Debugging and Runtime Analysis of Neural Networks with VLMs (A Case Study)

Boyue Caroline Hu
University of Toronto, Canada

Divya Gopinath
KBR Inc., NASA Ames, USA

Corina S. Păsăreanu
KBR Inc., Carnegie Mellon Univ., NASA Ames, USA

Nina Narodytska
VMware Research, USA

Ravi Mangal
Colorado State University, USA

Susmit Jha
SRI International, USA

***Abstract*—Debugging of Deep Neural Networks (DNNs), particularly vision models, is very challenging due to the complex and opaque decision-making processes in these networks. In this paper, we explore multi-modal Vision-Language Models (VLMs), such as CLIP, to automatically interpret the opaque representation space of vision models using natural language. This in turn, enables a semantic analysis of model behavior using human-understandable concepts, without requiring costly human annotations. Key to our approach is the notion of *semantic heatmap*, that succinctly captures the statistical properties of DNNs in terms of the concepts discovered with the VLM and that are computed off-line using a held-out data set. We show the utility of semantic heatmaps for fault localization – an essential step in debugging – in vision models. Our proposed technique helps localize the fault in the network (encoder vs head) and also highlights the responsible high-level concepts, by leveraging novel *differential heatmaps*, which summarize the semantic differences between the correct and incorrect behaviour of the analyzed DNN. We further propose a lightweight runtime analysis to detect and filter-out defects at runtime, thus improving the reliability of the analyzed DNNs. The runtime analysis works by measuring and comparing the similarity between the heatmap computed for a new (unseen) input and the heatmaps computed a-priori for correct vs incorrect DNN behavior. We consider two types of defects: misclassifications and vulnerabilities to adversarial attacks. We demonstrate the debugging and runtime analysis on a case study involving a complex ResNet-based classifier trained on the RIVAL10 dataset.**

***Index Terms*—Deep Neural Networks, Debugging, Multi-modal models, VLMs, Fault localization, Runtime Monitoring**

I. Introduction

DNNs are increasingly used in many applications, impacting many aspects of our life. Unfortunately. like any other software, DNNs can have bugs. DNNs are notoriously hard to debug due to their complex and opaque decision-making processes. There are a number of sophisticated approaches for debugging DNNs including spectrum-based [1, 2] and mutation-based [3] techniques. They focus on locating and correcting bugs in syntactic structural elements of DNNs, such as specific layers or neurons with faults, faulty activation functions, incorrect learning rates so on. However current techniques do not provide an intuitive, global understanding of the underlying problems, which may prevent developers in quickly identifying the characteristics of the inputs that lead to failures and promptly fixing the discovered faults. Attribution approaches [4, 5, 6] typically attempt to identify portions of an input that impact model prediction the most. This can help debug individual tests but again does not provide a global view of the (reason for) faulty behaviour.

To address this problem, we explore multi-modal Vision-Language Models (VLMs), such as CLIP [7], to automatically interpret the opaque representation space of vision models using natural language. VLMs are trained on a large corpus of images accompanied by captions describing the images and are thus aware of high-level, human-understandable concepts describing a variety of images. We leverage VLMs to perform a *semantic analysis* of a separate vision model with respect to a set of concepts. Key to our approach is the notion of a *semantic heatmap*, that succinctly captures the statistical properties of the analyzed model in terms of the concepts. The VLM and the heatmaps help give *semantically meaningful insights* for the understanding and debugging of the the model with respect to a *set of inputs*.

We first describe a novel technique for fault localization – which is a key step in any debugging activity. Given a set of mis-classified images, the proposed technique uses a map that aligns the model's internal encoding representation with the VLM representation to compare the outputs of the model and the VLM on the same inputs. This helps identifying if the reason for the mis-classification is due to a bug in the encoder of the vision model or in its classification head. We further use semantic heatmaps to validate the location of the bugs and identify the reason for failure in terms of the concepts. To this end, we compute *differential heatmaps*, which summarize the semantic differences between the correct and incorrect behaviour of the analyzed DNN and thus pinpoint the problematic concepts.

We also propose a lightweight runtime analysis to detect and filter-out defects at runtime, thus improving the reliability of the analyzed DNNs. The runtime analysis works by measuring and comparing the similarity between a heatmap computed for a new (unseen) input and the heatmaps computed a-priori for correct vs incorrect DNN behavior. If the heatmap of the input is found to be more similar to the heatmap that summarizes correct behaviour, we deem it as *good* and we conjecture that the DNN gives the correct answer for it; otherwise we flag it as problematic.

We illustrate the proposed techniques on a case study using the RIVAL10 dataset and a fairly complex image classification ResNet18 model. We study faults that are due to model inaccuracies or adversarial attacks. We provide all the artifacts for our experiments here.

We summarize our contributions as follows:

- **Novel fault localization**: We present an analysis technique that leverages VLMs to *localize* errors in a separate vision model, classifying them as *encoder* or *head* errors. The technique leverages novel *differential heatmaps* to isolate predicates in terms of high-level concepts responsible for errors. The technique can handle different *types of errors*, including misclassifications due to model inaccuracies and vulnerabilities to adversarial perturbations. In the latter case, the technique helps identify robust and non-robust features in the model.
- **Novel runtime defect detection**: We describe a lightweight technique for detecting misclassifications and adversarial inputs at runtime, by comparing the heatmap of a new input with *summary heatmaps* of correctly and incorrectly classified inputs – and thereby improving the robustness and reliability of analyzed models.
- **Case study**: We illustrate the proposed techniques on a case study from the image classification domain. Our results indicate that most vulnerability errors are due to encoder errors (90% of adversarial inputs), while for misclassifications this picture is more nuanced (approx. 40% encoder errors and 60% header errors). Further, we present results for runtime detection of misclassifications and adversarial perturbations, considering multiple norm-bounded attacks, achieving approx. 80% detection accuracy for both types of errors.

We envision that the fault localization results and semantic heatmaps computed with the help of VLMs can be further leveraged to perform targeted testing, re-training, and repairing of vision DNNs. We leave these activities for future work. We see our approach as complementary to existing approaches contributing towards the larger goal of safety assurance of DNN-enabled systems.

II. Preliminaries

Neural Networks. We consider feed-forward neural networks as functions $f: X \to Y$, where X is a (high-dimensional) space $\mathbb{R}^d$ and Y is $\mathbb{R}^{|C|}$ where C is a finite set of labels or classes. For classification, the output defines a score (or probability) across $|C|$; the class with the highest score is output as the prediction. Previous work [8, 9, 10, 11] indicates that neural classifiers can be seen as the composition of an encoder, denoted here as $f_{enc}: X \to Z$, and a head, denoted here as a $f_{head}: Z \to Y$, where Z ($\mathbb{R}^{d'}$) is the *embedding (or representation) space* of the network.

The encoder translates the inputs, e.g., pixels forming images, into higher-level representations, and the head computes the appropriate label based on these representations. As an example, in a typical convolutional neural network, the convolutional layers act as the encoder and are responsible for extracting high-level concepts from the inputs, while the fully-connected layers act as the head, and are responsible for classification based on the extracted concepts. The *encoding of an input* x is $f_{enc}(x)$.

Vision-Language Models. Vision-language models (VLMs) [7], denoted as g, consist of two encoders, one for images ($g_{enc}^{img}: X \to Z$) and one for text ($g_{enc}^{txt}: T \to Z$), both mapping to the same representation space Z. VLMs are trained on image-caption pairs such that, for each pair, the representation of the image and corresponding caption are as similar as possible (e.g., measured via cosine similarity $\cos$) in the common space Z. VLMs can be used for a variety of tasks such as image classification, visual question answering, or image captioning.

