TestifAI: A Comprehensive Testing Framework for Safe AI

Author Name

***Abstract*—Deep learning (DL) models are critical in high-stakes domains such as autonomous driving, medical diagnostics, and security systems, where their deployment in real-world scenarios requires rigorous robustness testing due to diverse environmental conditions. Traditional metrics like neuron coverage, while essential, do not fully capture all corner cases, which can lead to unexpected model failures. To address this gap, this research introduces a comprehensive testing framework that enhances the correctness evaluation of models through a structured five-stage process. The first stage is specification, defines essential system properties to guide the entire testing process and ensure comprehensive coverage. The second sampling stage, gathering relevant samples for exhaustive model testing. In the test case generation stage, the defined properties are applied to create targeted test scenarios. The testing and probabilistic graph stage validates the effectiveness of these test cases and conducts robustness assessments both locally (within individual category) and globally (across multiple scenarios), employing a Bayesian network for detailed probabilistic and quantitative analysis of performance. The final stage is error summarisation, compiles and analyzes recorded errors to generate actionable graphical error reports and recommendations, thus guiding the refinement of models. This framework not only fills existing gaps in DL testing but also supports the development of models that are correct across varied environmental conditions.**

I. Introduction

Deep neural networks (DNNs) are being more widely used in a variety of applications, yet their correctness in practical applications remains a challenge.

Unlike traditional software, DNNs do not have a clear control-flow structure. They learn their decision policy through training on a large dataset, adjusting parameters gradually using several methods to achieve desired accuracy. Consequently, traditional software testing methods like functional coverage, branch coverage, etc. cannot be applied to DNNs, thereby challenging their use for safety-critical applications. Traditional software testing methods fail when applied to DNNs because the code for DNNs holds no information about the internal decision-making logic.

- Input space is extremely large, **unguided simulation** are highly unlikely to find erroneous behaviour
- Standards available in industry but **lack of logcial structure** and **system specification**
- Heavily depend on manual collections of test data under different conditions which become expensive as number of test condition increases
- Existing coverage criteria are not detailed enough to notice precise behaviours exhibited by DL systems.

By systematically addressing each aspect from initial specifications to detailed error analysis, we aim to enhance the model's accuracy and reliability in real-world scenarios.

This research makes the following key contributions to the field of deep learning correctness evaluation:

- We design an **end-to-end pipeline** for evaluating the correctness of system.
- We propose a **conceptual framework** that quantifies both local and global correctness,with formalized Bayesian probabilistic approach to verify system robustness.
- A novel **error summarization** approach which allows better identification of model weaknesses related to class and property.
- We perform all our **experiments** using publicly available deep learning models and MNIST dataset.

II. Background and Related Research

A. AI Systems

We define AI systems refer to any software system that is capable to perform complex tasks through use of data, algorithms and high computational power, that typically require human intelligence. These tasks include problem solving, reasoning, decision making and natural language understanding.

Deep learning is a subset of AI, that utilise deep neural networks (DNNs) for complex patterns recognition. Some AI systems solely based on DNN components. In contrast, hybrid AI systems combine DNNs with traditional softwares to produce the final output.

The development of DNNs is significantly different form traditional softwares. The developers explicitly define logic in traditional softwares but DNNs learn logic rules from the raw data rather than explicitly programmed. The developers shape these rules by modifying the training data, selecting the features and designing the DNNs architecture such as number od neurons and layers

Since DNNs rules are non-transparent so it is crucial to testing and correcting its error, particularly in safety-critical systems. This paper focuses on ...

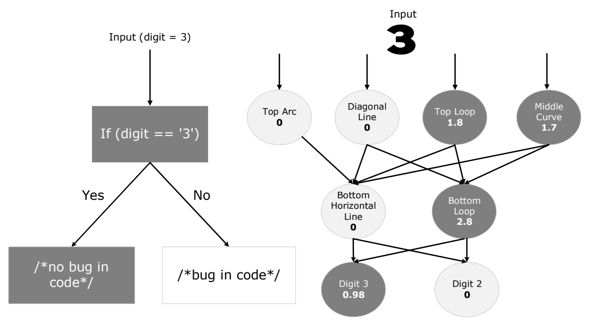

Fig. 1: Comparison between program flows of a traditional program (left) and a neural network (right). The nodes in gray denote the corresponding basic blocks or neurons that participated while processing an input.

