Politecnico Di Torino

Department of Electronics and Telecommunications

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Path: Software Technologies for Automation

Reliability Analysis techniques for Deep Learning-Based Automotive Perception

Tesi di Laurea Magistrale in Ingegneria Meccatronica

Conducted At: Center for Automotive Research and Sustainable Mobility (CARS)

Author:
Alafid Munir(305860)

Supervisor:
Matteo Sonza Reorda
Nicola Amati
Josie Esteban Rodriguez Condia
Shailesh Sudhakara Hegde

March 16, 2026

Reliability Analysis techniques for Deep Learning-Based Automotive Perception

A. Munir

March 16, 2026

Abstract

Artificial intelligence (AI)–based perception systems have become a fundamental component of advanced driver assistance and automated driving platforms. These systems rely on deep neural networks (DNNs) to perform safety-critical tasks, such as object detection, drivable area segmentation, and lane line estimation. Unfortunately, several studies have shown that the modern generation of devices, with the latest semiconductor technologies, are specially vulnerable to premature faults in the hardware. Such physical defects arise as a product of two main time-dependent phenomena: *i)* temporal impacts, and *ii)* environmental and operational impact variations. The temporal impacts are related to effects arising from long-term operations, such as premature silicon aging and wear-out. Instead, the second effect arises from the system's exposure to harsh operative conditions (e.g., external radiation, electromagnetic fields, or high operational temperatures). Additionally, post-production test scapes might contribute to higher levels of fault susceptibility on a device.

In particular, when deployed on embedded hardware, neural networks may experience hardware faults that perturb stored weights without causing immediate system crashes. Instead, such perturbations can silently propagate through the network and degrade perception outputs in subtle yet safety-critical ways. A systematic strategy that considers the interaction between the application and the underlying hardware characteristics remains partially explored and requires structured evaluation frameworks.

This thesis introduces the **FIERA** framework (***F**ault **I**njection and **E**valuation for **R**obust **A**utomotive Perception*), a configurable tool designed to evaluate the impact of errors on AI-based perception models. FIERA uses a unified real-time framework to perform the analysis and evaluation of object detection and dual segmentation tasks within a single forward pass. The framework injects controlled perturbations (i.e., soft-errors) into convolutional weight tensors and compares faulty outputs against golden references using multiple evaluation metrics.

More in detail, FIERA can instrument a neural-network application with up to five state-of-the-art structured error models from hardware-aware fault patterns: Single-Point, Row, Column, Bullet-Wake, and Shattered-Glass. The primary perturbation model employs intensity-based controlled Gaussian noise to enable progressive degradation analysis. Furthermore, conventional strategies, such as bit-flip injection, are also included to represent discrete hardware-like corruption. The output degradation is quantified using task-specific metrics, including mean Intersection over Union (mIoU), Pearson correlation, segmentation area difference, centroid shift, detection consistency rate, confidence drop, and detection count variation. A unified usability classification rule is defined to determine whether a scenario produces functionally unacceptable perception output. A classical proportion-based sampling method is applied independently to each fault pattern to ensure statistically meaningful results within the large experimental universe.

For the experimental analysis and evaluation of the framework, YOLOP was used as a representative AI-based perception model to validate the proposed approach. In particular, for each sub-universe of 204,000 possible configurations, 385 scenarios are executed to achieve a 95% confidence level with a 5% margin of error. The results demonstrate that application vulnerability is strongly dependent on the corruption fault model and its geometry. Indeed, structured errors, particularly Row and Shattered-Glass patterns,

produce significantly higher corruptions than isolated single-point faults. Moreover, lane-line segmentation is observed to be the most sensitive perception output, while detection confidence exhibits comparatively higher robustness. The findings indicate that spatially structured weight corruptions primarily induce geometric and structural degradation rather than abrupt detection collapse. By combining structured fault modeling with statistical inference and usability-based evaluation, FIERA provides a systematic methodology for assessing the impact of hardware-like disturbances on automotive AI perception systems.

The proposed framework is intended to establish a reproducible foundation for robustness profiling and contribute to the development of reliability-aware validation strategies in safety-critical AI-powered applications.

To my father, A. Munir
The silent architect of my resilience
and the steady compass of my life.

To my mother
My soul's sanctuary, who cradled my failures
and ignited my greatest triumphs.

To my brother Hamza
My eternal heartbeat. Though you walk in shadows,
you remain the light that guides my every step.

To my siblings
The keepers of my history and the unbreakable
threads that hold my world together.

To my family
The soil in which I grew. Today, I bloom
in the warmth of your love.

Acknowledgments

Reaching the completion of this thesis marks an important moment in my academic journey. Looking back, I realize that this achievement is not the result of my efforts alone. Along the way, I have been supported, guided, and encouraged by many people whose presence made this journey meaningful and possible. I would like to take this opportunity to express my sincere gratitude to all those who contributed to my growth, both academically and personally.

I would first like to express my deepest gratitude to **Professor Josie E. Rodriguez Condia**, my supervisor, whose continuous guidance and support accompanied me throughout the entire development of this thesis. His willingness to share knowledge, his constructive feedback, and his patience during the different stages of this work were invaluable. His mentorship helped me navigate challenges, refine my ideas, and approach research with greater clarity and discipline.

I would also like to extend my appreciation to all the professors who were directly and indirectly involved in my education during my studies. Their dedication to teaching and their passion for knowledge played a fundamental role in shaping my academic foundation. In particular, I would like to thank the Head of the Mechatronics, **Professor Marcello Chiaberge**, whose leadership and commitment to academic excellence have contributed to creating an inspiring learning environment for students in the program.

My deepest gratitude goes to my family, whose support has been the cornerstone of my journey. I am profoundly thankful to my Father **M. Munir**, who supported me in every stage of life and instilled in me the resilience and determination needed to face challenges with strength. His belief in my abilities has always motivated me to move forward even during difficult moments. I am equally grateful to my **Mother**, who taught me the value of perseverance and patience. Her encouragement and unwavering faith in me have always reminded me that dedication and persistence are the keys to achieving one's goals. I would also like to thank my **Siblings**, whose encouragement and joy in every success I achieved gave me confidence and motivation. Their support and belief in me have always meant more than words can express.

I would like to express my heartfelt appreciation to my closest friends, **Lutfur Rehman, Zia, Syed Hamza, and Fatir**. Their friendship has been one of the most valuable aspects of my life. They have stood by me through different phases of this journey and have always offered encouragement, understanding, and genuine companionship. Over time they have become more than friends to me. They are like brothers, and their support has played an important role in helping me remain motivated and grounded throughout my studies.

A special mention goes to my dear friend **Harsh**, with whom I spent countless hours studying, discussing ideas, and preparing for exams. The long days and nights of study that we shared were filled not only with academic effort but also with moments of laughter and positive energy that made even the most demanding periods enjoyable. Those memories will always remain special to me.

I would also like to thank my other friends **Hamza Rehman, Basit, Afzal, Ankit, Ali, and Junaid**, with whom I spent many enjoyable moments outside academic life. The time we shared together brought balance to this journey and created memories that I will always cherish.

I would like to express my sincere appreciation to **Shah Jahaan Ali**, who was like a brother to me during my early days in Turin when the city was still unfamiliar and new. His support, guidance, and companionship made that transition much easier. He stood beside me in both the good and the challenging moments, and I truly value his presence in my life.

My sincere thanks also go to my friend and dorm roommate **Fabio**, who became one of my closest companions during this period. We spent many hours studying together and supporting each other through the challenges of university life. Beyond academics, he introduced me to the world of card games, where his remarkable ability to always seem to win never ceased to amaze me. I will always cherish the late night conversations, coffee breaks, and the many moments we shared together. His friendship made everyday student life more enjoyable and truly memorable.

I would also like to express my gratitude to my friends from Mexico: **Carla, Giovanna, Eleonora, and Alessia**, with whom I shared one of the most meaningful experiences of my life. The time spent with them during my exchange period was filled with unforgettable moments, cultural discoveries, studying together, and friendships that will remain close to my heart. That experience was a transformative chapter of my life that broadened my perspective and enriched me both personally and academically.

In this context, I would also like to thank the professors from the **Universidad de las Americas Puebla (UDLAP)** who welcomed me during my semester exchange mobility in Mexico. Their kindness, professionalism, and openness created a supportive academic environment that allowed me to learn, grow, and fully benefit from that international experience.

Finally, I would like to express a special appreciation to the city of **Torino**, which became much more than a place where I studied. Living here, far from my parents and my home country, was a defining experience that helped me grow into a more independent, resilient, and mature person. Torino became the place where I pursued my goals with determination and where many of the dreams that motivated me to leave home began to take shape. For that, I will always carry a deep sense of gratitude toward this city.

To everyone who has been part of this journey in any way, thank you for the encouragement, support, and memories that made this chapter of my life truly meaningful.

Contents

List of Figures

List of Tables

List of Acronyms

ACLH Accelerate and Lane Change
ADAS Advanced Driver Assistance Systems
AI Artificial Intelligence
BW Bullet Wake
COL Column Fault
CTD City Drive
CUTIN Cut-In
CUTOUT Cut-Out
DA Drivable Area
DE Object Detection
DL Deep Learning
DNN Deep Neural Network
DSA Domain-Specific Accelerator
FIERA Fault Injection and Evaluation for Robust Automotive Perception
GPU Graphics Processing Unit
HPC High Performance Computing
HSV Hue, Saturation, Value
IoU Intersection over Union
LCH Lane Change
LL Lane Line
mIoU Mean Intersection over Union
NPU Neural Processing Unit
OD Object Detection
SDC Silent Data Corruption
SG Shattered Glass
SLURM Simple Linux Utility for Resource Management
SP Single Point
TPU Tensor Processing Unit
VPU Vector Processing Unit
YOLOP You Only Look Once for Perception

Chapter 1

INTRODUCTION

1.1 Deep Learning in Automotive Perception

The transformation of the automotive industry over the past decade has been driven by the rapid integration of artificial intelligence into vehicle perception and decision-making systems [1], [2], [3], [4]. Indeed, *Advanced Driver Assistance Systems* (ADAS) and automated driving platforms increasingly rely on deep learning models to interpret complex road environments in real time [5], [6]. Tasks such as object detection, drivable area estimation, and lane boundary recognition are no longer implemented using handcrafted features or classical computer vision pipelines. Instead, they are executed by deep neural networks that learn hierarchical feature representations from large datasets. This transition has enabled remarkable improvements in perception accuracy and system responsiveness, making autonomous functionality feasible in increasingly diverse operational conditions. Particularly, such developments have been possible due to the combination of two factors: *i)* effective and efficient deployment algorithms and *ii)* the availability of suitable hardware platforms and devices [7]. However, while performance improvements have received significant attention, technical concerns regarding functional safety, reliability, and propagation susceptibility under hardware faults remain partially unexplored for such complex systems [8].

In particular, modern deep-learning-based perception systems are deployed on high-performance Domain-Specific Accelerators (DSAs) and embedded hardware accelerators, including *Graphics Processing Units* (GPUs), *Vector Processing Units* (VPUs), and dedicated *Neural Processing Units* (NPUs), such as *Tensor Processing Units* (TPUs) or custom accelerators [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. These platforms are designed to deliver real-time inference within strict power and thermal constraints. As semiconductor technologies scale down to nanometer regimes, hardware components become more susceptible to transient physical defects, commonly referred to as soft errors [8], [19], [20], [21], [22], [23], [24]. In fact, soft errors may arise from radiation induced charge fluctuations, voltage instability, aging phenomena, or manufacturing variability. Unlike permanent faults, soft errors do not cause lasting hardware damage. Instead, they may temporarily corrupt stored values or intermediate computations.

In safety-critical domains, such as automotive perception, even transient corruptions may propagate through a neural network and influence the final output. The difficulty lies in the nonlinear and highly interconnected structure of deep models, where small numerical perturbations may either be masked internally or amplified through successive layers.

1.2 Reliability Challenges in Deep Neural Networks

Traditional reliability analysis in automotive systems has historically focused on deterministic control algorithms and hardware-level fault tolerance [8], [19], [25], [26], [27], [28]. Standards and safety guidelines emphasize functional safety, redundancy, and failure detection mechanisms for classical embedded software. However, deep neural networks introduce a fundamentally different paradigm. Indeed, their internal behavior is governed by millions of learned parameters stored in memory. These parameters are continuously accessed during inference and directly influence every computation in the network. A corruption in a single weight tensor may subtly alter feature extraction and ultimately modify object detection confidence, segmentation boundaries, or lane geometry estimation. Since neural networks are data-driven and nonlinear, predicting the impact of perturbations and corruptions analytically is extremely challenging [29], [30].
The automotive sector imposes strict requirements for dependability and robustness. ISO 26262 and related safety standards mandate systematic analysis of potential failure modes and their consequences [31]. At the same time, the Safety of the Intended Functionality standard addresses hazards arising from functional insufficiency and performance limitations rather than explicit hardware faults [32]. In fact, Deep learning based perception systems intersect both domains. They are vulnerable not only to environmental uncertainties but also to hardware-induced disturbances that may degrade their internal numerical stability. Consequently, evaluating reliability in this context requires methodologies that bridge hardware fault modeling and application-level performance assessment.

1.3 Limitations of Existing Reliability Studies

Existing literature has investigated soft error effects in digital circuits and microprocessors for decades [33, 34, 35, 36, 37, 38, 39, 40, 41]. With the growing deployment of neural networks, research has begun to examine how bit flips and numerical perturbations affect inference accuracy [30, 42, 43, 44, 45, 46]. Many studies focus on classification networks trained on benchmark datasets, where output correctness is measured in terms of top-one accuracy. In [47], [48], the authors analyze the effects of faults on an AI-powered perception application using closed-loop analysis. Unfortunately, the evaluation was limited to a few perception applications (mostly line-lane detection) in a mono-modal fashion. While such investigations provide insight into model sensitivity, they do not directly address the multi-task and spatially structured nature of automotive perception. In a driving context, degradation cannot be evaluated solely through classification labels. Instead, one must consider object detection stability, segmentation overlap, geometric displacement, and structural similarity of predicted masks. Moreover, automotive perception networks often integrate multiple tasks within a single architecture, meaning that a fault may affect object detection and lane detection differently.
Another limitation of current research lies in the absence of unified experimental infrastructures. Fault injection studies vary across the selected abstraction and method to inject faults, errors, or failures into a system, from bitflips on logic up to corruptions on interconnections or communications at system levels [8], [49], [50], [51], [52]. Furthermore, such frameworks are frequently performed using ad hoc instrumentation or are limited to specific technologies and platforms. In addition, some instruments can perturb (inject fault effects or errors) on selected parameters of a system or workload without systematic

sampling or statistically justified stopping criteria.
In some other cases, the results are often reported for limited sets of layers or intensities, making it difficult to generalize conclusions. In particular, on large-scale perception models containing hundreds of convolutional tensors, the experimental space becomes enormous. Each tensor may be perturbed at multiple intensity levels and random seeds, resulting in tens of thousands of potential scenarios, e.g., evaluations on simplified versions of a workload near the hardware might involve more than 10,000 days! [53]. Thus, exhaustively exploring this fault space universe is computationally infeasible, especially when inference must be performed over video sequences. Therefore, a statistically grounded sampling strategy is essential to obtain representative estimates of failure probability while maintaining practical feasibility.

1.4 Problem Statement and Motivation

The central problem addressed in this thesis arises from two main issues: **1)** current AI-powered applications are data-intensive, so conventional hardware-aware strategies for their analysis and evaluation of reliability can hardly be applied due to computational power and performance needs, and **2)** the characterization of hardware-aware error effects on multi-modal AI-powered perception applications might require specialized frameworks and level abstractions to automate and speed up the process.
While deep learning based automotive perception systems are increasingly deployed in safety-critical environments, there is a lack of systematic hardware-aware reliability analysis techniques that evaluate the impact of errors at the layer's tensor level. Moreover, quantifying perception degradation requires application-specific metrics and yields statistically interpretable results. Without such a framework, it is difficult to determine which internal components are most vulnerable, how often a system becomes unusable under representative disturbances, and whether the evaluation campaign has achieved sufficient statistical precision.
The motivation for this work is therefore rooted in the need to move beyond accuracy-centric evaluation toward reliability-centric assessment. Performance metrics under nominal conditions do not guarantee resilience against hardware disturbances. A perception model that achieves high accuracy on validation datasets may still exhibit significant vulnerability when its internal parameters are perturbed. For automotive applications, understanding this vulnerability is crucial. If certain convolutional layers are disproportionately sensitive, targeted hardening strategies may be developed. If particular fault patterns consistently lead to unusable outputs, system-level mitigation mechanisms may be prioritized accordingly. Ultimately, reliability analysis contributes to the safer deployment of autonomous driving technologies.

1.5 Introduction of the FIERA Framework

To address these challenges, this thesis introduces FIERA, a structured framework for Fault Injection and Evaluation for Robust Automotive Perception. FIERA is implemented through a set of Python-based tools that perform inference while injecting controlled faults into the neural network weights. The framework generates both quantitative outputs, in the form of performance metrics, and qualitative outputs through video-based

analysis, thereby enabling the systematic evaluation of the impact of injected faults on the perception system.
The complete codebase, fault injection pipelines, and experimental data developed for this research are publicly available online.[1]
The framework is designed to inject controlled perturbations into the trained weight tensors of a realistic multi-task perception network and evaluate the resulting degradation using both qualitative and quantitative methods. Rather than modifying the network architecture or retraining under noise, the framework focuses on post-training reliability evaluation. This perspective aligns with real-world deployment, where trained models are stored in memory and executed repeatedly under varying hardware conditions.
In addition, the choice of fault model is guided by the need for both realism and controllability. Gaussian distributed perturbations are adopted as the primary numerical disturbance model. While not a direct physical replication of transistor-level behavior, Gaussian noise provides a statistically consistent abstraction of aggregate numerical deviations. By scaling noise magnitude relative to the standard deviation of the original weights, perturbations remain proportional to the inherent numerical distribution of each tensor. In addition to Gaussian perturbations, bit-level modifications are supported as a complementary injection mode, enabling comparison between continuous and discrete disturbance models. In particular, five hardware-aware structured fault patterns are implemented to explore different spatial organizations of corruption within reshaped convolutional tensors. These patterns emulate isolated, row-oriented, column-oriented, and clustered disturbances, allowing investigation of how spatial structure influences propagation.

1.6 Statistical Methodology and Reliability Interpretation

In addition, this work adopts a statistical methodology to reduce the computational power and performance required to evaluate complete workloads. Rather than relying on iterative stopping rules tailored to simple Bernoulli experiments [42], a classical sample-size determination approach is employed to determine the number of scenarios required for each fault pattern. By selecting a confidence level of ninety-five percent and a conservative margin of error, the campaign ensures that estimated failure probabilities are statistically interpretable. This approach enables the definition of a clear stopping criterion once the measured margin of error falls below the target threshold. As a result, conclusions drawn from the experiments are supported by quantitative and representative trends.
Reliability in deep learning perception must also be interpreted through the lens of resilience. Indeed, resilience describes the ability of a system to maintain an acceptable level of functionality despite the presence of disturbances, faults, or unexpected perturbations [54]. For segmentation outputs, acceptable functionality may be defined in terms of minimum overlap with nominal masks and bounded geometric displacement. For object detection, resilience implies stable detection counts, consistent bounding box localization, and limited confidence degradation. By defining task-specific usability thresholds, the framework transforms continuous metric deviations into binary usability outcomes.

[1] Henceforth, every time the author refers to the implementation or the codebase, it implies the resources provided in the author's public repository, available at `https://github.com/alafidmunir94-collab/Thesis_Reliability-of-DL-Based-Automotive-Perception`.

This transformation enables estimation of the overall unusable probability and facilitates tensor level vulnerability ranking.
The broader significance of this thesis lies in bridging hardware reliability concepts with application-level perception evaluation. By focusing on weight-tensor perturbations and examining their impact on multi-task outputs, the work provides insight into how low-level disturbances manifest at the system level. It demonstrates that reliability analysis for deep learning cannot remain confined to hardware abstraction or simple classification accuracy metrics. Instead, it requires integrated frameworks that consider model structure, task-specific evaluation criteria, and statistically justified experimentation.
The remainder of this thesis is organized as follows. Chapter 2 provides the background necessary to understand the study, including an overview of deep learning based automotive perception, the YOLOP architecture, the organization of its weight tensors, and the fault models considered in this work. Chapter 3 presents the proposed FIERA framework and describes the fault injection mechanism together with the inference and evaluation pipelines used to analyze the impact of faults. Chapter 4 explains the experimental methodology, including scenario generation, evaluation metrics, statistical analysis procedures, and the high performance computing execution strategy used for large scale experiments. Chapter 5 presents the experimental results and discusses the reliability implications of the observed faults. Chapter 6 describes the Future Work. Finally, Chapter 7 concludes the thesis and outlines possible directions for future work.

Chapter 2

BACKGROUND

2.1 Deep Learning Based Automotive Perception

The development of automated driving systems has led to the integration of deep learning models as the core component of perception stacks. Modern perception systems must simultaneously detect surrounding vehicles and pedestrians, estimate drivable regions, and extract lane boundaries. These tasks require high spatial resolution, temporal consistency, and robust generalization under diverse environmental conditions. Multi-task learning architectures have emerged as an efficient solution by sharing feature extraction layers across several perception objectives. Such architectures reduce computational redundancy and enable joint optimization of correlated tasks.
Among the architectures designed for automotive perception, YOLOP is a representative example of an end-to-end multi-task model capable of performing object detection, drivable area segmentation, and lane line detection within a unified neural network [55]. The choice of YOLOP in this thesis is motivated by its realistic complexity and deployment relevance. This perception benchmark model is designed to operate in real-time scenarios. Therefore, it provides an appropriate basis for reliability and fault propagation analysis. The next subsection describes the YOLOP architecture, the underlying implementation platform, and the fault models used in this work.
A general perception pipeline for automated driving is illustrated in Fig. 2.1. The system begins with multiple sensors, such as cameras, radar, and LiDAR, which capture raw data from the surrounding environment. This data is first processed and filtered, and then combined in a sensor-fusion stage to produce a unified representation of the scene. The perception module uses deep learning models to extract meaningful information from this representation. Typical perception tasks include object detection (identifying vehicles, pedestrians, and other traffic participants), drivable area segmentation (classifying road regions that can be safely traversed), and lane line detection (identifying lane boundaries and road markings). These outputs provide a structured representation of the environment, which higher-level modules, such as planning and control, use to generate safe driving decisions.

2.2 YOLOP Architecture

YOLOP, which stands for You Only Look Once for Perception, extends the single stage object detection philosophy introduced in the original YOLO family of model [56, 57]

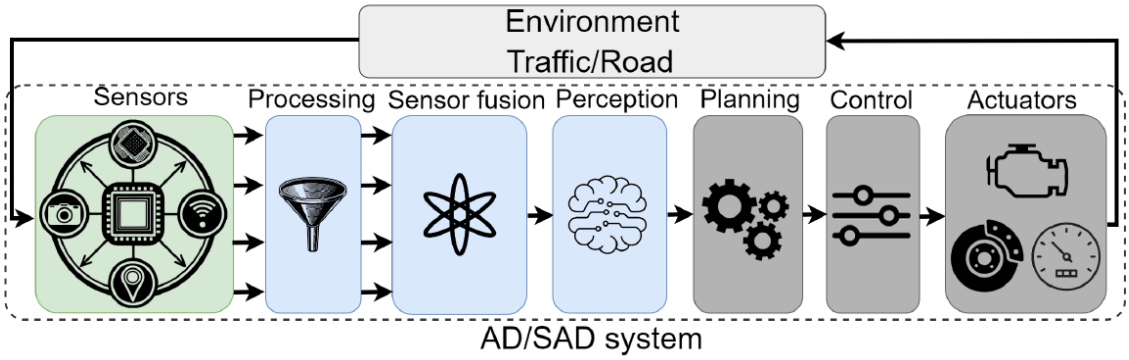


Figure 2.1
A general scheme of an AI-Based Perception system in automotive. Adapted from [48].

toward a multi task automotive perception context. The architecture integrates three perception tasks into a single inference pipeline:

- Object Detection (DE)
- Drivable Area Segmentation (DA)
- Lane Line Detection (LL)

Unlike traditional perception stacks where each task is processed by a separate neural network, YOLOP employs a shared backbone for feature extraction followed by task specific branches. This design improves computational efficiency and enables cross task feature reuse.

2.2.1 Backbone

The backbone is responsible for extracting hierarchical visual features from the input image. It consists of multiple convolutional layers that progressively reduce spatial resolution while increasing feature abstraction. Early layers capture low-level features such as edges and textures, whereas deeper layers represent high-level semantic information.

From a reliability perspective, backbone layers are particularly important because faults injected at early stages may propagate to all downstream tasks. Since the backbone feeds into every head, perturbations in these tensors have the potential to influence detection and segmentation simultaneously.

2.2.2 Neck

The neck stage performs multi-scale feature fusion. It aggregates information from different resolution levels to balance spatial precision and contextual awareness. This stage is critical for object detection performance, especially for small or distant objects.

Feature fusion layers typically contain convolutional weight tensors that combine spatial and channel information. Because these layers integrate features from multiple sources, they may exhibit different sensitivity characteristics compared to purely sequential convolutional blocks.

2.2.3 Task Specific Heads

The architecture branches into three heads:

- The object detection head predicts bounding boxes, class probabilities, and confidence scores.
- The drivable area head produces a pixel-wise segmentation mask indicating safe road regions.
- The lane line head outputs a segmentation map representing lane markings.

Each head contains convolutional weight tensors that specialize the shared features for task-specific outputs. Fault injection into head tensors may produce localized degradation limited to a single task, whereas backbone perturbations may have a multi-task impact. As shown in Figure 2.2.

The multi task structure of YOLOP makes it particularly suitable for studying cross task fault propagation and structured degradation patterns.

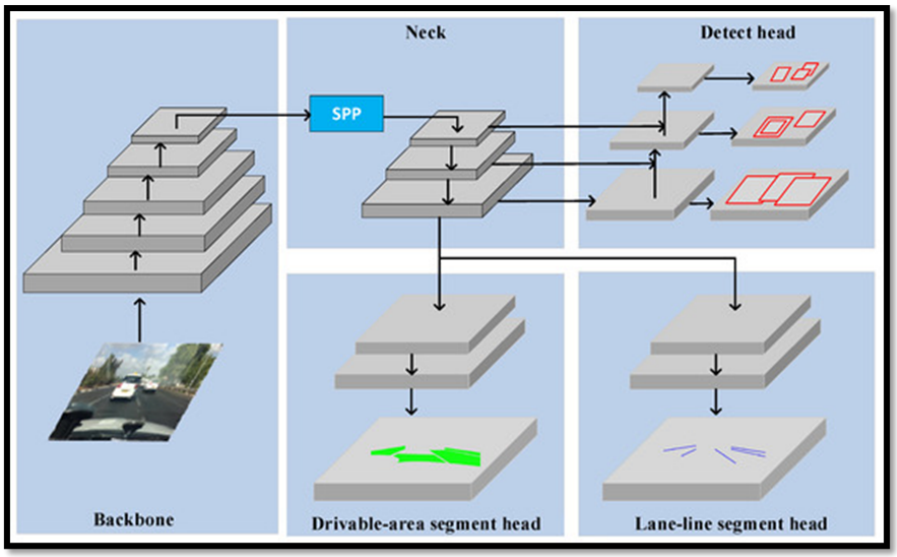


Figure 2.2
A general architectural scheme of YOLOP.

2.3 Role of YOLOP in This Thesis

YOLOP is adopted in this thesis as the **reference perception** model on which the proposed fault injection and evaluation framework is developed and validated. As a state-of-the-art multi-task automotive perception network, YOLOP performs object detection, drivable area segmentation, and lane line detection within a unified architecture. These characteristics make it representative of modern perception stacks used in automated driving systems.
The use of YOLOP in this work provides a realistic and complex benchmark for evaluating the effects of hardware-induced faults on deep learning–based perception. By operating on a multi-task perception architecture with shared feature representations, YOLOP allows the study of how low-level perturbations can propagate through different perception tasks and affect the overall understanding of the driving scene.

The main contribution of this thesis is the design and implementation of a custom fault injection and analysis framework targeting internal neural network parameters and tensors within the YOLOP model. Controlled faults are injected into selected components of the network to emulate hardware-induced perturbations such as transient bit corruptions. The resulting impact on inference behavior is then systematically analyzed.

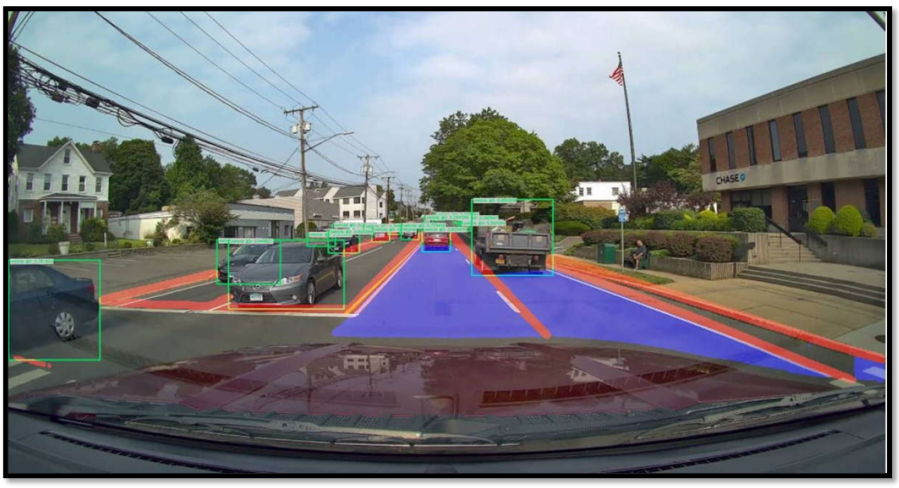


Figure 2.3
A general view of the YOLOP multi-modal inference operation on the frame of a video.