By default, a VLM can be used to select the caption (from a set of captions) which has the highest similarity with a given image. This strategy can be leveraged for *zero-shot classification* of images [7] (this is used to design g_{head} here). For instance, given an image from the ImageNet dataset, the label of each ImageNet class, e.g., *truck*, can be turned into a caption such as *"An image of a truck"*. Then one can compute the cosine similarity between the embedding of the given image with the text embeddings corresponding to the captions constructed for each class, and pick the class that has the highest similarity score.

Aligning a vision model with the VLM. Previous work [11] has shown that one can build an affine map r_{map} that links the representation spaces of the vision-based DNN and the VLM, $r_{map}: Z_f \to Z_g$. The map has the form $r_{map}(z) := Mz + d$ and is learned by solving an optimization problem:

$$M, d = \underset{M,d}{\operatorname{argmin}} \frac{1}{|D_{\text{train}}|} \sum_{x \in D_{\text{train}}} \|M f_{enc}(x) + d - g_{enc}^{img}(x)\|_2^2. \quad (1)$$

More complex maps can be defined and learned as well. We learn and leverage such a map in order to use the representation space of a VLM for interpreting the behavior of a separate vision model.

Concepts and Predicates. We aim to reason about vision models in terms of high-level, human understandable concepts. For instance, for a vision model tasked with distinguishing between several classes such as *cat, dog, bird, car,* and *truck*, the relevant concepts could be *metallic, ears,* and *wheels*. Inferring representations of the concepts learned by a model can be very challenging, typically requiring manual annotations of inputs with the presence/absence of relevant concepts [12, 13, 14, 9, 8, 15]. Previous work [16] has shown how to leverage a separate VLM and the affine map to reason about concepts. Essentially, this is achieved in a way that is similar to how VLMs perform zero-shot learning, using embeddings of concepts rather than of classes. While the goal of the work in [16] was to formally prove properties in terms of the extracted concepts, in this work we aim to use concepts in complementary activities, namely fault localization and runtime defect detection.

Given an image x, suppose one is interested in the presence or the absence of concepts con_1 and con_2 in the representation space of f for the image. Let $\overline{con}$ denote the mean of the text embeddings (in g) for a set of captions that all refer to a concept con in different ways. The mean is computed since there may be many different valid captions constructed for the same concept and each caption may lead to a slightly different embedding. One can then compute the similarity score between the (embedding of the) image (mapped to the VLM) and the (embedding of) each concept and pick the concept that has the highest score, i.e., compute and compare:

$$\cos(r_{map}(f^{img}_{enc}(x)), \overline{con_1}) > \cos(r_{map}(f^{img}_{enc}(x)), \overline{con_2})$$

Following [16], we use predicates of the form $con_1 > con_2$ as a shortcut for the above. As an example, for an image representing a *truck*, one may expect that the model is more aware of concepts *metallic* and *wheels* rather than *ears*, written as *metallic* > *ears* and *wheels* > *ears*, as *ears* are typically associated with an animal.

III. Case Study

We demonstrate the proposed debugging and runtime detection on a case study analyzing a vision model trained on the RIVAL10 dataset [17].

1) Dataset and Concepts: **RIVAL10** is a subset of ImageNet [18], categorized into 10 classes corresponding to those of CIFAR10, including *truck*, *car*, etc. The dataset includes annotations by domain experts, providing instance-wise labels for 18 attributes that reflect human-understandable concepts, including *wheels*, *tall*, etc. While our approach does not require concept annotations for each image, we use these annotations here to identify the relevant concepts for each class (which would normally be supplied directly by domain experts). The *dictionary* of relevant concepts for each class is provided below:

class truck:{wheels, text, metallic, rectangular, long, tall};
class car: {wheels, metallic, rectangular, };
class plane: {wings, metallic, long, tall};
class ship: {metallic, rectangular, wet, long, tall};
class cat: {ears, colored-eyes, hairy};
class dog: {long-snout, floppy-ears, ears, hairy};
class equine: {long-snout, ears, tail, mane, hairy};
class deer: {long-snout, horns, ears, hairy};
class frog: {wet};
class bird: {wings, beak, patterned};

For instance, for class *truck*, concepts *wheels, tall, long, rectangular, metallic* and *text* are all considered relevant whereas all other concepts (e.g., *tail* and *wings*) are irrelevant. Based on these concepts, we define predicates relating all the possible pairs of concepts (a total of $18 \times 18 = 324$ predicates). We also identify the relevant predicates (of the form $relevant > irrelvant$) for each class. For instance, for class *truck*, we identify 72 relevant predicates, e.g., $wheels > wings$ and $metallic > tail$.

2) Vision Model: We consider a vision classifier, ResNet18 [19], trained on the RIVAL10 dataset. The architecture of ResNet18 can be decomposed into an encoder and a head for classification based on the encoded images; Fig. 1 (left) shows such a decomposition. The encoder contains the initial convolutional and pooling layers, followed by four groups of residual blocks consisting of convolutional, batch normalization, and ReLU layers, and then finally an average pooling layer for dimension reduction. The output of the average pooling layer is the image embedding to be mapped to the VLM for analysis. The head is a single linear layer with no activation functions with inputs of type $\mathbb{R}^{512}$ and outputs of type $\mathbb{R}^{10}$.

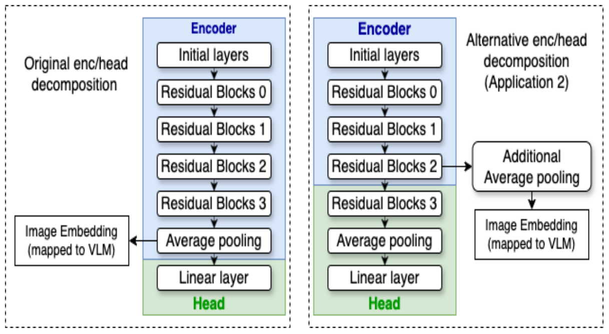

Original enc/head decomposition
Encoder
Initial layers
Residual Blocks 0
Residual Blocks 1
Residual Blocks 2
Residual Blocks 3
Average pooling
Image Embedding (mapped to VLM)
Linear layer
Head
Alternative enc/head decomposition (Application 2)
Additional Average pooling

Fig. 1: Architecture of the vision model ResNet18. The left side shows the encoder-head decomposition of ResNet18, and the right side shows an alternative decomposition.

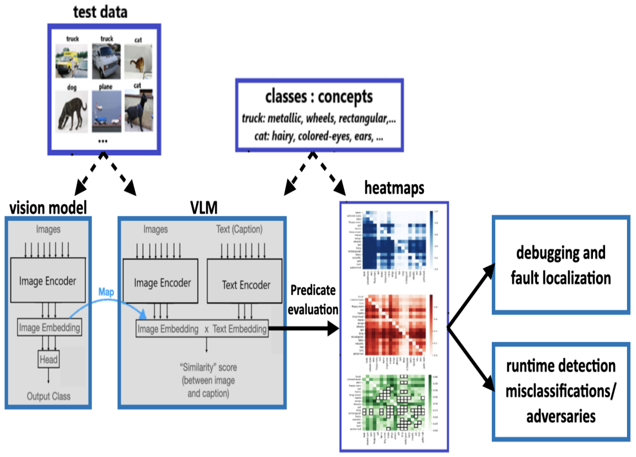

test data
truck
cat
dog
plane
classes : concepts
truck: metallic, wheels, rectangular,...
cat: hairy, colored-eyes, ears, ...
vision model
VLM
heatmaps
Images
Image Encoder
Image Embedding
Head
Output Class
Map
Text (Caption)
Text Encoder
Image Embedding x Text Embedding
"Similarity" score (between image and caption)
Predicate evaluation
debugging and fault localization
runtime detection misclassifications/ adversaries

Fig. 2: Leveraging VLMs for debugging and run-time analysis of vision models.

