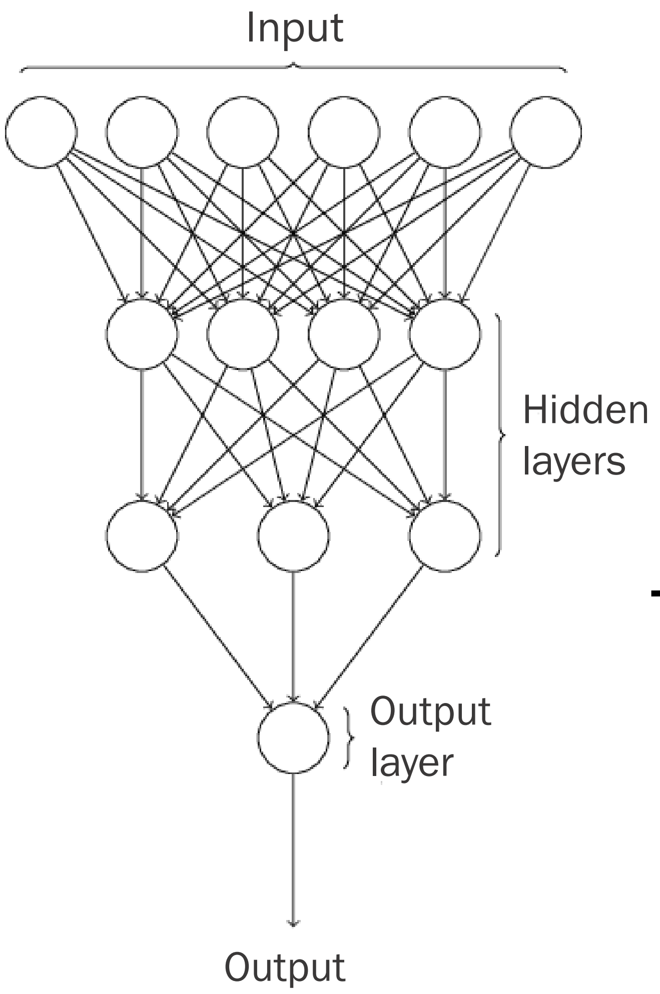
Towards Trustworthy Deep Learning Software System

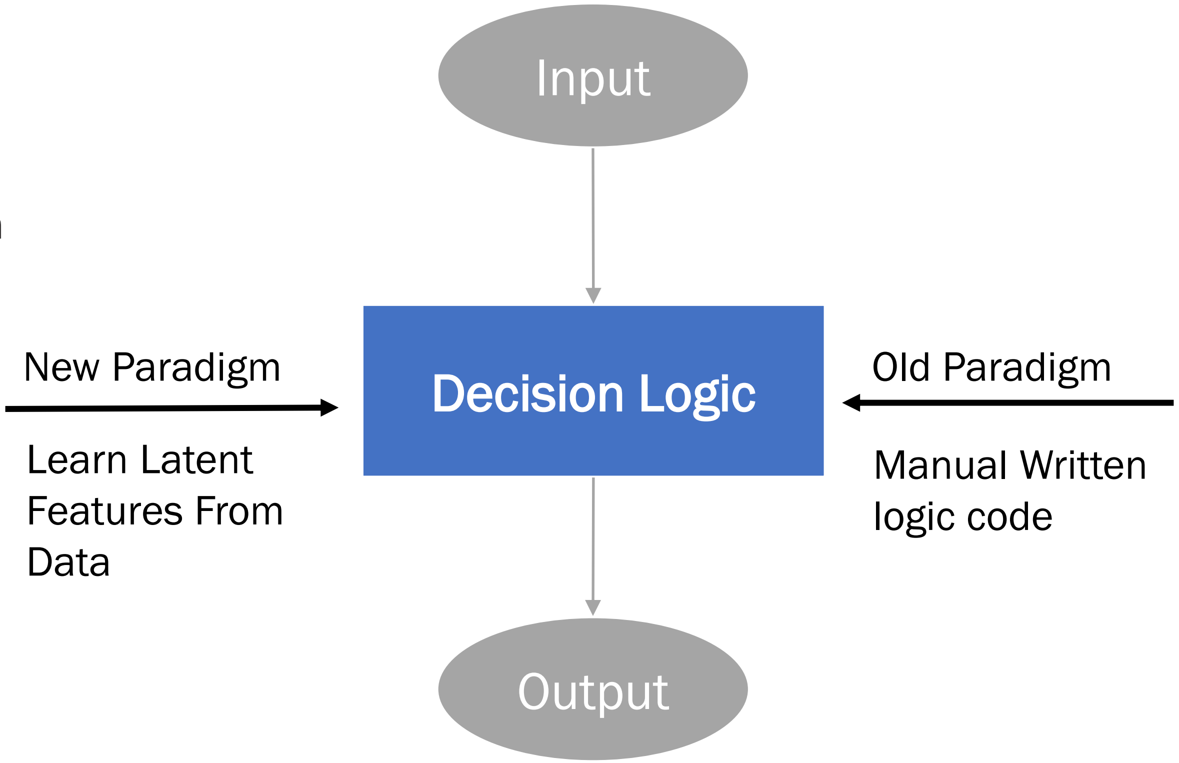
Houssem Ben Braiek, Ph. D.

Deep Learning Software vs Traditional Software

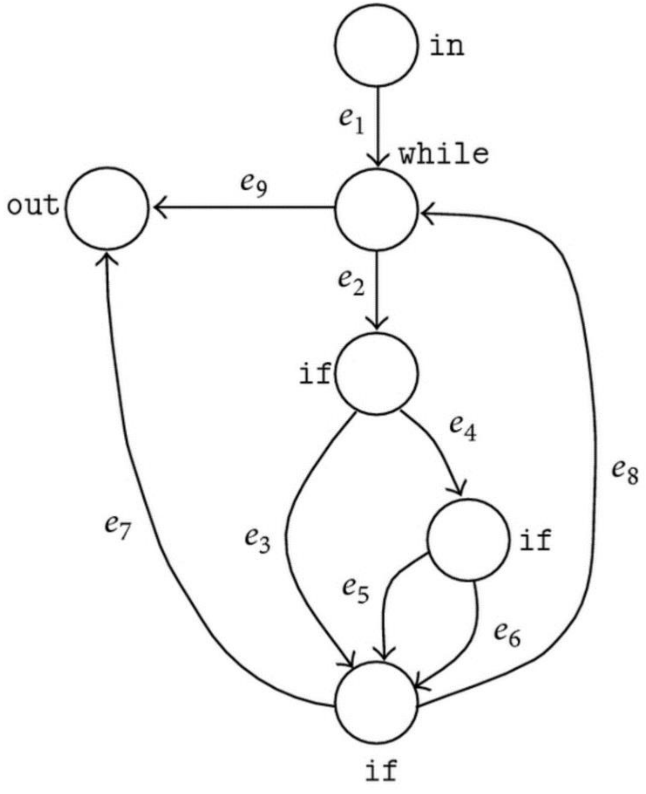
Neural Network Graph



Software Application Flow Graph

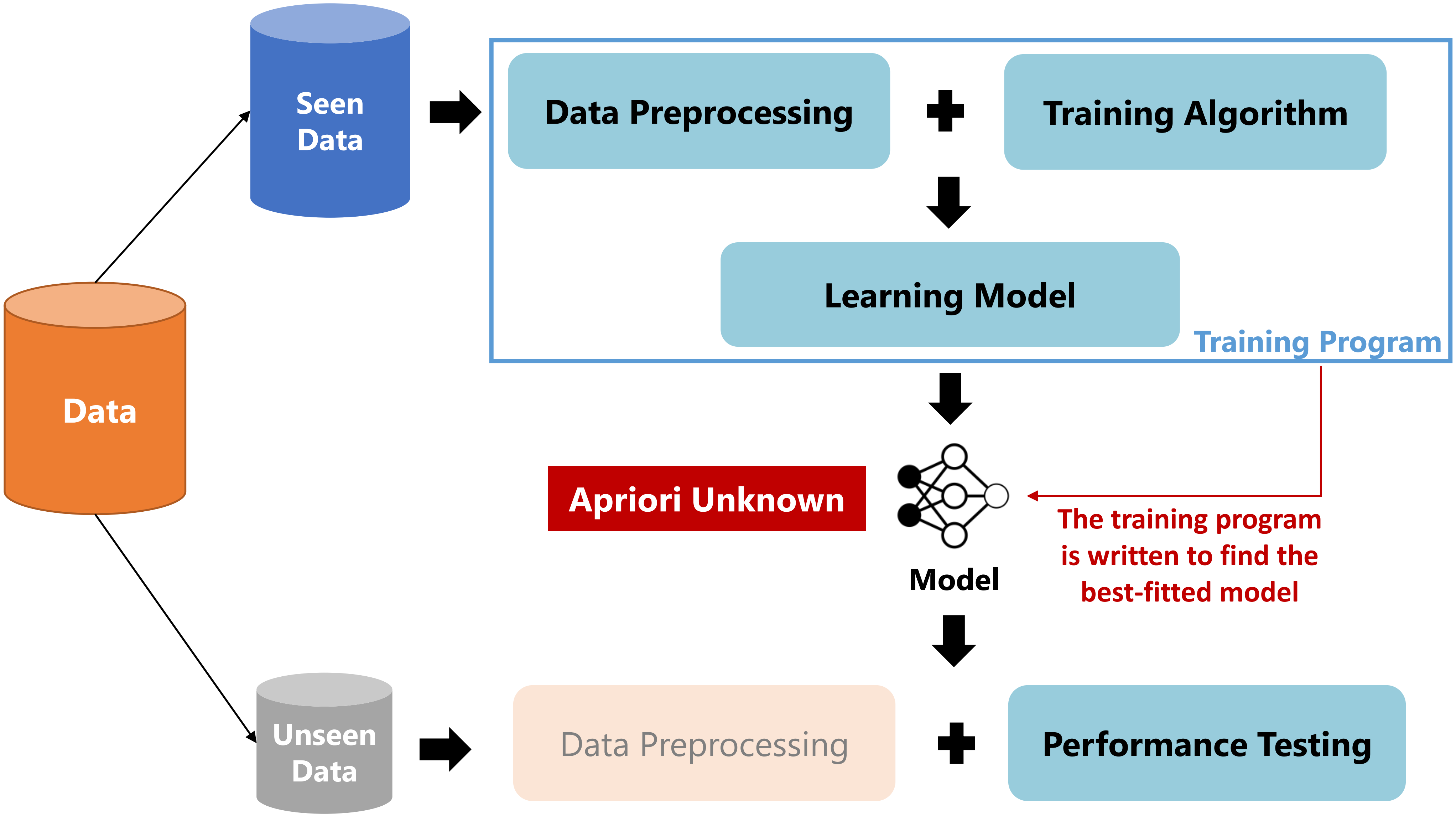


Control Flow Graph of Traditional Program



Challenges of Debugging DL Training Programs

The Training Program is "Untestable" Software



“There would be no need to write such programs, if the correct answer were known”

Davis and Weyuker, 1981

The Oracle Problem in Practice

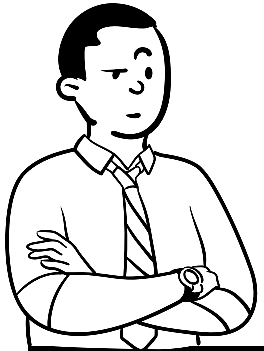
```python
# Load the dataset
(x_train, y_train),(x_test, y_test) = fashion_mnist.load_data()

# Normalize the pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

# Compile the model
model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with a callback
model.fit(x_train, y_train, epochs=10)
```



Can we have the expected values from an oracle ?

