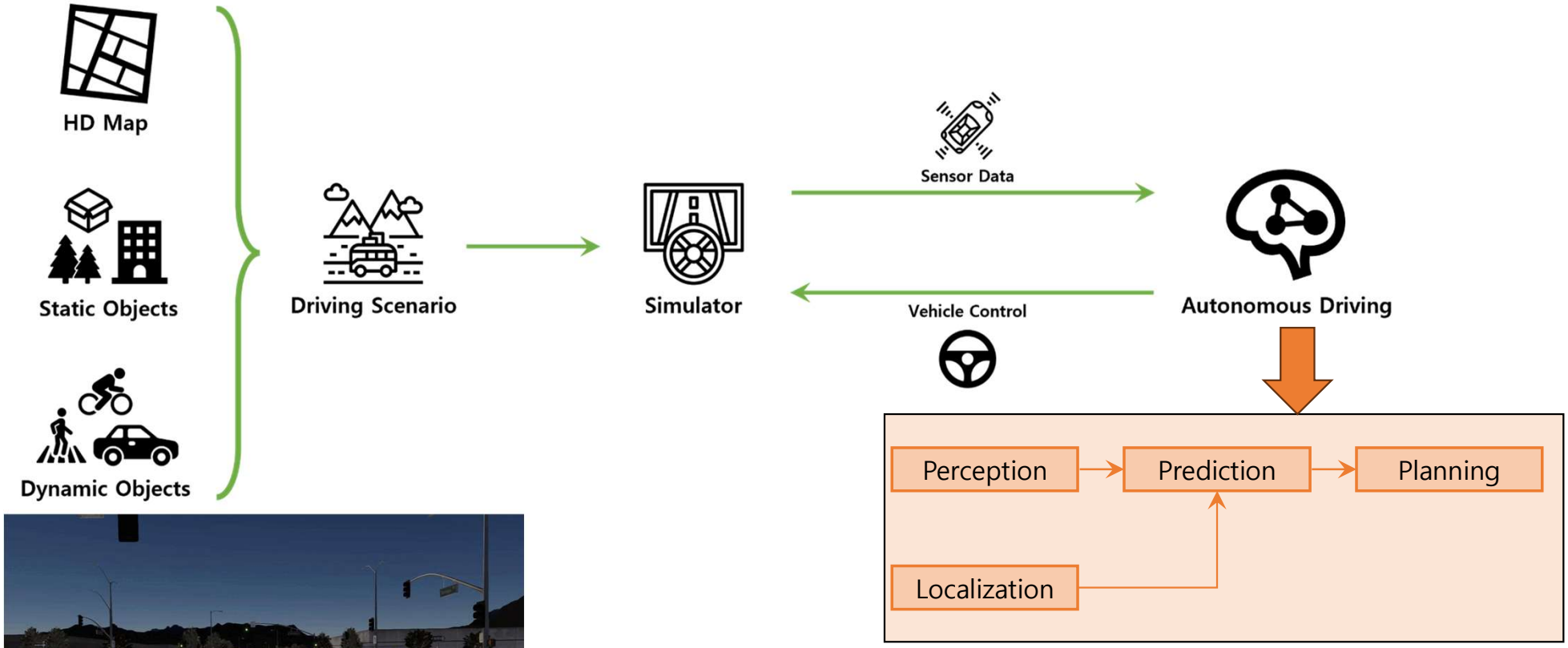
- **Generate Smart Route.**
- **Additionally, add or change other components that ensure validity.**
- **We can detect bugs of global planner using smart route.**

Ongoing: contribution on oracles



- Adopt differential fuzzing by generating **equivalent mutants**
- Various mutation targets available.
  - Static objects
  - Ego vehicle

Yet another limitation: AD system as a whole

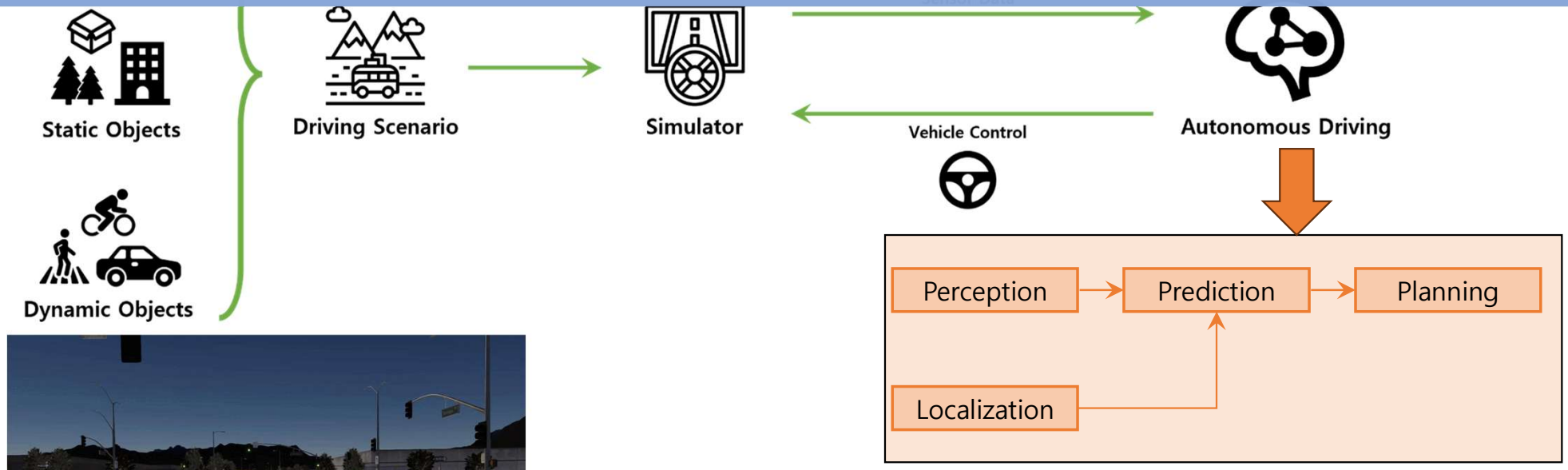


Suppose you report this scenario to developer.

How can this be delivered to the right person?

Yet another limitation: AD system as a whole

Locating the **faulty** component would be needed.



Suppose you report this scenario to developer.

How can this be delivered to the right person?

Another Poster: combining random and search-based

	Random Approach	Search-Based Approach
Method	• Select method sequence and corresponding argument inputs randomly	• Optimize search to the direction that maximize certain metric like coverage
Pros	• Simple and efficient	• Higher coverage
Cons	• Hard to cover specific parts of code	• More expensive • Coverage does not guarantee bug detection

Coverage is **necessary** for bug detection but **not enough.**
More **diverse** inputs are required for detecting missed bugs.

Search-Based + Random
Keep Coverage, Diversify inputs

Conclusion

- Input **validity** matters for fuzzing effectiveness
  - Extracted constraints from document help
  - Generated constraints by LLM help better
- Program synthesis enables **pseudo** dynamic symbolic execution
- Autonomous driving software needs **smart** scenario generation
  - Many opportunities in enhancing **oracles** as well
  - End-to-end testing gives both positive and negative effects