IV. Approach

Figure 2 provides a high-level overview of our approach to VLM-based debugging and run-time analysis of vision models. Assume we are given a vision model to analyze and a dictionary of high-level concepts that are relevant for each class in the vision task. We also have access to a set of test images (annotated with ground truth for each output class but not concepts).

We use a separate off-the-shelf VLM and build an *affine map* to align the embedding spaces of the vision model and the VLM. We leverage the map and the concept definition to define *concept predicates* as described in Section II. We evaluate these predicates on the set of test images to produce various *semantic heatmaps* that succinctly summarize information about the vision model in terms of high-level, human-understandable concepts. The VLM and the semantic heatmaps enable two types of novel analyses: (i) debugging and fault localization and (ii) runtime defect detection. The defects we consider are those that cause misclassifications, and vulnerabilities to adversarial attacks. We provide more details below.

A. Semantic Heatmaps

We build semantic heatmaps as 2-dimensional graphs where the X and Y axes contain all the concepts. Each cell represents a predicate $con_1 > con_2$, where con_1 is the concept on the Y-axis and con_2 is the concept on the X-axis.

Single-input Heatmaps. This simple heatmap is defined per input; every cell in the graph has a value of 0 or 1 corresponding to the valuation of the respective concept predicate.

Ground-truth Summary Heatmaps. This heatmap summarizes the satisfaction probabilities for the concept predicates for a set of inputs that *all have the same ground truth*. Given a set of inputs, the satisfaction probability for a concept predicate ($con_i > con_j$) measures the ratio of the inputs in that set that satisfy the predicate.

Throughout the paper we depict these heatmaps in blue. For example, Figure 3a presents the ground-truth heatmap for a set of images with ground-truth label *truck*. The dark color corresponding to *metallic* > *ears* indicates high satisfaction probability. This is as expected since concept *metallic* is relevant for class *truck* while concept *ears* is irrelevant; relevant and irrelevant concepts for a class are indicated by the user.

To enable different kinds of analyses,we compute the same kind of heatmap for different groups of inputs. For example, Figure 3b presents the ground-truth heatmap for the sub-set of images with ground-truth label *truck* which are all *correctly classified* by the vision model, while Figures 3c and 3d present the ground-truth heatmaps for sub-sets of images with ground-truth label *truck* which are *misclassified*, due to different errors.

Throughout the paper we depict predicates that relate *relevant and irrelevant concepts* for a particular class with a red outline in the different heatmaps.

Output-label Summary Heatmaps. This heatmap is similar to the previous one, but it summarizes the satisfaction probabilities for the predicates for a set of inputs that *all have the same output label* (but may have different ground-truth). Throughout the paper we depict these heatmaps in orange (see Figure 7a).

Differential Heatmaps. This type of heatmap enables a differential analysis between different categories of inputs. Given two heatmaps, H_1 and H_2, of the same dimensionality, a differential heatmap is constructed by calculating the magnitude of the difference between the values in each cell. $\forall i,j \; H_{diff}[i,j] = |H_1[i,j] - H_2[i,j]|$. The higher the value in a cell, the more the two heatmaps differ in the satisfaction of the corresponding predicate. The darkness of the color in each cell is proportional to the difference. Note that the heatmaps being compared could be of any type (single-input, ground-truth or output-label heatmaps). Throughout the paper we depict these heatmaps in green (see Figure 3e and Figure 3f for examples).

Binarized Heatmaps. For defect detection, we also find it useful to *binarize* the summary heatmaps, with a threshold t, i.e., 1 if the satisfaction probability is $\geq t$, and 0 otherwise.

B. Debugging and Fault Localization

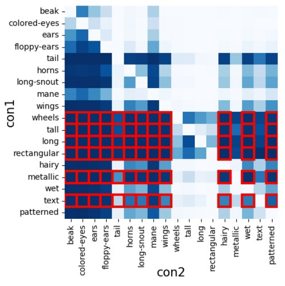
Debugging Misclassifications. We propose a novel technique that uses VLMs to perform fault localization, which is a key first step in any debugging task. Specifically the technique can determine if a wrong classification is due to a bug in the encoder or the head of a vision model, while considering different decompositions (head vs. encoder) for the model. Further, by determining which concept predicates are violated by misclassified inputs, we can also *localize the reasons for failure*, i.e., determine which concepts are poorly understood by the model. This information can in turn be used to improve the model, e.g., by selecting images with the desired concepts and using them for re-training.

1) Head vs. encoder errors: Consider an input x with ground truth class c such that $f(x) \neq c$, i.e., it is misclassified by the vision model. The misclassification can be caused by errors in the encoder or in the head. Note that, for an image x, encoder and head errors may be present at the same time. However, since updating the encoder would also require updating the head, the encoder errors should be addressed first, thus we consider such cases as exclusively encoder errors.

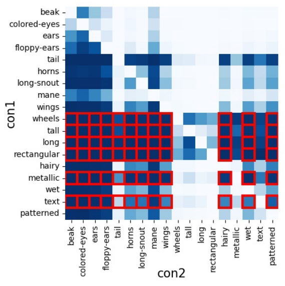
Assume that x is correctly classified by the VLM via zero-shot classification, i.e., $g_{head}(g_{enc}^{img}(x)) = c$. Further assume r_{map} is of high quality. Let $e_1 := r_{map}(f_{enc}(x))$ refer to the embedding of x produced by the vision model and mapped to the VLM space, and let $e_2 := g_{enc}^{img}(x)$ refer to the embedding of x directly produced by the VLM.

- *Encoder Error.* If e_1 and e_2 lead to different zero-shot classification outputs, i.e., $g_{head}(e_1) \neq g_{head}(e_2)$, then it is likely that the embedding e_1 is incorrect, and hence the error is in f_{enc}. We call it *encoder error*.
- *Head Error.* If e_1 and e_2 lead to the same zero-shot classification outputs, i.e., $g_{head}(e_1) = g_{head}(e_2)$, it is likely that the embedding computed by f_{enc} and mapped to the VLM's representation space via r_{map} is correct since it allows the VLM head to output the correct label. Hence the error is in f_{head}; we call it *head error*.

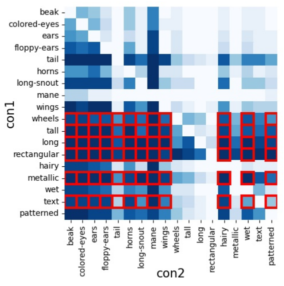
2) Analyzing Errors with Heatmaps: We compute differential heatmaps to help localize the predicates that distinguish between correct vs incorrect inputs, for different types of error (head vs encoder). Specifically, given the sets of misclassified inputs for a specific ground-truth label categorized as encoder and head errors respectively, we construct ground-truth summary heatmaps for each set. We then compare these heatmaps with the ground-truth summary heatmap for the images correctly classified to the specific label. We do this by constructing a differential heatmap which calculates the absolute difference between the summary heatmap for encoder errors and correctly classified inputs, and a similar differential



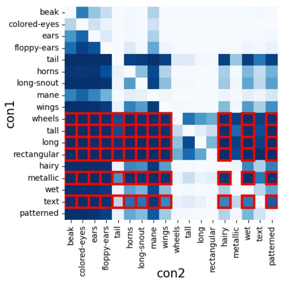
(a) With all images with ground truth *truck*.



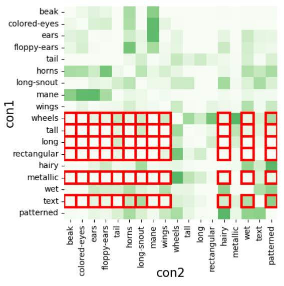
(b) With images correctly classified as *truck*.



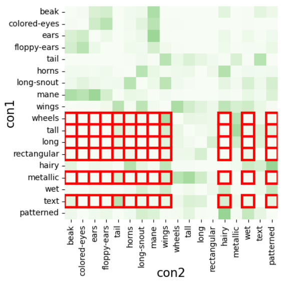
(c) With misclassified *truck* images due to encoder error.