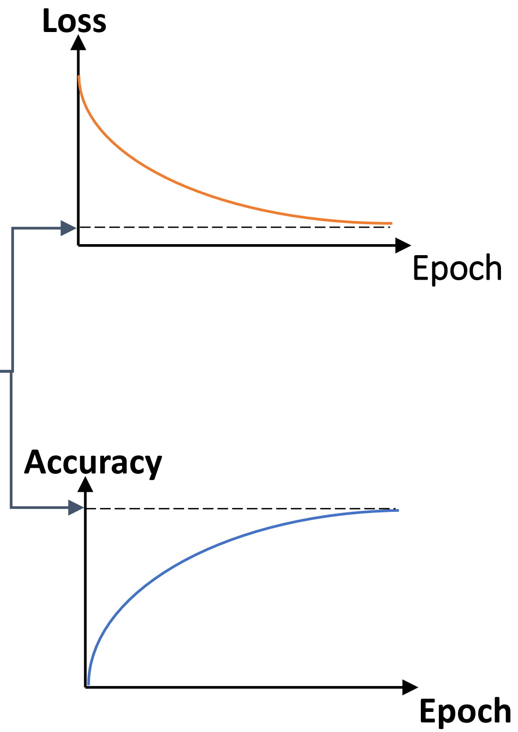
```python
def test_params_fashion_mnist_dense_512_relu_10_softmax(model,
                                                         oracle_params,
                                                         tolerance=0.01):

    for layer in model.layers[1:]:
        # Extract the learned W&B of each layer
        actual_weight, actual_bias = layer.get_weights()
        # Set your expectations for W&B of each layer
        expected_weight, expected_bias = oracle_params[layer.name]
        # Check if the weights are nearly equal
        tf.debugging.assert_near(actual_weight, expected_weight,
                                 atol=tolerance)
        # Check if the biases are nearly equal
        tf.debugging.assert_near(actual_bias, expected_bias,
                                 atol=tolerance)
```

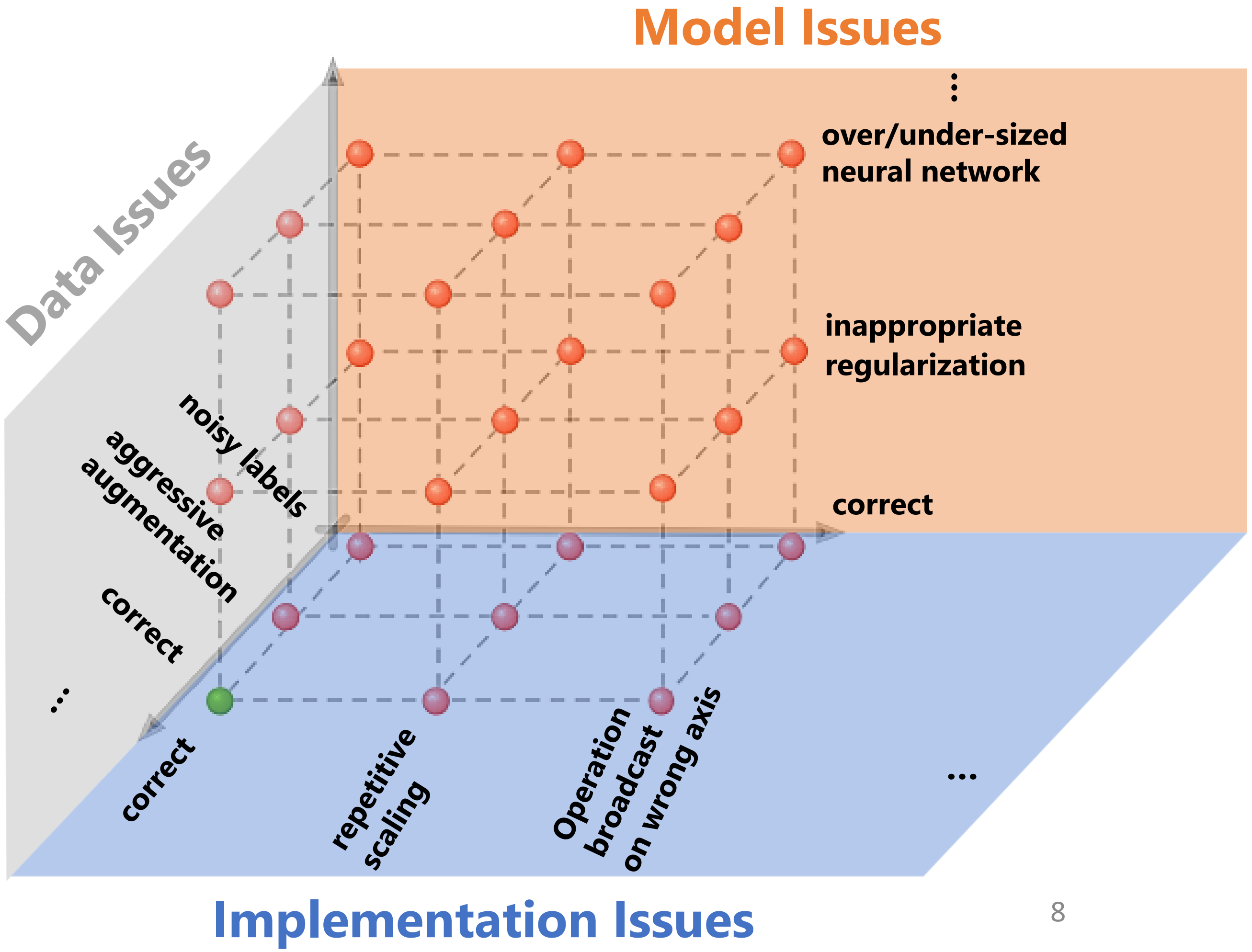
State-of-practice for Training Program Debugging

Watch the training performance, i.e., loss and accuracy evolution curves.

For **novel problems** or **data**, it is **challenging** to set **expectations** too…



If the training behaves poorly on the data ?



Property-based DL Training Program Debugging

Property-based Testing for Training Programs

Property-based testing for a function

for all (x, f(x), ...)
such as precondition(x, f(x), ...) holds
property(x, f(x), ...) is true

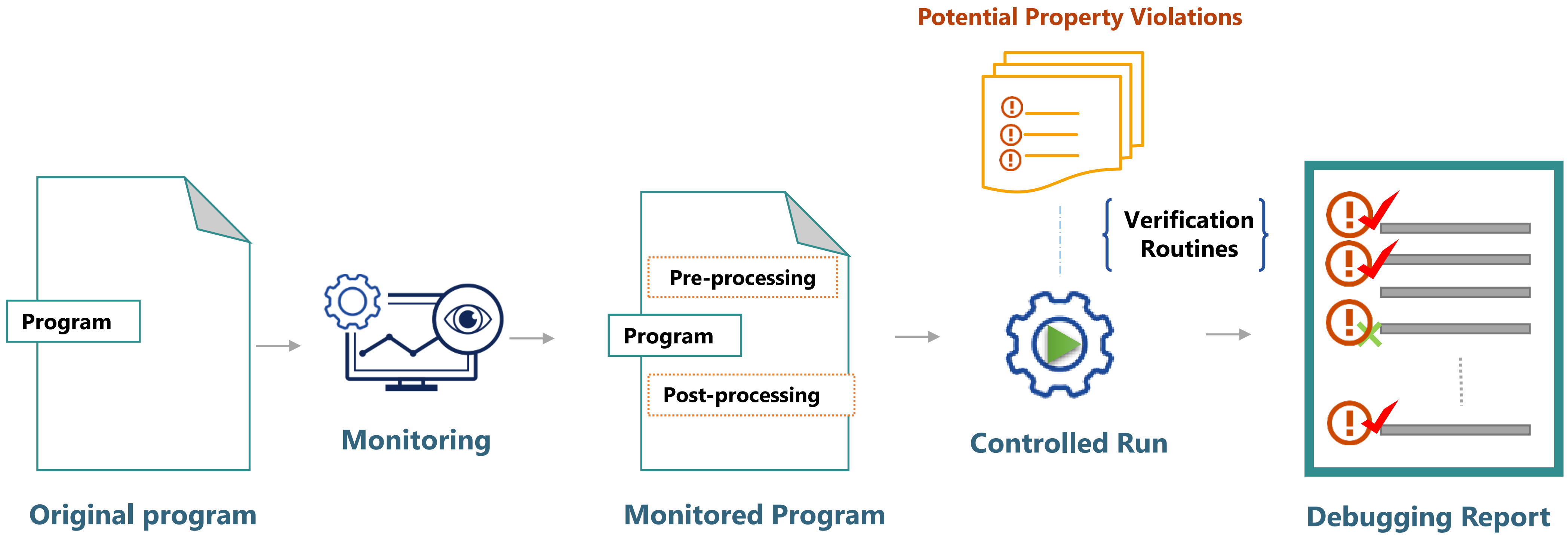
Its application for model testing is straightforward:
- Verify if all model outputs f(x) maintain specific properties such as smoothness, invariance, etc., across valid inputs x.

Property-based testing for components of the training program

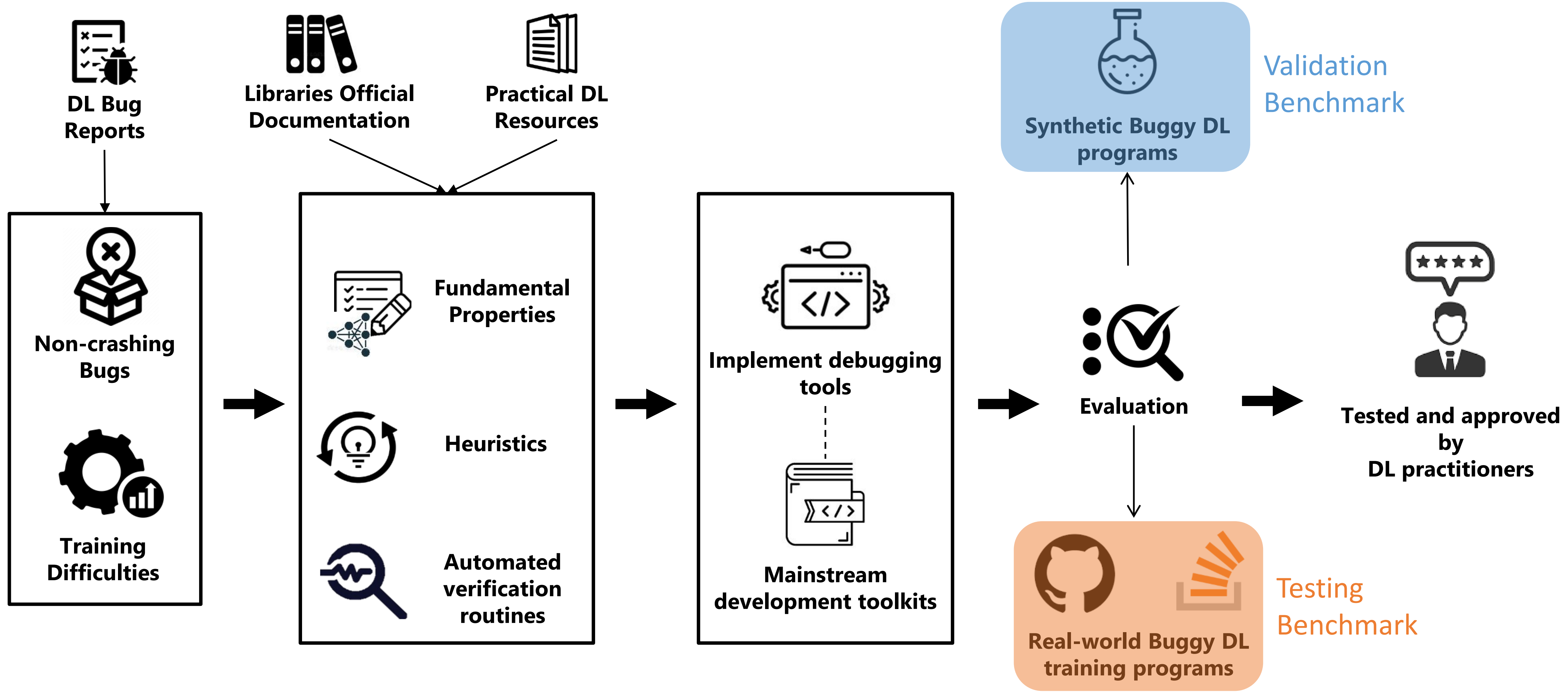
for all ($W^{(i)}, b^{(i)}, A^{(i)}, loss^{(i)}, acc^{(i)}$, ...)
such as precondition ($X^{(i)}, y^{(i)}, H^{(i)}$, ...) holds
property($W^{(i)}, b^{(i)}, A^{(i)}, loss^{(i)}, acc^{(i)}$, ...) is true

Its application to training program requires adaptation:
- Check if all the intermediate training program states hold some properties for all the valid input data and settings.