The objective of the fault injection campaign is to evaluate the sensitivity and vulnerability of the perception model to faults affecting its internal parameters. By systematically injecting faults and observing their effects, the framework enables:

- Identification of the most vulnerable components within the perception network.
- Analysis of how faults propagate across different perception tasks.
- Quantification of performance degradation under faulty conditions.

This evaluation provides the foundation for a controlled, reproducible, and statistically meaningful analysis of reliability in deep learning–based automotive perception systems, which is further explored in the subsequent chapters of this work.

2.4 YOLOP Weight File

The YOLOP model used throughout this thesis relies on a **pre-trained weight file, referred to as End-to-end.pth**. This file contains the complete set of learned parameters resulting from the end-to-end training of the YOLOP architecture, including the shared backbone, neck, and all task-specific heads for object detection, drivable area segmentation, and lane line detection.

The weight file represents the functional state of the perception system and encodes all the knowledge acquired during training. During inference, these parameters are continuously accessed by the hardware accelerator and directly influence the numerical computations performed in each layer. As a result, the integrity of this weight file is critical to the correctness and stability of the perception outputs.

In the context of this thesis, End-to-end.pth serves as the **golden reference model**, representing fault-free operation. All fault injection experiments are conducted by introducing controlled perturbations to selected parameters derived from this file, while preserving the original weights for comparison and repeatability. The structure and naming of tensors within the weight file also enable precise targeting of specific layers and components of the YOLOP architecture as shown in 2.4.

By consistently using the same pre-trained weight file across all experiments, this work ensures:

- Reproducibility of results,
- Fair comparison between fault-free and faulty executions,
- Isolation of fault effects from training-related variability.

For the remainder of this thesis, End-to-end.pth is therefore treated as the **baseline perception model**, and all evaluations of fault impact, robustness, and degradation are performed relative to its nominal behavior.

2.5 Tensor Organization within the YOLOP Weight File

The YOLOP weight file, End-to-end.pth, stores the trained parameters of the network in the form of tensors, which are multi-dimensional numerical arrays used by the neural network during inference. Each tensor corresponds to a specific functional element of the model, such as a convolutional filter, a normalization parameter, or a bias term.

From a structural point of view, tensors in the weight file can be broadly categorized based on their role in computation.

- **Convolutional weight tensors**, typically four-dimensional, which encode spatial filters applied to feature maps,
- **Batch normalization tensors**, which include scale, bias, and running statistics used to stabilize activations,

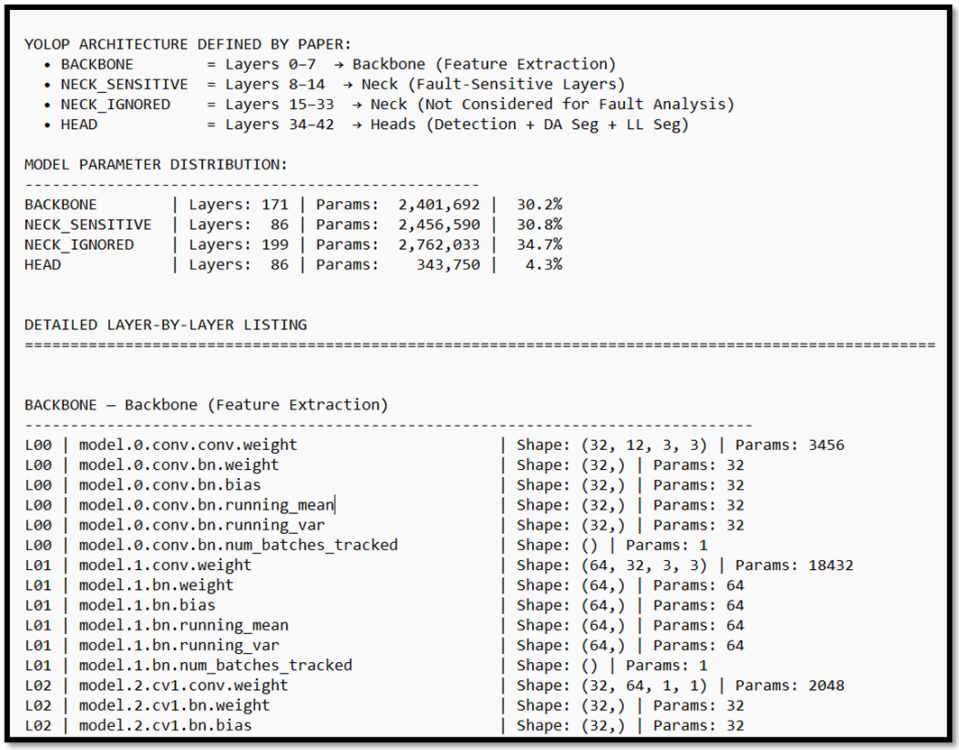
- **Bias and auxiliary tensors**, which provide additive or structural parameters for specific layers.

These tensors are organized hierarchically and are uniquely identified by descriptive names that reflect their position within the YOLOP architecture. This naming structure enables precise and reproducible access to individual tensors during inference, which is essential for controlled fault injection experiments.

During execution, tensor values are repeatedly read from memory and processed by the underlying hardware accelerator. As a result, they represent **a realistic abstraction of hardware-visible data**, making them particularly relevant for modeling the effects of transient hardware faults such as soft errors. A corruption affecting even a small subset of tensor elements can influence intermediate feature representations and propagate through subsequent layers, potentially altering perception outputs.

In this thesis, tensors are treated as the **primary unit of fault injection and analysis**. Faults are introduced by selectively perturbing tensor values during inference, allowing the study of localized and structured error patterns without modifying the network architecture or training process. This tensor-level perspective provides a flexible and fine-grained mechanism for evaluating how hardware-induced disturbances manifest at the level of AI-based perception.

To support clarity and reproducibility, a visual representation of the tensor structure within the YOLOP weight file is provided in this thesis. This illustration highlights how tensors are shaped figure 2.4, indexed, and associated with different architectural components, serving as a conceptual bridge between abstract neural network parameters and their concrete implementation in memory.
The figure 2.4 illustrates the internal structure of the YOLOP weight file and the distribution of its learned parameters (Indicates how many numerical values are contained in the tensor.) across the main architectural components of the network, including the backbone, neck, and task-specific heads. Each entry represents a parameter tensor stored in the .pth file, specifying its name, dimensional shape, and number of parameters. These tensors correspond to convolution weights, normalization parameters, and other trainable values learned during training. In this research, these tensors serve as the target elements for fault injection experiments to analyze how parameter perturbations affect the perception outputs.



```
YOLOP ARCHITECTURE DEFINED BY PAPER:
  • BACKBONE        = Layers 0–7  → Backbone (Feature Extraction)
  • NECK_SENSITIVE  = Layers 8–14  → Neck (Fault-Sensitive Layers)
  • NECK_IGNORED    = Layers 15–33  → Neck (Not Considered for Fault Analysis)
  • HEAD            = Layers 34–42  → Heads (Detection + DA Seg + LL Seg)

MODEL PARAMETER DISTRIBUTION:
--------------------------------------------------
BACKBONE        | Layers: 171 | Params:  2,401,692 |  30.2%
NECK_SENSITIVE  | Layers:  86 | Params:  2,456,590 |  30.8%
NECK_IGNORED    | Layers: 199 | Params:  2,762,033 |  34.7%
HEAD            | Layers:  86 | Params:    343,750 |   4.3%


DETAILED LAYER-BY-LAYER LISTING
================================================================================================


BACKBONE – Backbone (Feature Extraction)
------------------------------------------------------------------------------
L00 | model.0.conv.conv.weight                    | Shape: (32, 12, 3, 3) | Params: 3456
L00 | model.0.conv.bn.weight                      | Shape: (32,) | Params: 32
L00 | model.0.conv.bn.bias                        | Shape: (32,) | Params: 32
L00 | model.0.conv.bn.running_mean                | Shape: (32,) | Params: 32
L00 | model.0.conv.bn.running_var                 | Shape: (32,) | Params: 32
L00 | model.0.conv.bn.num_batches_tracked         | Shape: () | Params: 1
L01 | model.1.conv.weight                         | Shape: (64, 32, 3, 3) | Params: 18432
L01 | model.1.bn.weight                           | Shape: (64,) | Params: 64
L01 | model.1.bn.bias                             | Shape: (64,) | Params: 64
L01 | model.1.bn.running_mean                     | Shape: (64,) | Params: 64
L01 | model.1.bn.running_var                      | Shape: (64,) | Params: 64
L01 | model.1.bn.num_batches_tracked              | Shape: () | Params: 1
L02 | model.2.cv1.conv.weight                     | Shape: (32, 64, 1, 1) | Params: 2048
L02 | model.2.cv1.bn.weight                       | Shape: (32,) | Params: 32
L02 | model.2.cv1.bn.bias                         | Shape: (32,) | Params: 32
```

Figure 2.4
Weight File Contents in YoloP.

2.5.0.1 Tensor Structure Interpretation

Original Tensor Shape
Consider the convolutional weight tensor used as an example in this work:
(out_channels, in_channels, kernel_height, kernel_width) = (32, 12, 3, 3)
This tensor contains:

- 32 output channels (filters),
- 12 input channels per filter,
- each input channel associated with a spatial kernel 3×3.

2.5.1 Internal Two-Dimensional Tensor View

It is important to describe how the convolutional tensors of the YOLOP model are organized. In the original model, convolutional weights are stored as four-dimensional tensors with the structure:
(out_channels, in_channels, kernel_height, kernel_width)
which then reshapes the tensor into a two-dimensional form: $(32, 12 \times 3 \times 3) = (32, 108)$
This transformation is central to how faults are applied.
Interpretation:
Rows (32) Each row corresponds to one output channel, i.e., one complete convolutional filter. They are interpreted as Row 0 to Row 31.

Columns (108) Each column corresponds to one scalar weight value, obtained by flattening all input channels and spatial kernel positions. They are interpreted as Column 0 to Column 107.

Important:
Although the original tensor has 12 input channels, the framework does not treat "12" as columns, Instead, each input channel contributes 9 columns (from the 3×3 kernel), resulting in 108 columns total.

2.5.2 Conceptual Visual Representation

Original tensor (conceptual):

Filter 0:	Filter 1:	Filter 2:	Filter 31:
Channel 0 → 3 × 3	Channel 0 → 3 × 3	Channel 0 → 3 × 3	Channel 0 → 3 × 3
Channel 1 → 3 × 3	Channel 1 → 3 × 3	Channel 1 → 3 × 3	Channel 1 → 3 × 3
...	...	...	...
Channel 11 → 3 × 3	Channel 11 → 3 × 3	Channel 11 → 3 × 3	Channel 11 → 3 × 3

Table 2.1
A general scheme of the original Tensor Shape.

Reshaped 2D tensor used for fault injection:

Row 0 →	$[W_{0,0} \quad W_{0,1} \quad \dots \quad W_{0,107}]$
Row 1 →	$[W_{1,0} \quad W_{1,1} \quad \dots \quad W_{1,107}]$
...	...
Row 31 →	$[W_{31,0} \quad W_{31,1} \quad \dots \quad W_{31,107}]$

Each row is a flattened version of all $(12 \times 3 \times 3)$ weights of one output filter.The Figure (2.5) Shows the Original Tensor Shape. Also show in table 2.1

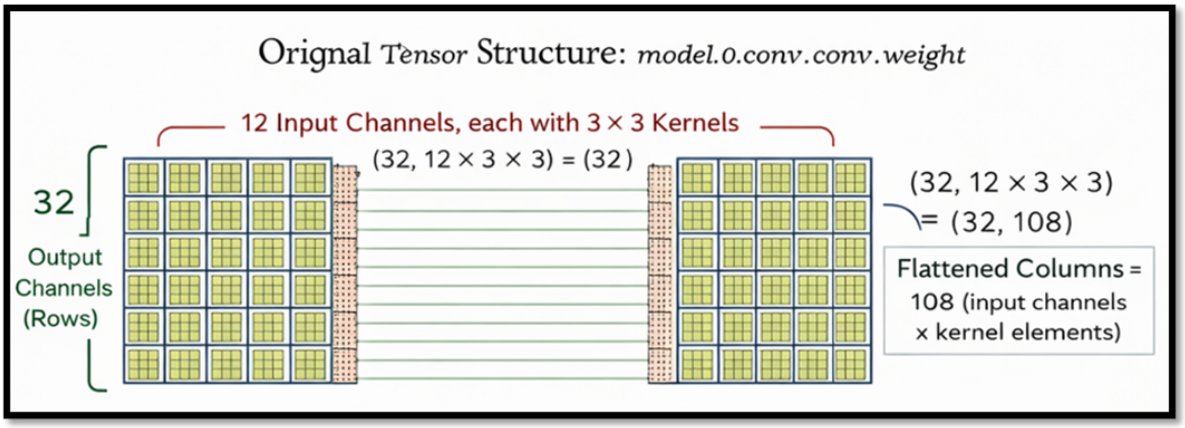


Figure 2.5
A general scheme of the original Tensor Shape.

2.5.3 Linearized Tensor View

For indexing and memory access, the two-dimensional tensor can be viewed linearly as:

$$\begin{bmatrix} Row_0\text{–}Col_0 & Row_0\text{–}Col_1 & \cdots & Row_0\text{–}Col_{107} \\ Row_1\text{–}Col_0 & Row_1\text{–}Col_1 & \cdots & Row_1\text{–}Col_{107} \\ \vdots & \vdots & \ddots & \vdots \\ Row_{31}\text{–}Col_0 & Row_{31}\text{–}Col_1 & \cdots & Row_{31}\text{–}Col_{107} \end{bmatrix}$$

This linear view explains how faults targeting contiguous indices may affect multiple spatial or channel-related weights.

2.6 PyTorch

The implementation of YOLOP used in this thesis is based on PyTorch, an open source deep learning framework widely adopted in both research and industrial applications [58]. PyTorch provides automatic differentiation, dynamic computation graphs, and efficient tensor operations optimized for both CPU and GPU execution. These features make it suitable for developing and evaluating deep learning based perception systems.

PyTorch represents neural network models as collections of tensors and computational modules that can be easily inspected and manipulated. All learned model parameters are stored within a *state dictionary*, where each parameter is represented as a multi dimensional tensor. This structure enables direct programmatic access to the internal weights of the network.

From an experimental analysis perspective, PyTorch also facilitates controlled inference execution. By disabling gradient computation and configuring deterministic execution settings, experiments can be performed in a reproducible and stable manner.

Furthermore, PyTorch provides flexible tensor operations that allow individual weight values or groups of parameters to be accessed and analyzed without modifying the network architecture. This capability makes the framework particularly suitable for studying the internal behavior of deep neural networks used in automotive perception systems.

2.7 Error Models and Fault Injection Patterns

To analyze reliability, it is necessary to define fault models that approximate hardware induced disturbances. In this thesis, two perturbation mechanisms are considered:

- **Gaussian** distributed numerical perturbations
- **Bit-level** modifications

While Gaussian perturbations constitute the primary research model, bit flips are included as a complementary mechanism. Both approaches target weight tensors during inference.

Beyond the perturbation mechanism itself, the spatial organization of faults within tensors plays a crucial role. Five structured fault patterns are implemented to emulate different corruption footprints [55]. These patterns are defined after reshaping convolutional weight tensors into a two-dimensional representation for indexing purposes.

2.7.1 Single Point Fault

The Single Point Fault [55] represents the most localized disturbance. In this pattern, only one scalar weight value within a tensor is perturbed. All other elements remain unchanged. The depiction of Single Point Fault is shown in Figure 2.6.

This pattern models isolated soft errors affecting a single memory cell. Its impact depends heavily on the importance of the specific weight location. In many cases, the effect may be masked by surrounding computations, but in sensitive regions it may propagate significantly.

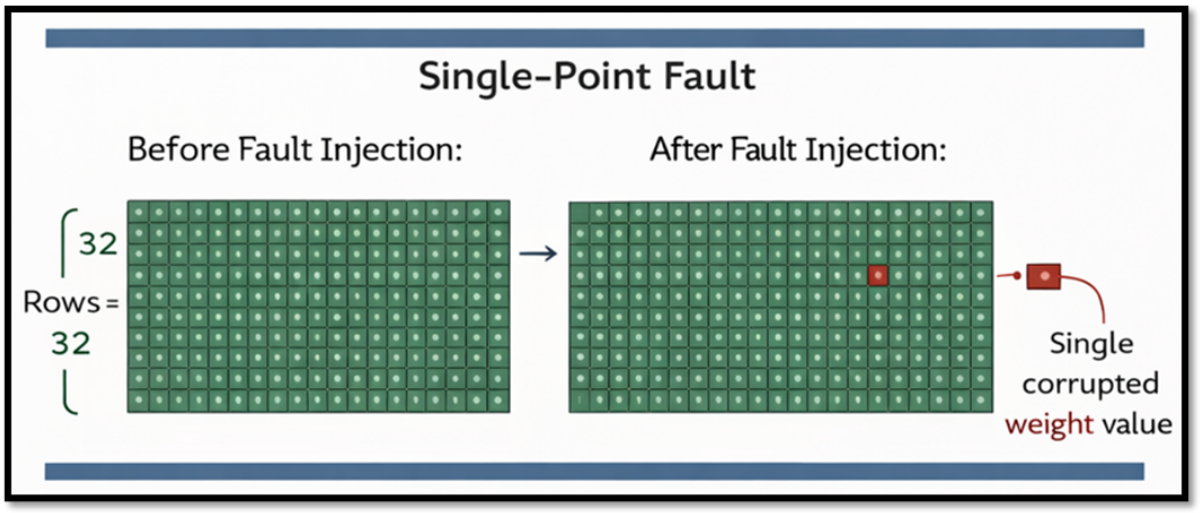


Figure 2.6
A general scheme of a Single Point Fault.

2.7.2 Row Wise Fault

In the Row Wise Fault pattern [55], a contiguous segment of elements within a selected row of the reshaped tensor is perturbed. Since each row corresponds to a flattened convolutional filter, this pattern affects multiple weights within a single output channel. The depiction of Row Fault is shown in Figure 2.7.

This can be interpreted as corruption of one convolutional filter. Because a filter contributes to the generation of a feature map, perturbations may alter entire activation patterns. This pattern approximates scenarios where a localized hardware disturbance affects multiple adjacent memory cells along a row buffer.

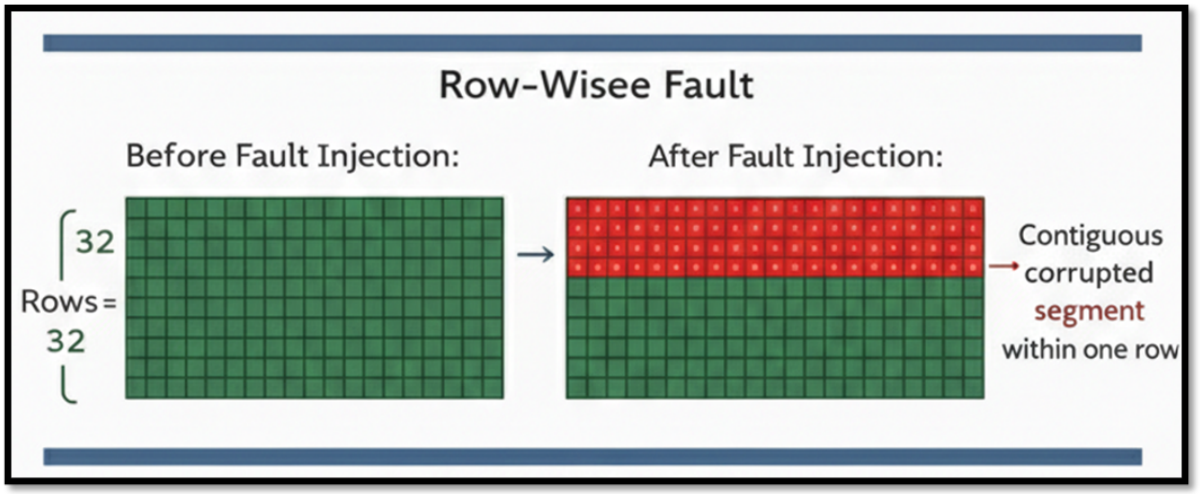


Figure 2.7
A general scheme of a Row Fault.

2.7.3 Column Wise Fault

The Column Wise Fault is an additional pattern added from the inspiration of paper [55], which affects a fixed column index across multiple rows. In the reshaped representation, a column corresponds to a specific spatial kernel position across all output channels.

This pattern models disturbances that impact a shared computational path or memory column. Since the same kernel position is corrupted across several filters, correlated degradation may occur across feature maps.

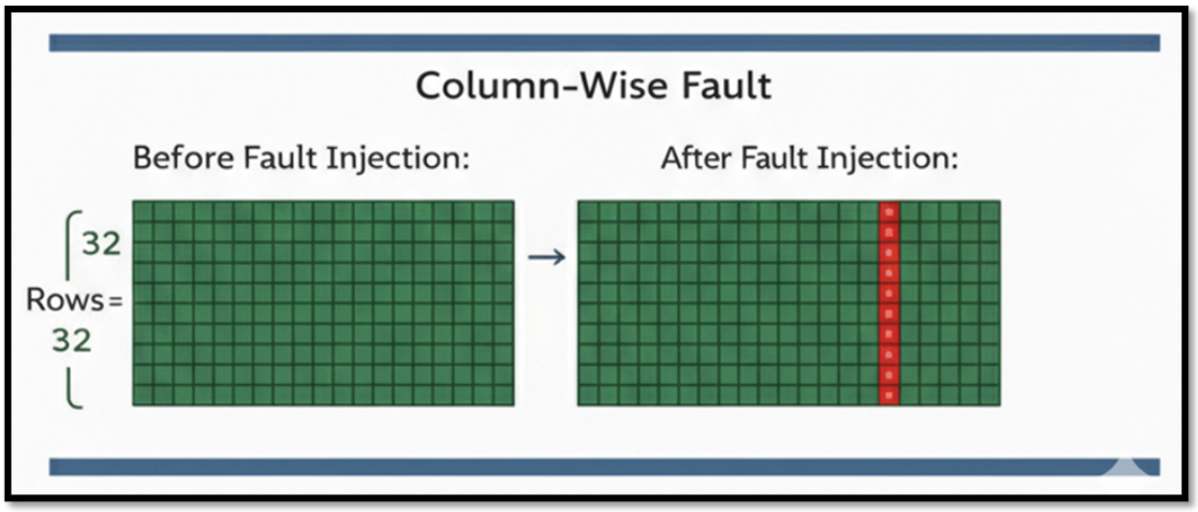


Figure 2.8
A general scheme of a Column Fault.

2.7.4 Bullet Wake Fault

The Bullet Wake [55] Fault Figure 2.9 introduces a structured diagonal pattern across the reshaped tensor. It simulates a disturbance that propagates vertically and horizontally with controlled sparsity.

This pattern reflects more complex hardware like effects where perturbations are not confined to a single row or column but follow a trajectory. Its structured nature allows investigation of correlated multi channel corruption and its propagation characteristics.

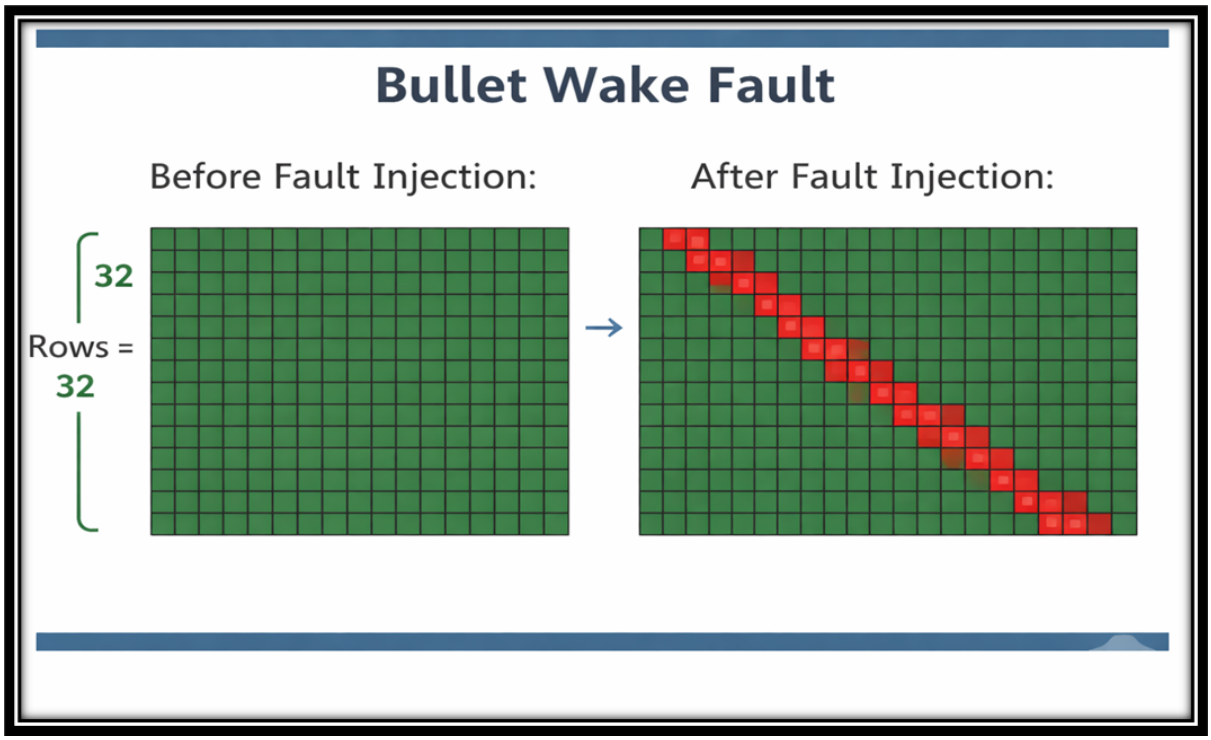


Figure 2.9
A general scheme of a Bullet Wake Fault.

2.7.5 Shattered Glass Fault

The Shattered Glass [55] Fault figure 2.10 represents clustered corruption. Multiple adjacent rows and columns are perturbed simultaneously, creating a dense region of disturbance.

This pattern approximates more severe localized hardware disturbances where several contiguous memory elements are affected. Due to its density and spatial coverage, it typically produces stronger degradation and serves as a stress test for resilience analysis.

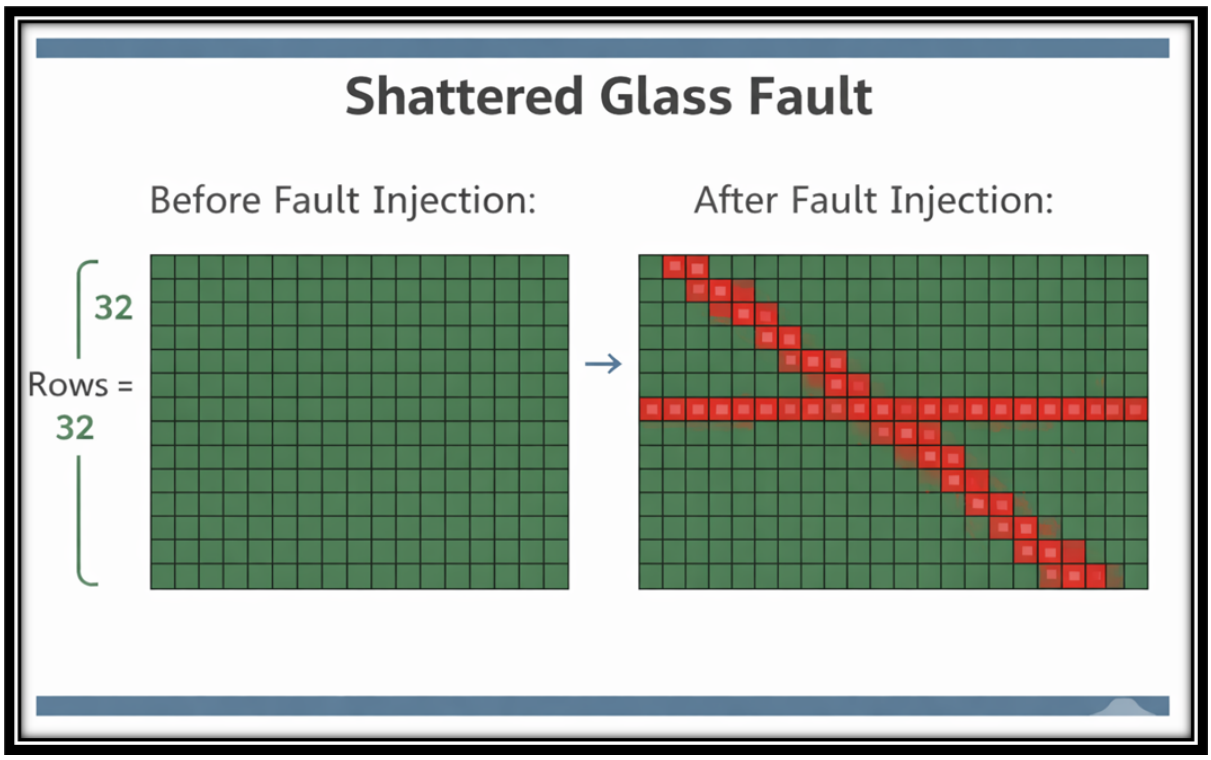


Figure 2.10
A general scheme of a Shattered Glass Fault.

Each of these five patterns enables systematic exploration of how spatial structure influences fault propagation in convolutional weight tensors.

2.8 Reliability Perspective on Weight Corruption

Weight corruption affects the internal representation learned by the network. Since convolutional filters encode spatial feature extraction patterns, perturbing these filters modifies the response of feature maps to input stimuli.

From a system level perspective, corruption may result in:

- Reduced segmentation overlap
- Structural distortion in masks
- Geometric displacement of lane boundaries

- Missing or unstable object detections
- Confidence score degradation

The nonlinear nature of deep neural networks makes it difficult to predict which tensors are most vulnerable. Some perturbations may remain internally compensated, while others may amplify through successive layers.

Understanding this behavior requires systematic experimentation across multiple tensors, intensities, and seeds. The five fault patterns described above provide a structured basis for analyzing vulnerability distribution across the network.