(d) With misclassified *truck* images due to head error.



(e) Differential heatmap for encoder errors.



(f) Differential heatmap for head errors.

Fig. 3: Ground-truth summary and Differential heatmaps for *truck* images. Red outlines indicate relevant predicates for *truck*.

heatmap for head errors; see Figure 3e and Figure 3f. A large difference indicates predicates that distinguish between correct versus incorrect classifications. Potentially, these predicates can be leveraged to retrieve more images for retraining the model.

Debugging Vulnerabilities and Identifying Robust and Non-Robust Features. We also consider another type of defect, namely vulnerabilities to adversarial perturbations. Vision DNNs are known to be vulnerable to *adversarial perturbations*; small changes (imperceptible to human eye) applied to validly classified images can fool the model into misclassification [20, 21, 22]. We can apply the same debugging approach for this type of defects. Recent work [23] points out that adversarial examples can be attributed to the presence of brittle *non-robust* features. We propose the use of semantic heatmaps to identify robust and non-robust features of a model, where we consider as features the human-understandable concepts as studied in this paper. This is another debugging activity, where the cause of misclassification is potentially the feature (or concept) found to be non-robust.

C. Runtime Defect Detection

We also propose the use of semantic heatmaps for runtime detection of adversarial inputs or of inputs that lead to misclassifications. This can potentially improve model robustness and reliability without incurring the cost of retraining.

Run-time detection of adversarial inputs. For each output class, we construct a ground-truth summary heatmap with all clean (i.e. unperturbed), correctly classified images (as in Figure 3b for *truck*) and an output summary heatmap with all adversarial inputs misclassified to the class (generated during offline analysis). Then, for each input encountered during runtime with output class c, this input is considered adversarial if its single-input heatmap is more similar to the perturbed heatmap than the clean heatmap of class c, and clean otherwise. To determine the similarity between a single-input heatmap and the above summary heatmaps, we first binarize the summary heatmap with a threshold t to highlight strength predicates with high satisfaction probabilities, i.e., 1 if satisfaction probability $\geq t$, and 0 otherwise. We compute Intersection over Union (IoU) as the similarity metric, between each single-input heatmap and the binarized summary heatmaps; we use IoU_p for similarity with the the binarized perturbed heatmap, and IoU_n for similarity with the binarized clean heatmap. An input with $IoU_p > IoU_n$ suggests it is adversarial, and clean otherwise.

Run-time detection of misclassifications. Even if the input is not adversarially or naturally perturbed, it may lead to misclassifications (due to inaccuracies in the model). Detecting (and possibly preventing) such inputs would improve the reliability of the model, particularly relevant in safety-critical settings. The problem is challenging since the ground-truth is not known at runtime. To detect misclassifications in the absence of ground truth, we can take a similar approach as above, by comparing the single-input heatmap with the summary heatmaps computed for correct vs incorrect behavior.

V. Experiments

A. Experimental Setup

We demonstrate our techniques on the RIVAL10 case study. The RIVAL10 dataset is divided into a training set with 21,000 images and a test set with 5,000 images. The ResNet18 model was pre-trained on the full ImageNet dataset and the final layer of the model, i.e., the head, was further fine-tuned on the RIVAL10 training set in a supervised fashion. The model has an accuracy of 96.73% on the test set.

1) Vision-Language Model: As mentioned, we use a pre-trained CLIP model for our experiments, which has a Vision Transformer (ViT-B/16) [24] as the image encoder and a Transformer[25] as the text encoder. The representation space of the CLIP model is of type $\mathbb{R}^{512}$. The head g_{head} is a zero-shot classifier, implemented as detailed in Section II. This classifier achieved an accuracy of 98.79% on the RIVAL10 test set. Our experimental setup is similar to previous work [16], which showed that CLIP can be used to correctly identify many relevant RIVAL10 concepts.

2) Map between ResNet18 and CLIP: The map is obtained by learning an affine map r_{map} from the representation space of the ResNet18 vision model to the representation space of the CLIP model [11]. We learn a map of type $\mathbb{R}^{512} \rightarrow \mathbb{R}^{512}$, with a low Mean Squared Error (MSE) of 0.963 and a high Coefficient of Determination (R^2) of 0.786 on the test data, suggesting good quality. The problem is solved via gradient descent on RIVAL10 training data for 50 epochs using a learning rate of 0.01, momentum of 0.9, and weight decay of 0.0005.

B. Debugging Misclassifications

ResNet18 misclassifies 163 images from the RIVAL10 test set. We investigate the cause of failure by locating if the bug lies in the encoder or in the classification head, and isolating the predicates that get violated by the misclassified inputs.

1) Localization to Encoder vs Head Errors: We apply our localization approach to categorize the misclassifications as encoder errors and head errors. Essentially the approach treats the CLIP model as an oracle to validate the ResNet18 model. Therefore, it requires that the CLIP model classifies accurately when fed directly with the images. Out of the 163 images, 145 images are correctly classified by CLIP. We use these images to localize the errors in the ResNet18 model. Out of these images, we found that 61 images are misclassified due to encoder errors and 84 images due to head errors, using the procedure described in Section IV.

As an example, consider the *ship* images on the left of Figure 4 that are misclassified by the ResNet18 model. The one on the left is misclassified due to an encoder error because $g_{head}(e_2)$ (i.e., CLIP) is *ship* but $g_{head}(e_1)$ (i.e., CLIP via r_{map}) is *frog*. The one on the right is misclassified due to a head error because $g_{head}(e_2)$ and $g_{head}(e_1)$ (i.e., CLIP via r_{map}) are both *ship* but the network output is *dog*.

2) Validating Fault Localization with Mutations: To validate our the fault localization technique, we employ a mutation-based approach by manually introducing errors in the encoder or the head of ResNet18: this is achieved by randomizing the weights of a small set of neurons. If our technique is valid, then by introducing errors in the encoder, the localization should result in statistically more encoder errors than head errors, and vice versa. From ResNet18, we created two different mutated models, one with randomized weights for two neurons in each convolutional layer in the encoder, and one with randomized weights introduced only in the final classification layer, i.e., in

TABLE I: Number of encoder and head errors found through error localization for all the images in the test set.

Mutation location	# encoder error	# head error
No mutation (original ResNet18)	61	84
Mutation in Encoder	4271	405
Mutation in Head	101	4571
Mutation in Residual Block 3	1183 (orig decomp) 438 (alt decomp)	3064 (orig decomp) 3809 (alt decomp)

the head. From row 2 of Tbl. I, we can see that introducing mutations in the encoder resulted in significantly more encoder errors than head errors, as expected. Similarly, from row 3, mutations in the head resulted in significantly more head errors than encoder errors, again as expected. This suggests that our localization approach, although not perfect, can provide useful insights into localizing the errors.

We also attempted to determine if our approach can perform a more precise layer-wise localization. For example, to localize errors specifically in Residual Block 3 as shown in Figure 1, we can consider an alternative decomposition for ResNet18 by moving Residual Block 3 from the encoder to the head. In this case, mutations introduced specifically in Residual Block 3 would be encoder errors with the original decomposition, but head errors with the alternative decomposition. Since all Residual Blocks contain convolutional layers, to obtain an accurate r_{map}, we added average pooling to reduce the dimensionality of the output of Residual Block 2 before the mapping to CLIP embedding space[1].

The bottom row of Tbl. I presents the number of encoder and head errors with the original and alternate decomposition. Surprisingly, we observe that regardless of the location of the mutation, i.e., mutation in the encoder or the head, the number of head errors exceeds encoder errors. At the same time, the original decomposition yields more encoder errors compared to the alternate decomposition, indicating a shift in the location of the mutation. This underscores the potential of employing different encoder-head decompositions for precise layer-wise error localization; we leave this exploration for future work.