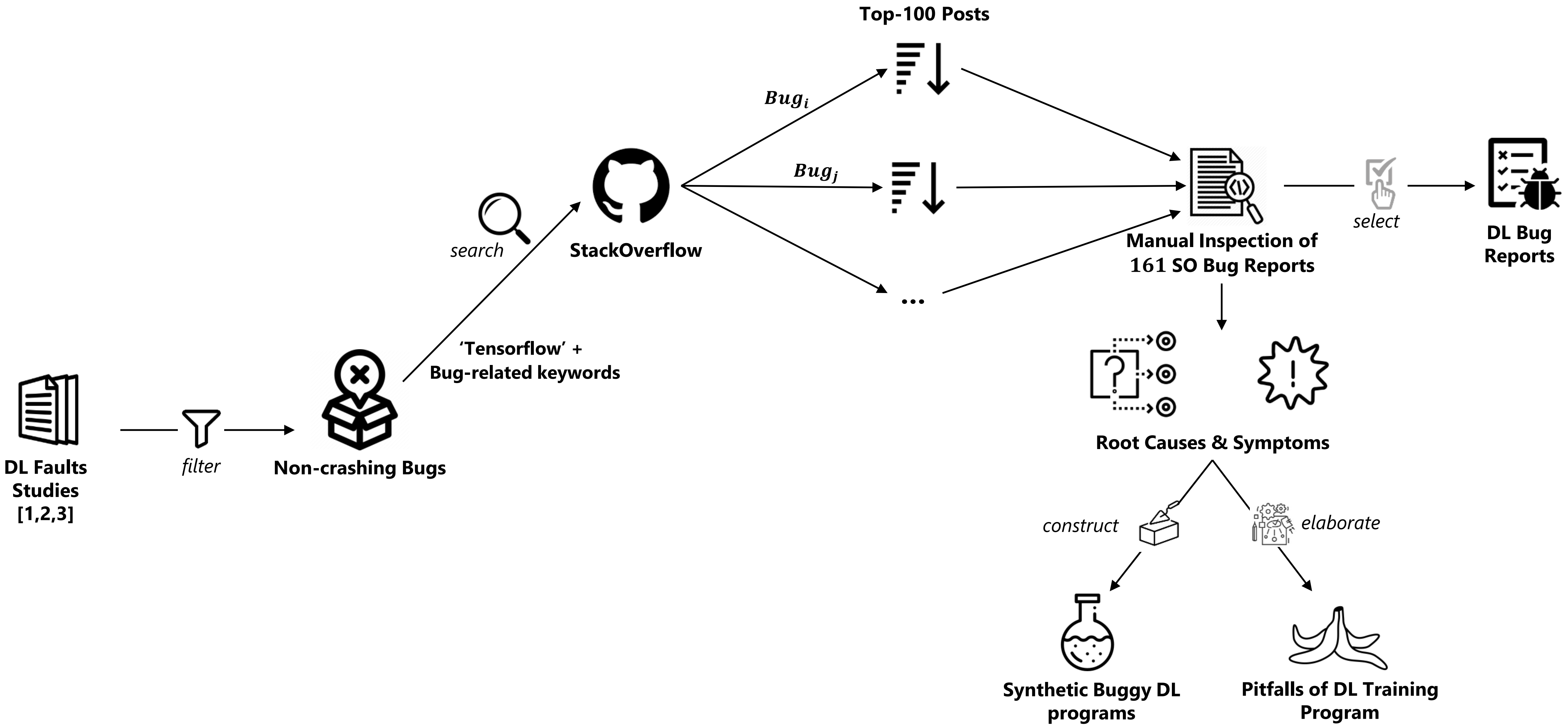
Property-based Testing for Training Programs



Methodology

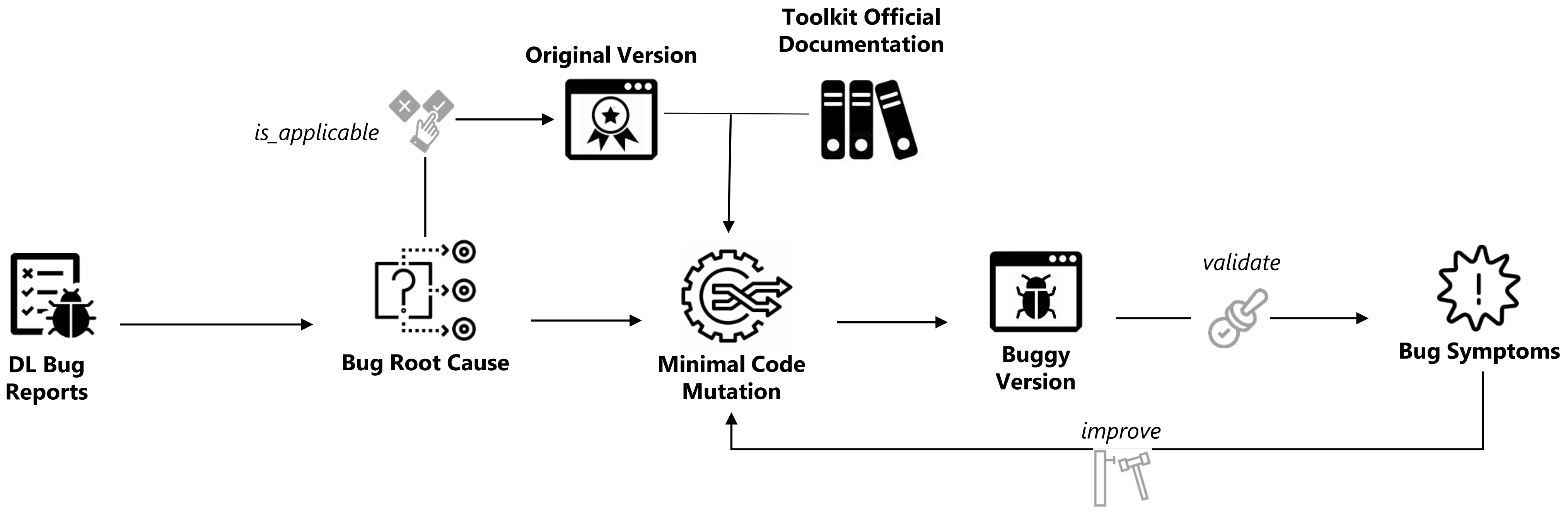


Examine Empirical Studies & StackOverflow



[1] Zhang el al., An Empirical Study on TensorFlow Program Bugs (2018)
[2] Islam et al., A comprehensive study on deep learning bug characteristics (2019)
[3] Humbatova et al., Taxonomy of real faults in deep learning systems (2020)

Build Synthetic Buggy DL training programs



The Catalog of Pitfalls in DL Training Program

# N	Principal Program Component Involved
4	Input Data-related Pitfalls
2	Connectivity & Custom Ops Pitfalls
2	Parameters-related Pitfalls
5	Optimization-related Pitfalls
2	Regularization-related Pitfalls
8	Activations-related Pitfalls

Examples

```
# Load the dataset
# x_train and x_test can be already normalized
(x_train, y_train), \
(x_test, y_test) = fashion_mnist.load_data()

# Normalize the pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the model
model = tf.keras.models.Sequential([
  # Some models start with rescaling the input
  tf.keras.layers.Rescaling(scale=1./255),
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

➔ The abuse of data re-scaling, often unintentionally, can result in an ill-conditioned loss minimization problem and, at best, a slow learning rate.

The Catalog of Pitfalls in DL Training Program

# N	Principal Program Component Involved
4	Input Data-related Pitfalls
2	Connectivity & Custom Ops Pitfalls
2	Parameters-related Pitfalls
5	Optimization-related Pitfalls
2	Regularization-related Pitfalls
8	Activations-related Pitfalls

Examples

```
class Custom_CE_Loss(tf.keras.losses.Loss):

    def call(self, y_true, y_pred, inv_rates):
        weights = tf.reduce_sum(inv_rates * y_true, axis=0)
        log_y_pred = tf.math.log(y_pred)
        elements = -tf.math.multiply_no_nan(x=log_y_pred,
                                            y=y_true)
        return tf.reduce_mean(
                    weights * tf.reduce_sum(elements, axis=0)
                )

model.compile(optimizer='adam',
              loss=Custom_CE_Loss(),
              metrics=['accuracy'])

# Train the model with a callback
model.fit(x_train, y_train, epochs=10)
```

➔ A custom operation may be buggy because the reduction is broadcast over the wrong axis. The result will be faulty due to this.

The Catalog of Pitfalls in DL Training Program

# N	Principal Program Component Involved
4	Input Data-related Pitfalls
2	Connectivity & Custom Ops Pitfalls
2	Parameters-related Pitfalls
5	Optimization-related Pitfalls
2	Regularization-related Pitfalls
8	Activations-related Pitfalls

Examples

```
# Define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu,
        kernel_initializer=RandomNormal(stddev=0.01)),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax,
        kernel_initializer=RandomNormal(stddev=0.01))
])
```

➔ The choice of initialization has a significant impact on the quality of training. Variance of random weights should be considered in relation to the size of the layer's input. This consideration reduces the risk of vanishing or exploding gradients during training.

The Catalog of Pitfalls in DL Training Program

# N	Principal Program Component Involved
4	Input Data-related Pitfalls
2	Connectivity & Custom Ops Pitfalls
2	Parameters-related Pitfalls
5	Optimization-related Pitfalls
2	Regularization-related Pitfalls
8	Activations-related Pitfalls

Examples

```
# Build the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.LSTM(64),
  tf.keras.layers.Dense(1),
])
# Set the training parameters
model.compile(loss=tf.keras.losses.Huber(),
              optimizer='sgd',
              metrics=["mae"])
```

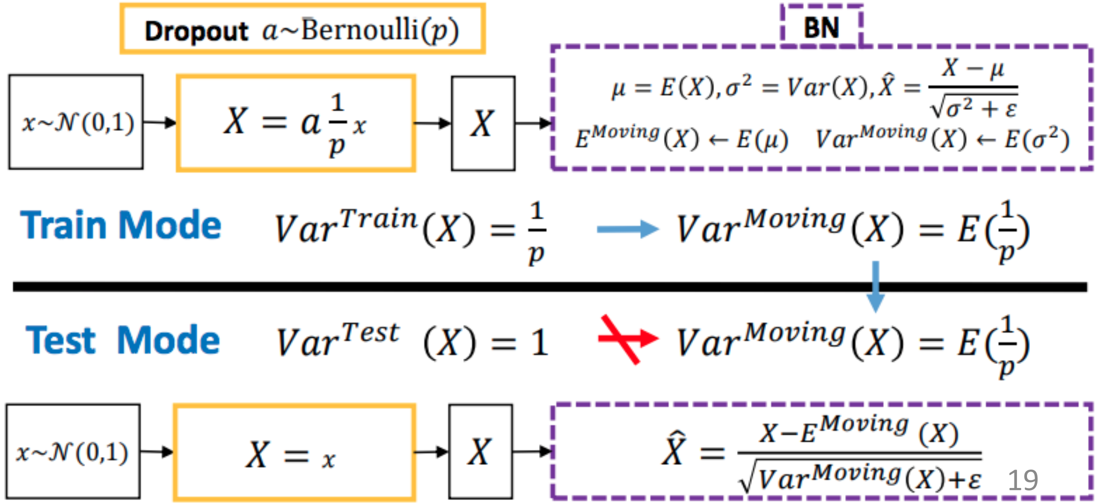
➔ A poor choice of loss function, such as Huber loss for demand forecasting (spikes are crucial), prevents the identification of useful patterns.

The Catalog of Pitfalls in DL Training Program

# N	Principal Program Component Involved
4	Input Data-related Pitfalls
2	Connectivity & Custom Ops Pitfalls
2	Parameters-related Pitfalls
5	Optimization-related Pitfalls
2	Regularization-related Pitfalls
8	Activations-related Pitfalls

Examples

```
# Define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  # Induce the discrepancy between train and test mode
  tf.keras.layers.Dropout(0.5)
  # Moving statistics will be shifted at test mode
  tf.keras.layers.BatchNormalization()
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```



The Catalog of Pitfalls in DL Training Program

# N	Principal Program Component Involved
4	Input Data-related Pitfalls
2	Connectivity & Custom Ops Pitfalls
2	Parameters-related Pitfalls
5	Optimization-related Pitfalls
2	Regularization-related Pitfalls
8	Activations-related Pitfalls

Examples

```
# Define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

# Compile the model
model.compile(optimizer=tf.optimizers.Adam(),
              loss=CategoricalCrossentropy(True),
              metrics=['accuracy'])
```

➔ The instantiated loss function expects logits, but the softmax is applied on the last layer. Thus, the logits are subject to redundant softmax applications, resulting in pathological learning.