Chapter 3

PROPOSED FRAMEWORK

3.1 Overview of the Framework

FIERA, which stands for Fault Injection and Evaluation for Robust Automotive Perception 3.1, is the reliability analysis framework developed in this thesis to systematically evaluate the impact of hardware disturbances on deep learning-based automotive perception systems.

FIERA is composed of the following tools;
1) A Random Scenarios Generator for fault injection 4.2.
2) Custom Scenario Generated by user with parameter setup of spatial coordinates 4.1.1.
3) A Video-based inference pipeline, used to visually observe perception degradation under fault conditions 3.6.4.
4) A Metrics-based inference pipeline, used to evaluate perception deviations using task-specific similarity and robustness measures quantitatively 3.6.5.
5) A video comparison tool to compare the Original vs Faulty Segmented videos4.1.6.

The FIERA framework consists of dedicated Python-based execution pipelines, which are the following:

- ***Video_Based_Pipeline.py*** for video-based inference pipeline.
- ***Metrics_Based_Pipeline.py*** for Metrics-based inference pipeline.
- ***GenerateScenarios.py*** for generating multiple scenarios as a txt file based on JSON lines
- ***Enhanced_Compare_videos.py*** for comparing the Golden Vs Faulty Video

FIERA has two operational modes. The first Operational mode is the Green Activity from the figure 3.1 that takes the Random Scenarios from the script GenerateScenarios.py and the second Operational mode is Blue Activity from figure 3.1 which is Custom based Approach, in which the user explicitly defines the exact coordinate locations to inject the Fault inside the Tensors, which is a slow process if done for numerous Tensors.

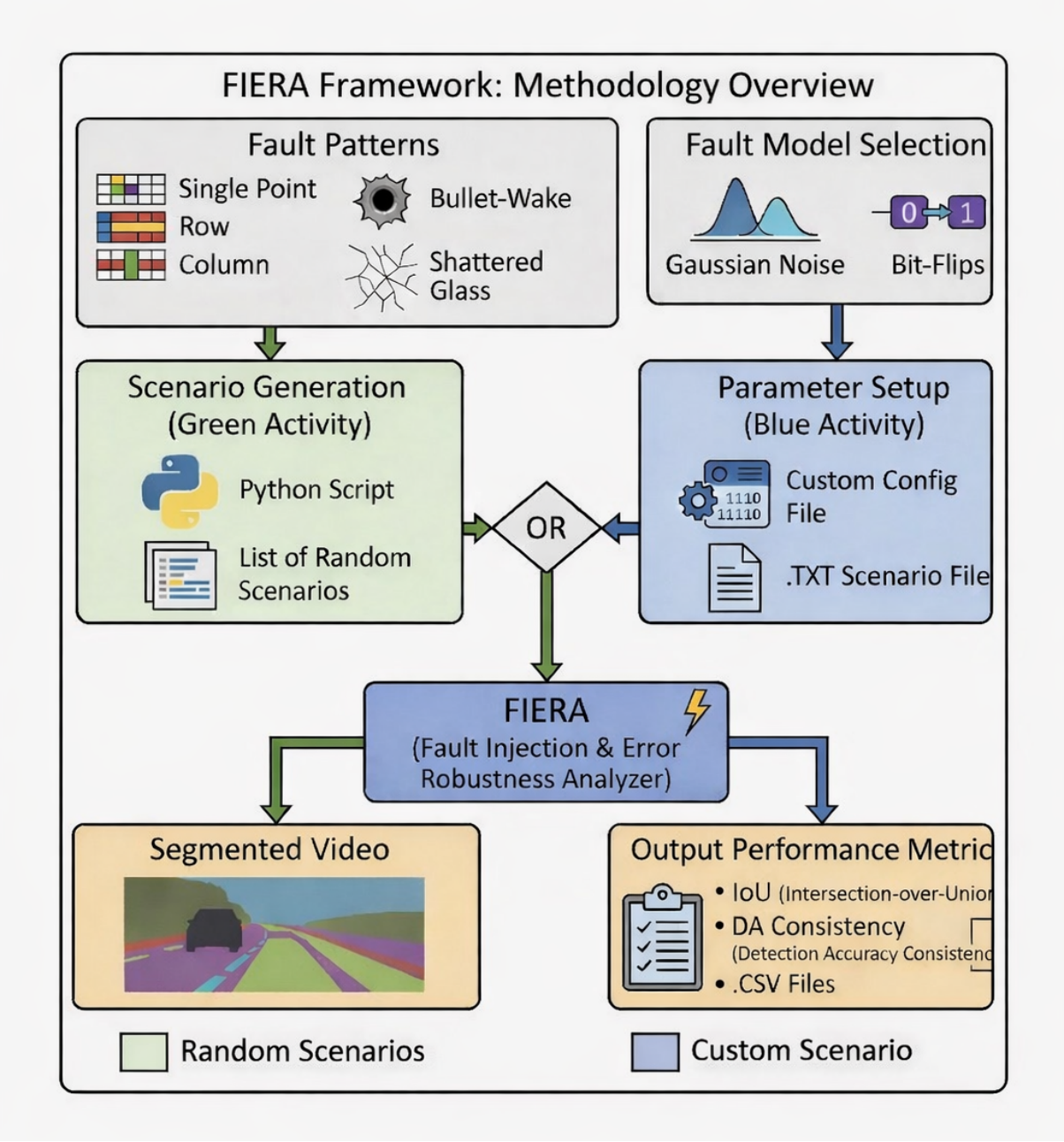


Figure 3.1
FIERA Framework

The framework 3.1 supports multiple structured fault patterns, including Single Point, Row, Column, Bullet Wake, and Shattered Glass [55], and allows the selection of different fault models, such as Gaussian noise and bit flips to corrupt neural network weight tensors. Scenarios are generated either by automated Python scripts or by predefined configuration files, thereby ensuring controlled randomness and reproducibility.
These scenarios define the target tensor, fault type, intensity, and other tuning parameters used during execution.

The framework executes the perception model under nominal and perturbed conditions, enabling direct comparison between fault-free and faulty outputs. It produces two main

types of results: Visually segmented outputs for qualitative inspection and quantitative performance metrics, such as IoU, Pearson Coefficient Metrics, and detection consistency, stored in structured data files for statistical analysis.

Through this integrated design, FIERA provides a systematic and reproducible methodology for identifying vulnerable weight tensors, estimating failure probability, and analyzing resilience in multi-task automotive perception models.

3.2 Fault Models

This thesis considers two fault models for evaluating reliability under hardware-like disturbances: **Gaussian noise injection** and **Bit flip injection**, both applied to neural network weight tensors during inference.

The Gaussian noise model adds zero-mean normally distributed perturbations to selected weight elements, with magnitude scaled according to the tensor statistics. It provides a continuous and controllable abstraction of transient numerical disturbances. For this research, Gaussian noise is the primary fault model and forms the basis of the statistical analysis.

The bit flip model modifies specific bits in the floating point representation of weight values, emulating memory level soft errors. In this thesis, bit flip injection is implemented as a complementary model, while the main experimental campaign relies on the Gaussian noise strategy.

3.3 Framework Strategy for Emulating Hardware Faults

The objective of the proposed framework FIERA is to evaluate the impact of hardware faults on AI-based perception systems by emulating realistic fault effects at the neural network level. Since direct experimentation on physical hardware accelerators is often impractical, this work adopts a software-based fault-injection strategy that targets the internal parameters of the perception model during inference. In particular, the framework operates by injecting controlled perturbations into selected weight tensors of the YOLOP model while preserving the original network architecture and training configuration. Faults are introduced dynamically and temporarily, allowing inference to proceed under faulty conditions and then restoring the fault-free state. This design choice ensures repeatability, isolation of fault effects, and fair comparison between nominal and faulty executions.

To emulate the behavior of transient hardware faults, the framework employs Gaussian noise-based perturbations, whose magnitude is scaled according to the statistical properties of the original weights. This approach allows the injection process to remain systematic, measurable, and proportional to the natural distribution of model parameters.

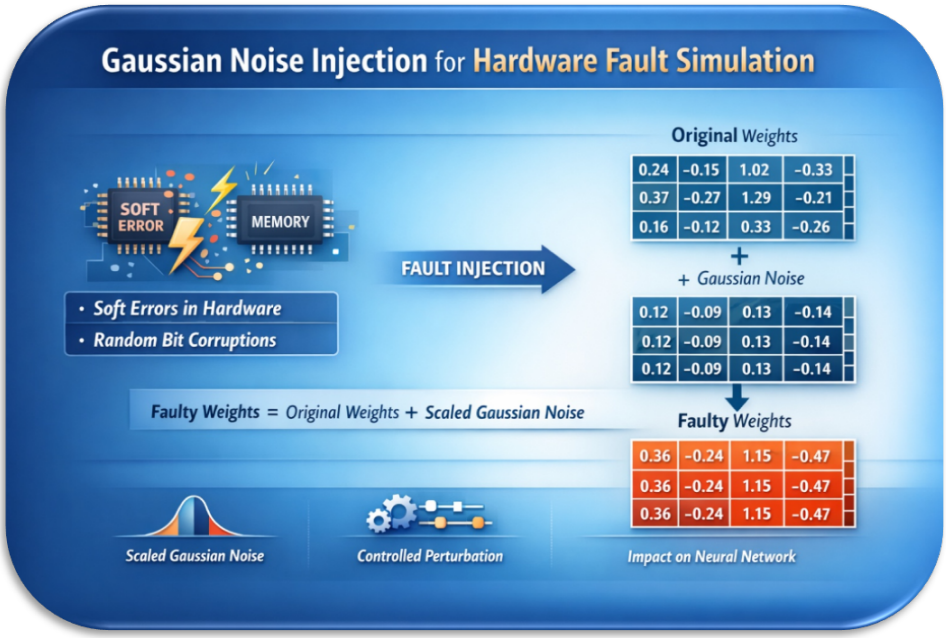


Figure 3.2
A general example of gaussian noise for Hardware fault simulation.

3.4 Gaussian Noise in magnitude error corruption

Gaussian noise is considered a safe and accepted choice for framework for several technical and methodological reasons.

First, it provides a statistically grounded abstraction of transient hardware disturbances. Real soft errors in modern semiconductor devices often manifest as small, unpredictable numerical deviations in stored or computed values. When observed at the algorithmic level, the aggregate effect of such disturbances can be reasonably approximated as stochastic noise. Gaussian distributions [59] are widely used in engineering to model random fluctuations due to their theoretical properties and empirical validity.

Second, Gaussian noise enables controlled and scalable perturbation intensity. By scaling the injected noise relative to the standard deviation of each weight tensor, the disturbance remains proportional to the natural numerical range of the model parameters. This avoids unrealistic distortions and ensures that the fault injection remains context aware rather than arbitrarily destructive.

Third, it supports smooth sensitivity analysis. Unlike discrete bit flips, which may cause abrupt and non uniform changes, Gaussian perturbations allow gradual variation of fault severity. This makes it possible to study degradation trends, vulnerability thresholds, and resilience behavior in a systematic and statistically interpretable manner.

Gaussian noise is widely used in robustness and reliability research [60], [61] as a proxy for hardware-induced perturbations in neural networks. Its use is academically accepted because it captures functional impact at the application level without requiring detailed transistor-level modeling.

Importantly, the goal of this noise model is not to simulate exact bit-level behavior, but to approximate the functional impact of hardware faults on neural computations. This abstraction is well suited for evaluating perception degradation and fault propagation at the application level.

3.5 Relationship to Real Hardware Faults

Although Gaussian noise is an abstract model, it exhibits strong conceptual alignment with real hardware fault behavior observed in modern computing systems.

In hardware accelerators used for AI inference:

- Soft errors may affect memory cells, registers, or arithmetic units.
- Such faults often result in temporary corruption of numerical values rather than permanent failure.
- Such faults often result in temporary corruption of numerical values rather than permanent failure.
- Such faults often result in temporary corruption of numerical values rather than permanent failure.
- The exact location and magnitude of the error are typically unpredictable.
- Such faults often result in temporary corruption of numerical values rather than permanent failure.

The proposed framework reflects these characteristics by:

- Injecting temporary perturbations during inference rather than permanently modifying weights,
- Allowing faults to affect localized or structured regions of weight tensors,
- Supporting both sparse and contiguous corruption patterns.

Additionally, by extending Gaussian noise injection to structured patterns (e.g., single-point, row-wise, column-wise, or spatially correlated perturbations), the framework approximates realistic fault scenarios such as memory row failures, column disturbances, or clustered corruption effects.

While this approach does not replicate the physical origin of hardware faults at the transistor or bit level, it captures their observable consequences at the algorithmic level, which is the primary concern for AI-based perception reliability, as reported by several works in the state-of-the-art [62], [61], [63]. As such, the framework provides a practical and effective means to evaluate how transient hardware faults can influence perception outputs in automotive systems.

3.6 Fault Injection Framework Implementation

As depicted in the **FIERA figure 3.1** the implementation of the proposed fault injection framework developed to analyze the impact of transient hardware faults on AI-based perception. The framework is built around the YOLOP perception model and is implemented through two complementary execution of Python scripts pipelines:

* **A Video-based inference pipeline,** used to visually observe perception degradation under fault conditions;
* **A Vetrics-based inference pipeline,** used to quantitatively evaluate perception deviations using task-specific similarity and robustness measures.

Both pipelines share a common inference structure and fault injection logic, while differing in the form of output they produce. Together, they enable a comprehensive evaluation of fault effects at both qualitative and quantitative levels.

3.6.1 General Inference Workflow

In both pipelines, inference follows a consistent execution flow. The YOLOP model is loaded once at startup using a pre-trained checkpoint and configured for evaluation. The input video is then processed sequentially, frame by frame, using the same preprocessing and post-processing steps as the standard YOLOP inference procedure. For each frame, the network produces three outputs: 1) Object detection (DE), 2) Drivable Area segmentation (DA), 3) lane line segmentation (LL).
Detection outputs are post-processed using non-maximum suppression, while segmentation outputs are resized and converted into pixel-level masks. This common inference path ensures that any observed deviation in the output can be attributed to the injected fault rather than to differences in preprocessing or post-processing.
The proposed framework allows the user to configure several parameters, including the target tensor or layer, the type of fault model (e.g., Gaussian noise or bit-flip), the fault location within the tensor, the perturbation intensity, the fault injection time window during inference, and the random seed used for scenario generation. These parameters enable controlled and reproducible experiments for evaluating the sensitivity of the perception model to injected faults. More details will be provided in Chapter 4.

3.6.2 Targeting of Network Parameters

Fault injection experiments are configured through external scenario descriptions, which specify the target parameter tensor, fault pattern, and injection configuration. Each target is identified by its unique parameter name within the YOLOP model, allowing direct and reproducible access to specific tensors.

During execution, the selected parameter tensor is retrieved from the model's parameter set. A fault-free copy of the tensor is preserved to allow restoration after each faulty inference step. This mechanism ensures that faults are introduced in a controlled and temporary manner, without permanently altering the model state.

The parameter selection strategy enables experiments to be conducted across different architectural components of YOLOP, including backbone, neck, and task-specific heads, without imposing architectural exclusions.

3.6.3 Fault Injection Timing and Application

Fault injection is performed during inference over a configurable frame interval, referred to as the fault window. For frames outside this window, inference proceeds under nominal conditions. For frames inside the window, the target tensor is modified immediately before the forward pass and restored immediately after inference.

This time-localized injection strategy allows the analysis of transient fault effects and supports frame-by-frame comparison between nominal and faulty outputs. It also enables the evaluation of temporal stability and recovery behavior once the fault window ends.

The framework does not alter the computational graph or network structure during inference. Instead, faults are applied directly to parameter values used during the forward computation, ensuring compatibility with standard inference pipelines.

3.6.4 Video-Based Evaluation Pipeline

The Video-Based pipeline (`Video_Based_Pipeline.py`) produces an annotated output video for each fault scenario. Each frame displays (OD), (DA) and (LL).

This pipeline is intended for qualitative analysis, enabling visual inspection of how perception outputs change under injected faults. It is particularly useful for identifying failure modes such as missing detections, distorted segmentation boundaries, or temporal instability.

3.6.5 Metrics-Based Evaluation Pipeline

The Metrics-Based Pipeline is (`Metrics_Based_Pipeline.py`) implemented as a dedicated script that executes the YOLOP perception model under predefined fault injection scenarios and records the resulting quantitative deviations in structured CSV files.

The script loads a pre-trained checkpoint, reads a scenario file describing the target tensor, fault type, noise model, perturbation intensity, and temporal fault window, and then processes the input video frame by frame. For each injected frame, the pipeline performs both golden and faulty inference, compares the corresponding outputs, and computes task-specific metrics for object detection, drivable area segmentation, and lane line segmentation.

The resulting measurements are aggregated per scenario and stored in `metrics_summary.csv`, while optional frame-level results can be exported to `metrics_per_frame.csv` for detailed analysis.

3.7 Summary

The framework presented in this chapter enables systematic experimentation on the effects of transient parameter perturbations during inference in a multi-task perception model. By combining controlled fault injections, time-localized activation, and both visual and quantitative evaluation pipelines, the framework provides a structured basis for analyzing how hardware-like faults influence perception outputs in automotive scenarios.

The results produced by this framework form the foundation for the experimental analysis and statistical evaluation presented in the following chapters.

Chapter 4

EXPERIMENTAL METHODOLOGY

4.1 A. Custom Scenarios Parameters (BLUE Activity)

The custom scenario parameters define how a fault is spatially placed and propagated inside a selected tensor. In the proposed framework 3.1, the user can directly specify the **spatial coordinates** of the fault location instead of relying only on random selection. This makes the injection process configurable and reproducible, since the same tensor region can be targeted repeatedly across different runs. The coordinates are provided through the scenario description and are read by both the metrics pipeline and the video pipeline, where they are passed to the selected fault injection function through the `coords` field.

These parameters are introduced to control the spatial location of the corruption inside the reshaped tensor representation. Depending on the selected fault pattern, the coordinates can identify a single point, a row interval, a column interval, or a structured region affected by broader spatial corruption.

In this way, the framework supports both random fault generation and user guided spatial targeting, allowing the same experimental infrastructure to be used for exploratory campaigns as well as controlled case studies.
The role of this subsection is to introduce the parameter group that governs spatial targeting. The next subsection explains how these spatial parameters are used for the different fault patterns and how the corresponding coordinate definitions are interpreted in practice.

4.1.1 Spatial Targeting Parameters

Convolutional layers in the perception model are represented as four-dimensional tensors of the form:
$(\textbf{out\_channels}, \textbf{in\_channels}, k_h, k_w)$

For fault injection purposes, these tensors are internally reshaped into a two-dimensional representation to facilitate row-wise and column-wise indexing. The following parameters control spatial fault localization:

Variable	Description / Meaning	Purpose in Fault Modeling
r	Selected row index in the reshaped 2D tensor representation	Targets a specific output feature map (filter) for row-oriented or structured fault patterns
$\mathbf{r_0}$	Starting row index of a contiguous fault region	Defines the beginning of a multi-row corruption span
$\mathbf{c_0}$	Starting column index of the fault region	Defines the horizontal start of structured corruption
$\mathbf{c_1}$	Ending column index of the fault region	Defines the horizontal boundary of structured corruption

Table 4.1
Fault Modeling Variables

r: Represents the selected row index within the logical two-dimensional representation of the tensor. In practice, this corresponds to selecting a specific output feature map or filter. This parameter is central to row-oriented and structured fault patterns. $\mathbf{r}_0$: Defines the starting row index when the injected fault spans multiple contiguous rows. It is primarily used in row-wise and shattered-glass patterns. $\mathbf{c}_0$, $\mathbf{c}_1$: Define the column boundaries of the fault region.

- $\mathbf{c}_0$ specifies the starting column index.
- $\mathbf{c}_1$ specifies the ending column index.

These variables can be set in two scripts in FIERA:
Video_Based_Pipeline.py and ***Metrics_Based_Pipeline.py***, which control the exact spatial location of the injected fault. In practice, they define the row and column boundaries of the corrupted region in the reshaped tensor, enabling controlled and repeatable spatial fault injection across different fault patterns and experiments, see the available open-source repository [1].

[1] `https://github.com/alafidmunir94-collab/Thesis_Reliability-of-DL-Based-Automotive-Perception`.

4.1.2 Repetition and Persistence Parameters

Transient hardware faults may manifest as isolated single events or as recurring disturbances over time. To capture this variability, the framework includes repetition-related parameters:

Table 4.2
Parameters used in the fault injection model.

Variable	Description / Meaning	Purpose in Fault Modeling
repeats	Number of times a fault is re-applied during inference.	Simulates persistent or recurring transient disturbances.
keep_prob	Probability that the injected fault remains active during repetitions.	Models intermittent faults and partial fault masking behavior.

repeats: Defines how many times a fault is re-applied to the same tensor location across consecutive inference passes. A value greater than one simulates persistent or recurring disturbances, whereas a value of one models a single-event transient fault.

keep_prob: Represents the probability that a fault remains active during repeated injections. This parameter introduces controlled stochasticity, allowing faults to probabilistically disappear between iterations. It models partial masking or intermittent hardware instability.

`keep_prob` takes values in the interval $(0, 1]$, where values close to 0 correspond to low fault persistence, and 1 corresponds to full persistence during repeated injections. The parameter `repeats` is selected according to the desired observability of the fault pattern. For example, in the Single Point pattern, a value such as `repeats = 10` produces only weak visible degradation, whereas `repeats = 50` makes the fault effect more clearly observable in the video outputs.

4.1.3 Fault Extent Parameters

This parameter is particularly relevant for row-wise and shattered-glass fault patterns, where spatial continuity is a defining characteristic and clustered perturbations more closely approximate correlated hardware disturbances. In particular, these patterns might affect contiguous regions. Thus, the parameter controlling this spatial extent is:

row_span: Defines the number of consecutive rows affected by a fault.

- A value of 1 corresponds to a single-row disturbance.
- Larger values produce extended row-wise corruption.

Table 4.3
Parameter controlling spatial fault propagation patterns.

Variable	Description / Meaning	Purpose in Fault Modeling
row_span	Number of consecutive rows affected by a fault.	Controls the spatial continuity of row-wise or clustered corruption patterns.

4.1.4 Temporal Control Parameters

To ensure controlled experimentation and fair comparison across scenarios, fault activation is temporally constrained during inference as follow:

Table 4.4
Temporal parameters controlling the fault injection window.

Variable	Description / Meaning	Purpose in Fault Modeling
Injection window	Range of inference steps or video frames during which faults are active.	Ensures that faults are injected only within controlled experimental intervals.
inject_every	Periodicity of fault injection within the defined window.	Simulates regularly occurring transient disturbances instead of continuous corruption.
fault_start	Starting frame index of the injection window.	Defines the beginning of temporal fault activation.
fault_end	Ending frame index of the injection window.	Defines the termination of fault activation.

Injection window: Defines the temporal interval, in terms of frames or inference steps, during which faults are active. Outside this window, the model operates under nominal conditions.

inject_every: Specifies the periodicity of fault application within the injection window. For example, injecting every Nframes simulates regularly occurring disturbances rather than continuous corruption.

Temporal control is particularly important in video-based perception, where stability and consistency across frames directly influence downstream planning and control systems.

4.1.5 Key Fault Injection Parameters

The tuning parameters are intentionally decoupled from the network architecture and inference logic. This separation ensures that identical configuration strategies can be consistently applied across different layers, tensors, and fault patterns.

By combining spatial targeting, persistence modeling, extent control, and temporal scheduling, the framework enables systematic exploration of the fault space while remaining

interpretable from a hardware reliability perspective. This structured parameterization supports both qualitative inspection and statistically grounded quantitative analysis.

Table 4.5
Fault injection configuration parameters used in the experimental setup.

Variable	Description / Meaning	Purpose in Experimental Design
intensity	Magnitude of Gaussian perturbation applied to selected tensor elements.	Controls the severity of numerical disturbance introduced during fault injection.
seed	Random seed used for perturbation reproducibility.	Ensures deterministic and statistically fair experimentation across runs.
fault_type	Selected fault pattern (Single, Row, Column, Bullet Wake, Shattered Glass).	Defines the spatial organization of the injected corruption.
noise_model	Fault mechanism used for perturbation (Gaussian noise or Bit-flip).	Determines the strategy used to perturb tensor values.
target_tensor	Selected convolutional weight tensor targeted for injection.	Identifies the specific model component under evaluation.

4.1.6 Visual Frame Difference Analysis

To complement the quantitative metric evaluation, a visual comparison tool was developed to analyze perception degradation at the frame level. The auxiliary script, `Enhanced_Compare_videos.py`, performs synchronized comparison between a fault-free inference video and its corresponding fault-injected counterpart.

1) Processing Pipeline: For each synchronized frame pair (golden vs faulty), the script performs the following steps:

- Resizes both frames to a uniform resolution to ensure pixel-wise comparability.
- Converts frames to HSV color space to facilitate robust extraction of perception overlays.
- Computes task-specific difference maps by isolating overlay regions corresponding to object detection bounding boxes, drivable area segmentation masks, and lane detection regions.
- Computes an overall pixel-wise absolute RGB difference map to capture global deviation.

This multi-level comparison enables separation of task-specific degradation from overall visual distortion.

2) Visualization Output:
The script ***Enhanced_Compare_videos.py*** generates a composite visualization containing the Original frame, the Fault-injected frame, the Overall difference map, and the Task-Specific difference maps. In particular, the differences are classified according to the perception targets:

- Object detection differences
- Drivable area differences
- Lane detection differences

Furthermore, for representation and analysis, each displayed frame includes metadata, including the frame index, overall difference magnitude, and per-task difference values.
Figure 4.1 illustrates an example of a typical frame-by-frame analysis between a reference video and the produced video when errors corrupt the operation of the AI-based perception task. As observed, the reference frame (*Left*) correctly identifies the drivable areas in the scenario. Instead, the faulty frame (*Center*) exhibits slight inconsistencies and corrupted regions due to error effects. On the right, the difference frame highlights the areas and regions in the scenario that are affected by the error in the IA-powered perception application.
The red map represents differences in object detection, with highlighted red regions indicating changes in detected bounding boxes or object-related overlays. The green map represents differences in drivable area; the green regions highlight where the estimated road region differs between the reference and faulty outputs. The blue map represents differences in lane detection, with the blue regions indicating changes in the detected lane structure.

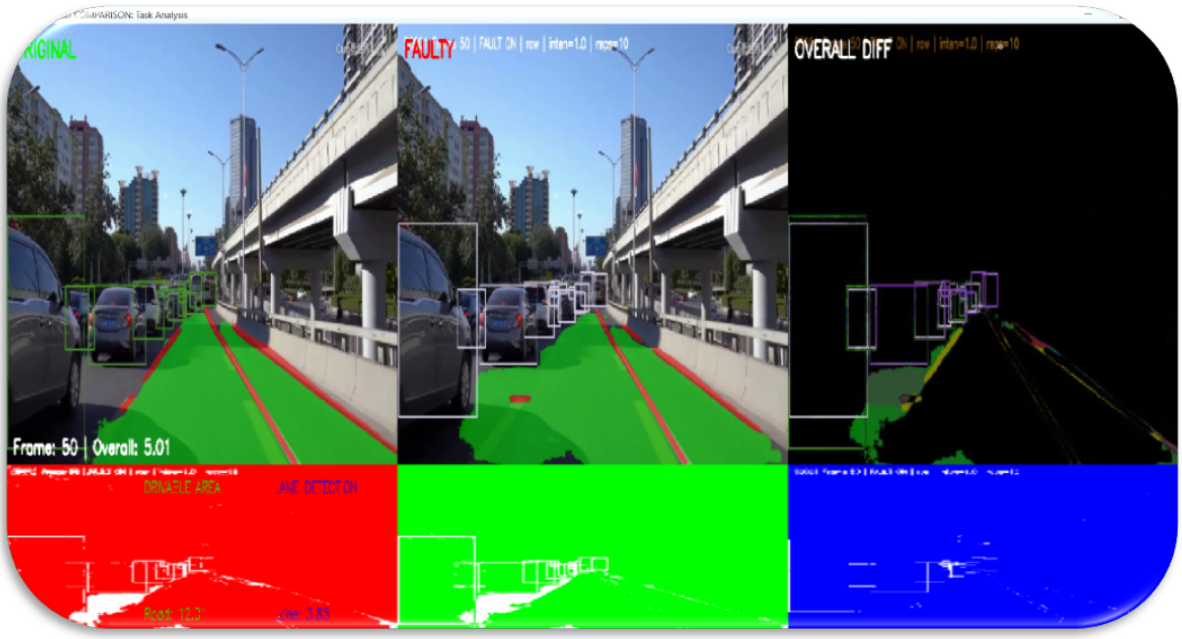


Figure 4.1
An example of the performed comparison between reference videos (*left*), faulty videos (*Center*), and the overall pixel-per-pixel area difference (*right*).

4.2 B. Random Scenario Generation (Green Activity)

The scenario generation stage, illustrated in Figure 3.1, constitutes the input preparation phase of the FIERA framework. Its primary objective is to construct statistically fair and reproducible fault-injection configurations before the execution of the inference pipeline.

The scenario generation produces a structured, configuration-based JSON text file that defines how faults are injected into the neural network weights during inference. Each experimental campaign requires preparing a batch of randomized scenarios for each of the five supported fault patterns: single point, row, column, bullet wake, and shattered glass. For this purpose, a JSON-based scenario definition file is employed.

4.2.1 JSON-Based Scenario Definition:

Each scenario is represented as a single JSON object, stored on a separate line in a plain-text file generated by GenerateScenarios.py.
An example configuration file, such as ***Scenarios.txt***, contains multiple JSON entries, each describing an independent fault injection experiment to be performed by the FIERA tool. In particular, organizing the fault injection experiments in this format ensures modularity, human-readable documentation, and compatibility with automated parsing mechanisms, as illustrated in Listing 1.

The `Scenarios.txt` file is used as input to the following scripts:
`Video_Based_Pipeline.py`, which generates the video outputs, and
`Metrics_Based_Pipeline.py`, which generates the CSV results.