B. Architecture of Deep Neural Network

DNNs are inspired by human brains with millions of interconnected neurons. They extract high level features from raw input using labeled training data without any human interference. DNN consists of atleast three layers: input, output and one or more hidden layers. Each neuron has direct connection to the neuron in next layer. The number of neurons on each layer can vary across different DNNs. The connections between neurons are known as edges and their associated weights called as DNN parameters. Neuron recieves input as a weighted sum of output from previous layer. It then applies non-linear activaion funciton (sigmoid, hyperbolic tangent, or Rectified Linear Unit (ReLU)) to this input to generate output. The objective of DNNs training is to learn parameters during training in order to make accurate prediction on unseen data during real world deployment.

C. DNN Testing Techniques

1) Coverage Criteria: Neuron coverage (NC) is the first coverage metric proposed in the literature to test DNNs. It is defined as the ratio of neurons activated by a test input to the total number of neurons in the model, where a neuron is activated when its activation value exceeds a predefined threshold. K-multisection neuron coverage (KMNC) calculates coverage by dividing the interval between lower and upper bounds into k-bins and measuring the number of bins activated by the test inputs. For a test suite, it is the ratio of uniquely covered bins to the total bins in the model. Neuron boundary coverage (NBC) measures the ratio of corner case regions covered by test inputs, with corner cases defined as activation values below or above those observed during training. Strong neuron activation coverage (SNAC) similarly measures how many upper corner cases, defined as activation values above the training range, are covered by test inputs. Modified Condition/Decision Coverage (MC/DC) metrics capture causal changes in test inputs based on the sign and value change of a neuron's activation. Likelihood-based Surprise Adequacy (LSA) uses Kernel Density Estimation (KDE) to estimate the likelihood of a test input during the training phase, prioritizing inputs with higher LSA scores as they are closer to classification boundaries. Distance-based Surprise Adequacy (DSA) is an alternative to LSA that uses the distance between activation traces of new test inputs and those observed during training.

2) DNN Testing Generation:

D. Limitations of Existing DNN Testing

Existing coverage criteria are often coarse-grained, offering a broad overview that lacks detail and may miss subtle yet important aspects of model behavior.

E. Realistic Synthetic Images

The properties that are exisiting in literature to verify any real world scenario AI systems include:

- **Noise**: Assessing performance under various types of noise (e.g., Gaussian noise, salt-and-pepper noise).
- **Rotation**: Recognizing digits regardless of their orientation.
- **Scaling**: Handling variations in the size of the digits.
- **Occlusion**: Maintaining recognition accuracy with partially occluded digits.

Let I represent the input image and $\hat{I}$ the transformed image subjected to various conditions. The transformations can be modeled as follows:

$$\hat{I}_{noise} = I + N \tag{1}$$
$$\hat{I}_{rotation} = R(\theta) \cdot I \tag{2}$$
$$\hat{I}_{brightness} = B(b) \cdot I \tag{3}$$
$$\hat{I}_{scaling} = S(s) \cdot I \tag{4}$$
$$\hat{I}_{occlusion} = I \cdot O \tag{5}$$

Where:

- N represents the noise added to the image.
- $R(\theta)$ denotes a rotation matrix with angle θ.
- $B(b)$ denotes brightness with factor b.
- $S(s)$ is a scaling matrix with scale factor s.
- O is an occlusion mask.

The performance of the AI system is evaluated based on its ability to correctly recognize the digit in $\hat{I}$ for each transformation.

F. Bayesian Networks

AI system is $op(f_1, \ldots, f_n)$, where op is some operation.

Specification is...

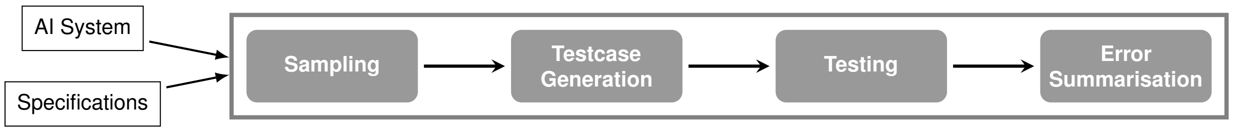

Fig. 2: Overview of Testing Framework

III. Approach

This section introduces our comprehensive approach to evaluating an AI system performance, summarised in Figure 3. It takes as input the description of an AI system and a set of relevant specifications. Then it has 4 major components. This step-by-step method highlights the strengths and weaknesses of the model.