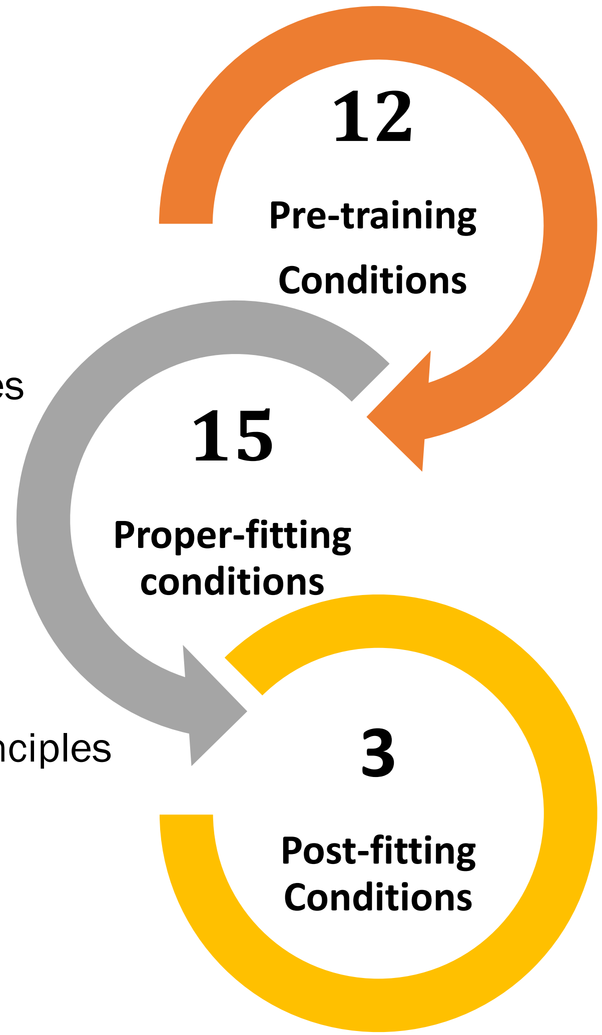
Property-based Verification Routines & Phases

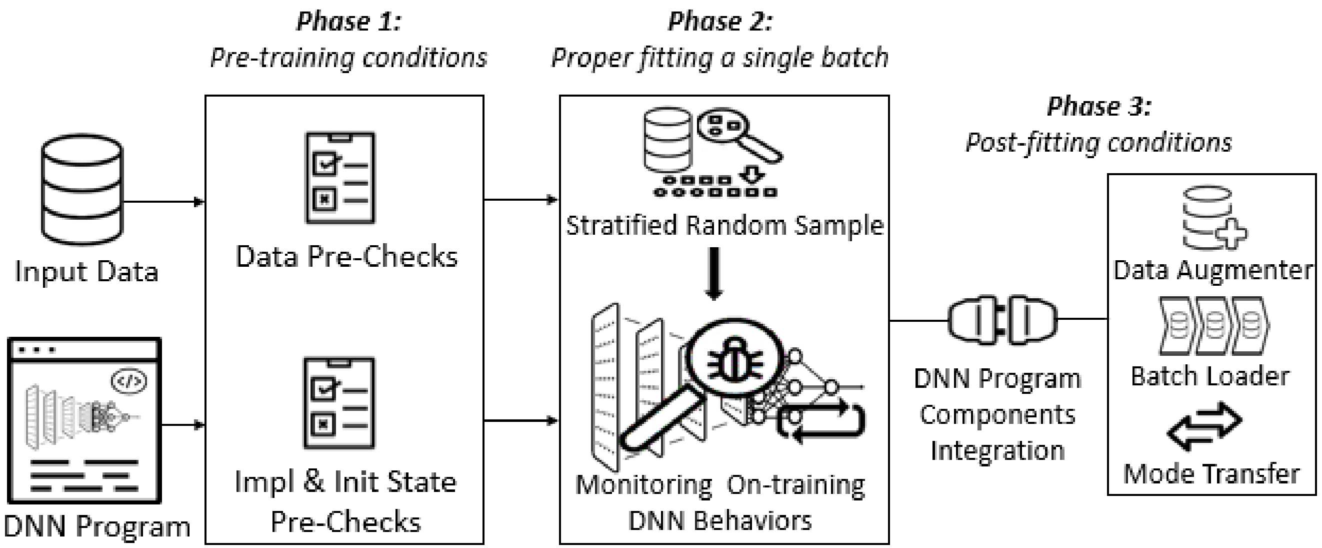
Origins

- Applied DL textbooks
- Academic DL lectures
- Industry practical courses
- DL experts' blogs
- DL practitioners' forums

Types

- Fundamental Design Principles
- Inefficient Training Traits



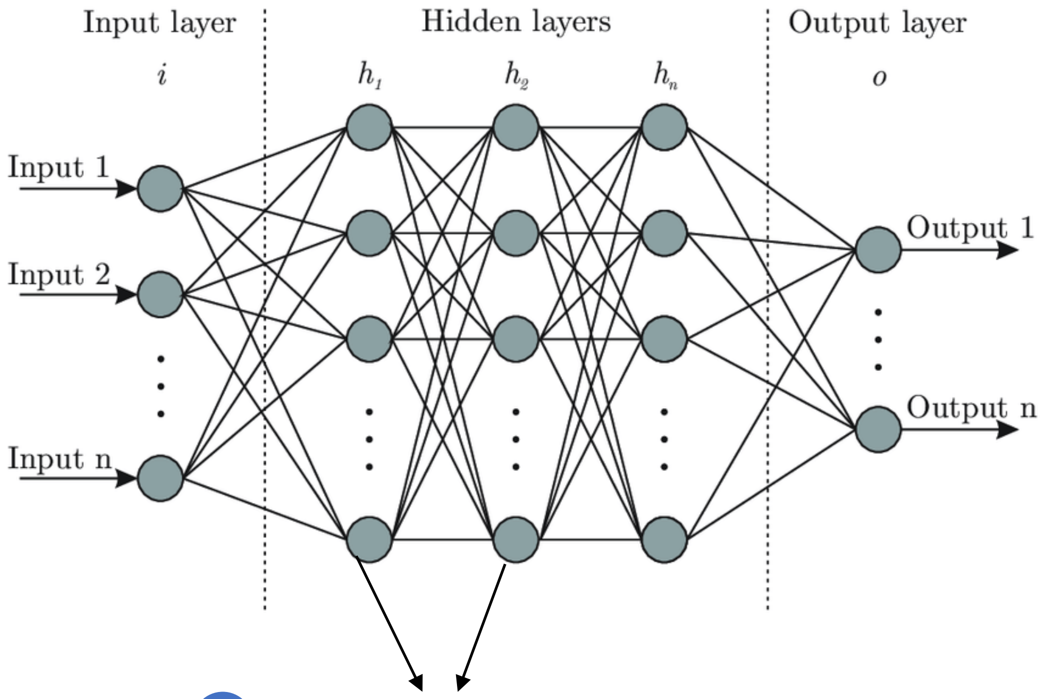


Pre-training Verification Examples

Initialized & first pass

A

Validate the scale of inputs (Normalized)



Loss Estimation
+
Accuracy Evaluation

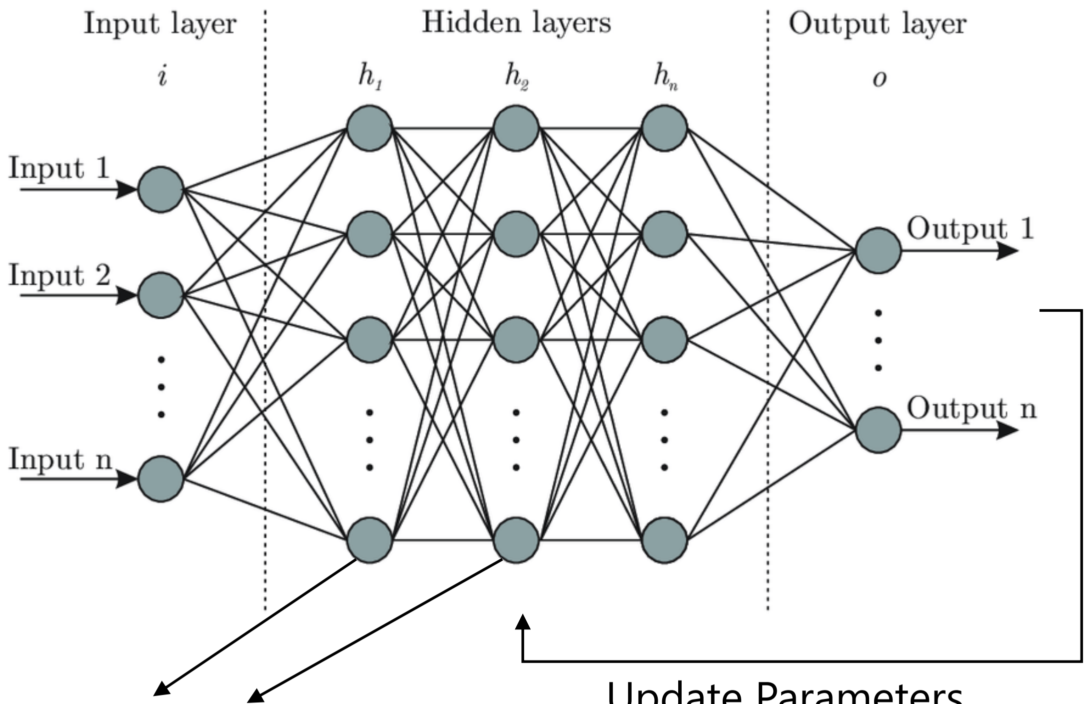
B **random weights (appropriate variance magnitude...)**

C

Initial Loss (Random Guess)
+
Data Dependency: Loss w.r.t Output 1 should depend only on Input 1. (The only non-zero derivative)