Each JSON entry includes:

target_tensor	Specific convolutional weight tensor selected for injection
noise_model	Fault mechanism (e.g., Gaussian or bit-flip)
fault_type	Spatial fault pattern (e.g., row, column, or another)
intensity	Perturbation magnitude (0.0 - 1.0)
seed	Random seed for reproducibility (1 - 20)

And additional tuning parameters as described earlier in section 4.1 such as ***repeats, keep_prob, row_span, inject_every, fault_start, and fault_end.***
This line-wise JSON representation allows efficient batch processing while preserving traceability of individual experimental conditions.

Listing 1
JSON format for Batch Scenarios

```
# Fixed parameters:
# - noise_model: gaussian
# - repeats: 10
# - keep_prob: 0.1
```

```
# - row_span: 0.7
# - coords: null
# - inject_every: 1
# - min_seg_len: 2
# Random ranges:
# - intensity: 100 levels from 0.01 to 1.00
# - seed: integers 1..20
# - target_tensor: cycles in the same order as tensors file
# Key mapping:
# - noise_model = gaussian
# - fault_type  = one of: single,row,column,bullet_wake,shattered_glass

{"target_tensor": "model.0.conv.conv.weight", "noise_model": "gaussian", "fault_type": "bullet_wake", "intensity": 0.19, "seed": 4, "repeats": 10, "keep_prob": 0.1, "row_span": 0.7, "coords": null, "inject_every": 1, "min_seg_len": 2, "fault_start": 1, "fault_end": 400, "gauss_mean": 0.0}
{"target_tensor": "model.1.conv.weight", "noise_model": "gaussian", "fault_type": "bullet_wake", "intensity": 0.02, "seed": 3, "repeats": 10, "keep_prob": 0.1, "row_span": 0.7, "coords": null, "inject_every": 1, "min_seg_len": 2, "fault_start": 1, "fault_end": 400, "gauss_mean": 0.0}
{"target_tensor": "model.0.conv.conv.weight", "noise_model": "gaussian", "fault_type": "shattered_glass", "intensity": 0.49, "seed": 5, "repeats": 10, "keep_prob": 0.1, "row_span": 0.7, "coords": null, "inject_every": 1, "min_seg_len": 2, "fault_start": 1, "fault_end": 668, "gauss_mean": 0.0}
{"target_tensor": "model.1.conv.weight", "noise_model": "gaussian", "fault_type": "shattered_glass", "intensity": 0.71, "seed": 12, "repeats": 10, "keep_prob": 0.1, "row_span": 0.7, "coords": null, "inject_every": 1, "min_seg_len": 2, "fault_start": 1, "fault_end": 668, "gauss_mean": 0.0}
{"target_tensor": "model.2.cv1.conv.weight", "noise_model": "gaussian", "fault_type": "bullet_wake", "intensity": 0.23, "seed": 12, "repeats": 10, "keep_prob": 0.1, "row_span": 0.7, "coords": null, "inject_every": 1, "min_seg_len": 2, "fault_start": 1, "fault_end": 400, "gauss_mean": 0.0}
{"target_tensor": "model.0.conv.conv.weight", "noise_model": "gaussian", "fault_type": "bullet_wake", "intensity": 0.06, "seed": 12, "repeats": 10, "keep_prob": 0.1, "row_span": 0.7, "coords": null, "inject_every": 1, "min_seg_len": 2, "fault_start": 1, "fault_end": 668, "gauss_mean": 0.0}
{"target_tensor": "model.1.conv.weight", "noise_model": "gaussian", "fault_type": "bullet_wake", "intensity": 0.8, "seed": 3, "repeats": 10, "keep_prob": 0.1, "row_span": 0.7, "coords": null, "inject_every": 1, "min_seg_len": 2, "fault_start": 1, "fault_end": 668, "gauss_mean": 0.0}
```

4.2.2 Fair Random Sampling Strategy

To ensure statistical validity, the scenarios are generated using a controlled random sampling procedure implemented in the script ***GenerateScenarios.py***

The design follows three principles:

a. Uniform Tensor Coverage

Target tensors are selected from a predefined tensor list. The generator cycles through the tensor list in file order to guarantee balanced coverage and prevent selection bias.

b. Controlled Randomization of Fault Parameters

For each scenario:

* Intensity is selected from a predefined discrete set of levels.
* Random seeds are drawn from a bounded range to ensure reproducibility.
* Fault patterns are sampled from the specified subset corresponding to the experiment.

c. Fixed Structural Parameters
Certain parameters such as repeats, keep_prob, and row_span remain constant across scenarios to isolate the effect of the primary variable under investigation. This structured randomness ensures that no tensor is disproportionately represented while maintaining independence between scenarios.

4.2.3 Integration with the Metrics-Based Inference Pipeline

Once generated, the scenario text file (***Scenarios.txt***) is provided as input to the main execution script ***Metrics_Based_Pipeline.py***
During runtime, the script:

* Reads each JSON line sequentially.
* Validates and normalizes the configuration.
* Applies the specified perturbation to the selected weight tensor.
* Executes inference over the defined injection window.
* Computes performance degradation metrics.
* Stores summary statistics in CSV format.

Thus, the scenario file acts as a declarative experiment specification layer, decoupling configuration from execution logic. This separation enhances reproducibility, enables large-scale batch experimentation, and simplifies statistical post-processing.

4.2.4 Role in Statistical Reliability Analysis

The scenario generation stage is essential for achieving statistically meaningful reliability estimation. Instead of manually selecting tensors or perturbation conditions, the framework produces a sufficiently large and randomly distributed set of configurations, using state-of-the-art strategies [64].
This enables:

* Estimation of failure probability per fault pattern
* Vulnerability ranking of weight tensors.
* Confidence interval computation for degradation metrics.
* Controlled stopping criteria based on margin of error.

By ensuring fair random sampling across tensors and parameter combinations, the Green Activity guarantees that subsequent statistical conclusions are not biased by manual configuration choices.

4.3 Core Framework Metrics Used

The core of the framework is implemented in the script ***Metrics_Based_Pipeline.py***, which automates three critical tasks:
1. Applying the fault injection to selected weight tensors.
2. Executing inference on the same input sequence under golden vs faulty conditions.
3. Computing and logging metrics that quantify perception degradation.

4.3.1 "Golden vs Faulty" Evaluation Concept

A key design choice in this thesis is to evaluate faults against the model's own **golden output** (Fault-free), rather than ground truth labels. In other words, the framework compares:

$$M_{\textbf{golden}} \textbf{ vs } M_{\textbf{faulty}}$$

This is intentional because the primary research goal is to determine the fault-induced deviation measurement under identical inputs and inference settings. Therefore, all metrics are implemented in a differential form:

$$\Delta M = M_{\text{faulty}} - M_{\text{golden}} \tag{4.1}$$

This design isolates the effect of injected perturbations and avoids confounding effects due to dataset annotation noise.

4.3.1.1 Segmentation Overlap – Mean Intersection over Union (mIoU)

For segmentation outputs (Drivable Area and Lane Line), the overlap between golden and faulty segmentation masks is computed using Intersection over Union (IoU) [65].

$$\text{IoU} = \frac{|M_g \cap M_f|}{|M_g \cup M_f|} \tag{4.2}$$

where M_g denotes the golden segmentation mask and M_f denotes the faulty segmentation mask.
The scenario-level mean IoU over all injected frames is:

$$\text{mIoU} = \frac{1}{N} \sum_{i=1}^{N} \text{IoU}_i \tag{4.3}$$

Segmentation degradation relative to the golden inference is defined as:

$$\text{Degradation} = 1 - \text{mIoU} \tag{4.4}$$

Unusable condition [66]: mIoU < 0.90

4.3.1.2 Structural Correlation – Pearson Correlation Coefficient

To measure structural similarity between golden and faulty segmentation masks, the Pearson correlation coefficient [67] is computed between flattened masks.

$$\rho = \frac{\sum (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum (x_i - \mu_x)^2} \cdot \sqrt{\sum (y_i - \mu_y)^2}} \tag{4.5}$$

where x_i and y_i represent corresponding pixels from golden and faulty masks, while μ_x and μ_y denote the respective mean values.
Unusable condition: $\rho < 0.85$ is empirically selected to flag substantial structural deviation.

4.3.1.3 Detection Consistency Rate

Object detection outputs from golden and faulty inference are matched using an IoU-based rule [68].

$$\text{IoU}(\text{box}_g, \text{box}_f) \geq \tau \tag{4.6}$$

where $\tau = 0.5$ is the matching threshold used in the implementation.
The detection consistency rate is defined as:

$$\text{Consistency} = \frac{N_{\text{match}}}{N_g} \tag{4.7}$$

where N_{match} represents the number of golden detections successfully matched with faulty detections and N_g represents the number of golden detections.
Unusable condition: Consistency < 0.90 is empirically selected

4.3.1.4 Area Difference (Segmentation Size Distortion)

To capture segmentation size distortion caused by fault injection, the framework computes the relative area difference between the golden and faulty segmentation masks [66]. The area of a segmentation mask is defined as the number of foreground pixels predicted by the network.

Let M_g and M_f denote the golden and faulty segmentation masks for a given perception task (drivable area or lane line). The corresponding mask areas are defined as:

$$A_g = \sum \mathbf{1}(M_g \neq 0), \qquad A_f = \sum \mathbf{1}(M_f \neq 0) \tag{4.8}$$

where A_g represents the number of foreground pixels in the golden mask and A_f represents the number of foreground pixels in the faulty mask.
The relative area difference is then computed as:

$$\Delta A = \frac{|A_f - A_g|}{A_g + \varepsilon} \tag{4.9}$$

where ε is a small constant used to prevent division by zero (in the implementation, $\varepsilon = 10^{-12}$).
The scenario-level area distortion is obtained by averaging the per-frame area differences across the injected frames:

$$\overline{\Delta A} = \frac{1}{N} \sum_{i=1}^{N} \Delta A_i \tag{4.10}$$

where N denotes the number of frames within the fault injection window.
Unusable condition as Threshold Calibration:

$$\overline{\Delta A} > 0.10 \tag{4.11}$$

A relative area difference greater than 10% indicates that the segmentation mask has expanded or contracted significantly compared to the golden reference. Such distortion may lead to incorrect estimation of drivable space or lane geometry, potentially affecting downstream decision-making modules in autonomous driving systems.

4.3.1.5 Position Shift – Normalized Centroid Displacement

To quantify geometric displacement between segmentation masks, the centroid shift between golden and faulty masks is computed. The position-shift metric is adapted from position-based segmentation monitoring ideas in [66].

$$d = \sqrt{(x_f - x_g)^2 + (y_f - y_g)^2} \tag{4.12}$$

The displacement is normalized by the image diagonal in order to obtain a resolution-independent metric.

$$\text{PositionShift} = \frac{d}{\sqrt{H^2 + W^2}} \tag{4.13}$$

where H and W denote the height and width of the segmentation mask.

4.3.1.6 Confidence Drop (Object Detection Reliability)

$$\text{ConfDrop} = \text{conf}_g - \text{conf}_f \tag{4.14}$$

where conf_g represents the confidence score in the golden inference and conf_f represents the corresponding score in the faulty inference [69].
The scenario-level confidence degradation is computed as the mean confidence drop across matched detections and injected frames.
Unusable condition: ConfDrop > 0.15 is empirically calibrated

4.3.1.7 Detection Count Difference

$$\Delta Count = Count_f - Count_g \tag{4.15}$$

where N_f is the number of detected objects in the faulty inference and N_g is the number detected in the golden inference.
This metric captures instability caused by disappearance or spurious creation of detections during fault injection.

4.3.1.8 Combined Usability Decision Rule

A scenario is classified as unusable if at least one of the following conditions is satisfied:

- mIoU < 0.90
- Pearson correlation < 0.85
- Detection consistency < 0.90
- Confidence drop > 0.15
- Area Difference > 0.10

Otherwise, the scenario is considered usable. These metrics jointly capture overlap degradation, structural distortion, geometric displacement, and detection instability. The thresholds were selected using a two-step procedure. Metrics related to segmentation degradation, such as mIoU and area difference, were initialized from the 10% criticality criterion reported in prior literature, while the remaining thresholds were tuned on a representative subset of faulty scenarios to separate minor deviations from substantial corruption. Since the final rule is OR-based, conservative thresholds were adopted so that noticeable degradation in the faulty output is not overlooked

4.4 HPC-Based Execution and SLURM Workflow

4.4.1 Introduction

The large-scale fault injection experiments conducted in this thesis required repeated execution of the YOLOP inference framework across hundreds of randomly generated fault scenarios. Executing these experiments manually on a local machine would be inefficient due to the computational cost associated with video processing, neural network inference, and metric computation. Therefore, the experiments were executed on Politecnico di Torino's High Performance Computing (HPC) infrastructure (HPC@PoliTo[2]), which handles support for SLURM scheduling.
SLURM provides automated job scheduling, resource allocation, and execution monitoring. Using the HPC infrastructure allowed the reliability analysis experiments to be executed in a scalable and reproducible manner while efficiently utilizing available computational resources.

4.4.2 HPC Computing Environment

The experiments were conducted on the HPC cluster using the `cpu_sapphire` partition. This partition provides compute nodes equipped with modern multi-core processors suitable for CPU-based machine learning inference tasks.
The configuration used in this work was:

- Architecture: CPU Sapphire 45
- Access mode: Open
- Priority class: Standard
- Maximum walltime: 24 hours

Each compute node provides up to 45 CPU cores which can be requested through SLURM resource directives. However, instead of requesting the full node capacity, the jobs were configured to request between 24 and 28 CPUs per task. This configuration provided a balanced trade-off between execution speed and queue waiting time.

4.4.3 Software Environment and Dependencies

The execution of the YOLOP fault injection framework required a dedicated software environment containing the necessary Python libraries and machine learning dependencies. A Conda environment was used to ensure consistent package management across all HPC jobs.
The environment used for the experiments was named `yolop313`. The primary packages included:

- Python 3.13
- PyTorch for loading and executing the YOLOP neural network model

[2]https://hpc.polito.it/

- Torchvision for preprocessing and image transformations
- OpenCV for video frame extraction and processing
- NumPy for numerical computations
- SciPy for mathematical and statistical operations
- Pandas for metrics aggregation and CSV file generation
- Openpyxl for exporting results to Excel format
- PIL and ImageIO for image and video data handling

Additionally, the YOLOP project modules were made accessible by exporting the project root directory to the `PYTHONPATH` variable during job execution.

4.4.4 SLURM Job Scheduling

SLURM (Simple Linux Utility for Resource Management) is a widely used workload manager in HPC systems. It allows users to submit computational jobs which are automatically scheduled and executed when the required resources become available.
In this work, SLURM was used to:

- allocate computational resources
- schedule fault injection experiments
- manage job queues
- record execution logs
- automate repeated experiment runs

This approach enabled the reliability analysis experiments to be executed in a fully automated and reproducible manner.

4.4.5 Structure of the SLURM Batch Script

The experiments were executed using a SLURM batch script which defines both the computational resources required and the sequence of commands executed during the job.

4.4.5.1 Resource Allocation

Resource requirements are specified using `#SBATCH` directives which inform the SLURM scheduler about the resources needed for the job. The parameters defined include:

- job name
- compute partition
- number of CPUs per task

- memory allocation
- maximum runtime
- output and error log paths
- email notification settings

These directives allow the scheduler to place the job appropriately within the cluster queue.

4.4.5.2 Path Initialization

The batch script defines the required paths for the experiment, including:

- the YOLOP project root directory
- the Python metrics-generation script
- the scenario file directory
- the output directory
- the YOLOP model weights
- the input video file

Defining absolute paths ensures that the job can execute correctly regardless of the directory from which the job is submitted.

4.4.5.3 Dynamic Parameter Passing

The batch script uses environment variables passed through the SLURM submission command using the `--export` flag. Two important variables are passed during job submission:

- `SCEN_FILE`, which specifies the scenario file containing the fault injection configurations
- `PAT_NAME`, which specifies the name of the fault pattern being executed

This approach allows the same batch script to be reused for multiple experiments without modification.

4.4.5.4 Environment Initialization

Before running the Python framework, the batch script initializes the software environment by loading Conda shell support and activating the `yolop313` environment. Additionally, the `PYTHONPATH` variable is set to the YOLOP project directory so that all project modules can be imported correctly.

4.4.6 Execution of the Fault Injection Framework

After the environment is prepared, the batch script executes the custom Python scripts responsible for performing the fault injection experiments:

`Video_Based_Pipeline.py`

and

`Metrics_Based_Pipeline.py`

This script performs the following operations:

1. Loads the trained YOLOP model weights.
2. Reads the input video frame by frame.
3. Parses the fault injection scenario file.
4. Injects the specified fault pattern into the selected neural network tensor.
5. Executes YOLOP inference for the corrupted model.
6. Computes evaluation metrics comparing faulty and golden outputs.

4.4.7 Results Storage and Post-Processing

After processing all scenarios, the computed metrics are aggregated and stored in a CSV file within the output directory associated with the current fault pattern. Each row in the CSV file corresponds to a single fault injection scenario.
Following this step, an additional Python routine converts the CSV file into Excel format to facilitate easier inspection and statistical analysis.

4.4.8 CPU Resource Selection Strategy

Although the CPU Sapphire nodes provide up to 45 CPU cores, requesting all available cores for a job is not always optimal.

4.4.8.1 Scheduler Constraints

If a job requests all 45 CPUs, the SLURM scheduler must wait until all cores on a node become available simultaneously. In busy cluster environments this may significantly increase queue waiting time.

4.4.8.2 Queue Waiting Time

The total turnaround time of a job consists of both queue waiting time and execution runtime. Requesting fewer CPUs often reduces queue waiting time, allowing the job to start earlier.

4.4.8.3 Efficient Cluster Utilization

Requesting fewer CPUs improves cluster utilization and allows multiple jobs to run simultaneously. In this work, requesting approximately 28 CPUs provided a suitable balance between computational performance and scheduling efficiency.

4.4.9 Overall Execution Workflow

The complete HPC workflow used in this thesis can be summarized as follows:

1. A SLURM job is submitted for each fault pattern using the batch script.
2. SLURM schedules the job based on the requested computational resources.
3. The batch script initializes the execution environment.
4. The YOLOP fault injection framework processes the video and applies the specified fault scenarios.
5. Performance metrics are computed for each scenario.
6. Results are aggregated and saved as CSV and Excel files.
7. The generated datasets are used for statistical reliability analysis.

4.4.10 Summary

The use of SLURM and HPC infrastructure enabled efficient execution of large-scale fault injection experiments for the YOLOP perception framework. The automated batch-based workflow ensured reproducibility, efficient resource utilization, and scalable execution across multiple scenario configurations. By carefully selecting computational resources and leveraging the HPC scheduling system, the experimental campaign was completed efficiently while producing reliable datasets for subsequent statistical reliability analysis.

4.5 Statistical Analysis Choice: Classical One-Step Sampling

In [42], [64], the authors propose an iterative statistical fault injection approach in which the sample size is dynamically refined according to the observed failure rate and margin of error at each iteration.
Such methodology is particularly suitable when the following conditions hold:

- The experimental outcome is binary (success or failure).
- The experiment directly estimates a failure probability, such as the Silent Data Corruption (SDC) rate.
- Bernoulli trial assumptions are strictly satisfied.

However, the FIERA evaluates a significantly more complex output space. Specifically, the analysis includes:

- Continuous degradation metrics such as mIoU, Pearson correlation, centroid shift, and confidence drop.
- Multi-task perception outputs including Object Detection (OD), Drivable Area (DA), and Lane Line (LL).
- A composite usability decision rule derived from multiple metrics rather than a single binary outcome.

Since the evaluation space is multi-dimensional and not strictly Bernoulli distributed at the metric level, an iterative SFI stopping rule based purely on binary success probability would not adequately capture the experimental objective.
For this purpose, FIERA adopts a classical statistical inference approach based on fixed-sample size estimation[64].

$$n = \frac{Z^2 p(1-p)}{e^2} \tag{4.16}$$

Symbol	Description
n	Required sample size
Z	Standard normal value for selected confidence level
p	Assumed probability of occurrence
e	Desired margin of error

For this thesis the following parameters are selected:

Parameter	Value
Confidence Level	95%
Z value	1.96
Margin of Error	$e = 0.05$
Assumed Probability	$p = 0.5$

The assumption $p = 0.5$ represents the maximum uncertainty scenario, which produces the largest possible sample size and therefore ensures a conservative estimation.

4.6 Universe Definition, Tier-1 Tensor Selection, and Sample Size Determination

4.6.1 Sub-Universe Definition per Fault Pattern

The complete experimental space is divided into five independent sub-universes, each corresponding to one fault pattern evaluated in the framework: Single Point Fault, Row Fault, Column Fault, Bullet-Wake Fault, and Shattered-Glass Fault.

Each sub-universe contains the same number of potential scenarios since the following parameter dimensions are identical across patterns:

- Target tensors (Tier-1)
- Intensity levels
- Random seeds

Therefore, statistical sampling is performed independently for each fault pattern.

4.6.2 Tier-1 Tensor Selection Rationale

Tier-1 tensors figure 4.2 consist primarily of `conv.weight` parameters from convolutional layers. The selection of Tier-1 tensors is not based on their backbone position but on the following technical rationale:

- Convolutional weight tensors directly determine learned feature extraction.
- Perturbations in these tensors propagate through multiple downstream layers.
- They represent the dominant trainable parameter group influencing perception outputs.
- Faults in these tensors produce observable degradation across OD, DA, and LL tasks.

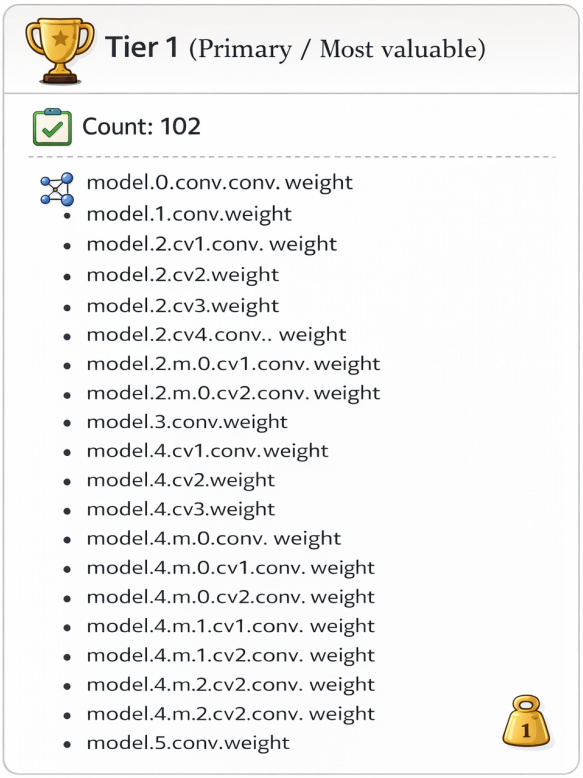


Figure 4.2
Tier1 Tensors.

4.6.2.1 Tier-2 tensors

Tier-2 tensors include BatchNorm weights and BatchNorm bias parameters.

4.6.2.2 Excluded tensors

The following parameters are excluded from the campaign:

- Running mean
- Running variance
- Tracking counters

BatchNorm running statistics are excluded because they are not directly analogous to stored weight memory elements under the chosen fault model.

4.6.3 Population Size for One Fault Pattern

Each fault-pattern sub-universe is defined by three independent parameters:

Parameter	Value
Tier-1 tensors in YoloP	102
Intensity levels	100
Random seeds	20

$$N = 102 \times 100 \times 20 \tag{4.17}$$

$$N = 204{,}000 \tag{4.18}$$

$$N_{pattern} = 204{,}000 \tag{4.19}$$

4.6.4 Sample Size Calculation

$$n = \frac{Z^2 p(1-p)}{e^2} \tag{4.20}$$

$$n = \frac{(1.96)^2 \times 0.5(1-0.5)}{(0.05)^2} \tag{4.21}$$

$$n = \frac{3.8416 \times 0.25}{0.0025} \tag{4.22}$$

$$n = 384.16 \approx 385 \tag{4.23}$$

Therefore the required representative sample per fault pattern is:

$$n = 385 \tag{4.24}$$

Since five independent fault patterns are evaluated:

$$n_{total} = 5 \times 385 \tag{4.25}$$

$$n_{total} = 1,925 \tag{4.26}$$

Thus, 1,925 statistically representative scenarios are executed across all fault patterns.

4.6.5 Fixed Parameters and Fair Randomness

To ensure fair comparison across fault patterns and tensor groups, several framework parameters are kept constant throughout the campaign.

Table 4.6
Fixed Parameters (Section 4.6.5.1)

Parameter Name	Description
repeats	Number of times the injection is repeated.
keep_prob	Probability of keeping original values.
row_span	Range of rows affected by the operation.
inject_every	Frequency of the fault injection.
min_seg_len	Minimum segment length for the operation.
fault_start	Starting point of the fault injection.
fault_end	Ending point of the fault injection.
Gaussian mean	The mean value for Gaussian distribution faults.

4.6.5.1 Variable parameters

Table 4.7
Variable Parameters (Section 4.6.5.2)

Parameter Name	Description
target_tensor	The specific tensor to which the fault is applied.
fault_type	The category or nature of the fault being injected.
intensity	The magnitude or strength of the fault.
seed	Random seed for reproducibility of the results.

The scenario generation script `GenerateScenarios.py` guarantees:

- Uniform sampling across the Tier-1 tensor list
- Controlled randomness through seed specification
- Deterministic cycling of tensor order
- Reproducibility of experiments

This design prevents overrepresentation of specific tensors, clustering of intensity levels, and bias toward specific seed values.

4.7 Statistical Results and Interpretation

This chapter explains the results produced by the statistical fault injection campaign using the Excel workbook `ShatteredGlass_SFI_Approach.xlsx`.

4.7.1 Workbook Structure

Sheet	Description
Data	Scenario-level dataset with metrics and failure flags
Stats	Statistical estimation of failure probability
Documentation	Notes on formulas and compatibility
Tensors Vulnerability	Tensor ranking based on failure statistics

4.7.2 Failure Rate Estimation

The estimated failure probability is

$$\hat{p} = \frac{x}{n} \tag{4.27}$$

The measured margin of error is

$$\hat{e} = z\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \cdot FPC \tag{4.28}$$

where the finite population correction factor is

$$FPC = \sqrt{\frac{N-n}{N-1}} \tag{4.29}$$

The confidence interval is

$$[\max(0, \hat{p} - \hat{e}), \min(1, \hat{p} + \hat{e})] \tag{4.30}$$

The stopping rule is defined as

$$\text{STOP if } \hat{e} \leq E_{goal} \tag{4.31}$$

4.7.3 Failure flags (binary usability evaluation)

For each metric, the workbook assigns a threshold-based failure indicator as discussed in Section 4.3.

Table 4.8
Failure Flags Used for Binary Usability Evaluation

Category	Failure Indicators	
Segmentation (Drivable Area)	Fail_mean_iou_da	Fail_mean_pearson_da
	Fail_mean_pos_shift_da	Fail_mean_area_diff_da
Segmentation (Lane Line)	Fail_mean_iou_ll	Fail_mean_pearson_ll
	Fail_mean_pos_shift_ll	Fail_mean_area_diff_ll
Object Detection Stability	Fail_det_consistency_rate	Fail_mean_det_conf_drop
	Fail_mean_det_iou	

Finally, **Fail_ANY = 1** if at least one failure flag is triggered for that scenario.
This is the "overall unusable" label used in the statistical estimation.

Chapter 5

EXPERIMENTAL RESULTS

5.1 Overview of the Chapter

This chapter presents the results of the experimental campaign carried out with the proposed FIERA framework and is organized to move from scenario-level evidence to cross-scenario comparative analysis. The first part focuses on selected video-based scenario the results obtained from the fault injection campaign conducted using the proposed FIERA framework. The experiments analyze the reliability of the YOLOP perception model under controlled perturbations injected into its weight tensors, where three representative fault patterns are discussed in detail: Single Point as the low-severity reference case, and Bullet Wake and Shattered Glass as the two most severe structured fault patterns. For these patterns, the chapter reports the statistical results, representative visual comparisons, and the corresponding tensor-level observations for the **City Driving scenario** which comes from the **Samples of YOLOP model**. The remaining detailed results for **Lane Change (LCH), CUT-OUT (CUTOUT), CUT-IN (CUTIN), and Accelerate and Lane Change (ACLH)** Driving Scenarios (were generated from the **SCANer Tool**) are provided in the Appendix, while the main chapter concentrates on the most representative cases and their interpretation.

The second part of the chapter performs the **fault-pattern based analysis**, in which the results of all five fault error shapes/geometries are compared across all five driving scenarios through summary tables and aggregated rankings. This section highlights how the severity of degradation changes with the spatial organization of the injected corruption and how different driving contexts respond to the same fault pattern.

The third part presents the **metrics-based and task-based analysis**, to identify which evaluation metrics are the most sensitive to tensor perturbations and which perception task—Drivable Area, Lane Line, or Object Detection—is most affected under the studied fault conditions. In this way, the chapter progresses from showing representative scenario outcomes, to comparing the global severity of fault patterns, and finally to examining the sensitivity of individual metrics and perception tasks. Throughout the chapter, the focus remains on presenting and interpreting the observed results, while the final synthesis and broader implications are reserved for the dedicated conclusion chapter that follows.

5.2 Statistical Analysis Results for Reliability Assessment

The results presented in this chapter are derived from the statistical dataset generated during the experimental campaign and stored in the corresponding Excel workbooks. These datasets include scenario level measurements, statistical estimations of failure probability, and tensor level vulnerability rankings derived from the evaluation metrics. The analysis combines statistical reliability estimation, graphical visualization of failure trends, tensor vulnerability ranking, and qualitative inspection of perception outputs through video comparison.

5.2.1 Statistical Validation and Stopping Criterion

The first step in the results analysis is to verify that the experimental campaign satisfies the statistical requirements defined in the methodology. As described in the previous chapter, the number of executed scenarios was determined using a classical statistical sampling approach.
The sampling process was configured using a confidence level of ninety five percent and a target margin of error of five percent as shown in the Table 5.1. Using these parameters, the classical sample size formula was applied to estimate the number of scenarios required for each fault pattern. The resulting campaign produced a representative dataset that satisfies the predefined stopping criterion once the measured margin of error falls below the target threshold.