3) Analysis of Errors with Heatmaps: With semantic heatmaps, we can visualize and validate localization results and understand the reasons for failure. We first perform an analysis using the heatmaps for inputs that are correctly classified to validate that the behavior of the vision model can indeed be analyzed semantically in terms of domain-specific concepts.

We use images and concepts related to class *truck* to illustrate. The RIVAL10 test dataset has 502 images with ground-truth *truck*. The ResNet18 model classifies 474 correctly as *truck*. Figure 3a shows the ground-truth summary heatmap of all images with ground-truth *truck* and Figure 3b depicts the heatmap computed with all *truck* images that are correctly classified by ResNet18. We can observe that both heatmaps are visually similar to each other in terms of the satisfaction probabilities of the concept predicates. We can observe from Figure 3b most relevant predicates have high satisfaction probabilities, i.e., the ones involving concepts relevant to *trucks*

[1]The resulting mapping has MSE of 1.128 and R^2 of 0.749, comparable with the mapping obtained for the original decomposition

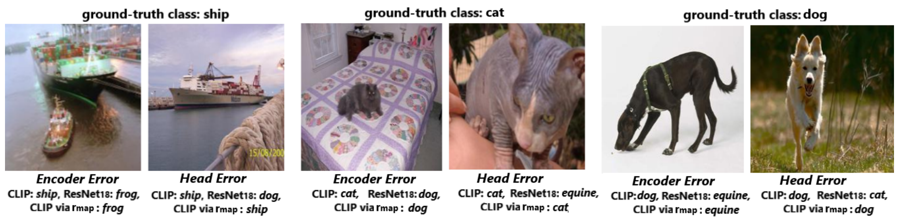

ground-truth class: ship
ground-truth class: cat
ground-truth class: dog
Encoder Error
CLIP: ship, ResNet18: frog, CLIP via r_{map} : frog
Head Error
CLIP: ship, ResNet18: dog, CLIP via r_{map} : ship
Encoder Error
CLIP: cat, ResNet18: dog, CLIP via r_{map} : dog
Head Error
CLIP: cat, ResNet18: equine, CLIP via r_{map} : cat
Encoder Error
CLIP: dog, ResNet18: equine, CLIP via r_{map} : equine
Head Error
CLIP: dog, ResNet18: cat, CLIP via r_{map} : dog

Fig. 4: Examples of misclassified inputs of ResNet18, error localized to the encoder (encoder error) and the head (head error).

(highlighted with red outlines), except for the predicate *metallic > tail*, and predicates involving concept *text*, e.g., *text> tail* and *text > horns*. This suggests that when ResNet18 correctly classifies images as *trucks*, it is indeed able to recognize the strong presence of concepts that are relevant for that class, such as *long, rectangular, tall, metallic* and *wheels* in these images. However the model struggles to effectively identify *text* in images, and distinguish *metallic* and *tails*. Additionally, we observe that some non-relevant predicates also have high satisfaction probabilities, e.g., *tail > beak*. This could be the result of other concepts in the images, and the dependency between relevant and non-relevant concepts. This analysis highlights how the respective heatmaps could be used to semantically explain and validate the behavior of the model.

We proceed to use the heatmaps to specifically investigate the misclassified inputs due to encoder and head errors to understand the reasons for failure semantically in terms of high-level concepts. There are 28 inputs with ground-truth *truck* that are mis-classified to other labels, out of which 22 are correctly classified by CLIP. Our localization technique categorizes 8 of these mis-classifications as encoder errors and 14 as head errors. Figures 3b, 3c, and 3d depict the ground-truth summary heatmaps for images correctly classified as truck, misclassified due to encoder errors, and to head errors, respectively. Figures 3e and 3f depict the differential heatmaps (between the heatmaps computed for correctly classified vs misclassified inputs) for the encoder and head errors, respectively. We found that the average difference value across all 324 strength predicates in Figure 3e was 0.115 (120 predicates with values > 0.115), while for Figure 3f the average difference value was smaller 0.077 (102 predicates with values > 0.077). Intuitively, this is because heatmaps summarize the satisfaction probability of concept predicates by the encoder's output. Thus, for Figure 3e, the error is in the encoder, so more predicates are violated, while for Figure 3f, the error is in the head, so fewer predicates are violated.

Focusing on the 72 predicates relevant to class *truck* (highlighted by the red boxes in the figures), we find that the differential heatmap for head errors shows zero or very small differences; only two predicates show a difference greater than 0.2, *wheels > wings* and *text > tail*. On the other hand, in the differential heatmap for encoder errors, many more predicates, namely *wheels > tail, wheels > long-snout, wheels > wings, wheels > patterned, text > horns, text > long-snout, text > wet, text > patterned, tall > long-snout, long > tail*, exhibit relatively large differences, ranging from 0.25 to 0.45. The results seem to indicate that the model is specifically faulty when processing images containing *wheels* or *text*, as these concepts frequently appear in the above predicates. Note also that some other predicates such as those comparing the concept *mane* with *beak*, *colored-eyes* and *ears* (which seemingly are irrelevant to class *truck*) also exhibit large values in the differential heatmaps, indicating that the model's output is potentially impacted by unintended features.

Isolation of the concepts and predicates potentially responsible for the failures can help with generating new inputs that satisfy the respective predicates for further debugging, repair and re-training of the model.

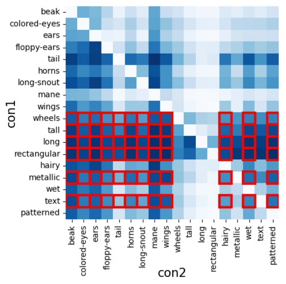
C. Debugging Vulnerabilities

To analyze the model with respect to adversarial examples, we generated adversarial images for every image from the RIVAL10 test set. We perturbed each image with Projected Gradient Descent (PGD) (perturbation radius $\epsilon = \frac{8}{255}$) [26]. We considered both l_∞ and l_2 PGD attacks.

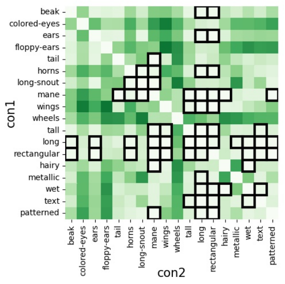
ResNet18 has 0% accuracy on all the images perturbed with the l_∞ PGD, i.e., it is not robust against the l_∞ PGD attack. We applied our localization technique to classify the errors as encoder vs head errors; 93.27% of all the errors were categorized as encoder errors. Figure 3a shows the ground-truth summary heatmap obtained with all original *truck* images without any perturbations, and Figure 6a shows the heatmap obtained with all *truck* images perturbed with the l_∞ PGD attack. The heatmaps indicate that the satisfaction of most strength predicates is affected by the perturbation. By visualizing the absolute difference between these two heatmaps (see Figure 6b, lighter green indicates a smaller difference), we can identify a small set of predicates that remained constant, i.e., with absolute difference ≤ 0.05, that are robust predicates, e.g., *rectangular > horns, long > ears*, and the rest are non-robust predicates, e.g., *wheels > tail, metallic > long-snout*. Notice that the majority of all relevant predicates for *truck* are not robust, suggesting non-robust features.

Considering the l_2 PGD attack, ResNet18 is robust—it has a 96.3% accuracy on perturbed images. In addition to high accuracy, we can validate using the heatmap obtained for all *truck* images perturbed with l_2 PGD attack (Figure 6c) as well as the differential heatmap between Figure 3a and Figure 6c that all relevant strength predicates are satisfied.