Proper-fitting Verification Examples

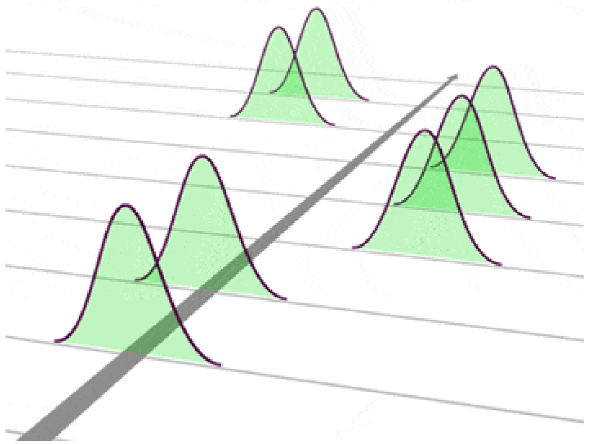
On-Fitting



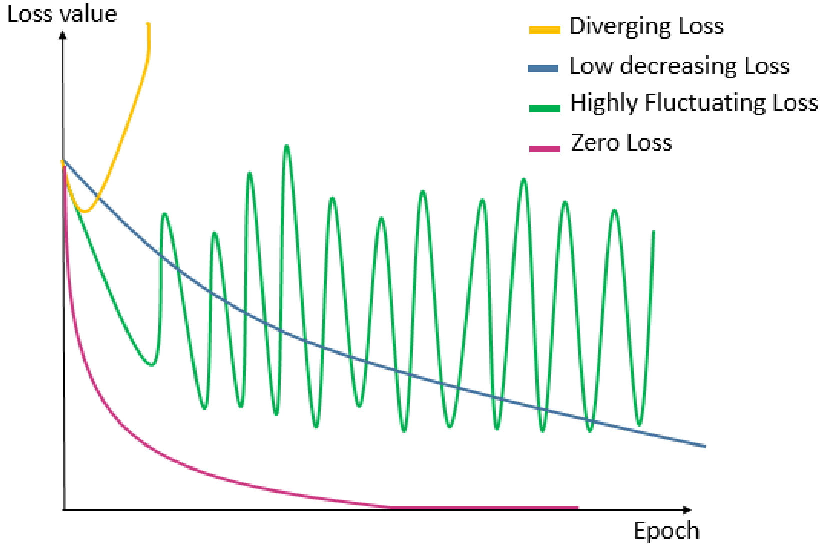
Loss Estimation + Accuracy Evaluation

Update Parameters

B Unstable Activation Distributions



C Abnormal Loss Curves



A Unstable Params Update

Given a layer i

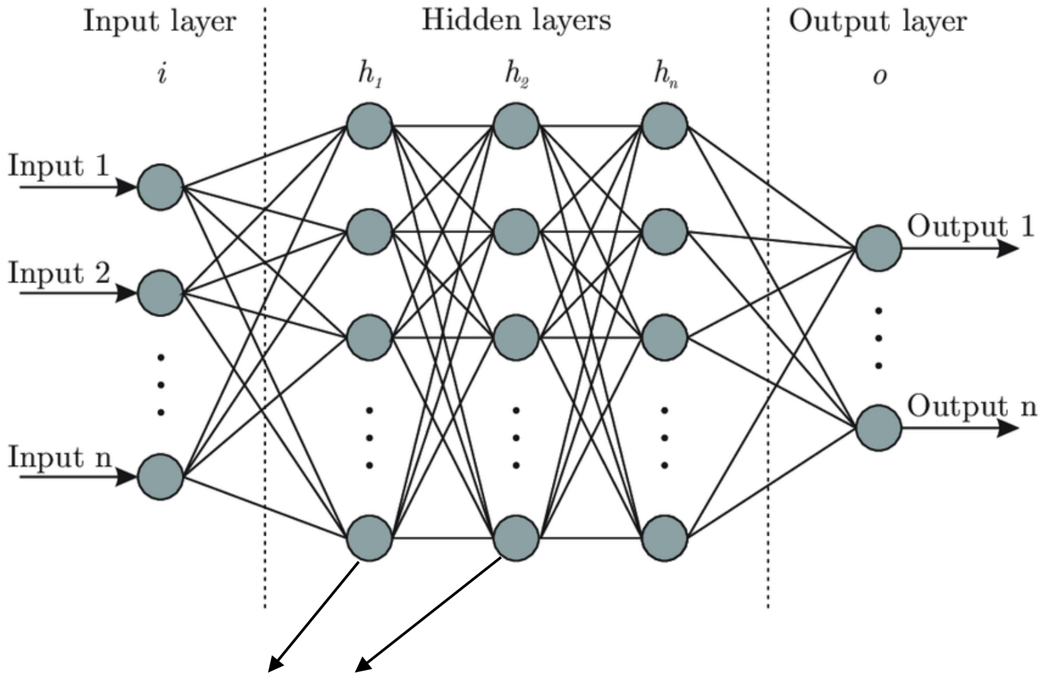
$$-4 < \log_{10}\left(\frac{mean(abs(updates_i))}{mean(abs(parameter_i))}\right) < -1$$

D

Uncorrelated Metrics (Loss & Accuracy)

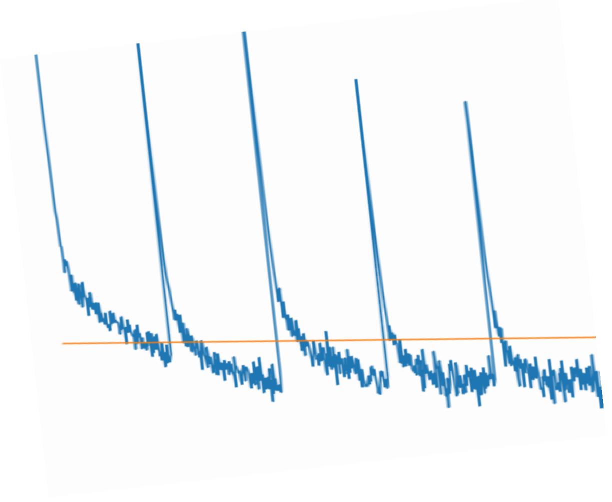
Post-fitting Verification Examples

Post-Fitting



Accuracy Evaluation On Validation Data

A Post-shuffle Loss Spikes



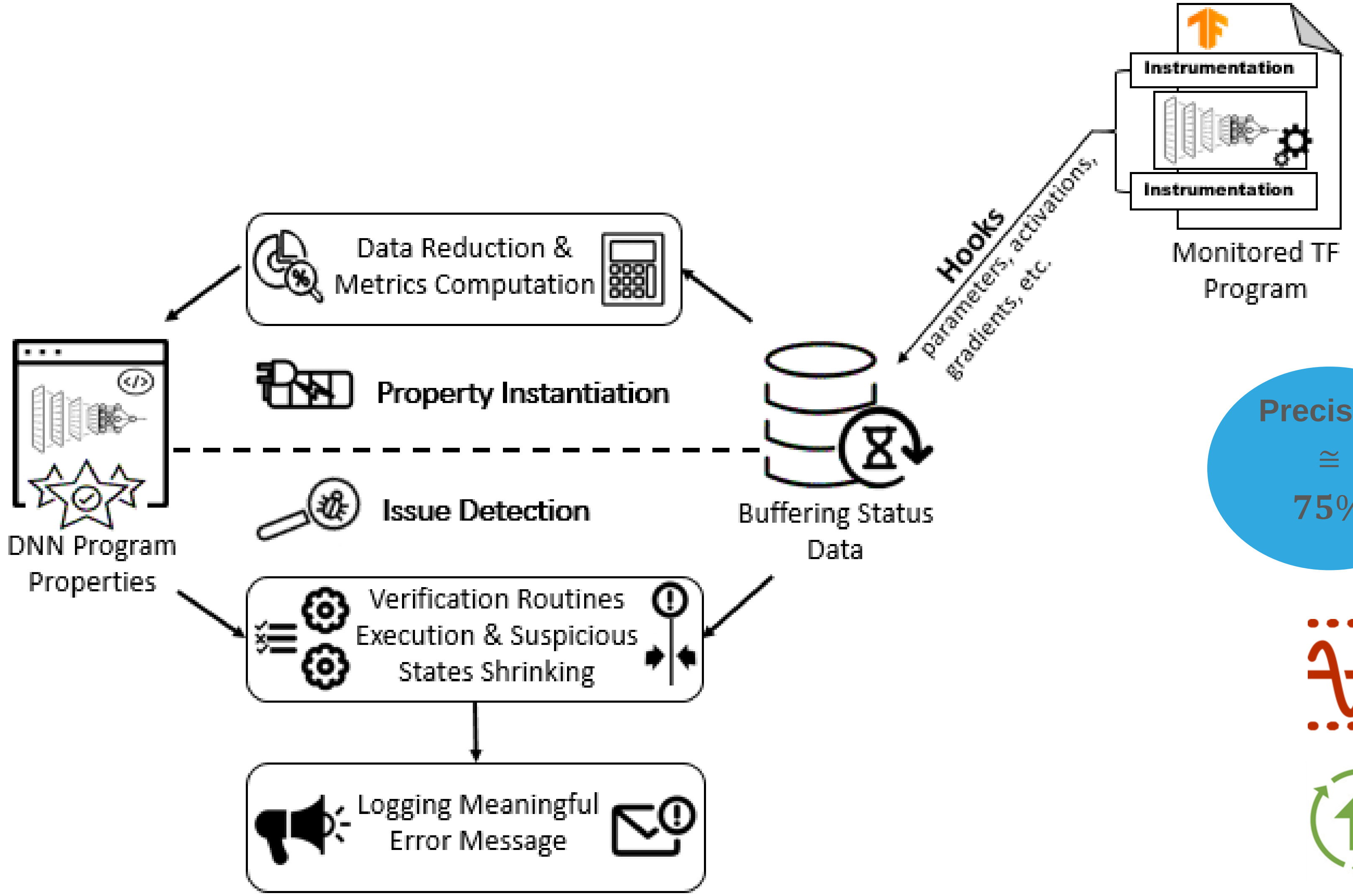
B Perf. Degradation With Augmentation

$$accuracy_{decay} < max$$

$$accuracy_{decay} = \frac{valid\_accuracy\_with\_augmentation}{valid\_accuracy\_without\_augmentation}$$

C **Dissimilarity of activation patterns between test and train modes** using Centered Kernel Alignment (CKA) for representational similarity measure.

Implementation & Performance Evaluation



Amazon SageMaker

Precision ≅ 60%

Vs

Precision ≅ 75%

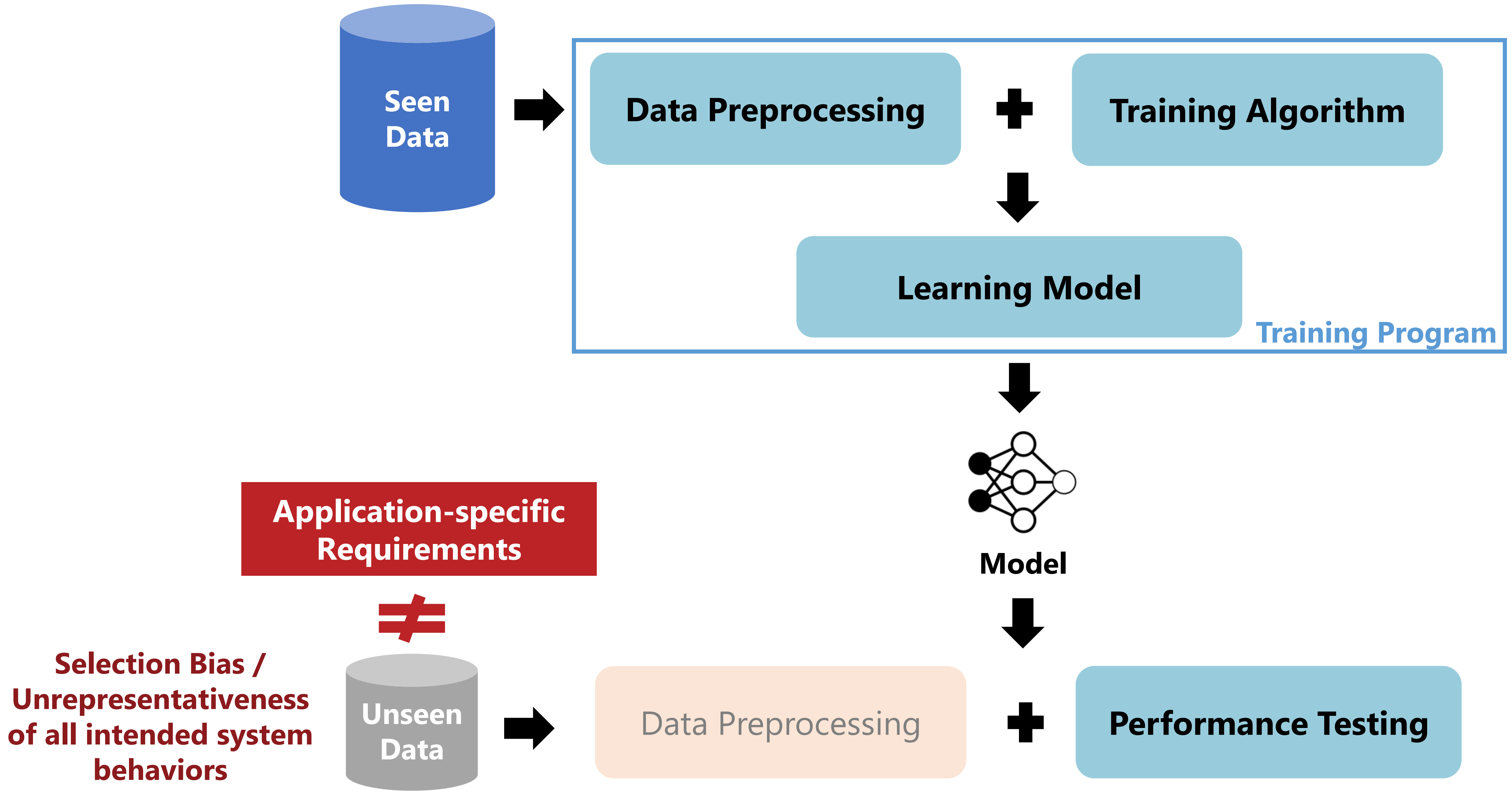
20 Real-world Buggy DL training programs

The choice of Thresholds in Dynamic Property Checking.