The statistical validation confirms that the collected experimental data provides reliable estimates of system failure probability within the defined experimental universe. The workbook sheet dedicated to statistical estimation computes the estimated failure rate, the measured margin of error, and the associated confidence interval for each monitored metric. Once the calculated margin of error becomes smaller than the target threshold, the campaign is considered statistically sufficient and the experiments are stopped.

Table 5.1
Statistical description of the driving scenarios and number of planned experiments per scenario (n).

Data	Values
Population_N (N)	204,000
Sample_n (rows)	385
Confidence Level $(1-\alpha)$	0.95
t_{crit} $(z_{\alpha/2})$	1.959963985
Target_MOE (Egoal)	0.05
Assumed p (planning)	0.5
Planned n (Eq.4)	383.4257448
FPC = sqrt((N-n)/(N-1))	0.999058376

5.2.2 Failure Metrics and Binary Usability Evaluation

To evaluate whether a fault injection scenario leads to an unacceptable degradation of perception performance, threshold based failure indicators are defined for each monitored metric. These thresholds convert continuous metric values into binary failure flags.

Metric_Label	Data_Column	Direction	Threshold
Seg IoU – Drivable Area	mean_iou_da	<	0.9
Seg IoU – Lane Line	mean_iou_ll	<	0.9
Pearson – Drivable Area	mean_pearson_da	<	0.85
Pearson – Lane Line	mean_pearson_ll	<	0.85
Area RelDiff – Drivable Area	mean_area_diff_da	>	0.1
Area RelDiff – Lane Line	mean_area_diff_ll	>	0.1
Pos Shift – Drivable Area	mean_pos_shift_da	>	0.05
Pos Shift – Lane Line	mean_pos_shift_ll	>	0.05
Detection Consistency Rate	det_consistency_rate	<	0.9
Detection Mean IoU	mean_det_iou	<	0.9
Detection Mean Conf Drop	mean_det_conf_drop	>	0.15

If any of these metrics violates its predefined threshold as discussed in section **??**, the corresponding failure indicator is triggered. These individual failure flags are then combined into a global usability indicator denoted as **Fail_ANY**, which represents whether the perception system becomes unusable for that scenario.

5.3 Graphical Analysis and Interpretation of Experimental Results

Following the definition of the failure metrics and usability criteria described in the previous section, the remaining analysis of the experimental campaign is primarily presented through graphical representations derived from the statistical dataset. The purpose of these graphs is to visualize the failure behavior of the perception system under the five evaluated fault patterns and to identify the internal model components that exhibit the highest sensitivity to injected perturbations.
First, the distribution of metric failures is analyzed by plotting the relationship between the evaluated metrics and the number of observed failures across the executed scenarios. These graphs provide an overview of how frequently each metric violates its corresponding usability threshold and therefore indicate which perception properties are most affected by internal parameter perturbations.
The experimental results are then used to evaluate the vulnerability of the model's weight tensors. Using the statistical dataset, tensors are ranked according to their observed failure behavior using two complementary evaluation approaches. The first method evaluates vulnerability using the global usability indicator Fail_ANY, which represents the occurrence of an overall system failure when at least one metric violates its threshold. The second method analyzes failures at the individual metric level, allowing the identification of tensors that are particularly sensitive to specific perception degradations. Based on this analysis, the top ranked vulnerable tensors are visualized through dedicated graphs.

In addition to tensor ranking, the experimental results are also analyzed separately for the three perception tasks performed by the YOLOP network: object detection, drivable area segmentation, and lane line estimation. This separation allows the identification of which perception components are most exposed to failures under the injected fault conditions.

Finally, qualitative observations are obtained through visual comparison of the inference of unmasked Driving Scenario (Figure 5.1) generated during the experiments. These masked videos (Figure 5.5) comparisons provide additional insight into how the injected perturbations propagate through the neural network and affect the perception outputs in real driving scenes.

All analyses described in this section are repeated for each of the five evaluated fault patterns and across the different driving video scenarios used in the experimental campaign. The results are summarized through graphical comparisons that allow the identification of the fault pattern that produces the most significant reliability degradation in the perception system.

5.3.1 City Driving Scenario

Figure 5.1
A general overview of a frame in the City Drive Scenario.

5.3.1.1 Single Point Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	12	385	0.0312	0.0173	0.0138	0.0485	STOP	46	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Area RelDiff – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Area RelDiff – Lane Line	10	385	0.0260	0.0159	0.0101	0.0418	STOP	39	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	11	385	0.0286	0.0166	0.0119	0.0452	STOP	43	0

Figure 5.2
Single Point Stop Criterion

The Table in Figure 5.2 shows that the margin of error is smaller than the target threshold and the **STOP Criterion is met**, the campaign is considered statistically sufficient, and the experiments are stopped. Furthermore, the statistical Analysis was done twice with 2nd order Scenarios.txt for all the Fault Pattern and they meet the same STOP criterion. The per-metric statistical results show that most metrics did not violate their defined thresholds, indicating stable behavior of the perception outputs under the evaluated perturbations.
Among the evaluated metrics, Lane Line IoU, Lane Line Area Difference, and Lane Line Pearson correlation recorded the highest number of failures, highlighting the higher sensitivity of lane-line segmentation compared to drivable area and detection metrics.
All detection-related metrics remained within acceptable limits across the sampled scenarios. When applying the combined usability rule (Fail_ANY), 11 failures were observed out of 385 scenarios, resulting in an estimated overall failure probability of 2.86%. This indicates that only a small fraction of scenarios produced outputs classified as unusable under the defined thresholds.

Top Vulnerable Tensors for Single-Point Fault Injection: The statistical analysis was further used to identify the **top 20 most vulnerable weight tensors** as shown in the Table of Figure 5.3 under the single-point fault injection pattern. The ranking was based on the estimated failure probability $p_{\text{hat\_any}}$, which represents the proportion of scenarios where at least one metric violated its defined threshold. Among the analyzed tensors, **model.40.cv1.conv.weight** showed the highest vulnerability with a failure probability of **1.0**, followed by **model.40.cv3.weight**, **model.40.cv4.conv.weight**, and **model.40.m.0.cv1.conv.weight**, each with a probability of **0.67**. These tensors mainly exhibited failures associated with **lane-line segmentation metrics**, particularly the area difference and Pearson correlation. The remaining tensors showed lower failure probabilities or no observed failures within the sampled scenarios, indicating relatively higher robustness to single-point perturbations.

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.cv1.conv.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.cv3.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.40.cv4.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.666666667	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.31.cv4.conv.weight	3	1	0.333333333	0	0	0	0	0.333333	0	0	0	0	0	0
model.42.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5.3
SP Top 20 Vulnerable Tensors

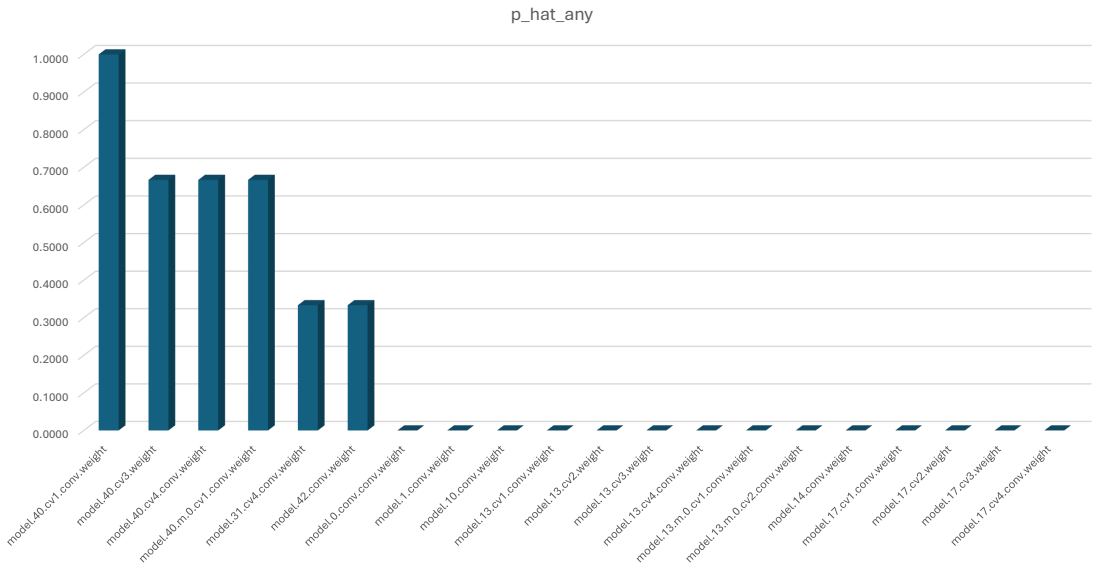


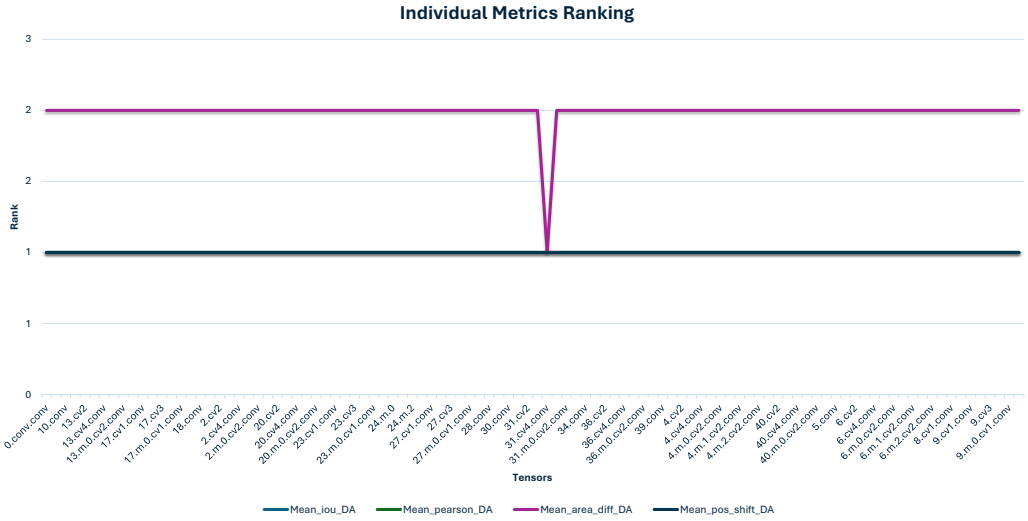
Figure 5.4
SP Bar Graph

Ranking of Top 20 Vulnerable Tensors: The Bar Graph 5.4 shows the Ranking of Top 20 Vulnerable Tensors. From the Graph we see that model.40.cv1.conv.weight with Rank 1 is most Vulnerable weight Tensor that was affected by Single Point Fault Injection.

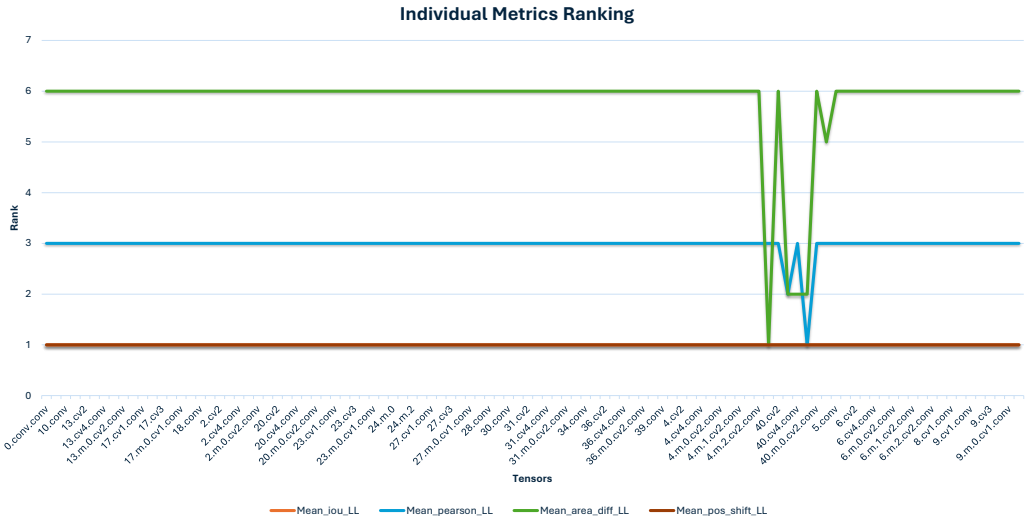
Ranking of Tensors Based on Individual Metrics: To further investigate the sensitivity of different tensors, the results of the fault injection experiments were analyzed separately for each evaluation metric Charts 5.5. These Charts are divided in a Task Based Charts each belonging to metrics related to Drivable Area, Lane Line and Object Detection. The x axis of the charts consists of Tensors whose suffix (.weight) and prefix (model.) are truncated for better readability. For every metric, the failures observed during the experiments were aggregated for each tensor to determine how frequently a specific tensor contributed to a violation of that metric.
Based on these observations, a probability of failure was computed for each tensor with respect to the corresponding metric. The tensors were then ranked in descending order.

- **Rank 1 (The Dip):** This identifies the single tensor that is most vulnerable to fault injection.
- **Rank 2 (The Flat Line):** A higher number indicates lower vulnerability. In this case, all other tensors tied for second place for Drivable Area, Sixth place for Lane Line and first place for Object Detection, meaning they share the exact same, lower rate of failure.

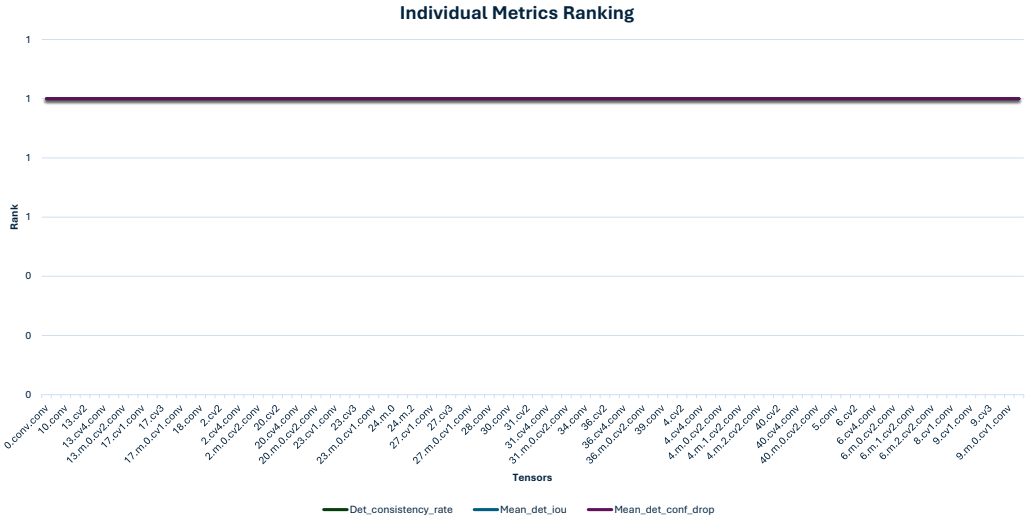
In subfigure 5.5a the vulnerable Tensor is **31.cv4.conv**, while in 5.5b **40.cv1.conv** is the most vulnerable Tensor with Rank1 and finally in 5.5b none of the Tensors is vulnerable. Using this ranking method, the most vulnerable tensors are identified for each metric individually. This analysis provides deeper insight into how faults affecting specific tensors influence different perception tasks of the network.



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure 5.5
SP Task Based Charts

City Drive Video Comparison Single Point Fault

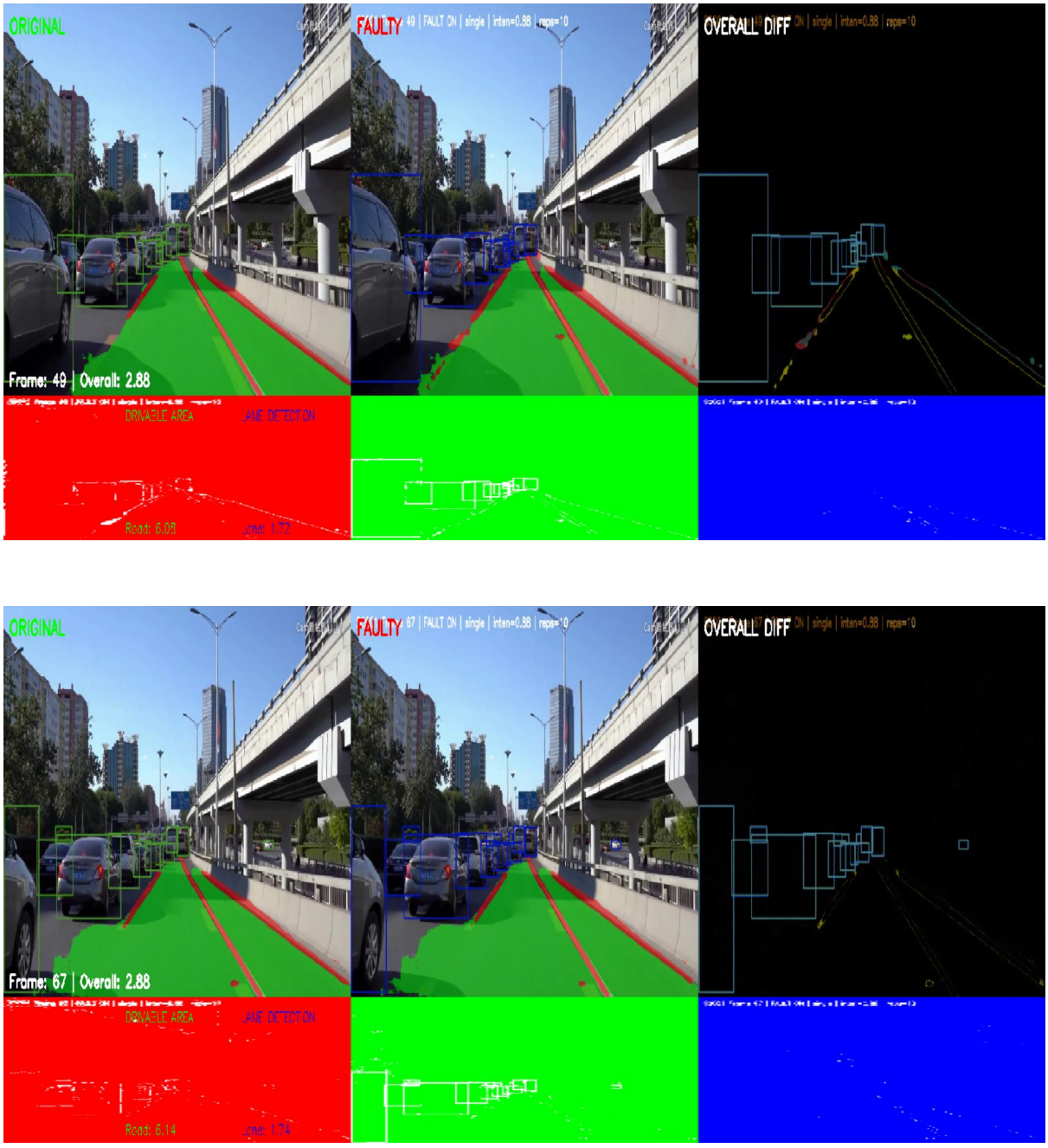


Figure 5.6
SP CTD Comparison

From the Figure 5.6 it is observed that the Lane Line was affected from the Single Point Fault Injection. The object detection always shows a difference in position which is negligible and the Drivable Area is Robust in this case.

5.3.1.2 Bullet Wake Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	17	385	0.0442	0.0205	0.0237	0.0647	STOP	65	0
Seg IoU – Lane Line	73	385	0.1896	0.0391	0.1505	0.2287	STOP	236	0
Pearson – Drivable Area	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Pearson – Lane Line	23	385	0.0597	0.0237	0.0361	0.0834	STOP	86	0
Area RelDiff – Drivable Area	6	385	0.0156	0.0124	0.0032	0.0279	STOP	24	0
Area RelDiff – Lane Line	37	385	0.0961	0.0294	0.0667	0.1255	STOP	133	0
Pos Shift – Drivable Area	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pos Shift – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Detection Consistency Rate	22	385	0.0571	0.0232	0.0340	0.0803	STOP	83	0
Detection Mean IoU	12	385	0.0312	0.0173	0.0138	0.0485	STOP	46	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	42	385	0.1091	0.0311	0.0780	0.1402	STOP	149	0

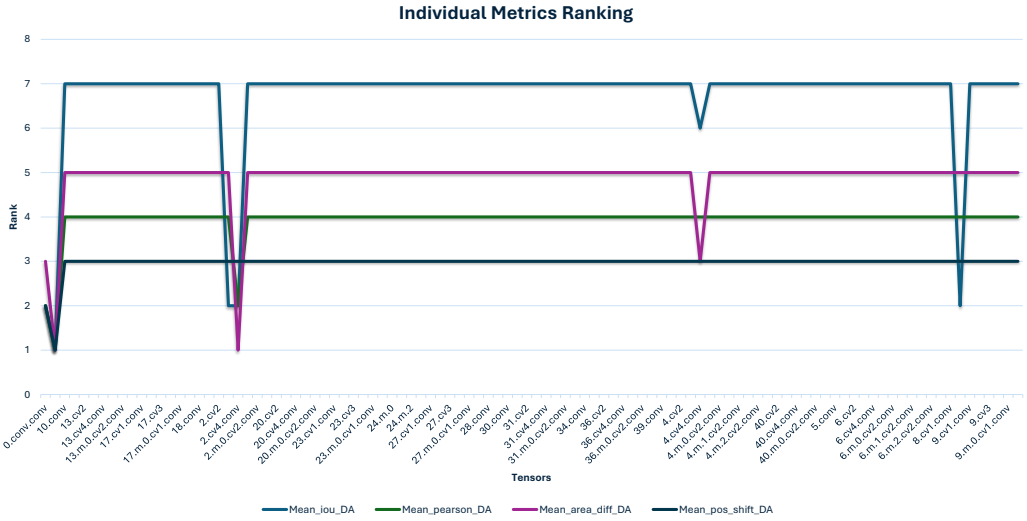
Figure 5.7
Bullet Wake Stop Criterion

The Table in Figure 5.7 shows that the margin of error is smaller than the target threshold and the **STOP Criterion is met**, the campaign is considered statistically sufficient, and the experiments are stopped. Furthermore, the statistical Analysis was done twice with 2nd order Scenarios.txt for all the Fault Pattern and they meet the same STOP criterion. The per-metric statistical results show that most metrics did not violate their defined thresholds, indicating stable behavior of the perception outputs under the evaluated perturbations.
All the metrics were affected with number of failures, highlighting the higher sensitivity of lane-line segmentation compared to drivable area and detection metrics. When applying the combined usability rule (Fail_ANY), 42 failures were observed out of 385 scenarios, resulting in an estimated overall failure probability of 10.91%. This indicates that only a small fraction of scenarios produced outputs classified as unusable under the defined thresholds.

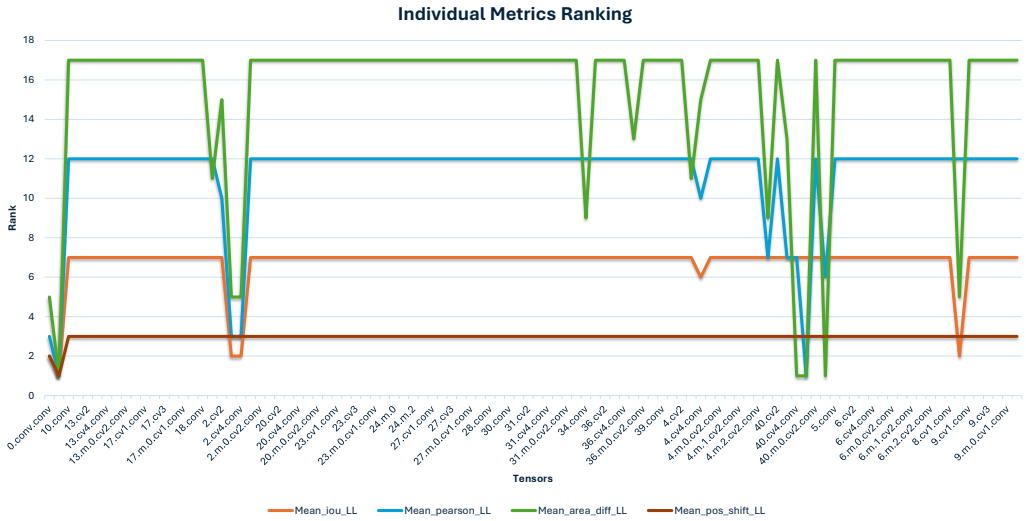
Top Vulnerable Tensors for Bullet Wake Fault Injection: From the Figure 5.8 it is observed that there are several ties in rank 1 which shows the top vulnerable Tensors affected with BW Fault injection.

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.1.conv.weight	4	4	1.0000	1.0000	1.0000	0.5000	1.0000	0.5000	1.0000	0.5000	0.5000	1.0000	0.7500	0.0000
model.40.cv4.conv.weight	3	3	1	0	0	0	0.333333	0	1	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	3	1	0	0	0	1	0	1	0	0	0	0	0
model.42.conv.weight	3	3	1	0	0	0	0.666667	0	1	0	0	0	0	0
model.0.conv.conv.weight	4	3	0.75	0.75	0.75	0.25	0.75	0.25	0.75	0.25	0.25	0.75	0.25	0
model.2.cv3.weight	4	3	0.75	0.75	0.75	0	0.75	0	0.75	0	0	0.75	0.25	0
model.2.cv4.conv.weight	4	3	0.75	0.75	0.75	0.25	0.75	0.5	0.75	0	0	0.75	0.75	0
model.8.cv2.conv.weight	4	3	0.75	0.75	0.75	0	0	0	0.75	0	0	0.75	0	0
model.34.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.cv1.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.2.cv1.conv.weight	4	2	0.5	0	0	0	0	0	0.5	0	0	0.5	0	0
model.24.m.2.weight	4	2	0.5	0	0	0	0	0	0	0	0	0	0.5	0
model.3.conv.weight	4	2	0.5	0	0	0	0	0	0	0	0	0.5	0	0
model.4.cv3.weight	4	2	0.5	0	0	0	0	0	0.5	0	0	0	0	0
model.36.m.0.cv1.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.40.cv3.weight	3	1	0.333333333	0	0	0	0.333333	0	0.333333	0	0	0	0	0
model.2.cv2.weight	4	1	0.25	0	0	0	0.25	0	0.25	0	0	0.25	0	0
model.24.m.1.weight	4	1	0.25	0	0	0	0	0	0	0	0	0	0.25	0
model.4.cv4.conv.weight	4	1	0.25	0.25	0.25	0	0.25	0.25	0.25	0	0	0.25	0.25	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

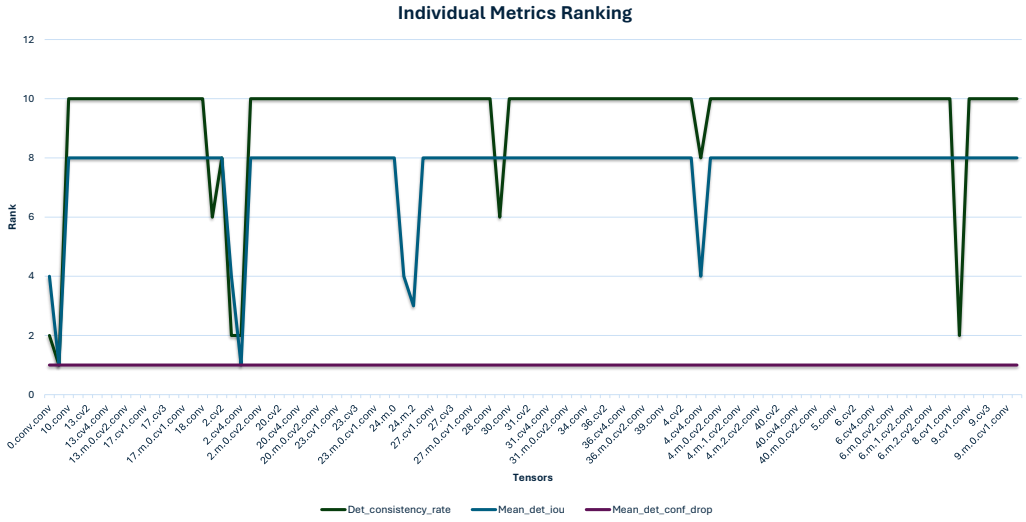
Figure 5.8
BW Top 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure 5.9
BW Task Based Charts

In subfigure 5.9a it is observed that Mean_iou_DA, Mean_pearson_DA and Mean_pos_shift_DA has vulnerable Tensor 1.conv while in Mean_area_diff_DA has a tie of rank 1 between 1.conv and 2.cv4.conv. Similarly we see Vulnerable Tensors in Subfigure 5.9b and 5.9c

Figure 5.10 illustrates a comparison of Golden vs Faulty Videos on Frame 42 and 64. On frame 42 it is observed that the BW Fault injection heavily degraded the AI perception of Lane line and Drivable area. The Green color of the (DA) is masked on the (LL) and the Red Color of (LL) is masked on the whole frame. There is a slight Shift in the Object detection bounding Box. On Frame 64 there is a slight degradation in (DA) and a slight shift in Bounding boxes of Object Detection.