- We sample inputs for the AI system
- To thoroughly assess the model, we generate (testcase generation) and test (testing) specific cases based on these properties.
- Following this, we summarise the errors, focusing on how individual image performance (local correctness) translates to the overall system performance (global correctness).

A. Specification

This step is crucial to specify the deployment environment of the AI system. This allows us to identify and define the relevant properties to ensure correctness in real-world situations. For instance, consider an AI system designed to recognize digits in images, such as in a calculator (sum) application.

B. Sampling

The sample selection process involves a random but balanced choice of samples from each class, focusing exclusively on instances that the model has correctly predicted. This method ensures a representative and fair distribution of data across all classes.

- Model Utilization:
  - A pre-trained CNN model is utilized to select samples.
  - Let $X = \{x_1, x_2, \ldots, x_N\}$ denote the set of MNIST images. The model function f predicts:

    $$f(x_i) \rightarrow y_i$$

  - The filter function g identifies accurate predictions, defined as:

    $$g(x_i) = \begin{cases} 1 & \text{if } f(x_i) = \text{true label of } x_i \\ 0 & \text{otherwise} \end{cases}$$

  - The subset S includes only correctly predicted images:

    $$S = \{x_i \in X \mid g(x_i) = 1\}$$

- Random Selection of Samples:
  - Randomly selects 100 samples from each class in S, totaling 1000 samples.
  - The random selection function R is defined to ensure:

    $$R(S_c, 100) \text{ for each class } c \text{ in } S$$

    where S_c represents the samples of class c within S.

C. Test Case Generation

This section outlines the generation of test cases to assess model robustness through properties such as noise, rotation, brightness adjustments, occlusion, and scaling.

- **Noise Addition:** Noise is added to an image x_i using a Gaussian noise model. The noise function p_n is defined as:

  $$p_n(x_i, \sigma) = x_i + \epsilon$$

  where $\epsilon \sim \mathcal{N}(0, \sigma)$ denotes the Gaussian noise with mean zero and standard deviation σ.
- **Rotation:** The rotation of an image x_i by an angle θ is modeled by the rotation function p_r:

  $$p_r(x_i, \theta) = \text{rotate}(x_i, \theta)$$

  where rotate$(\cdot, \theta)$ represents the rotation operation.
- **Brightness Adjustment:** Brightness adjustment of an image x_i is controlled by a multiplicative factor β, which scales the intensity of all pixels. The brightness function p_b can be defined as:

  $$p_b(x_i, \beta) = \beta \cdot x_i$$

  where $\beta > 1$ increases brightness, and $\beta < 1$ decreases it. This approach ensures that the image's contrast is preserved while adjusting its brightness.
- **Occlusion:** Occlusion is simulated by covering parts of the image x_i with a mask M. The occlusion function p_o is defined as:

  $$p_o(x_i, M) = x_i \cdot M$$

  where M is a binary mask that obscures parts of the image, with $M = 0$ for occluded pixels and $M = 1$ for visible pixels.
- **Scaling:** Scaling of an image x_i is controlled by a scaling factor s, which resizes the image. The scaling function p_s is defined as:

  $$p_s(x_i, s) = \text{scale}(x_i, s)$$

  where scale$(\cdot, s)$ represents the scaling operation, resizing the image by a factor of s.

D. Testing

The Testing section evaluates how accurately and confidently the AI subsystem's predicts under various properties applied to images from each class. This phase focuses on directly measuring and quantifying the correctness of the AI subsystem's.

After generating test cases, measure the AI subsystem's confidence for each class under each type of property.