These choices are mitigated by considering the persistence of issues over iterations.

Limitations of DL Model Testing

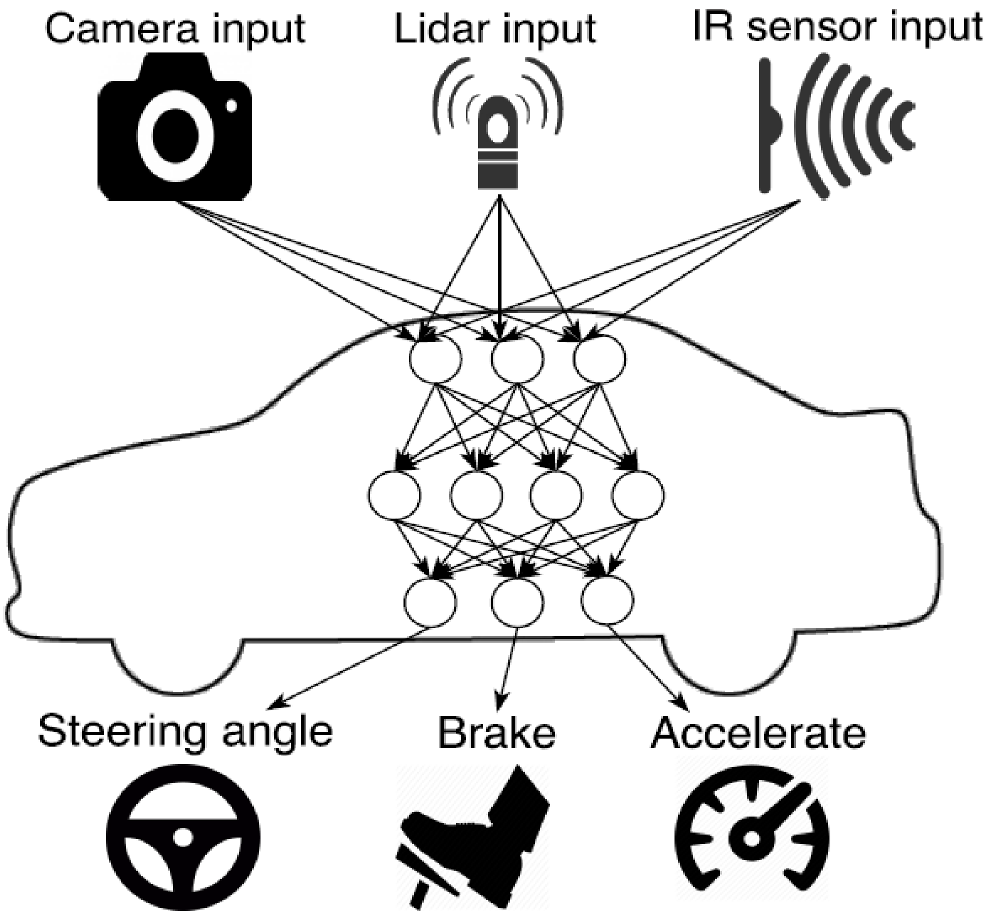
Underspecification Issues of Unseen Datasets

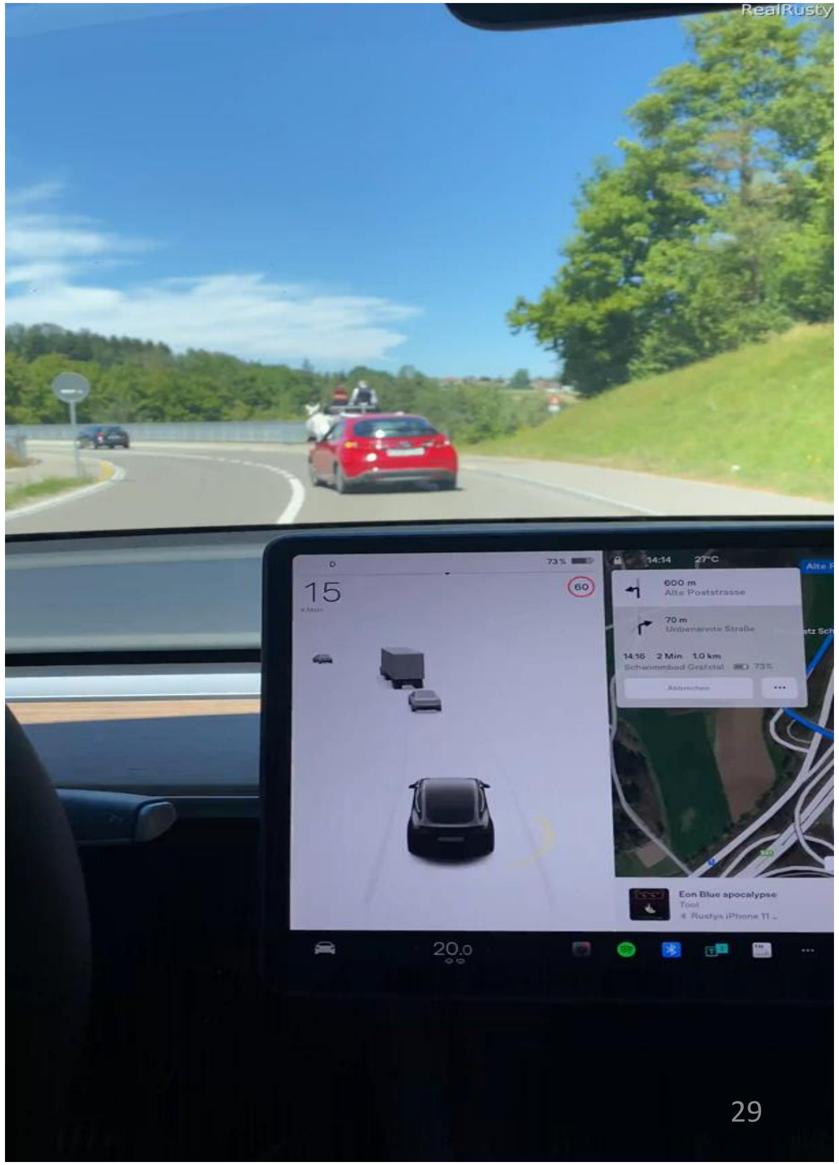


Why do DL practitioners perceive the value of DL testing differently?

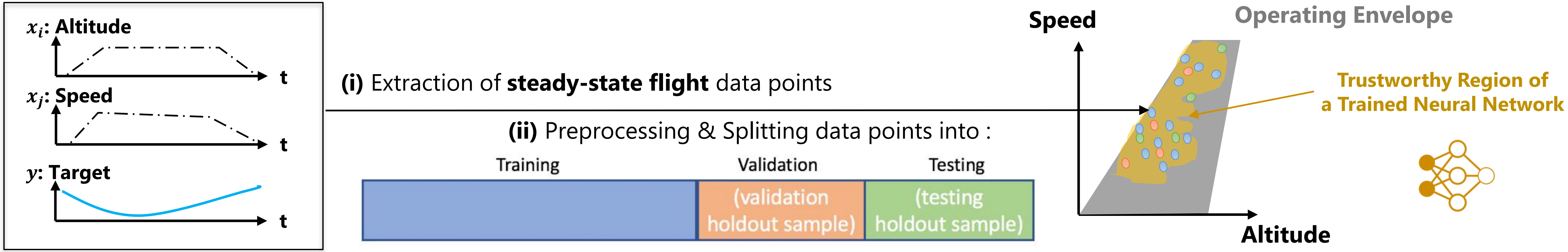
	Low Risk	High Risk
Quantifiable Performance	Outperform the **state-of-the-art** on **testing benchmarks**, e.g., ImageNet, Coco, etc.	Maintain an **acceptable performance** for a **critical function** under **carefully controlled conditions**, e.g., a custom-made cobot that performs repetitive tasks in a manufacturing facility.
Non-Quantifiable Performance	Provide added **value** over **legacy baselines** or **fill a gap**, e.g., filtering ads, recommending movies, etc.	Guarantee an **acceptable performance** for a **critical** function under **all foreseeable operational conditions**, e.g., a generic-purpose cobot that assists the elderly with household duties.

High Risk, Non-Quantifiable Performance ...





The Case of Aircraft System Performance Models



Timeseries Data Flights

➔ A trained NN could illustrate the system performance over the range of included-or-close operational conditions.

'the equipment, systems, and installations must be designed and installed to ensure they perform their intended functions under all foreseeable operating conditions.' U.S Code of Federal Regulations, parts 23, 25, 27, 29

A trustworthy performance model must be qualified to be representative of system behavior throughout the range of foreseeable operational conditions.

Domain-Aware DL Model Testing

The Need for Domain-Aware DL testing Methods

Tradeoff

Statistical Testing

Estimate the **iid performance** of the model for completely **new inputs**.

$$\boldsymbol{Err} = \sum_{i \in \boldsymbol{D_{test}}} \left(\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}\right)^2$$

Use unseen test data $\boldsymbol{D_{test}}$ as a proxy for future entries ($\boldsymbol{x_{new}}$).

$$\boldsymbol{D_{test}} = \left\{\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)\right\}_{\mathrm{i} \in [1,\mathrm{N}]}$$

Collection of D_{test} is costly in aircraft industry

Model Testing

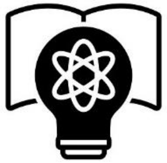
Test the **internal logic/mappings** of the model against the prior knowledge on the nature of the relation between $\boldsymbol{x}$ and $\boldsymbol{y}$.

Oracle Problem $\quad \boldsymbol{f}^*\,? \quad \boldsymbol{w}^*?$

$\boldsymbol{f}, \boldsymbol{w}$

$\boldsymbol{y} \leftarrow \boldsymbol{x}$

Domain-Aware Testing

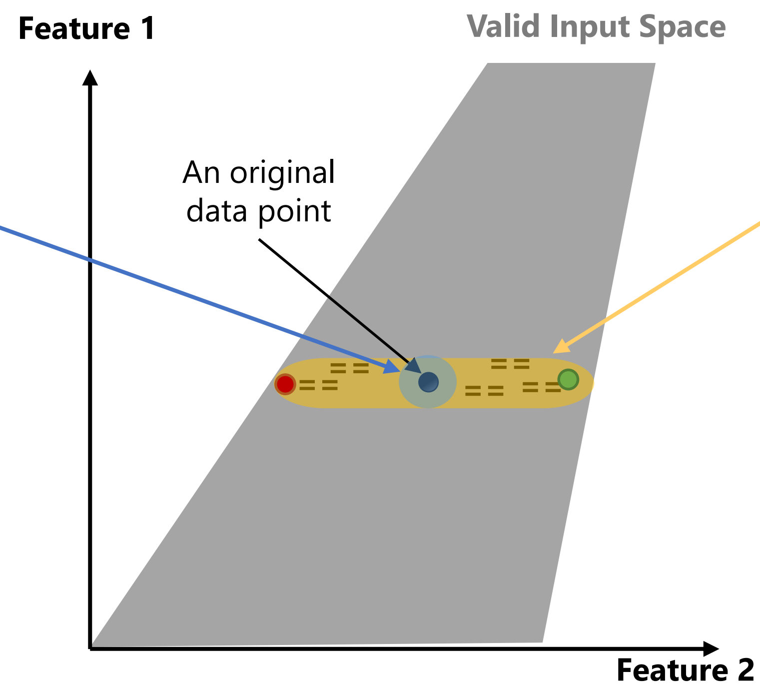


Domain Knowledge

Invariance Tests

Perturbation Test Definition

$$\forall\, x' \colon \|x - x'\|_p \le \delta \Rightarrow |f(x) - f(x')| \le \epsilon$$