City Drive Video Comparison Bullet Wake Fault

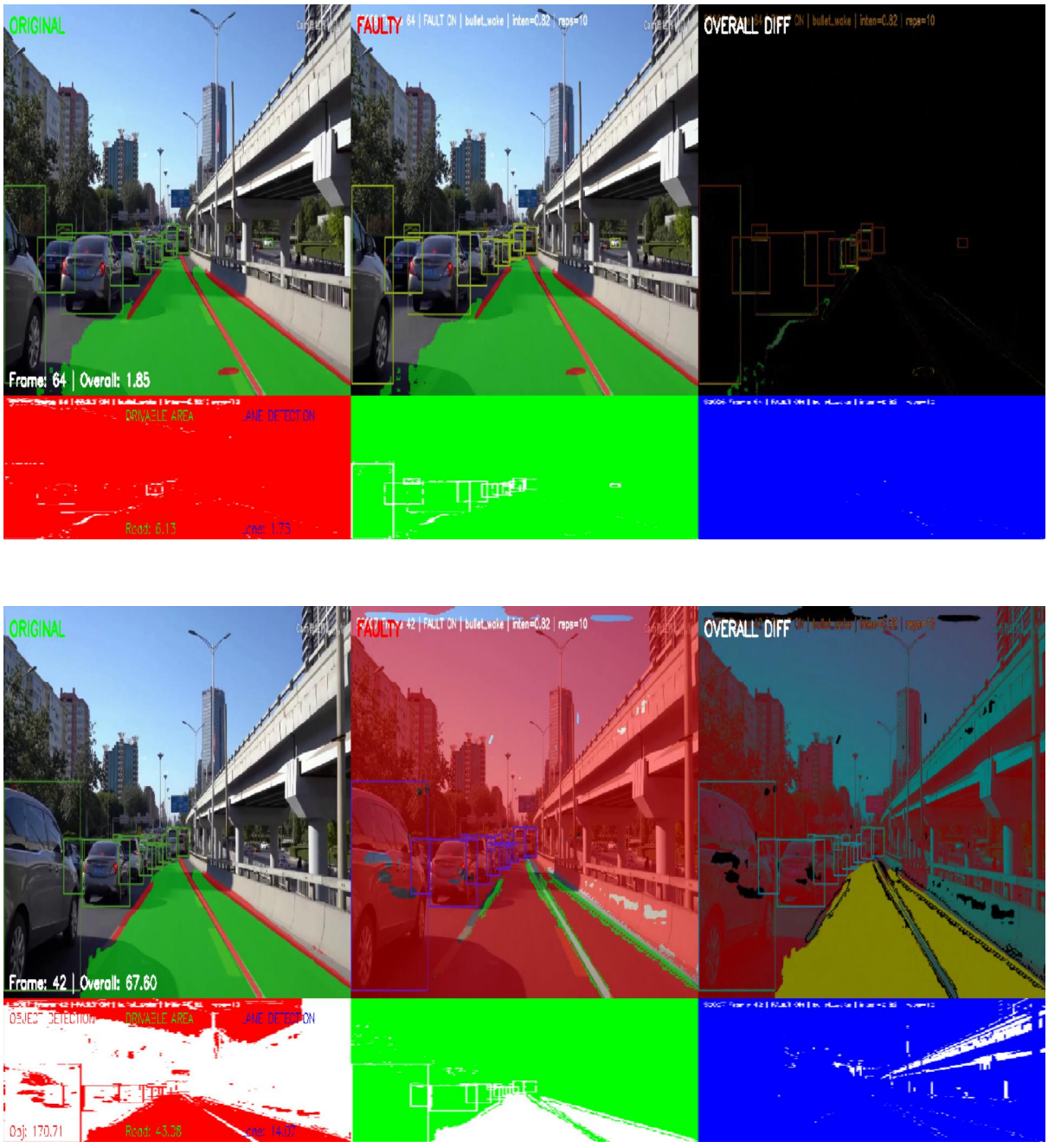


Figure 5.10
BW CTD Comparison

5.3.1.3 Shattered Glass Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	16	385	0.0416	0.0199	0.0216	0.0615	STOP	61	0
Seg IoU – Lane Line	68	385	0.1766	0.0381	0.1386	0.2147	STOP	223	0
Pearson – Drivable Area	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pearson – Lane Line	19	385	0.0494	0.0216	0.0277	0.0710	STOP	72	0
Area RelDiff – Drivable Area	14	385	0.0364	0.0187	0.0177	0.0550	STOP	54	0
Area RelDiff – Lane Line	34	385	0.0883	0.0283	0.0600	0.1166	STOP	124	0
Pos Shift – Drivable Area	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Pos Shift – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Detection Consistency Rate	20	385	0.0519	0.0221	0.0298	0.0741	STOP	76	0
Detection Mean IoU	14	385	0.0364	0.0187	0.0177	0.0550	STOP	54	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	44	385	0.1143	0.0318	0.0825	0.1460	STOP	155	0

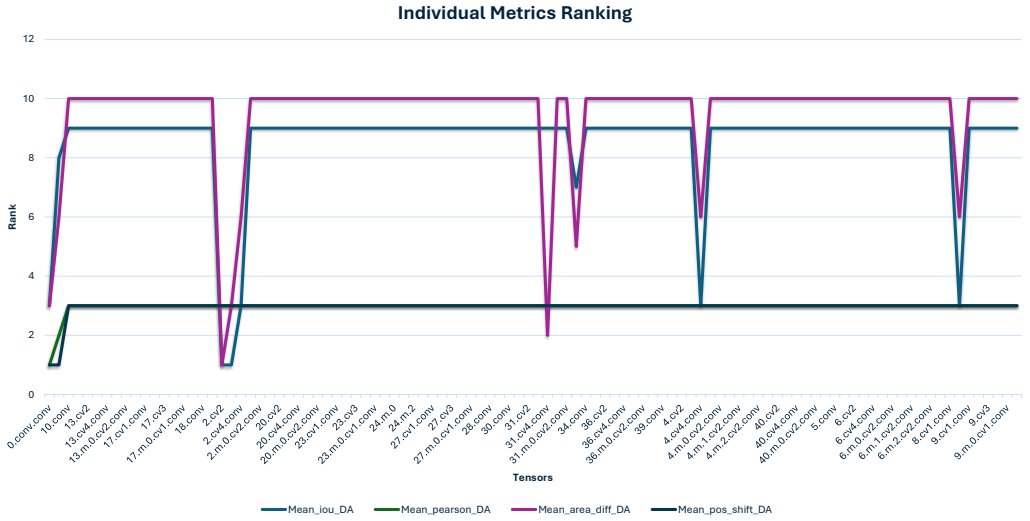
Figure 5.11
Shattered Glass Stop Criterion

The Table in Figure 5.11 shows that the margin of error is smaller than the target threshold and the **STOP Criterion is met**, the campaign is considered statistically sufficient, and the experiments are stopped. Furthermore, the statistical Analysis was done twice with 2nd order Scenarios.txt for all the Fault Pattern and they meet the same STOP criterion.
The per-metric statistical results show that most metrics did not violate their defined thresholds, indicating stable behavior of the perception outputs under the evaluated perturbations.
All the metrics excluding Detection_Mean_Conf_Drop were affected with number of failures, highlighting the higher sensitivity of lane-line segmentation compared to drivable area and detection metrics. When applying the combined usability rule (Fail_ANY), 44 failures were observed out of 385 scenarios, resulting in an estimated overall failure probability of 11.43%. This indicates that only a small fraction of scenarios produced outputs classified as unusable under the defined thresholds.

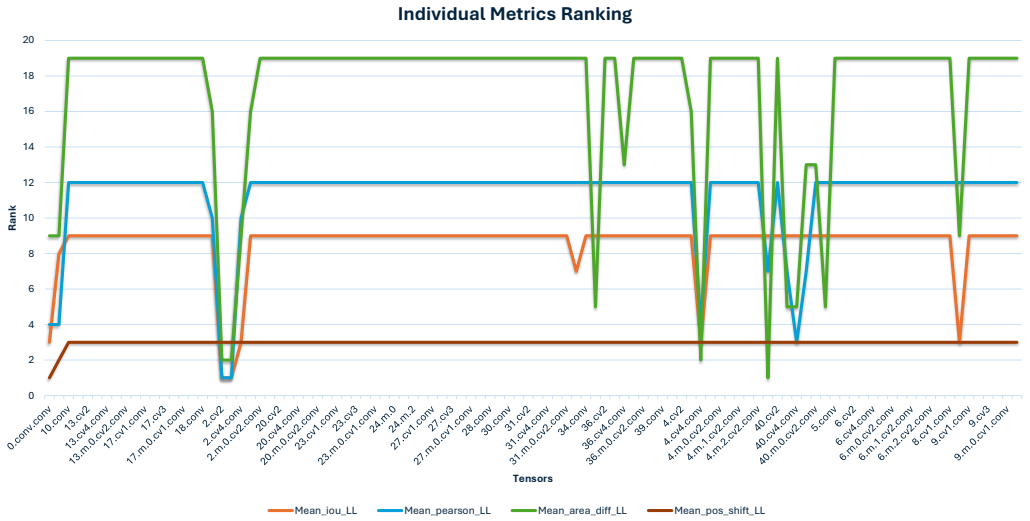
Top Vulnerable Tensors for Shattered Glass Fault Injection: The Bar Graph 5.12 shows the Ranking of Top 20 Vulnerable Tensors. From the Graph we see that the most vulnerable Tensor is model.40.cv1.conv.weight ranked 1, along with other vulnerable tensors with lower ranking.

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.cv1.conv.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.3333	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.2.cv2.weight	4	3	0.75	0.75	0.75	0	0.75	0.75	0.75	0	0	0.75	0.75	0
model.2.cv3.weight	4	3	0.75	0.75	0.75	0	0.75	0.5	0.75	0	0	0.75	0.5	0
model.24.m.1.weight	4	3	0.75	0	0	0	0	0	0	0	0	0	0.75	0
model.4.cv4.conv.weight	4	3	0.75	0.5	0.5	0	0.5	0.25	0.75	0	0	0.5	0	0
model.31.cv4.conv.weight	3	2	0.666666667	0	0	0	0	0.666667	0	0	0	0	0	0
model.36.cv1.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.cv3.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.40.cv4.conv.weight	3	2	0.666666667	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.0.conv.conv.weight	4	2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.25	0.5	0.5	0.5	0
model.1.conv.weight	4	2	0.5	0.25	0.25	0.25	0.5	0.25	0.5	0.25	0.25	0.5	0.25	0
model.2.cv4.conv.weight	4	2	0.5	0.5	0.5	0	0.25	0.25	0.5	0	0	0.5	0.25	0
model.2.m.0.cv2.conv.weight	4	2	0.5	0	0	0	0	0	0	0	0	0.5	0	0
model.24.m.2.weight	4	2	0.5	0	0	0	0	0	0	0	0	0	0.5	0
model.8.cv2.conv.weight	4	2	0.5	0.5	0.5	0	0	0.25	0.5	0	0	0.5	0	0
model.33.conv.weight	3	1	0.333333333	0.333333333	0.333333	0	0	0.333333	0	0	0	0	0	0
model.36.cv4.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	1	0.333333333	0	0	0	0.333333	0	0.333333	0	0	0	0	0
model.40.m.0.cv2.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0

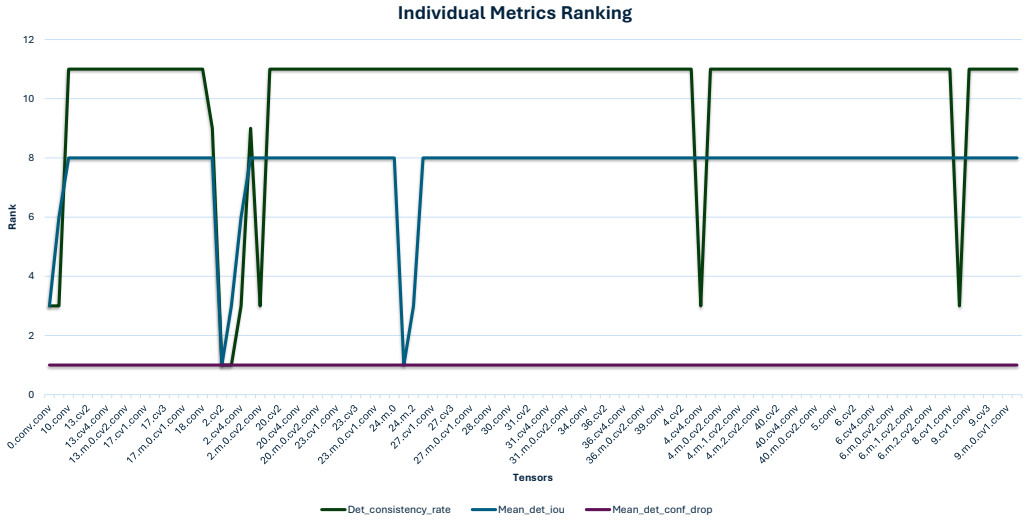
Figure 5.12
SG Top 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure 5.13
SG Task Based Charts

City Drive Video Comparison Shattered Glass Fault

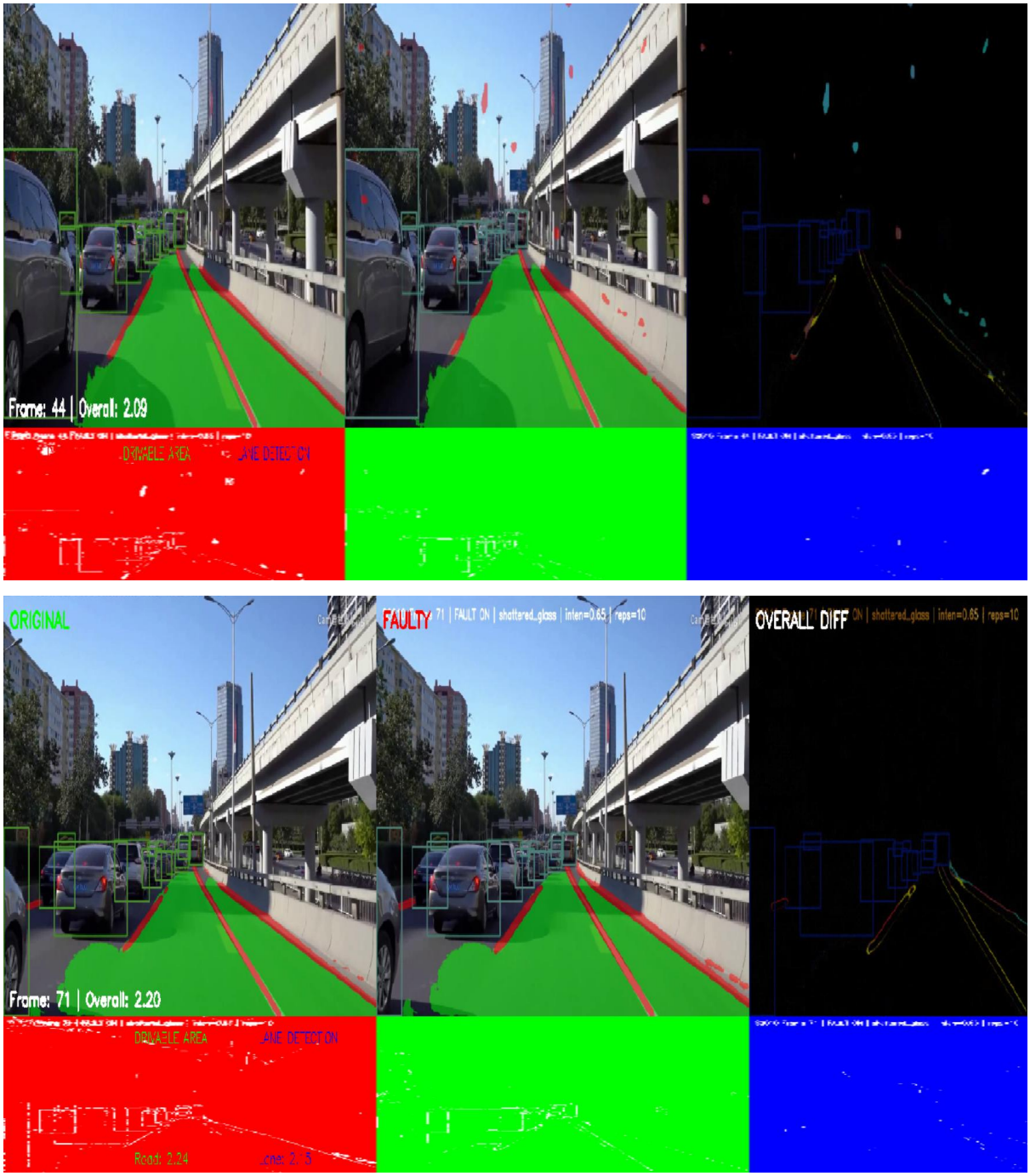


Figure 5.14
SG CTD Comparison

Figure 5.14 illustrates a comparison of Golden vs Faulty Videos on Frame 44 and 71. On frame 44 it is observed that the SG Fault injection degraded the AI perception of Lane line. Some Red Color dots of (LL) is masked on the frame. There is a slight Shift in the Object detection bounding Box. On Frame 71 there is a slight shift in Bounding boxes of Object Detection and Lane Line perception.

5.4 Fault Pattern Based Analysis

This section compares the five evaluated fault patterns across all driving scenarios in order to examine how the spatial organization of corruption influences the overall severity of perception degradation. The comparison is performed at three complementary levels. First, the *overall failure analysis* compares the probability that a scenario becomes unusable according to the combined binary usability rule, where a scenario is classified as failed when at least one monitored metric violates its threshold.

Second, a *task-balanced failure analysis* is introduced to verify whether the same trend remains visible when the degradation is aggregated separately over the three perception tasks, namely Drivable Area, Lane Line, and Object Detection.

Third, the *rank-1 tensor analysis* is used to identify which tensors emerge as the most vulnerable for each fault-pattern and driving-scenario combination, together with a frequency-based summary of how often these tensors recur across the complete experimental matrix.

5.4.1 Overall Failure Comparison Across Fault Patterns

The first level of comparison is based on the overall failure probability of each fault pattern across the five driving scenarios as shown in Table 5.15. This view captures the global severity of the injected perturbations and provides the clearest summary of how likely each fault geometry is to make the perception output unacceptable under the adopted usability rule. In this representation, each row corresponds to one fault pattern and each column corresponds to one driving scenario, while the final columns report the mean score and the associated ranking. This table therefore serves as the primary global comparison of fault-pattern severity.

1) Overall Failure Analysis for all videos considering p_hat_any

Fault	CTD	LCH	CUTOUT	CUTIN	ACLH	Mean	Std	Rank
SP	0.029	0.023	0.036	0.034	0.042	0.033	0.007018	3
Row	0.023	0.018	0.021	0.010	0.021	0.019	0.004996	4
Column	0.008	0.008	0.003	0.005	0.013	0.007	0.003853	5
BW	0.109	0.068	0.083	0.052	0.182	0.099	0.051031	2
S Glass	0.114	0.088	0.112	0.055	0.158	0.105	0.038112	1

Figure 5.15
Overall Fault Pattern Failure Analysis

The results show a clear separation between localized fault geometries and structured spatial corruptions. The two most severe patterns are the structured perturbations, while the more localized patterns remain substantially lower in mean severity.
The analysis shows a clear and consistent ranking of the five fault patterns. Shattered Glass is the most destructive pattern overall, followed closely by Bullet Wake. Single Point shows moderate severity, while Row is less harmful and Column is the least destructive pattern. This ranking is obtained from the overall failure analysis based on p_hat_any.
Most Affected Driving Scenario: From Table 5.15 it is clear that the (ACLH) Driving Scenario is the most vulnerable to rank 1 Fault Pattern i.e. Shattered Glass with mean value of **0.158**, followed by (CTD), (CUTOUT), (LCH) and (CUTIN)

Figure 5.16 presents the mean overall failure severity of the five fault patterns, where the comparison is based on the global failure probability considering p_hat.any. The bar heights show that Shattered Glass has the highest mean overall failure, followed very closely by Bullet Wake. In contrast, Column has the lowest mean value, while Row remains low and Single Point occupies an intermediate position. The vertical error bars represent the standard deviation across the five driving scenarios. Their magnitude indicates how much the failure severity varies from one scenario to another: larger error bars mean stronger scenario dependence, while smaller ones indicate more stable behavior. In particular, Bullet Wake and Shattered Glass not only show high mean failure severity but also larger variation, whereas Column and Row remain comparatively more consistent across scenarios.

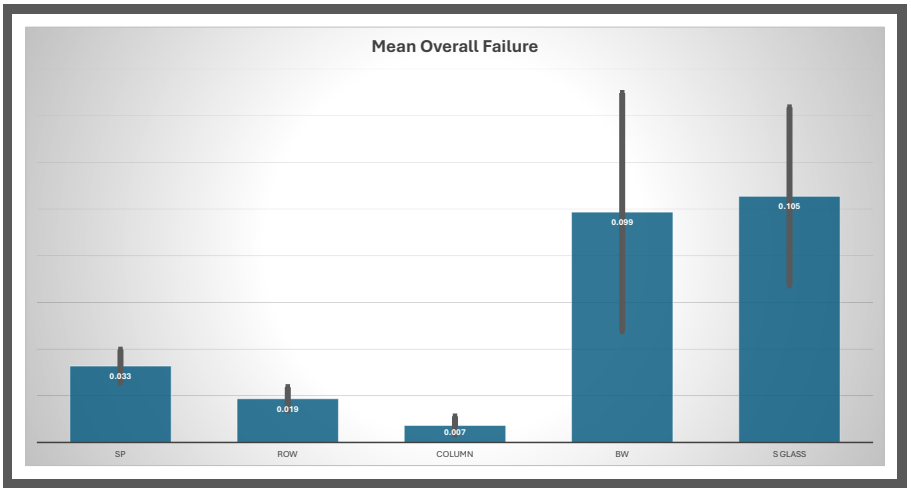


Figure 5.16
Overall Fault Pattern Failure Bar Graph

5.4.2 Task-Balanced Failure Analysis

While the overall failure table provides the most direct indication of global severity, it is also useful to verify whether the same trend remains visible when the degradation is aggregated at the level of perception tasks. For this purpose, a task-balanced severity score is computed Figure 5.17, by first grouping the metric failure probabilities into the three perception branches—Drivable Area, Lane Line, and Object Detection—and then averaging them with equal weight. In this way, the comparison does not depend only on the global usability outcome, but also reflects how broadly each fault pattern affects the

three functional components of the perception system.

2) Secondary Analysis Task Based Failure Analysis

Fault	CTD	LCH	CUTOUT	CUTIN	ACLH	Mean	Std	Rank
SP	0.0056	0.0041	0.0061	0.0056	0.0080	0.0059	0.001398	3
Row	0.0052	0.0030	0.0030	0.0019	0.0037	0.0034	0.001191	4
Column	0.0015	0.0013	0.0004	0.0009	0.0018	0.0012	0.000541	5
BW	0.0457	0.0196	0.0258	0.0164	0.0543	0.0324	0.016762	2
S Glass	0.0442	0.0259	0.0286	0.0164	0.0558	0.0342	0.015685	1

Figure 5.17
Task Based Fault Pattern Failure

The same ranking is also confirmed by the secondary task-balanced failure analysis. This means that the conclusion does not depend on only one aggregation method. Both the global usability-based analysis and the task-level analysis identify Shattered Glass and Bullet Wake as the most critical fault geometries, while Column remains the least harmful. This agreement makes the result more robust and reliable.
Figure 5.18 shows the mean task-based failure severity, where the results are aggregated across the three perception tasks: Drivable Area, Lane Line, and Object Detection. The same trend is observed again. Shattered Glass remains the most severe fault pattern, followed by Bullet Wake, while Column remains the least severe. The relative positions of Single Point and Row are also preserved, confirming that the ranking is consistent even when the degradation is measured from a task-balanced perspective rather than from the global usability rule alone.

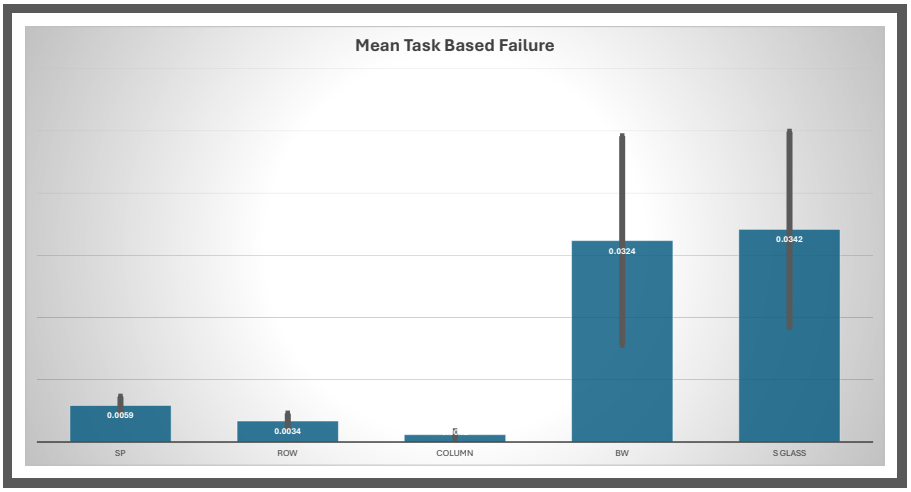
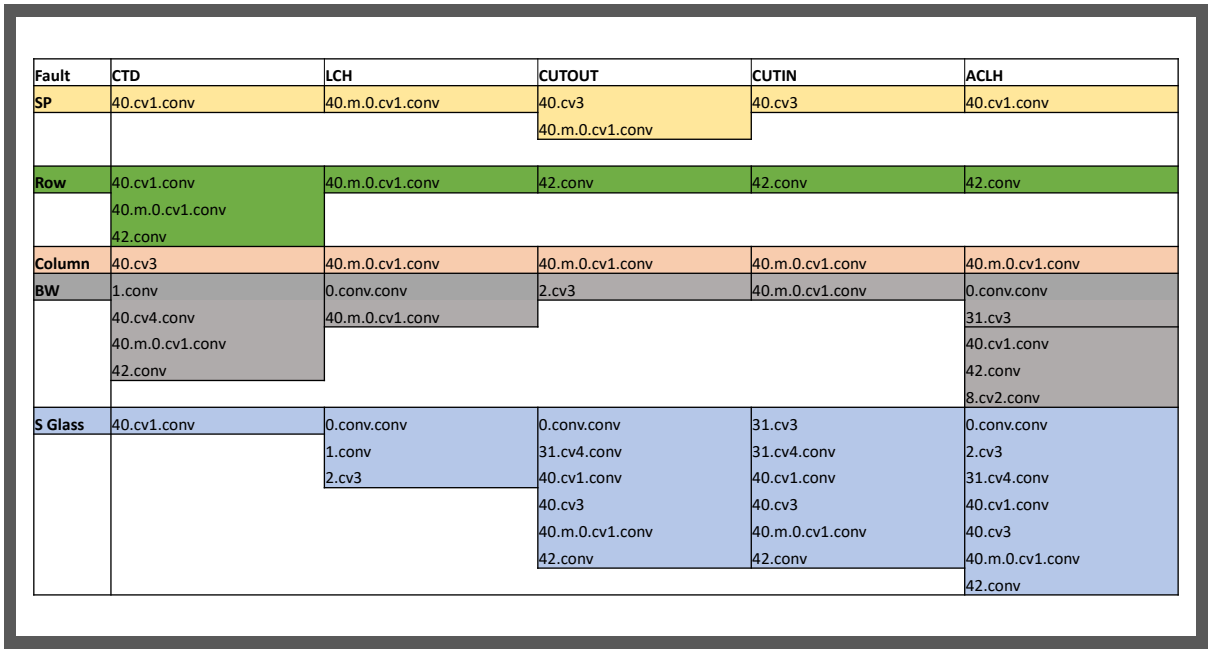


Figure 5.18
Task Based Fault Pattern Failure Bar Graph

5.4.3 Rank-1 Vulnerable Tensor Matrix

After comparing the fault patterns at the scenario and task levels, the next step is to identify which tensors become the most vulnerable under each fault-pattern and scenario combination. For this purpose, a rank-1 tensor matrix is taken from the Method-2 of Statistical Analysis workbook. The matrix typically shows that the most vulnerable tensor is not identical in all cases. Instead, the rank-1 set depends on both the fault pattern and the driving context as shown in **Figure 5.19**.



Fault	CTD	LCH	CUTOUT	CUTIN	ACLH
SP	40.cv1.conv	40.m.0.cv1.conv	40.cv3 40.m.0.cv1.conv	40.cv3	40.cv1.conv
Row	40.cv1.conv 40.m.0.cv1.conv 42.conv	40.m.0.cv1.conv	42.conv	42.conv	42.conv
Column	40.cv3	40.m.0.cv1.conv	40.m.0.cv1.conv	40.m.0.cv1.conv	40.m.0.cv1.conv
BW	1.conv 40.cv4.conv 40.m.0.cv1.conv 42.conv	0.conv.conv 40.m.0.cv1.conv	2.cv3	40.m.0.cv1.conv	0.conv.conv 31.cv3 40.cv1.conv 42.conv 8.cv2.conv
S Glass	40.cv1.conv	0.conv.conv 1.conv 2.cv3	0.conv.conv 31.cv4.conv 40.cv1.conv 40.cv3 40.m.0.cv1.conv 42.conv	31.cv3 31.cv4.conv 40.cv1.conv 40.cv3 40.m.0.cv1.conv 42.conv	0.conv.conv 2.cv3 31.cv4.conv 40.cv1.conv 40.cv3 40.m.0.cv1.conv 42.conv

Figure 5.19
Rank 1 Vulnerable Tensors in All Scenarios

The occurrence of tied rank-1 tensors is also an important observation, since it shows that multiple tensors may reach the same maximum vulnerability level within the same experimental condition.

5.4.4 Tensor Frequency Analysis

To complement the rank-1 matrix, a tensor frequency summary is used to count how often each tensor appears in the rank-1 set across the complete combination of fault patterns and videos. Figure 5.20 provides a global view of tensor recurrence and helps distinguish between tensors that are critical and tensors that repeatedly emerge as highly vulnerable across multiple conditions.

Tensor	Frequency as Rank-1
40.m.0.cv1.conv	14
42.conv	9
40.cv1.conv	8
40.cv3	6
0.conv.conv	5

Figure 5.20
Rank1 Tensors Occurence

A corresponding frequency chart as shown in **Figure 5.21** used to visualize these recurrence counts. This chart is especially effective because it condenses the entire rank-1 matrix into a single frequency-based comparison, making it easier to identify the tensors that should be regarded as globally critical from a reliability perspective.