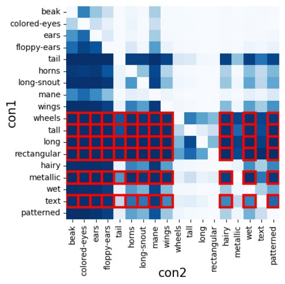
Analyzing model robustness through concepts, we can assign meaning to statistical results, allowing us to better



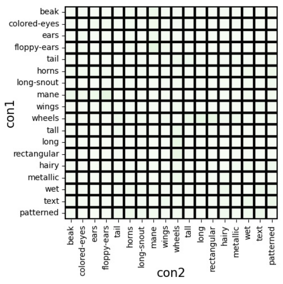
(a) Ground-truth heatmap for all perturbed images (PGD l_∞).



(b) Differential heatmap for absolute difference between Figures 3b and 6a.

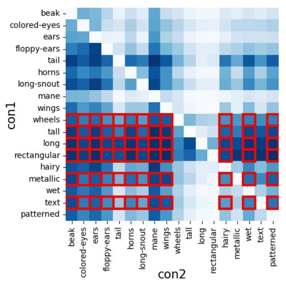


(c) Ground-truth heatmap for all perturbed images (PGD l_2).

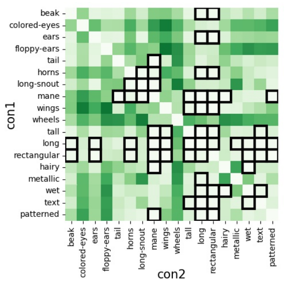


(d) Differential heatmap for absolute difference between Figures 3b and 6c.

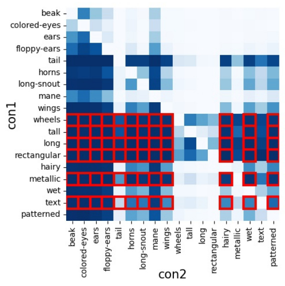
Fig. 5: Heatmaps for identifying robust and non-robust features against Projected Gradient Descent (PGD) attack (both l_∞ and l_2). All heatmaps consider images with ground truth *truck*. The black outline shows robust strength predicates where the absolute difference in satisfaction probability ≤ 0.05. The red outline indicates relevant strength predicates for class *truck*.



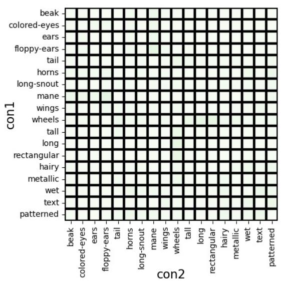
(a) Ground-truth heatmap for all perturbed images (PGD l_∞).



(b) Differential heatmap for absolute difference between Figures 3b and 6a.



(c) Ground-truth heatmap for all perturbed images (PGD l_2).



(d) Differential heatmap for absolute difference between Figures 3b and 6c.

Fig. 6: Heatmaps for identifying robust and non-robust features against Projected Gradient Descent (PGD) attack (both l_∞ and l_2). All heatmaps consider images with ground truth *truck*. The black outline shows robust strength predicates where the absolute difference in satisfaction probability ≤ 0.05. The red outline indicates relevant strength predicates for class *truck*.

understand the effects of the adversarial perturbations, and thus allowing developers to design more meaningful detectors and attacks. This experiment also highlights that most adversarial examples get misclassified due to encoder errors. While the encoder of ResNet18 is nearly accurate for in-distribution inputs, adversarial perturbations can lead to incorrect encoding as suggested by the non-robust features. Such insights can facilitate re-training or repair efforts by directing attention to the encoder and the specific non-robust strength predicates.

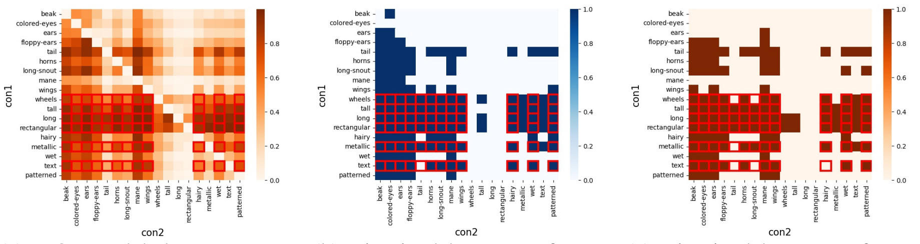
D. Runtime Defect Detection

So far, we have primarily relied on the ground-truth summary heatmaps for interpreting model behaviour. When ground truth information is unavailable, e.g., during runtime, we can utilize output-label summary heatmaps for analysis, enabling runtime detection of defects such as adversarial inputs and misclassifications.

1) Runtime Detection of Adversarial Inputs: As we observed before, adversarial perturbations lead to encoder errors. Therefore, with output-label summary heatmaps of a particular class, the relevant predicates would appear satisfied. For example, consider the PGD l_∞ attack ($\epsilon = \frac{8}{255}$); Figure 7a shows the summary heatmap for all perturbed images misclassified to *truck* by ResNet18. Most relevant predicates for *truck* are satisfied. However, since these images are in fact incorrectly mapped to *trucks* (from different ground-truth classes), the predicates that are not relevant to *truck* but that are relevant to other classes, show a more different and random satisfaction rate. Zooming in on this for adversarial detection, for each input, we identify such differences in its single-input heatmap.

To this end, we leverage a held-out dataset to construct, binarized ground-truth summary heatmaps and output summary heatmaps for clean and perturbed images classified to the same class by the model. Then, for each input encountered at runtime with output class c, it is considered adversarial if its single-input heatmap is more similar to the perturbed heatmap than the clean heatmap of class c, and clean otherwise. The similarity is computed using Intersection over Union (IoU).

We constructed sets of perturbed inputs ($\epsilon = \frac{8}{255}$) using common adversarial attacks such as PGD (both l_∞ and l_2) [26], FGSM (l_∞) [27], FAB (l_∞) [28], and a random mixture of all the above. One perturbed image was created for every input in the RIVAL10 test set. We used 75% clean test images and their perturbed versions to obtain the heatmaps offline, and



(a) Output-label summary heatmap for perturbed images classified as *truck*.

(b) Binarized heatmap of correctly classified clean images of *trucks* (binarized Figure 3b).

(c) Binarized heatmap of perturbed images classified as *truck* (binarized Figure 7a).

Fig. 7: Heatmaps used for runtime detection of adversarial inputs. Adversarial inputs generated with l_∞ PGD attack, $\epsilon = \frac{8}{255}$. Figure 7b and Figure 7c binarized with threshold 0.6. Red outlines indicate relevant strength predicates for class *truck*.

TABLE II: Runtime Defect Detection Accuracy: First five columns show results for adversarial inputs, each cell contains (a_p, a_n) where a_p is the percentage of perturbed images correctly detected as adversarial inputs ($IoU_p > IoU_n$), and a_n percentage of clean images correctly detected ($IoU_p \leq IoU_n$). ϵ is $\frac{8}{255}$ for all experiments. Last column shows results for misclassifications; (a_m, a_c), where a_m is the % misclassifications correctly detected and a_c is the % correct classifications not flagged as faulty.