Invariance Test Definition

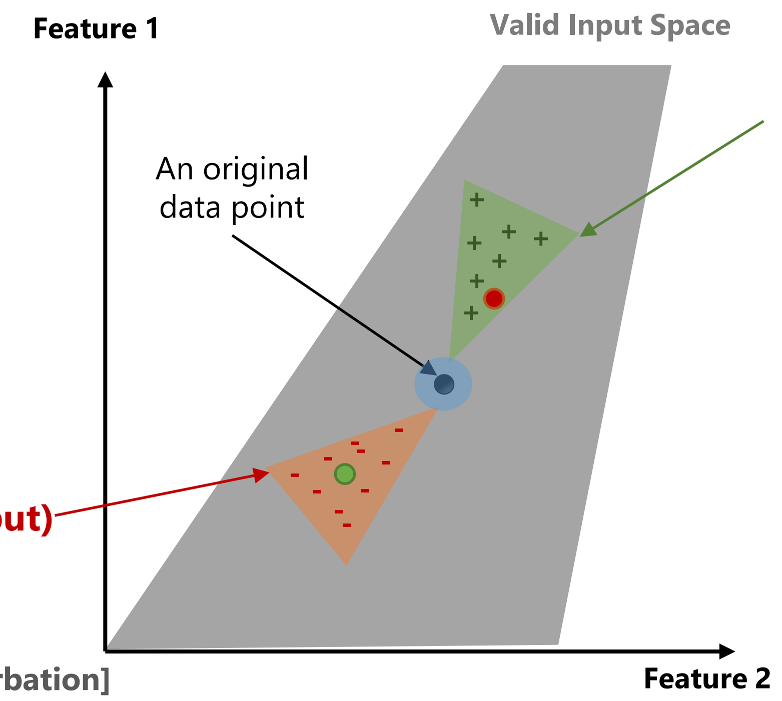
$\boldsymbol{x_i \nearrow, x_{i+1} \searrow, \ldots, x_n \leftrightarrow \Rightarrow f \leftrightarrow}$ [Rule Spec]

$\boldsymbol{\forall\, x', \forall i \in I_{pr}\colon (x_i - x_i') \le \delta_i}$ [Signed Perturbation]

$\boldsymbol{\Rightarrow |f(x) - f(x')| \le \epsilon}$ [Test Assertion]

These represent the failed inputs x for which the predictions are not consistent with the derived invariance tests.

Directional Expectation Tests



Directional Expectation Test (Increasing output)

$\boldsymbol{x_i \nearrow, x_{i+1} \searrow, \ldots, x_n \leftrightarrow \Rightarrow f \nearrow}$ **[Rule Spec]**

$\boldsymbol{\forall\, x', \forall i \in I_{inc}, (x_i - x_i') \leq \delta_i}$ **[Signed Perturbation]**

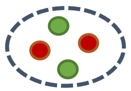
$\boldsymbol{\Rightarrow f(x) \geq f(x')}$ **[Test Assertion]**

Directional Expectation Test (Decreasing output)

$\boldsymbol{x_i \nearrow, x_{i+1} \searrow, \ldots, x_n \leftrightarrow \Rightarrow f \searrow}$ **[Rule Spec]**

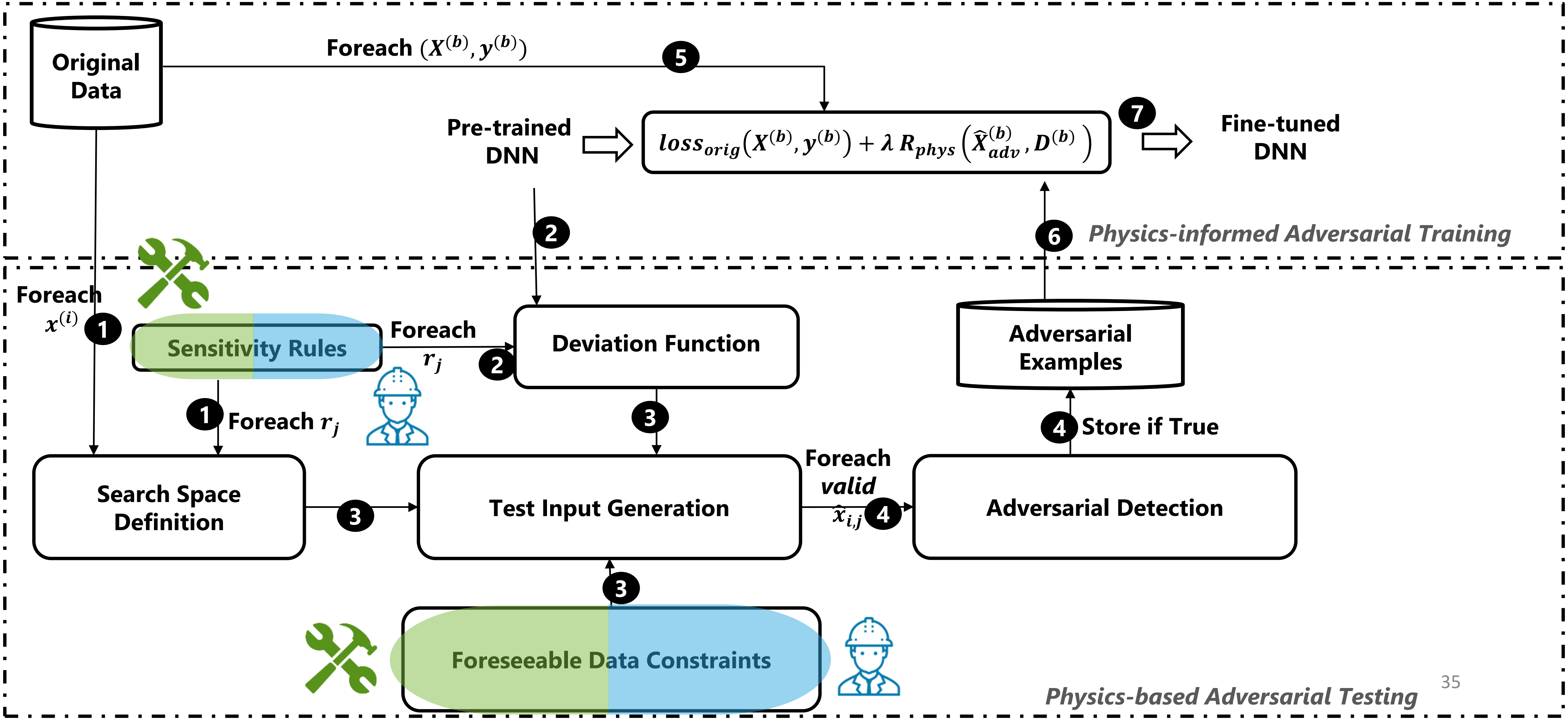
$\boldsymbol{\forall\, x', \forall i \in I_{dec}, (x_i - x_i') \leq \delta_i}$ **[Signed Perturbation]**

$\boldsymbol{\Rightarrow f(x) \leq f(x')}$ **[Test Assertion]**



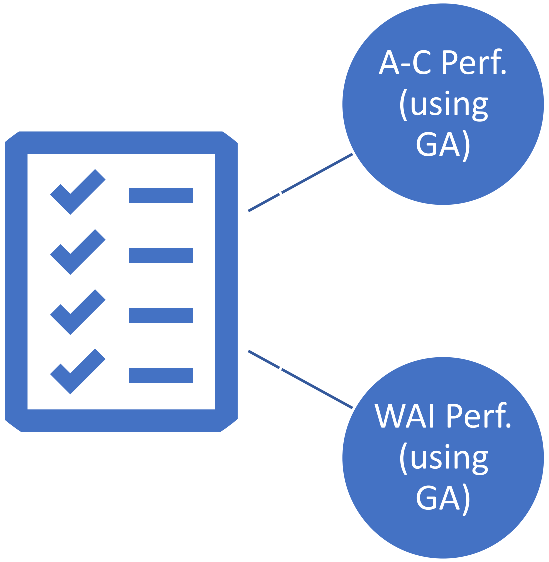
These represent the failed inputs x for which the predictions are not consistent with the derived directional expectation tests.

End-to-End Workflow of the Proposed Method



Evaluation Models & Results

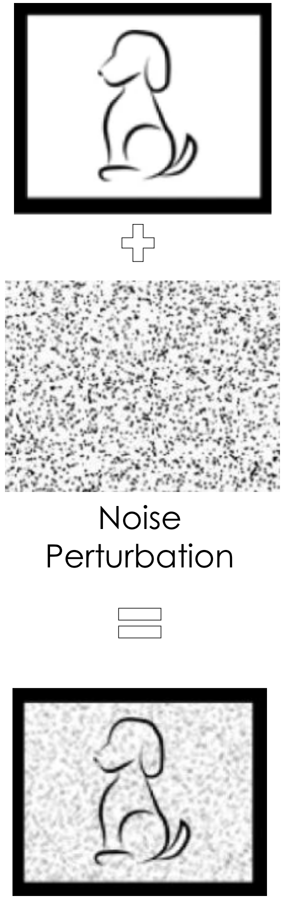
Model	Predicted Target	Description
Aircraft(A-C) Performance Model	α: angle of attack	The model maps steady-state angle of attack (α) to features related to flight conditions and wing configurations.
Wing Anti-Icing (WAI) Performance Model	T^{b}_{skin}: A-wing leading-edge skin temperature	The model maps the states of skin temperature sensors to features related to flight conditions, wing configurations, and high-pressure pneumatic system conditions at the wing root.
	T^{b}_{skin}: B-wing leading-edge skin temperature	