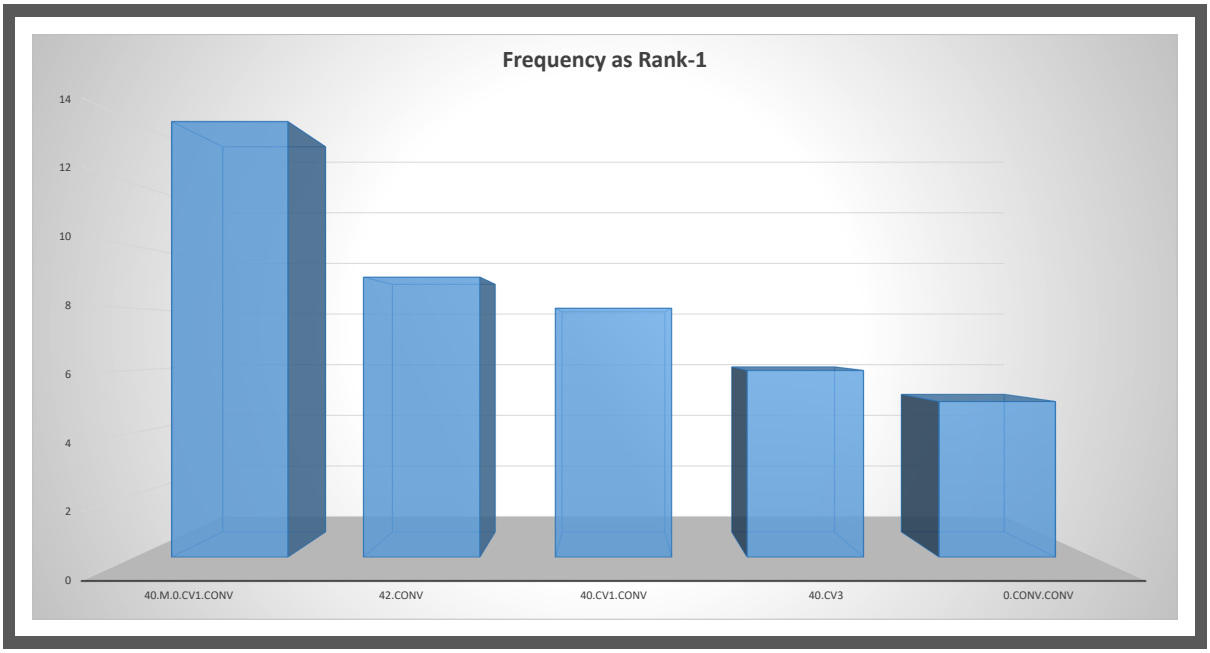


Figure 5.21
Rank1 Tensors Occurence Bar Graph

5.5 Metrics Based Analysis

The metrics-based analysis is structured in two complementary stages. The first stage examines the behavior of each evaluation metric across the five driving scenarios for every fault pattern separately. The second stage aggregates these scenario-level results into mean values in order to compare the sensitivity of the metrics across all fault patterns and to obtain an overall ranking of metric vulnerability.

5.5.1 Scenario-Level Metric Analysis

At the first stage, the analysis is carried out independently for each fault pattern. For a given fault geometry, the eleven evaluation metrics are observed across the five driving scenarios, namely CTD, LCH, CUTOUT, CUTIN, and ACLH. The value associated with each metric in a given scenario represents the corresponding failure probability. By considering these values together, it becomes possible to identify how strongly a given fault pattern activates each metric under different driving conditions.
This representation serves two purposes. First, it preserves the scenario-dependent variability of the degradation, showing how the same fault pattern may affect different perception properties with different intensity depending on the video context. Second, by averaging the scenario-level values for each metric, it provides a mean severity score that allows the metrics to be ranked within the same fault pattern. In this way, the analysis retains the detail of the individual scenarios while also identifying the dominant metrics associated with each specific fault geometry.
Within the main body of the chapter, it is appropriate to present the detailed scenario-level metric tables 5.22, 5.23 only for **Single Point** and **Shattered Glass**, as these two patterns represent the low-severity and high-severity ends of the evaluated fault spectrum. The remaining detailed scenario-level results for Row, Column, and Bullet Wake can be placed in the Appendix.

1) Single Point Fault

Metric_Label	p_hat (CTD)	p_hat (LCH)	p_hat (CUTOUT)	p_hat (CUTIN)	p_hat (ACLH)	p_hat mean	Rank
Seg IoU – Drivable Area	0.0000	0.0000	0.0026	0.0000	0.0052	0.0016	5
Seg IoU – Lane Line	0.0312	0.0208	0.0286	0.0260	0.0364	0.0286	1
Pearson – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0026	0.0005	6
Pearson – Lane Line	0.0078	0.0052	0.0078	0.0078	0.0104	0.0078	3
Area RelDiff – Drivable Area	0.0026	0.0026	0.0078	0.0078	0.0130	0.0068	4
Area RelDiff – Lane Line	0.0260	0.0208	0.0260	0.0260	0.0286	0.0255	2
Pos Shift – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	7
Pos Shift – Lane Line	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	7
Detection Consistency Rate	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	7
Detection Mean IoU	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	7
Detection Mean Conf Drop	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	7

Figure 5.22
SP Fault Vulnerable Metrics

5) Shattered Glass Fault

Metric_Label	p_hat (CTD)	p_hat (LCH)	p_hat (CUTOUT)	p_hat (CUTIN)	p_hat (ACLH)	p_hat mean	Rank
Seg IoU – Drivable Area	0.0416	0.0545	0.0649	0.0104	0.0416	0.0426	3
Seg IoU – Lane Line	0.1766	0.0987	0.1143	0.1039	0.2052	0.1397	1
Pearson – Drivable Area	0.0078	0.0156	0.0156	0.0026	0.0208	0.0125	8
Pearson – Lane Line	0.0494	0.0286	0.0260	0.0208	0.0649	0.0379	5
Area RelDiff – Drivable Area	0.0364	0.0468	0.0442	0.0104	0.0571	0.0390	4
Area RelDiff – Lane Line	0.0883	0.0494	0.0571	0.0416	0.0805	0.0634	2
Pos Shift – Drivable Area	0.0052	0.0104	0.0078	0.0000	0.0026	0.0052	10
Pos Shift – Lane Line	0.0078	0.0000	0.0026	0.0000	0.0234	0.0068	9
Detection Consistency Rate	0.0519	0.0000	0.0000	0.0000	0.1065	0.0317	6
Detection Mean IoU	0.0364	0.0052	0.0078	0.0052	0.0234	0.0156	7
Detection Mean Conf Drop	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	11

Figure 5.23
SG Fault Vulnerable Metrics

The Complementary Tasked Based Ranking in **Figure 5.24** shows that the Single Point has (LL) as the most vulnerable Metric and the (OD) the least Vulnerable. However in the the subfigure (b) the least vulnerable metrics is the (DA). This analysis show that the driving scenario also influences the severity of degradation of Tasks.

1) Single Point Fault

Tasks	Mean	Rank
Drivable Area (DA)	0.0022	2
Lane Line (LL)	0.0155	1
Object Detection (OD)	0.0000	3

(a) Single point Task Based Ranking.

5) Shattered Glass Fault

Tasks	Mean	Rank
Drivable Area (DA)	0.0248	3
Lane Line (LL)	0.0619	1
Object Detection (OD)	0.0433	2

(b) Shattered Glass Task Based Ranking.

Figure 5.24
Task Based Ranking

5.5.2 Aggregated Metric Ranking Across Fault Patterns

The second stage of the analysis focuses on the comparison of metric sensitivity across all fault patterns as illustrated in Figure 5.25. For each fault pattern, the mean failure probability of every metric is first obtained by averaging its values across the five driving scenarios. These mean values are then placed side by side so that the same metric can be compared under Single Point, Row, Column, Bullet Wake, and Shattered Glass perturbations.

An overall mean is subsequently computed for each metric by averaging its fault-pattern means. This value provides a global sensitivity score for the metric across the entire experimental campaign, and the metrics are ranked accordingly. This ranking makes it possible to determine which evaluation measures are the strongest indicators of tensor vulnerability when all fault patterns are considered together.
The aggregated results show that **lane-line related metrics occupy the highest ranks overall**. In particular, **Seg IoU – Lane Line** emerges as the most sensitive metric, followed by **Area RelDiff – Lane Line**. This indicates that the degradation induced by tensor perturbations is expressed most strongly through loss of lane-line overlap and distortion of lane-line segmentation extent. At the opposite end of the ranking, **Detection Mean Confidence Drop** remains negligible, indicating that confidence degradation is the weakest indicator of tensor vulnerability in the present campaign.

	Mean Values						
Metric_Label	SP	ROW	Column	BW	S Glass	Overall Mean	Overall Rank
Seg IoU – Drivable Area	0.0016	0.0010	0.0000	0.0384	0.0426	0.0167	3
Seg IoU – Lane Line	0.0286	0.0182	0.0062	0.1408	0.1397	0.0667	1
Pearson – Drivable Area	0.0005	0.0000	0.0000	0.0094	0.0125	0.0045	8
Pearson – Lane Line	0.0078	0.0036	0.0005	0.0327	0.0379	0.0165	4
Area RelDiff – Drivable Area	0.0068	0.0010	0.0005	0.0317	0.0390	0.0158	5
Area RelDiff – Lane Line	0.0255	0.0166	0.0062	0.0592	0.0634	0.0342	2
Pos Shift – Drivable Area	0.0000	0.0000	0.0000	0.0052	0.0052	0.0021	9
Pos Shift – Lane Line	0.0000	0.0000	0.0000	0.0031	0.0068	0.0020	10
Detection Consistency Rate	0.0000	0.0000	0.0005	0.0369	0.0317	0.0138	6
Detection Mean IoU	0.0000	0.0000	0.0000	0.0140	0.0156	0.0059	7
Detection Mean Conf Drop	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	11

Figure 5.25
Aggregated Vulnerable Metrics

5.5.3 Task Based Analysis

In addition to the metric-level ranking, the same analysis is extended to the task level by grouping the eleven metrics according to the three perception branches of the network:

- **Drivable Area (DA)**, including the four drivable-area segmentation metrics,
- **Lane Line (LL)**, including the four lane-line segmentation metrics,
- **Object Detection (OD)**, including the three detection-related metrics.

For each task, a mean severity value is computed from the overall metric means that belong to that group. A ranking (Figure 5.26) is then assigned to the three task means. This provides a compact interpretation of which perception branch is most affected by the fault injection campaign when all metrics are considered together.

Aggregated Metrics Analysis

Tasks	Mean	Rank
Drivable Area (DA)	0.0098	2
Lane Line (LL)	0.0298	1
Object Detection (OD)	0.0066	3

Figure 5.26
Aggregated Task Based Vulnerability

The task-based results show that **Lane Line** has the highest mean severity and therefore ranks first, followed by **Drivable Area**, while **Object Detection** ranks last. This indicates that the impact of tensor perturbations is concentrated more strongly in the segmentation branches, and particularly in lane-line estimation, than in the detection branch. The Bar Graph in Figure 5.27 shows the overall Ranks of all the Metrics.

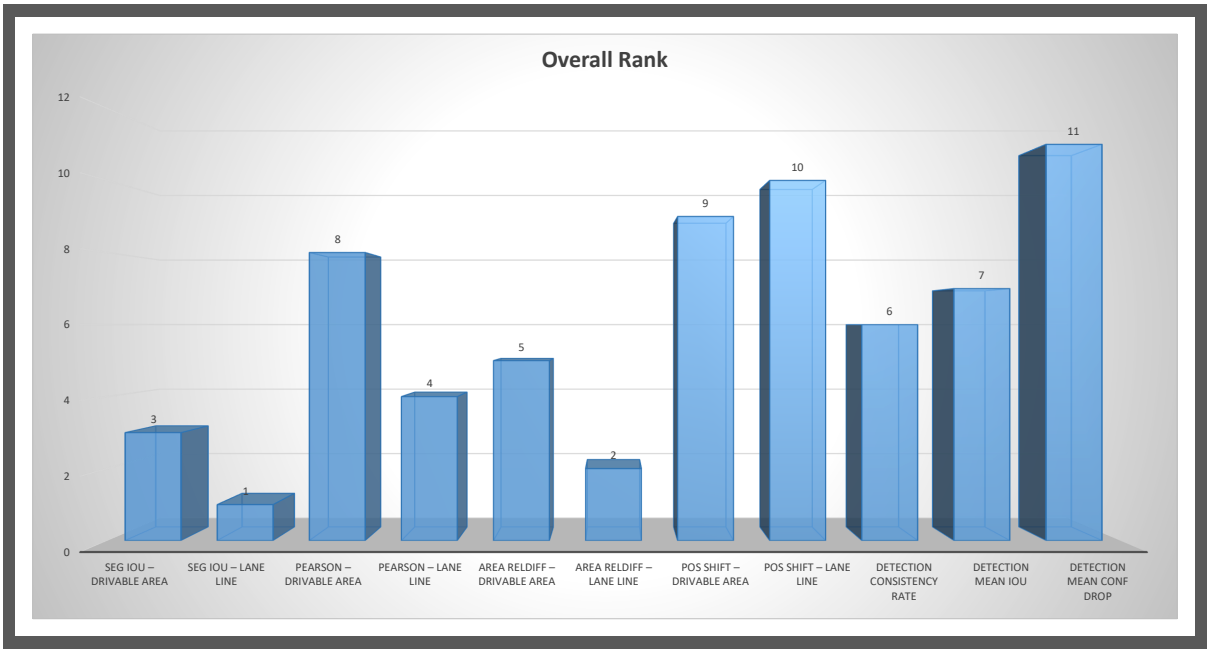


Figure 5.27
Aggregated Vulnerable Metrics Bar Graph

Chapter 6

FUTURE WORK

Although the proposed framework provides a structured methodology for analyzing the reliability of deep learning based automotive perception under tensor-level perturbations, several important extensions remain open for future investigation. The present work focuses primarily on controlled corruption of tensors stored in the weight file and evaluates their effect on object detection, drivable area segmentation, and lane line estimation. A natural continuation of this research is to broaden the fault space and study additional sources and locations of corruption within the inference pipeline.

6.0.1 Analysis of Feature Maps

A promising extension concerns the analysis of feature maps. Since the present study is based on faults injected into tensors of the weight file, the internal activations of the network are only observed indirectly through the final output metrics. Future work could instead target the feature maps directly, with the objective of understanding how corrupted intermediate representations evolve through the different layers of the network. This would allow the study of error amplification, masking, and propagation inside the model, and would help clarify whether specific layers or branches of the architecture are more sensitive because of the weights themselves or because of the dynamics of their generated activations.

6.0.2 New Fault Patterns

Another useful extension would be the introduction of additional fault geometries and perturbation patterns. The current framework evaluates five structured corruption models, namely Single Point, Row, Column, Bullet Wake, and Shattered Glass. Future work could define new patterns with different spatial densities, temporal persistence properties, or correlations across channels and layers. This would enable the framework to represent a broader range of realistic disturbances and to study whether certain perception tasks are more vulnerable to specific classes of structured corruption. Future work could expand this approach by adding other error models attached to GPUs, Vector processors, domain-specific hardware accelerators, or embedded platforms.
Such an analysis would make it possible to observe how faults propagate through convolution, activation, normalization, pooling, and feature fusion stages, and to identify which operators are intrinsically more exposed to functional degradation. This would provide a more complete view of reliability, moving from parameter-level perturbation toward computation-level vulnerability analysis.

6.0.3 Extension of Evaluation Metrics

The evaluation methodology can also be enriched through the introduction of additional performance metrics. While the present work already combines overlap, structural, geometric, and detection-based measures, further metrics could be included to better characterize degradation in temporal consistency, object localization stability, false positive behavior, or sensitivity of downstream driving decisions. A richer metric space would improve the interpretability of the results and provide a more detailed understanding of how faults affect different aspects of perception quality.

6.0.4 Sensor-Side Noise and Propagation into Tensors

A further extension concerns the inclusion of disturbances originating from the sensor side. In the current framework, corruption is introduced internally at the tensor level. Future work could investigate the effect of sensor noise and disturbances entering the network from the input stage, such as noise in camera frames, illumination variations, blur, or compression artifacts. Studying how noise arriving from sensors propagates into tensors and feature maps would create an important bridge between input-side uncertainty and internal model vulnerability. This would make the framework more representative of real automotive operating conditions, where both hardware-level perturbations and imperfect sensor signals may interact.

6.0.5 Closed-loop evaluation on automotive scenarios

A final extension might involve the closed-loop evaluation of driving agents, sensors, and dynamic driving scenarios, interacting in a closed-loop.

6.0.6 Overall Perspective

Overall, these extensions would broaden the framework from a weight-tensor reliability analysis tool into a more comprehensive platform for studying robustness and fault propagation across the full perception pipeline. Such developments would support a deeper understanding of how faults emerge, propagate, and ultimately affect AI-based automotive perception in realistic deployment conditions.

Chapter 7

Conclusion

This thesis presented **FIERA** as a structured framework for studying the reliability of deep learning-based automotive perception under controlled corruption of weight tensors. The experimental campaign showed that faults injected into the internal parameters of **YOLOP** do not affect all components of the perception system in the same way. Instead, the observed degradation depends strongly on the **type of fault pattern injected**, the **tensor being corrupted**, and the **perception property used to evaluate the output**. Across the complete evaluation campaigns, the results consistently showed that **structured corruptions** (e.g., shattered glass) produce broader and more severe degradation than localized errors (e.g., single faults), and that the response of the model is concentrated in **specific tensors** and **specific perception outputs** rather than being uniformly distributed across an AI-based perception network.

7.1 Fault Pattern Severity

The comparison of fault patterns (5.4.1) identified **Shattered Glass** as the **most critical fault pattern** and **Column** as the **least critical fault pattern**. In the overall failure analysis, Shattered Glass reached the highest mean severity, while **Bullet Wake** followed very closely. **Single Point** occupied an intermediate position, **Row** remained lower, and Column produced the smallest mean severity. The same ordering was preserved in the task-based comparison, where Shattered Glass again remained first, and Column remained last.
This repeated ordering is significant because it shows that the ranking is not dependent on only one aggregation rule. It indicates that **dense and spatially extended corruption patterns** are substantially more harmful than **localized or narrowly constrained perturbations**. Therefore, the results show that **fault geometry** is a dominant factor in determining the severity of perception degradation.

7.2 Most Corrupted Tensors

The tensor ranking results 5.4.3 showed that vulnerability is concentrated in a **limited subset of tensors** rather than being distributed evenly across the whole network. In the representative Single Point analysis for the City Driving scenario, **model.40.cv1.conv.weight** emerged as the most vulnerable tensor, followed by **model.40.cv3.weight**, **model.40.cv4.conv.weight**, and

model.40.m.0.cv1.conv.weight. These tensors were mainly associated with failures in lane line related metrics.
The Figure 5.4.3 also shows the Vulnerable Tensors for each Fault Pattern Belonging to different Video Scenarios.
When the broader tensor ranking results are considered across fault patterns and scenarios, repeated appearances were observed for tensors such as **model.40.m.0.cv1.conv.weight** and **model.42.conv.weight**. Under more severe structured perturbations, earlier tensors such as **model.0.conv.conv.weight** and **model.1.conv.weight** also appeared among the critical cases. Taken together, these findings indicate that tensors around the **model.40** and **model.42** regions form the most repeatedly exposed vulnerable subset, while a few earlier backbone tensors become critical under stronger structured fault conditions.

7.3 Metric Vulnerability

The metric based analysis in section 5.5.2 showed that the most sensitive degradation indicators are those related to **lane line segmentation**. Among all the evaluated metrics, **Seg IoU for Lane Line** emerged as the **most vulnerable metric**, followed by **Area Relative Difference for Lane Line**. These results indicate that the effect of tensor perturbations is expressed most strongly through **loss of lane line overlap** and **distortion in lane line segmentation extent**. In addition, **Pearson correlation for Lane Line** also showed noticeable sensitivity, further confirming the fragility of the lane line branch.
By contrast, **Detection Mean Confidence Drop** remained the **least vulnerable metric**. Its values stayed negligible throughout the reported analyses, showing that confidence degradation was not the dominant manifestation of tensor corruption in the present campaign. The position shift metrics also remained comparatively low, which suggests that the dominant effect of the faults was not large geometric displacement, but rather structural degradation of the predicted outputs.
Overall, the metric based results demonstrate that the strongest signs of vulnerability are captured by **lane line overlap** and **lane line area distortion**, whereas **detection confidence degradation** provides the weakest indication of failure.

7.4 Task Level Vulnerability

At the task level, the results 5.5.3 consistently showed that **Lane Line** is the **most vulnerable task**, while **Object Detection** is the **least vulnerable task**. **Drivable Area** occupies an intermediate position between the two. This pattern is clearly supported by the ranking of the metrics, where lane line related measures dominate the upper part of the table and detection related measures remain comparatively more stable.
The task based aggregation therefore confirms that the degradation is concentrated more strongly in the **segmentation branches**, and particularly in **lane line estimation**, than in the detection branch. This means that, under tensor perturbation, the network is least able to preserve reliable lane line perception, moderately affected in drivable area estimation, and most robust in object detection.

7.5 Final Remarks

In summary, this thesis shows that the reliability of YOLOP under fault injection is governed by a clear interaction between **fault geometry**, **tensor location**, and **perception task sensitivity**. The most severe degradation is produced by **Shattered Glass**, followed closely by **Bullet Wake**, while **Column** faults remain the least harmful. The most repeatedly exposed tensors are concentrated around the **model.40** and **model.42** regions, with some earlier tensors becoming critical under stronger structured patterns. At the output level, **lane line metrics** are the most sensitive indicators of corruption, while **Detection Mean Confidence Drop** is the least sensitive. At the task level, **Lane Line** is the most vulnerable branch, **Drivable Area** follows, and **Object Detection** is the least vulnerable overall. These findings demonstrate that reliability analysis for automotive perception must go beyond a single global accuracy measure and instead consider the **internal structure of the model**, the **geometry of the fault**, and the **specific perception function under evaluation**.

Appendix

Appendix A

Overview

The Appendix provides the results for the remaining driving scenarios across all five fault patterns. For each scenario, the procedure used to identify and rank the vulnerable tensors is consistent with the methodology outlined in Section 5.3.

Furthermore, the Appendix contains the remaining three out of five scenario-level metric analysis rankings for the Row, Column, and Bullet Wake fault patterns, along with their corresponding task-based rankings, as discussed in Section 5.5.1. The related tables are presented in Section A.110.

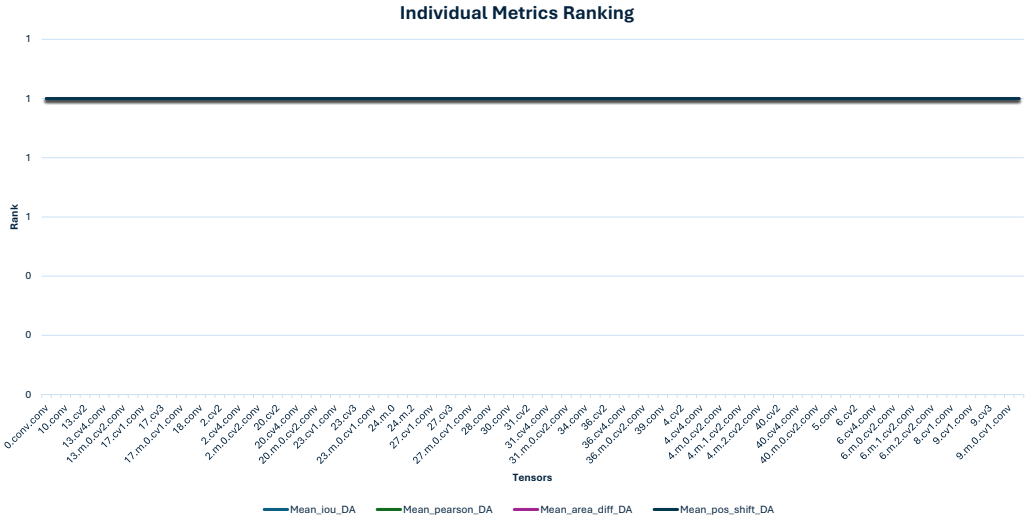
A.0.0.1 Row Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	11	385	0.0286	0.0166	0.0119	0.0452	STOP	43	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	9	385	0.0234	0.0151	0.0083	0.0385	STOP	35	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	9	385	0.0234	0.0151	0.0083	0.0385	STOP	35	0

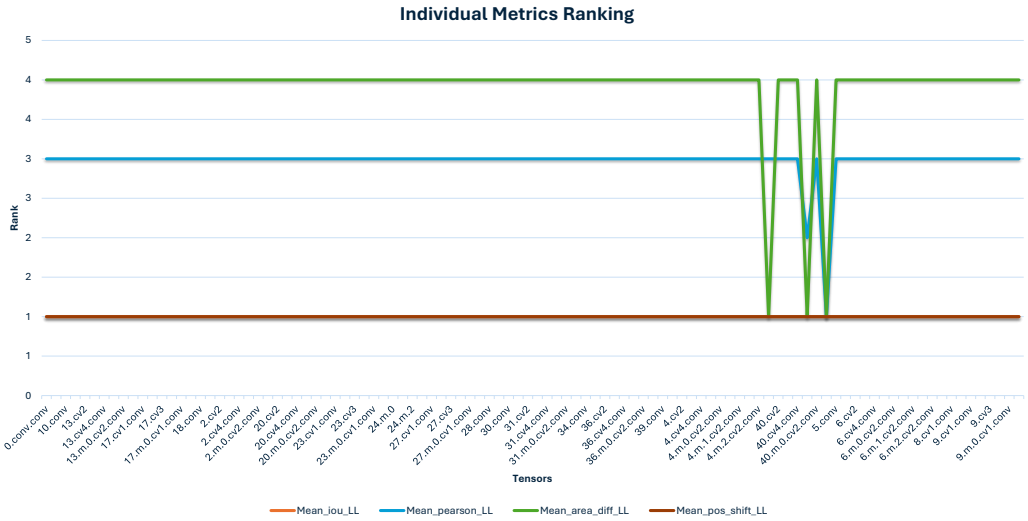
Figure A.1
CTD Row Stop Criterion

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.cv1.conv.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.m.0.cv1.conv.weight	3	3	1	0	0	0	0.333333	0	1	0	0	0	0	0
model.42.conv.weight	3	3	1	0	0	0	1	0	1	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

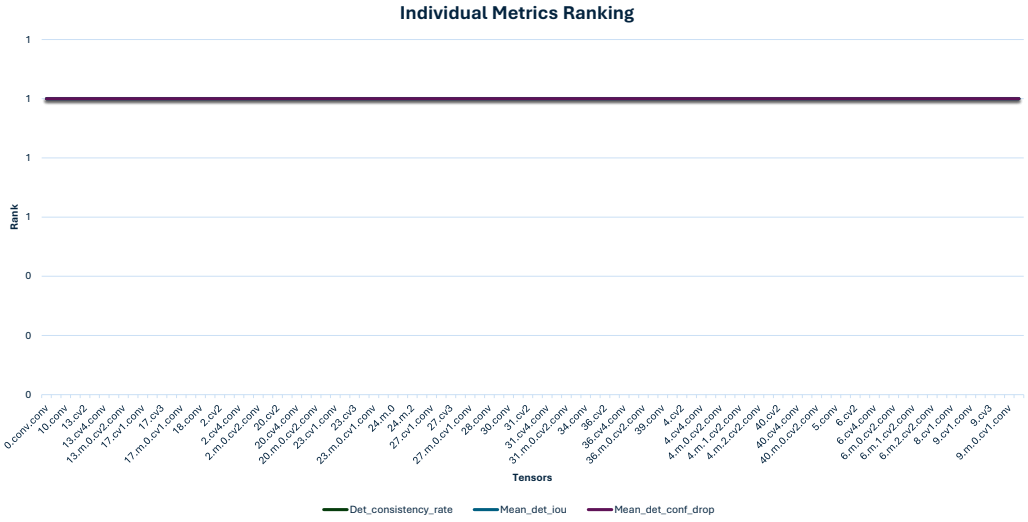
Figure A.2
CTD ROW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.3
CTD ROW Task Based Charts

City Drive Video Comparison Row Fault

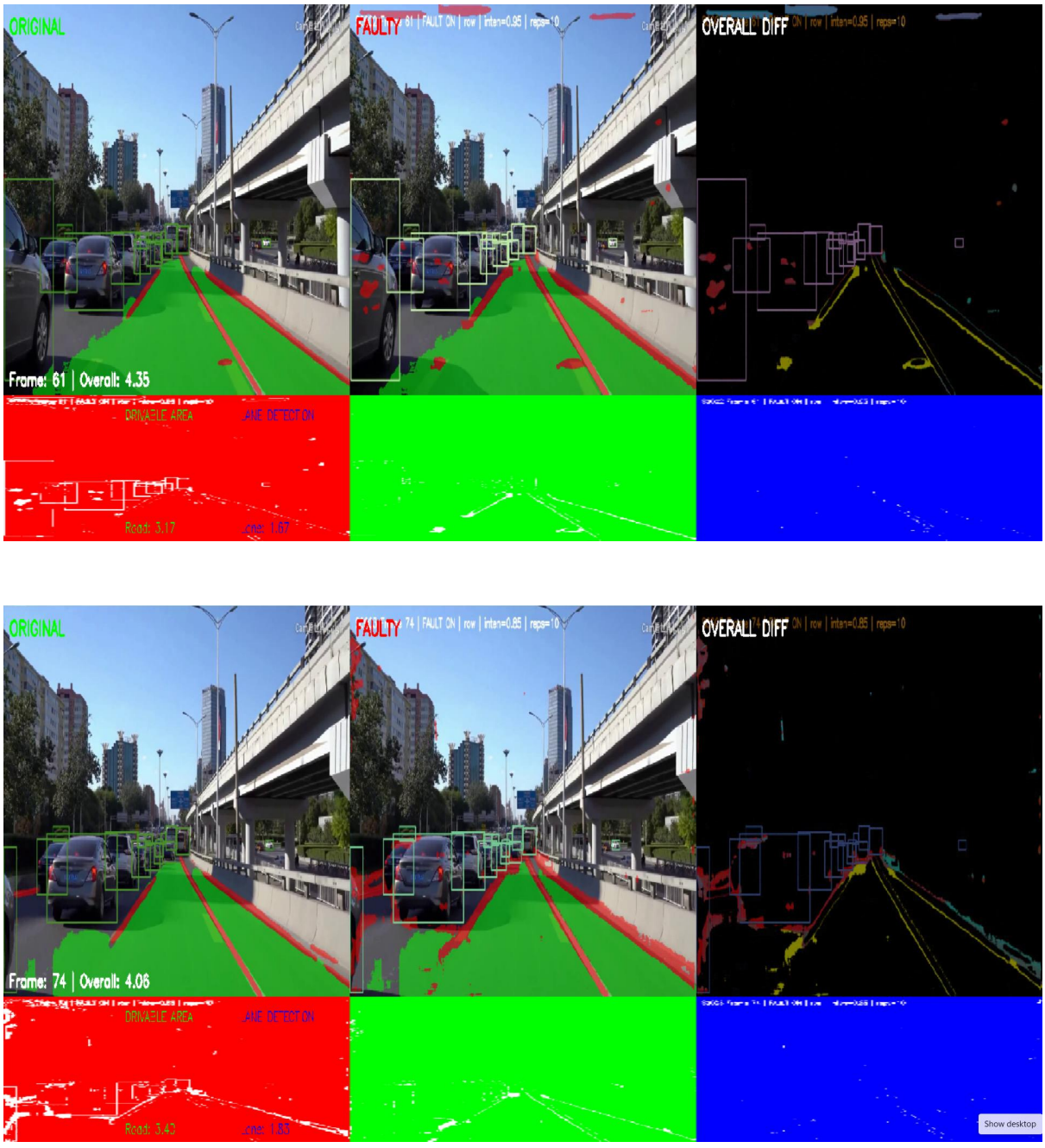


Figure A.4
Visual comparison of the city driving scenario under Row fault injection.