	PGD l_∞	FAB l_∞	FGSM l_∞	PGD l_2	Mixture	Misclassifications
truck	(0.519,0.898)	(0.621,0.961)	(0.897,0.916)	(0.234,0.782)	(0.528,0.808)	(0.80 , 0.610)
car	(0.55,0.964)	(0.541,0.992)	(0.807,0.906)	(0.083,0.926)	(0.583,0.944)	(0.50 , 0.846)
plane	(0.656,0.884)	(0.595,0.97)	(0.745,0.96)	(0.032,0.973)	(0.735,0.927)	(1.0 , 0.921)
ship	(0.748,0.891)	(0.724,0.938)	(0.934,0.966)	(0.163,0.853)	(0.794,0.907)	(0.50 , 0.427)
cat	(0.954,0.447)	(0.76,0.94)	(0.684,0.914)	(0.052,0.956)	(0.756,0.54)	(1.0 , 0.938)
dog	(0.535,0.788)	(0.634,0.947)	(0.567,0.955)	(0.04,0.965)	(0.46,0.881)	(0.75 , 0.867)
equine	(0.808,0.624)	(0.815,0.917)	(0.858,0.925)	(0.159,0.887)	(0.705,0.827)	(0.50 , 0.077)
deer	(0.58,0.922)	(0.714,0.94)	(0.581,0.962)	(0.048,0.967)	(0.713,0.909)	(0.667 , 0.855)
frog	(0.721,0.798)	(0.595,0.932)	(0.692,0.965)	(0.123,0.886)	(0.38,0.957)	(0.667 , 0.718)
bird	(0.679,0.71)	(0.794,0.968)	(0.554,0.965)	(0.033,0.968)	(0.611,0.767)	(1.0 , 0.961)
Total	(0.678,0.792)	(0.683,0.95)	(0.74,0.943)	(0.097,0.917)	(0.631,0.847)	(0.163 , 0.951)

the rest 25% clean images and perturbed images are used for runtime evaluation. The threshold t should be sufficiently high to capture the differences between clean and perturbed heatmaps while avoiding the inclusion of predicates satisfied by only a small set. In our experiment, we set $t = 0.6$, which resulted in the highest detection accuracy. The binarized clean heatmap and the binarized perturbed heatmap for class *truck* are shown in Figure 7b and Figure 7c respectively.

In Tbl. II, in the first five columns we show the accuracy of identifying perturbed and clean images per class. Overall, we achieved high accuracy for most classes and most perturbations, showing the potential of our lightweight approach. The detection accuracy for PGD l_2 is low. As we discussed in Sec. V-C, ResNet18 is highly robust to PGD l_2 attacks, thus the perturbed and clean heatmaps are too similar to enable differentiating adversarial inputs.

2) Runtime Detection of Misclassified Inputs: The same method as outlined above can also be used to monitor for inputs that are not perturbed but may still be misclassified, due to model inaccuracies. The challenge is that the goundtruth is not known at runtime; we can leverage semantic heatmaps as follows. We use an offline dataset comprising of in-distribution data that are classified correctly and mis-classified by the ResNet18 model to construct the respective binarized output-label summary heatmaps. At runtime, the single input heatmap of every new input is compared with the correct and misclassified heatmaps to detect mis-classifications. The last column of Table II shows the results. We constructed the offline dataset with 75% of correctly-classified inputs and 75% of the mis-classified inputs from the RIVAL10 test dataset. A dataset comprising of the remaining 25% of correctly classified and misclassified inputs were employed at runtime to obtain the accuracy results. Note that since the in-distribution accuracy of ResNet18 is high, there were very few misclassifications compared to correct classifications in the offline dataset. This impacts the precision of the heatmaps, which in turn impacts the total detection accuracy. However, the misclassification detection accuracies for specific labels are high (such as 80% for truck and 75% for dog).

Note that VLMs such as CLIP can themselves be deployed as runtime detectors to catch inputs misclassified by the vision model. However, foundation models such as CLIP are too computationally heavy to be deployed during runtime, while our heatmaps can be precomputed, only the single-layer linear aligner r_{map} is required during runtime, which enables a more lightweight analysis.

VI. Threats to Validity

Our approach relies on the set of human-defined concepts which may be incorrect. In this paper, we use high-quality information about concepts from a previous study [17]. In the future, we plan to explore VLMs to *elicit* such concepts automatically. Note that our approach does not require images annotated with concepts.

We only demonstrated our approach on one (albeit complex) model and one dataset. Thus we can not claim that the results would generalize to other models and/or other datasets. More work is planned for the future that involves other image datasets and vision models. However, we note that the methodology and the analysis applications presented in this paper are generic and are not tied to a particular model or DNN architecture.

The choice of VLM is vital to our approach, as it limits us to inputs the VLM correctly classifies and perturbations to which the VLM is robust. During our case study, we noticed that the version of CLIP used in our experiments is not robust to natural perturbations such as brightness (with a robust accuracy of 14.8%). Additionally, our approach depends on the quality of r_{map}. To achieve an effective mapping, the dimension of the representation space of the vision model should be similar to that of the VLM. We leave exploring other VLMs to build semantic maps and developing other mapping techniques between the representation spaces for future work.

VII. Related Work

Concept-based Analysis of Neural Networks. Closest to our work are the ideas presented in [16] including the $\texttt{Con}_{\texttt{spec}}$ specification language with the notion of concept-based strength predicates. While [16] aims for formal verification of properties expressed $\texttt{Con}_{\texttt{spec}}$, we aim for statistical analysis of concept predicates, which in turn enables novel techniques for fault localization and runtime monitoring. The field of concept representation analysis in neural networks, as explored in [9, 14, 8, 15, 10], seeks to extract representations of concepts in the representation spaces of neural networks, and use these extracted representations to quantify the impact of high-level concepts on model output via attribution-based techniques. However these techniques usually require manual annotations for each image in a test set. There is also a growing body of work on extracting concept representations learned by language models in order to understand and control them [29, 30, 31, 32, 33, 34].

VLMs for Analyzing Vision Models. Recent work [11] first observed that the representation spaces of vision models and VLMs can be linked to each other via a linear map. They used this mapping for a number of use-cases—extending the vision model to identify new classes in a zero-shot fashion, diagnosing data distribution shifts in terms of human concepts, retrieving images satisfying a set of text-based constraints, and so on. However, they did not propose to leverage this mapping for the analysis of vision models as we do here. Other recent works [35, 36, 37] have proposed the use of VLMs to find a coherent, human-understandable description for inputs that are misclassified by a vision model. Given such a set of misclassified inputs, these techniques directly use the VLM to extract coherent textual descriptions that characterize such inputs. Unlike our semantic maps, these techniques do not seek to understand the internal behavior of vision models in a human-understandable fashion.

Debugging for Neural Networks. Techniques for fault localization have been proposed to localize errors in trained models [1, 2, 3] or in programs and libraries that are used to build the models [38, 39, 40]. Our work is closest to the former. One closely related work [41] also uses heatmaps to group inputs that have similar neuron profiles at inner layers. These heatmaps help identify portions of input or neurons that impact the output, but rely on visual interpretability to understand the results. In contrast, we use heatmaps in terms of concepts in the VLM embedding space. We see our work as complementary to other techniques. Our debugging and semantic heatmaps can help give a global view of the reason for failure, in terms of human-understandable concepts. Recently, model editing [42, 43] techniques have been proposed that perform causal analysis, by intervening on neural activations, to isolate neurons in language models that lead to specific buggy behaviors. These techniques can be expensive in practice and we view them as complementary to our approach.

Runtime Detection. Previous techniques for runtime detection of misclassifications, see e.g., [44, 45, 46, 47], are generally based on the idea of capturing a training profile (probability distributions, confidence values, or data preconditions) and measure differences for new inputs. In our case we leverage heatmaps to capture correct/incorrect execution profiles and measure semantic difference for new inputs. There is a large body of literature on detecting adversarial examples (see [48] for a survey). We take a more semantic approach, that discovers and leverages robust and non-robust features for detection.

VIII. Conclusion and Future Work

In this paper, we proposed novel techniques for debugging and runtime defect detection for vision models. The techniques leverage a separate VLM and *semantic heatmaps* that summarize statistical properties of DNNs using human-understandable concepts. We demonstrated the proposed techniques through a case study with a ResNet classifier and CLIP.

In future work, we aim to develop methods that enable more precise and accurate error localization to specific layers, possibly by learning more complex alignment maps between the model under analysis and the VLM. We also plan to examine the role of robust features in creating more effective adversarial attacks and explore new methods to enhance adversarial detection accuracy. Lastly, we plan to explore how to use our results for localized repair and also for testing of vision models, including test generation and data selection for retraining with violated strength predicates.