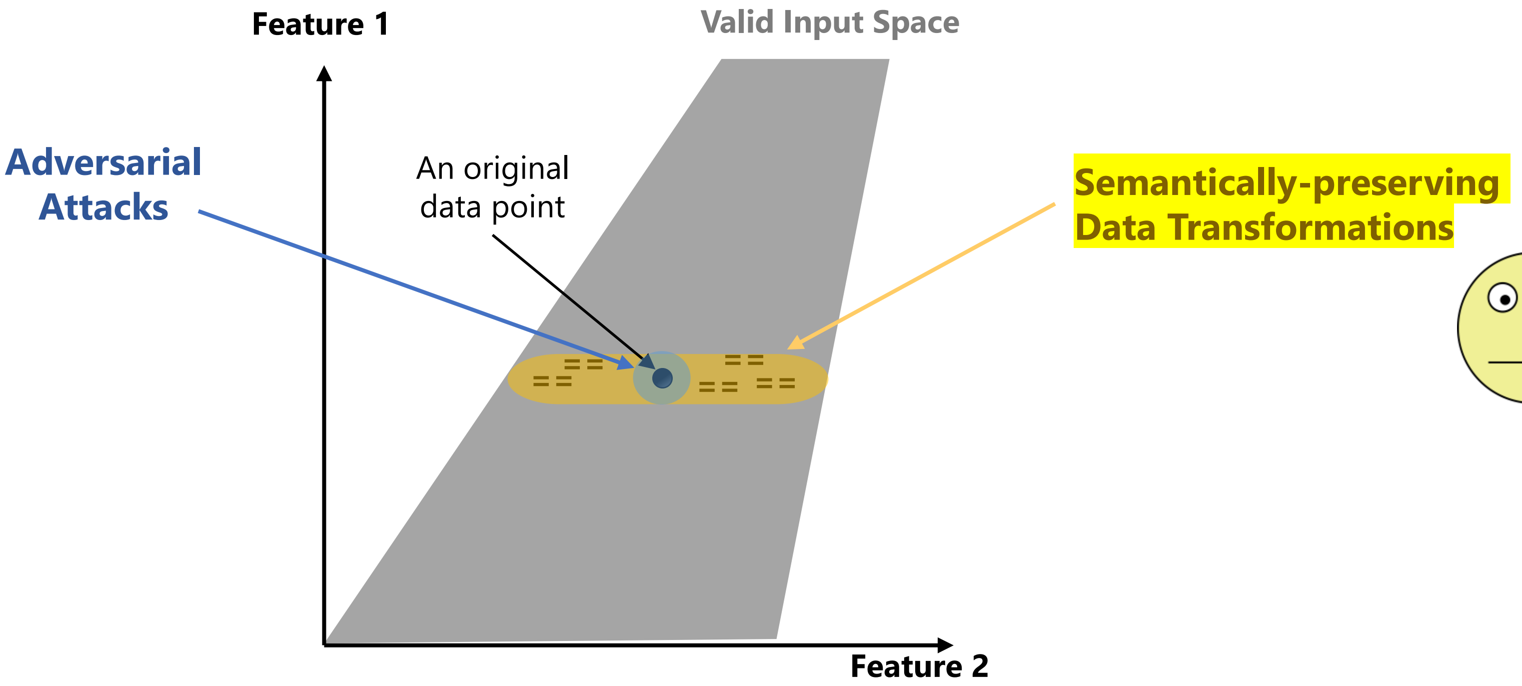


- **%Fixed Failures** 77.68%
- **%Err. Reduction** 10.8%

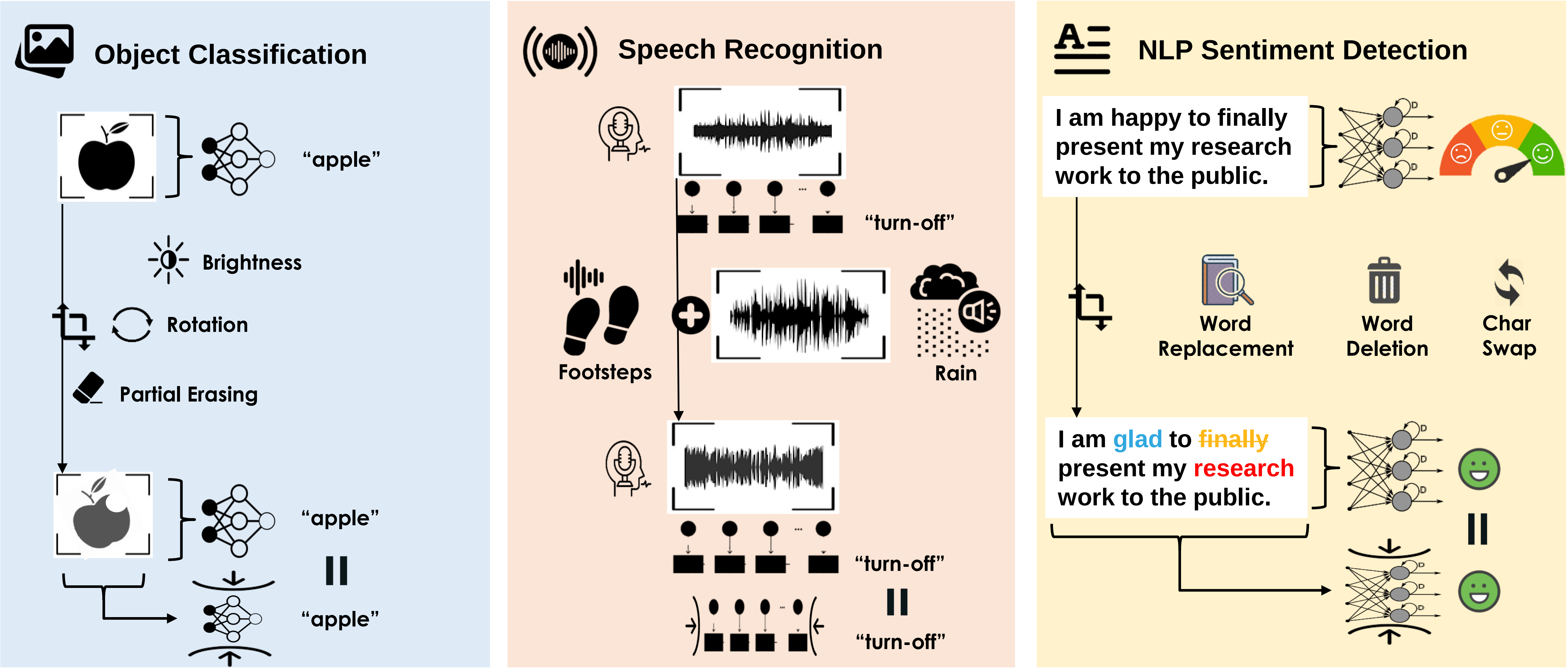
- **%Fixed Failures** 99.12%
- **%Err. Reduction** 4.8%

Analogies with other DL Applications





Semantically-preserving Data Transformations



Semantically-preserving Data Transformations

For Images:

Pixel-value Transformations:

Brightness

Contrast

Blurring

Partial Erasing

Pixel Perturbation

Affine Transformations:

Translation

Shearing

Scaling

Rotation

For Audio Speeches:

Signal-wise Conversions:

Speed

Pitch

Loudness

Additive Noise Signals:

Random noisy perturbations

Colored noises: white, pink, brown.

Indoor Noises: breathing, footsteps, laughing, clock-tick, etc.

Outdoor Noises: Engine, Fireworks, Rain, Train, etc.

For Natural Language Texts :

Char-level Transformations:

Random Insertion

Random Swap

Random Deletion

Word-level Transformations:

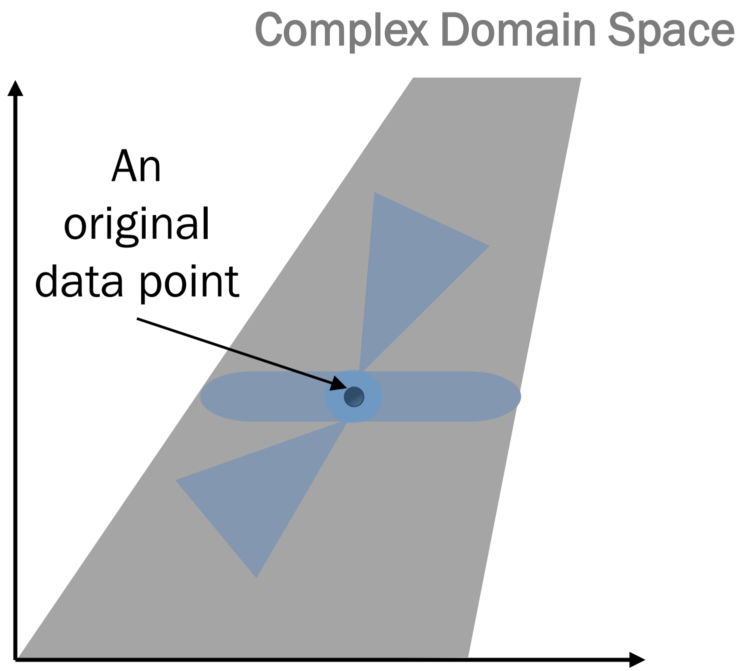
Synonym/Embedding Replacement

Random Insertion

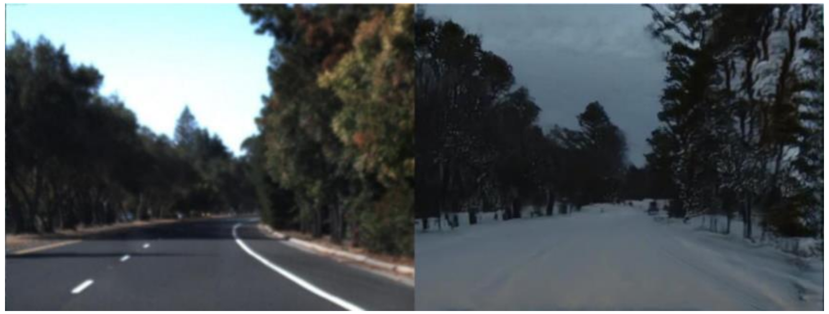
Random Swap

Random Deletion

How can we generate valid inputs from complex domains?

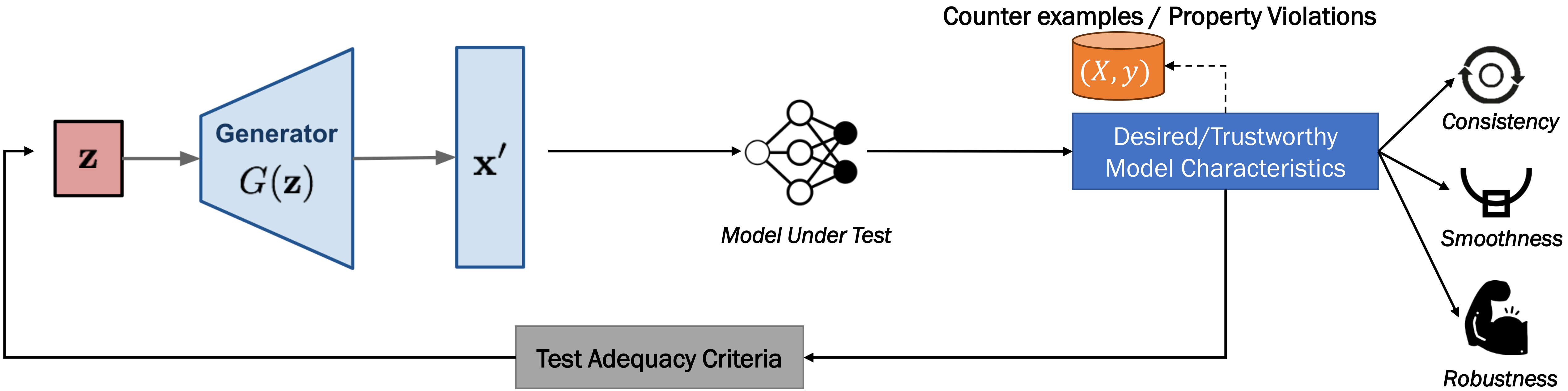


As software tests are written in code, DL tests can be produced by DL models !



DeepRoad [1] use GANs:

- map image from source domain to latent domain.
- generate image in the new domain from latent domain.



[1] Zhang et al. DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing. arXiv:1802.02295]

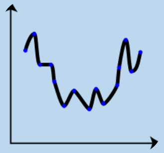
Conclusion

Training Program Bugs

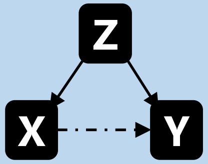
- Coding Mistakes
- Misconfigurations
- Toolkits' Misuse

Property-based Debugging Approach

Model Misconceptions

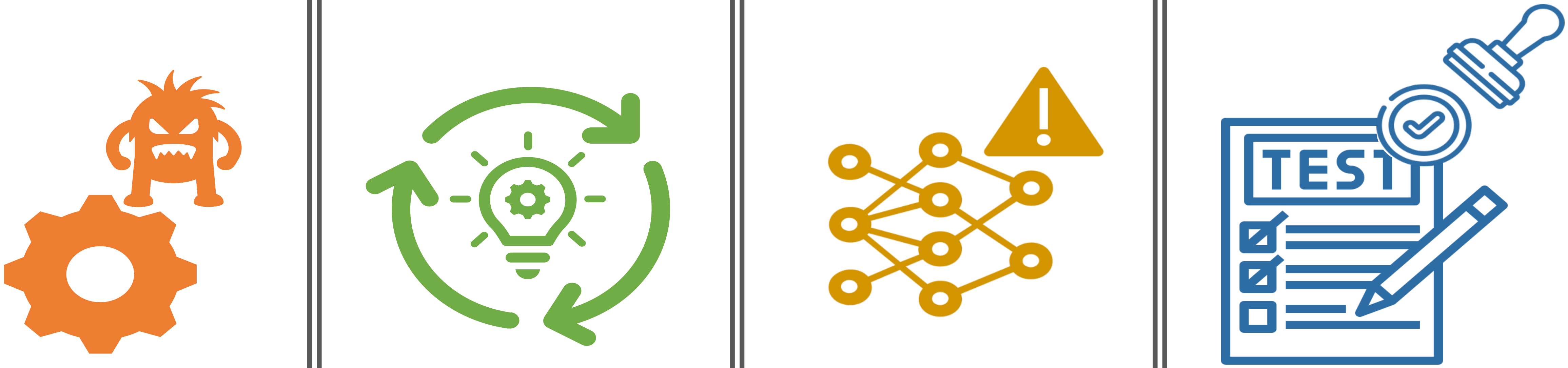


- Overfitting



- Spurious Correlation
- Shortcut Learning
- Pipeline Underspecification

Domain-Aware Testing Method



Towards Trustworthy DL Software System

Houssem Ben Braiek, Ph. D.