1) Local Correctness: Local correctness involves checking the AI subsystem's performance on individual images subjected to different transformations. For each image x from a set of samples X, the AI subsystem produces a confidence score $f(x)$ representing its certainty in recognizing the digit. The local correctness for a transformation T applied to an image x is defined as:

$$\text{Local Correctness}(x, T) = f(T(x))$$

Where T can be any transformation such as noise addition, rotation, brightness adjustment, occlusion, or scaling. Each transformation is evaluated to determine its impact on the confidence score.

2) Global Correctness: Global correctness evaluates the AI subsystem's performance on combinations of images and transformations, such as adding two digits under different transformations. Given a set of 100 samples for each digit (0-9), we randomly select images to form digit pairs. The global correctness assesses the combined confidence for recognizing the result of operations like addition under each transformation.

For each transformation T, we define local correctness for digits d_1 and d_2 as follows:

$$\text{LC}_T(d_1) = \text{Local Correctness}(x_1, T)$$

$$\text{LC}_T(d_2) = \text{Local Correctness}(x_2, T)$$

Where x_1 and x_2 are randomly selected images representing digits d_1 and d_2.

The global correctness for each transformation is then defined as the product of the local correctness values:

$$\text{GC}_T = \text{LC}_T(d_1) \times \text{LC}_T(d_2)$$

To determine the optimistic and pessimistic global correctness across all transformations, we calculate the following:

- **Optimistic Global Correctness**:

$$\text{GC}_{opt} = \max_T(\text{GC}_T)$$

- **Pessimistic Global Correctness**:

$$\text{GC}_{pes} = \min_T(\text{GC}_T)$$

For the addition operation, if we denote the sum of digits d_1 and d_2 as d_{sum}, the overall global correctness for the operation is:

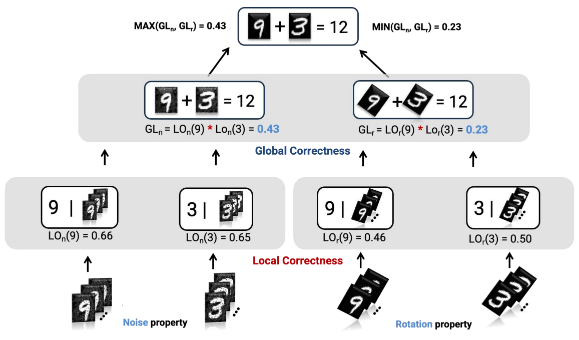

Fig. 3: Graphical View of Local and Global Correctness

$$\text{Overall Global Correctness} = \min(\text{GC}_{opt}, \text{GC}_{pes})$$

Where each transformation is evaluated separately, and the global correctness is determined based on the individual probabilities of correct recognition for each digit under each transformation.

3) how to write bayesian story, either it would be proper section or adust each baesian equation in all steps??? :

E. Error Summarization

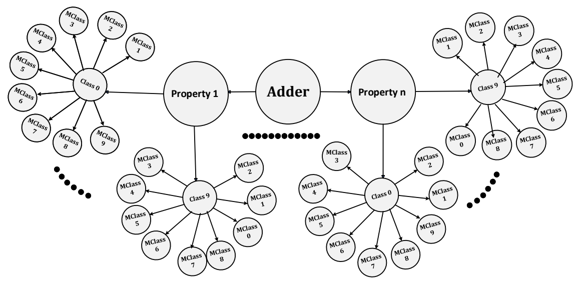

Fig. 4: Diagram of Error Summarization Highlighting Class-Property Impact

IV. Experiments

V. Research Questions

This paper addresses the following research questions applicable to various vision models and datasets:

- How can we design a comprehensive framework to test system correctness?
- How can we systematically evaluate the correctness both at local and global levels within framework?
- How can error summarization be employed to quantify the impacts on model correctness?

VI. Threats to Validity

This section outlines significant limitations and assumptions in our study that may affect the validity and reliability of our findings.

- **Random Sampling:** Our current approach assumes a uniform distribution of samples across all classes, which may not represent the true complexity and variability within real-world data. This uniform sampling can lead to biased evaluations if the class distribution in practical applications is skewed or non-uniform. We plan to enhance our sampling techniques to better capture the diversity and distribution of data in realistic scenarios. Improved sampling strategies will help in developing more robust and generalizable error summarization methods.

VII. Related Work

VIII. Conclusion

References

[1] Reference details