Acknowledgements

This work was partly supported by the US Air Force and DARPA under Contract No. FA8750-23-C-0519, and US ARL Cooperative Agreement W911NF-17-2-0196. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the Department of Defense, DARPA, or the US Government.

REFERENCES

[1] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "Mode: automated neural network model debugging via state differential analysis and input selection," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 175–186.

[2] H. F. Eniser, S. Gerasimou, and A. Sen, "Deepfault: Fault localization for deep neural networks," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2019, pp. 171–191.

[3] A. Ghanbari, D. Thomas, M. A. Arshad, and H. Rajan, "Mutation-based fault localization of deep neural networks," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023, pp. 1301–1313. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00171

[4] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: visualising image classification models and saliency maps," in *Proceedings of the International Conference on Learning Representations (ICLR)*. ICLR, 2014.

[5] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.

[6] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 336–359, 2020. [Online]. Available: https://doi.org/10.1007/s11263-019-01228-7

[7] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[8] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. sayres, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV)," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2668–2677. [Online]. Available: https://proceedings.mlr.press/v80/kim18d.html

[9] B. Zhou, Y. Sun, D. Bau, and A. Torralba, "Interpretable basis decomposition for visual explanation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[10] J. Crabbé and M. van der Schaar, "Concept activation regions: A generalized framework for concept-based explanations," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2590–2607, 2022.

[11] M. Moayeri, K. Rezaei, M. Sanjabi, and S. Feizi, "Text-to-concept (and back) via cross-model alignment," in *International Conference on Machine Learning*. PMLR, 2023, pp. 25 037–25 060.

[12] J. Crabbé and M. van der Schaar, "Concept activation regions: A generalized framework for concept-based explanations," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/11a7f429d75f9f8c6e9c630aeb6524b5-Abstract-Conference.html

[13] B. Kim, M. Wattenberg, J. Gilmer, C. J. Cai, J. Wexler, F. B. Viégas, and R. Sayres, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV)," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 2673–2682. [Online]. Available: http://proceedings.mlr.press/v80/kim18d.html

[14] A. Ghorbani, J. Wexler, J. Y. Zou, and B. Kim, "Towards automatic concept-based explanations," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 9273–9282.

[15] C. Yeh, B. Kim, and P. Ravikumar, "Human-centered concept explanations for neural networks," in *Neuro-Symbolic Artificial Intelligence: The State of the Art*, ser. Frontiers in Artificial Intelligence and Applications, P. Hitzler and M. K. Sarker, Eds. IOS Press, 2021, vol. 342, pp. 337–352. [Online]. Available: https://doi.org/10.3233/FAIA210362

[16] R. Mangal, N. Narodytska, D. Gopinath, B. C. Hu, A. Roy, S. Jha, and C. Pasareanu, "Concept-based analysis of neural networks via vision-language models," 2024.

[17] M. Moayeri, P. Pope, Y. Balaji, and S. Feizi, "A comprehensive study of image classification model sensitivity to foregrounds, backgrounds, and visual attributes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 19 087–19 097.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," arXiv: 1312.6199 [cs.CV], December 2013.

[21] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6706414

[22] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *CoRR*, vol. abs/1607.02533, 2016. [Online]. Available: http://arxiv.org/abs/1607.02533

[23] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," *Advances in neural information processing systems*, vol. 32, 2019.

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[26] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rJzIBfZAb

[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," *CoRR*, vol. abs/1312.6199, 2014.

[28] F. Croce and M. Hein, "Minimally distorted adversarial examples with a fast adaptive boundary attack," 2020.

[29] A. Bai, C.-K. Yeh, N. Y. Lin, P. K. Ravikumar, and C.-J. Hsieh, "Concept gradient: Concept-based interpretation without linear assumption," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=_01dDd3f78

[30] N. Nanda, A. Lee, and M. Wattenberg, "Emergent linear representations in world models of self-supervised sequence models," in *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, 2023, pp. 16–30.

[31] Z. Wang, L. Gui, J. Negrea, and V. Veitch, "Concept algebra for (score-based) text-controlled generative models," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[32] K. Park, Y. J. Choe, and V. Veitch, "The linear representation hypothesis and the geometry of large language models," in *Causal Representation Learning Workshop at NeurIPS 2023*, 2023.

[33] R. Huben, H. Cunningham, L. R. Smith, A. Ewart, and L. Sharkey, "Sparse autoencoders find highly interpretable features in language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=F76bwRSLeK

[34] A. Zou, L. Phan, S. Chen, J. Campbell, P. Guo, R. Ren, A. Pan, X. Yin, M. Mazeika, A.-K. Dombrowski, S. Goel, N. Li, M. J. Byun, Z. Wang, A. Mallen, S. Basart, S. Koyejo, D. Song, M. Fredrikson, J. Z. Kolter, and D. Hendrycks, "Representation engineering: A top-down approach to ai transparency," 2023.

[35] Y. Zhang, J. Z. HaoChen, S.-C. Huang, K.-C. Wang, J. Zou, and S. Yeung, "Diagnosing and rectifying vision models using language," in *The Eleventh International Conference on Learning Representations*, 2022.

[36] S. Eyuboglu, M. Varma, K. K. Saab, J.-B. Delbrouck, C. Lee-Messer, J. Dunnmon, J. Zou, and C. Re, "Domino: Discovering systematic errors with cross-modal embeddings," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=FPCMqjI0jXN

[37] S. Jain, H. Lawrence, A. Moitra, and A. Madry, "Distilling model failures as directions in latent space," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=99RpBVpLiX

[38] M. Wardat, W. Le, and H. Rajan, "Deeplocalize: Fault localization for deep neural networks," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 251–262. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00034

[39] M. Wardat, B. D. Cruz, W. Le, and H. Rajan, "Deepdiagnosis: Automatically diagnosing faults and recommending actionable fixes in deep learning programs," 2021. [Online]. Available: https://arxiv.org/abs/2112.04036

[40] A. Nikanjam, H. B. Braiek, M. M. Morovati, and F. Khomh, "Automatic fault detection for deep learning programs using graph transformations," vol. 31, no. 1, 2021. [Online]. Available: https://doi.org/10.1145/3470006

[41] H. M. Fahmy, F. Pastore, M. Bagherzadeh, and L. C. Briand, "Supporting deep neural network safety analysis and retraining through heatmap-based unsupervised learning," *IEEE Trans. Reliab.*, vol. 70, no. 4, pp. 1641–1657, 2021. [Online]. Available: https://doi.org/10.1109/TR.2021.3074750

[42] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in gpt," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 359–17 372, 2022.

[43] K. Meng, A. S. Sharma, A. J. Andonian, Y. Belinkov, and D. Bau, "Mass-editing memory in a transformer," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=MkbcAHIYgyS

[44] C. Corbière, N. THOME, A. Bar-Hen, M. Cord, and P. Pérez, "Addressing failure prediction by learning model confidence," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/757f843a169cc678064d9530d12a1881-Paper.pdf

[45] Y. Xiao, I. Beschastnikh, D. S. Rosenblum, C. Sun, S. G. Elbaum, Y. Lin, and J. S. Dong, "Self-checking deep neural networks in deployment," *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 372–384, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:230125000

[46] S. Ahmed, H. Gao, and H. Rajan, "Inferring data preconditions from deep learning models for trustworthy prediction in deployment," in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, 2024, pp. 440–452.

[47] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 359–371. [Online]. Available: https://doi.org/10.1145/3377811.3380353

[48] A. Aldahdooh, W. Hamidouche, S. A. Fezza, and O. Déforges, "Adversarial example detection for dnn models: A review and experimental comparison," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4403–4462, 2022.