A.0.0.2 Column Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0

Figure A.5
CTD Column Stop Criterion

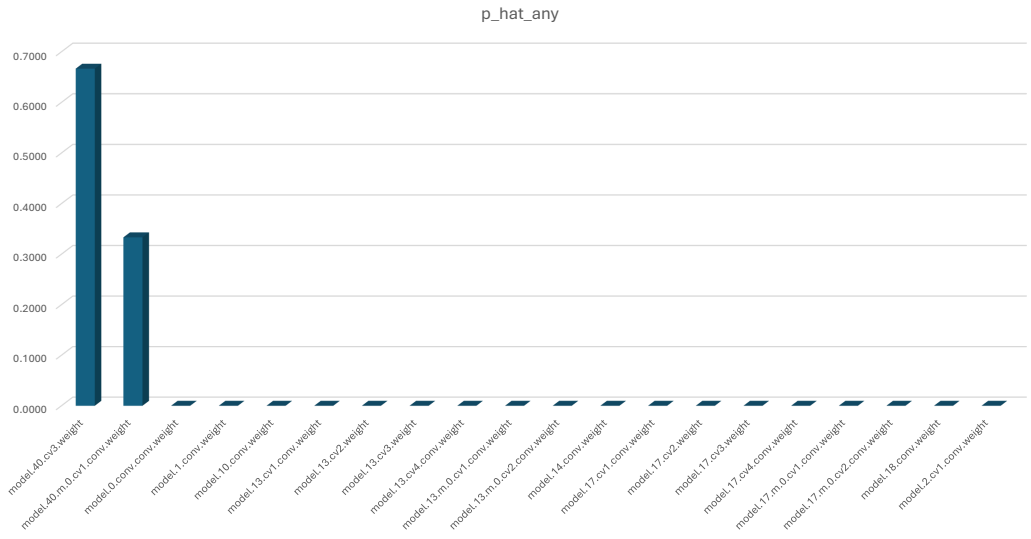
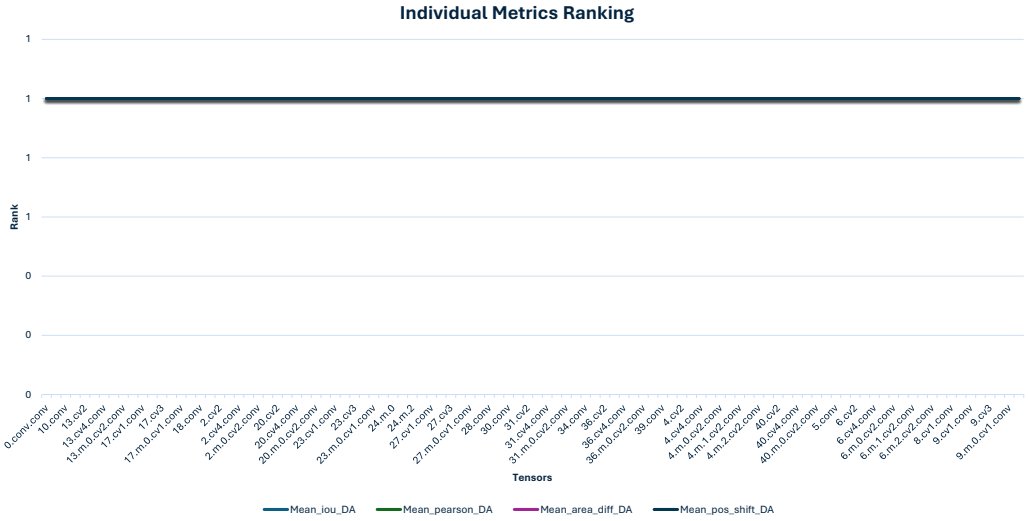


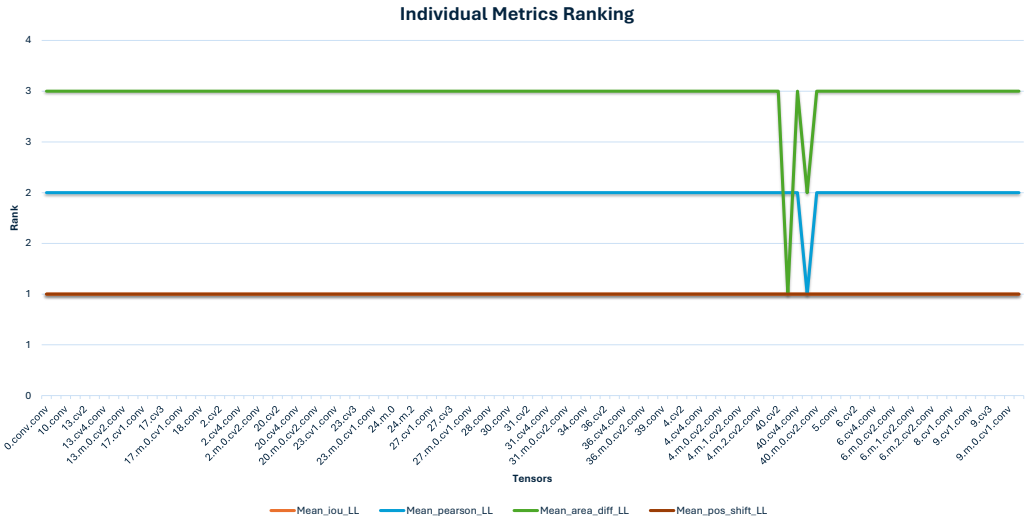
Figure A.6
CTD COL BAR Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.cv3.weight	3	2	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.m.0.cv1.conv.weight	3	1	0.333333333	0	0	0	0.333333	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.2.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

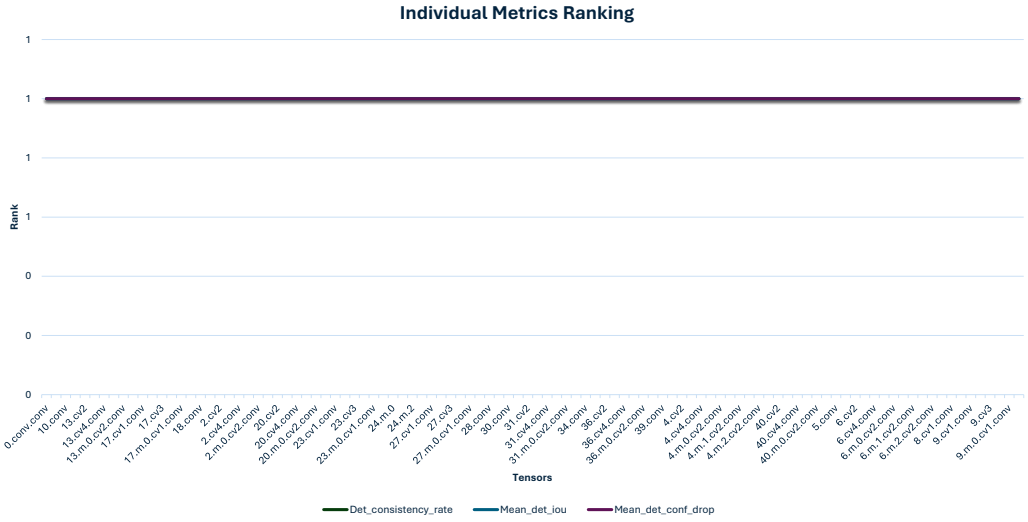
Figure A.7
CTD COL TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.8
CTD COL Task Based Charts

City Drive Video Comparison Column Fault

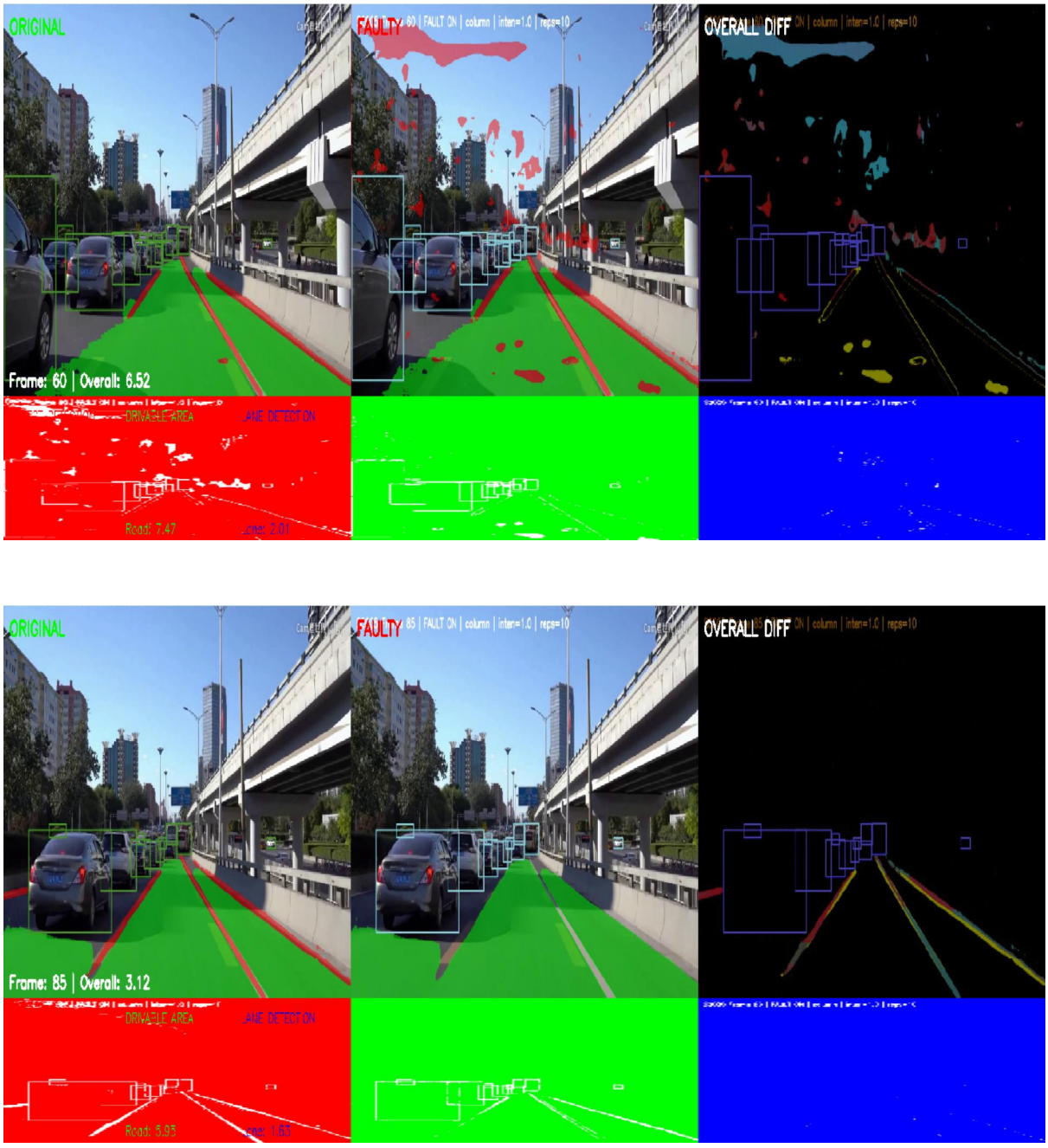


Figure A.9
Visual comparison of the city driving scenario under column fault injection.

A.0.1 Lane Change Driving Scenario

Figure A.10
Lane Change Driving Scenario (LCH)

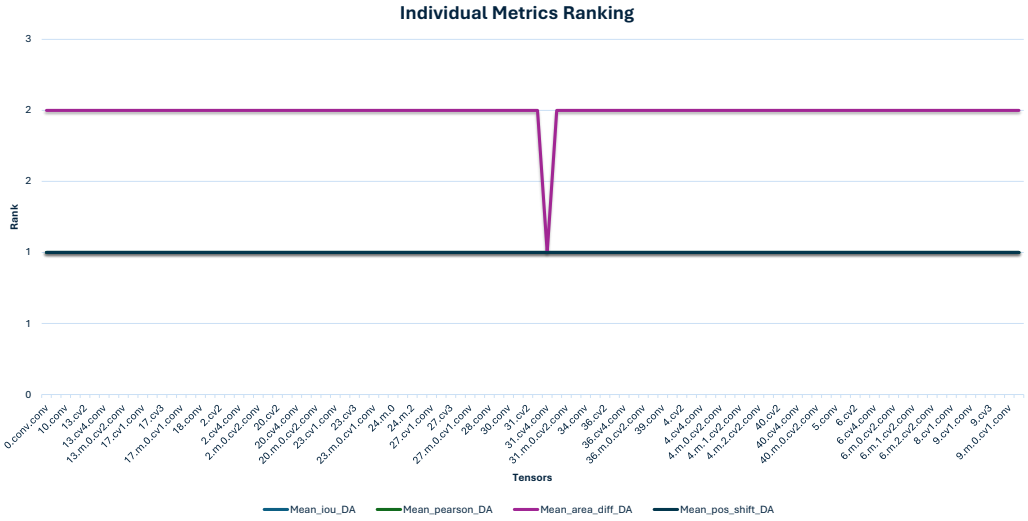
A.0.1.1 Single Point Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	8	385	0.0208	0.0142	0.0065	0.0350	STOP	31	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Area RelDiff – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Area RelDiff – Lane Line	8	385	0.0208	0.0142	0.0065	0.0350	STOP	31	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	9	385	0.0234	0.0151	0.0083	0.0385	STOP	35	0

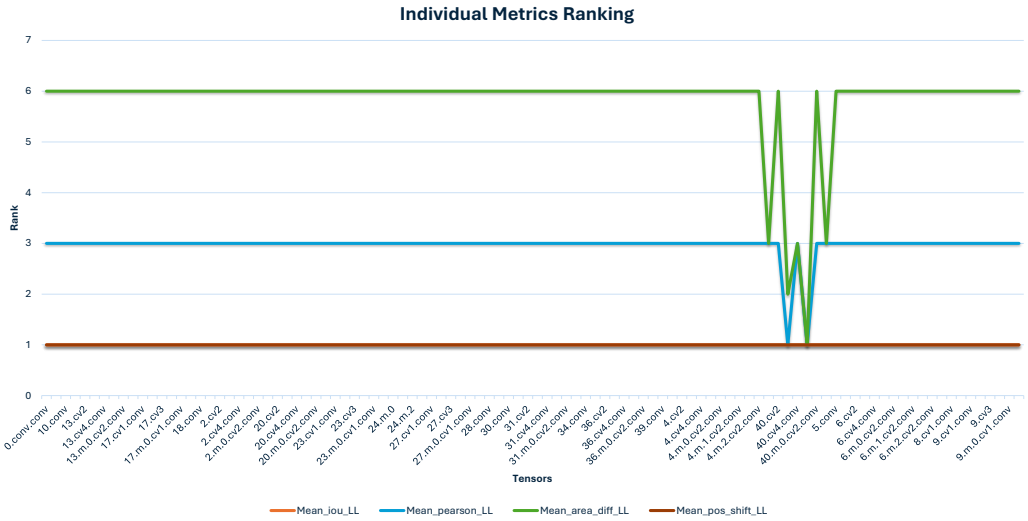
Figure A.11
LCH Single Point Stop Criterion

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.m.0.cv1.conv.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.3333	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.cv3.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.31.cv4.conv.weight	3	1	0.333333333	0	0	0	0	0.333333	0	0	0	0	0	0
model.40.cv1.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.40.cv4.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.42.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

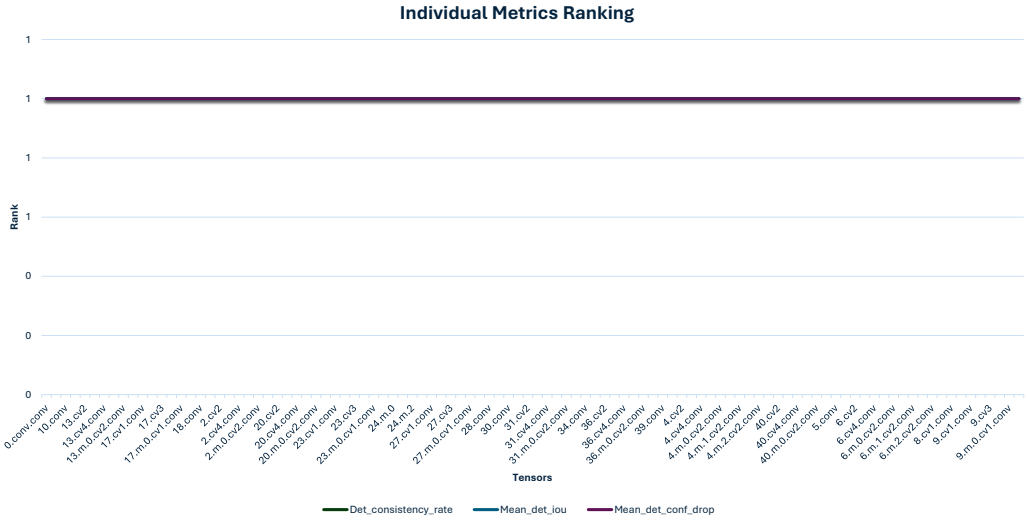
Figure A.12
LCH SP Top 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.13
LCH SP Task Based Charts

LCH Drive Video Comparison Single Point Fault

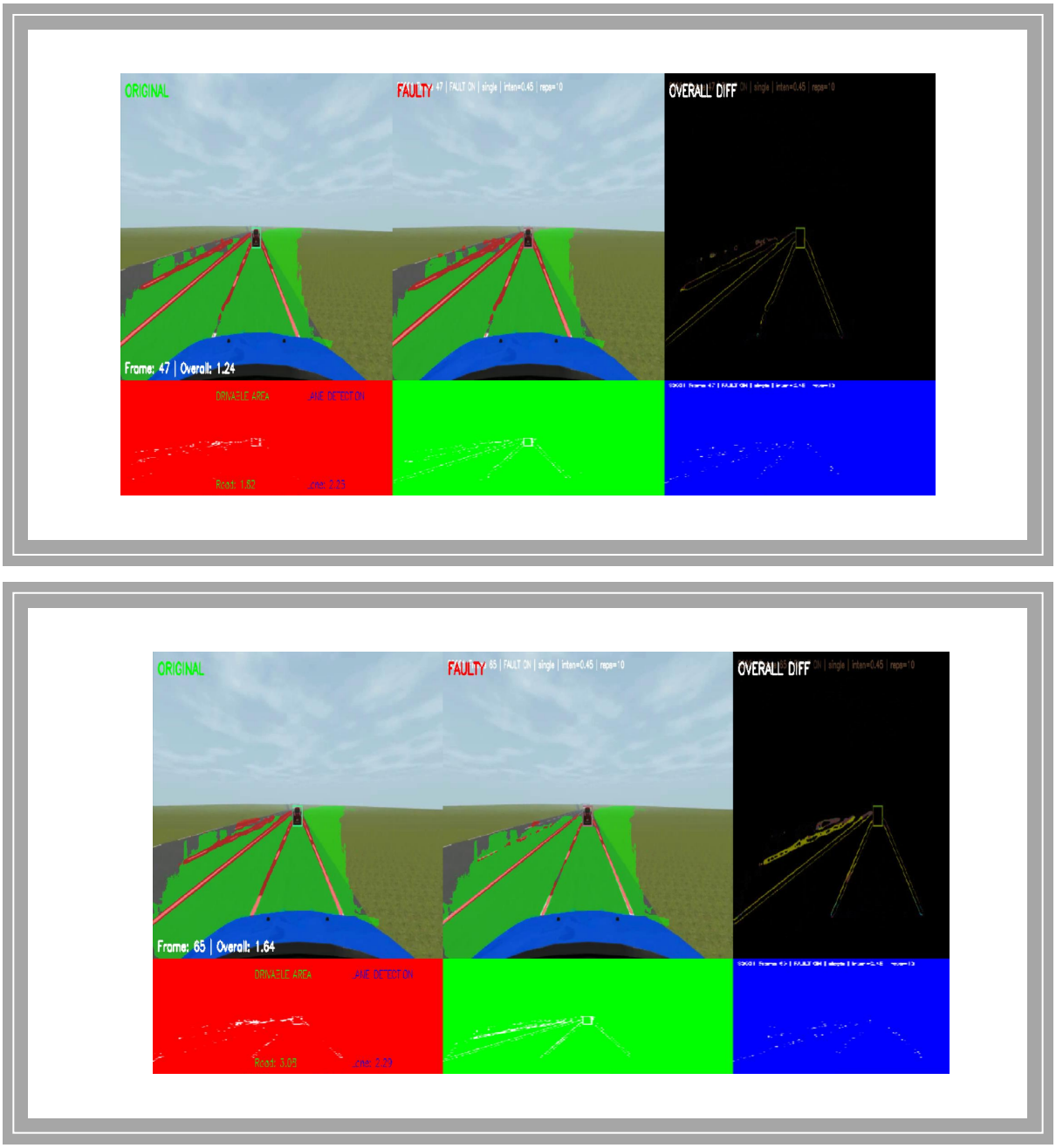


Figure A.14
Visual comparison of the LCH driving scenario under SP fault injection.

A.0.1.2 Row Fault Injection LCH

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	7	385	0.0182	0.0133	0.0048	0.0315	STOP	27	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	7	385	0.0182	0.0133	0.0048	0.0315	STOP	27	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	7	385	0.0182	0.0133	0.0048	0.0315	STOP	27	0

Figure A.15
LCH Row Stop Criterion

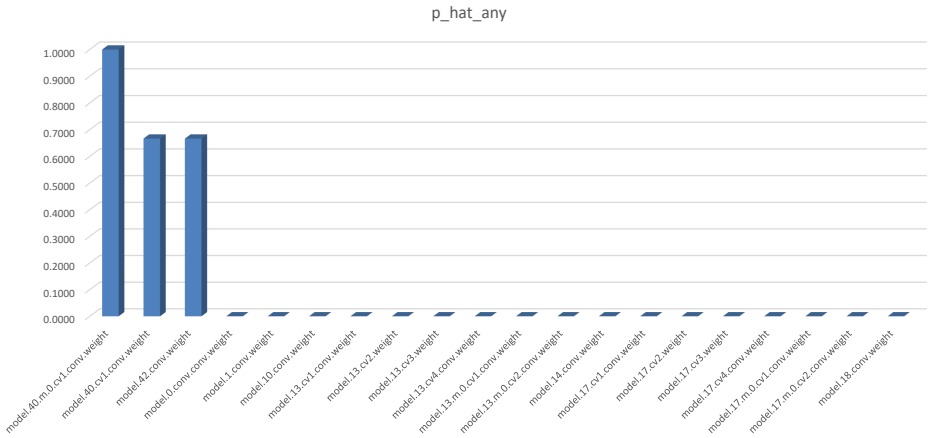
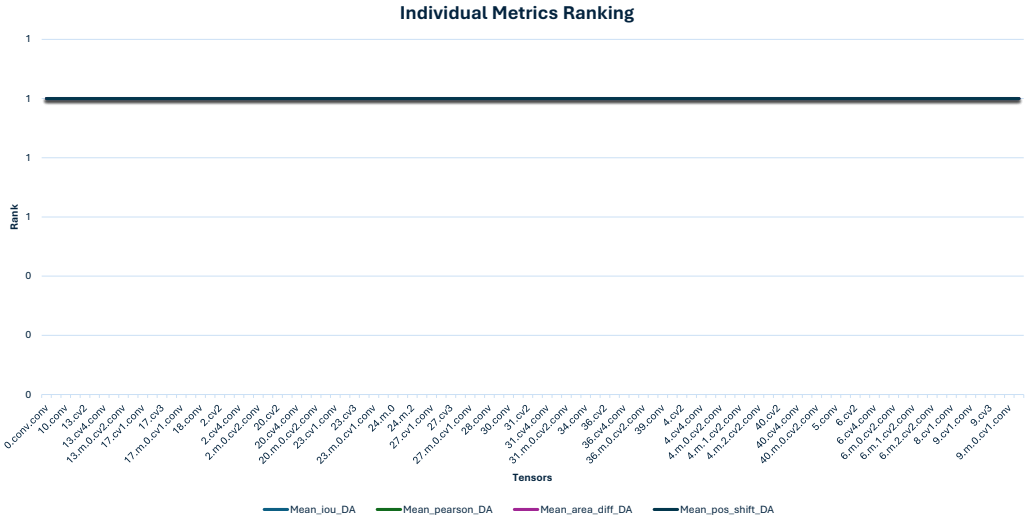


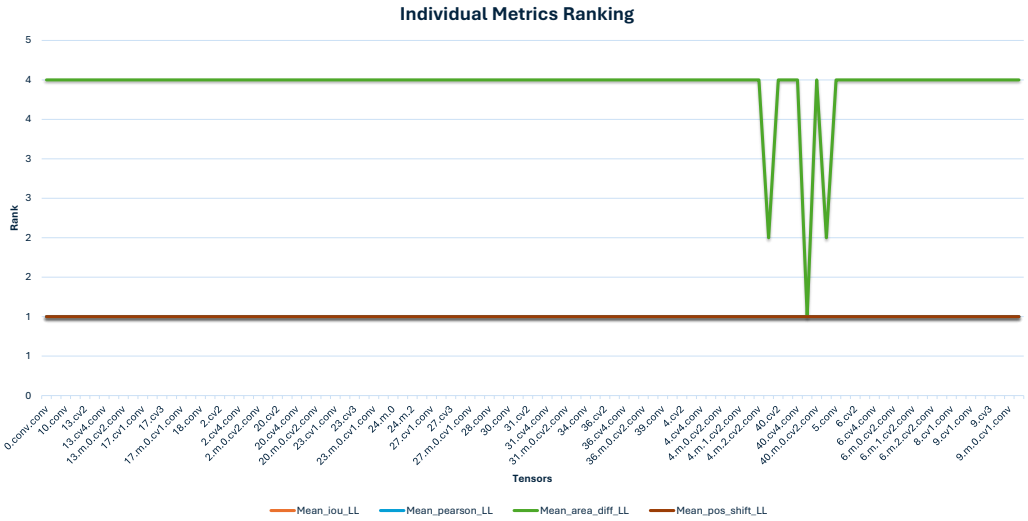
Figure A.16
LCH ROW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.m.0.cv1.conv.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.cv1.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

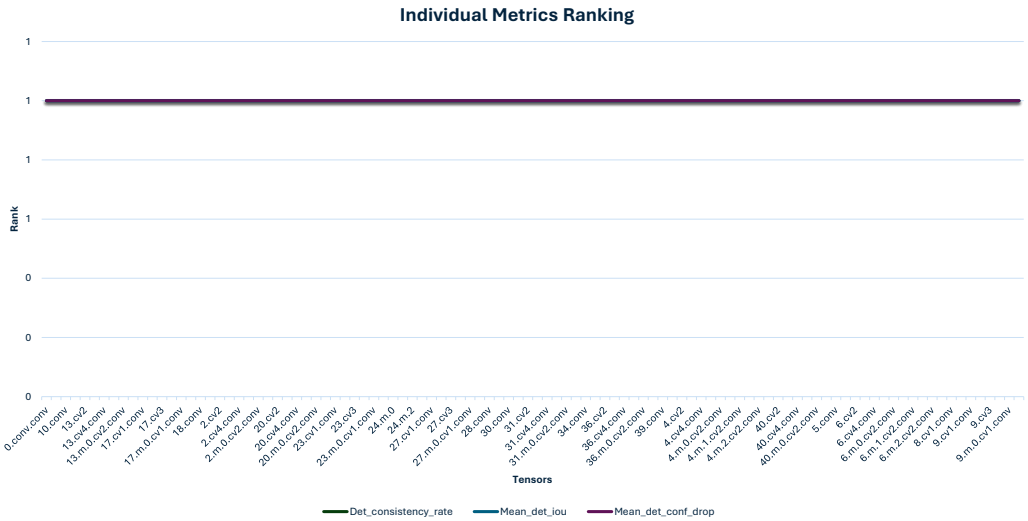
Figure A.17
LCH ROW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.18
LCH ROW Task Based Charts

LCH Drive Video Comparison Row Fault Fault

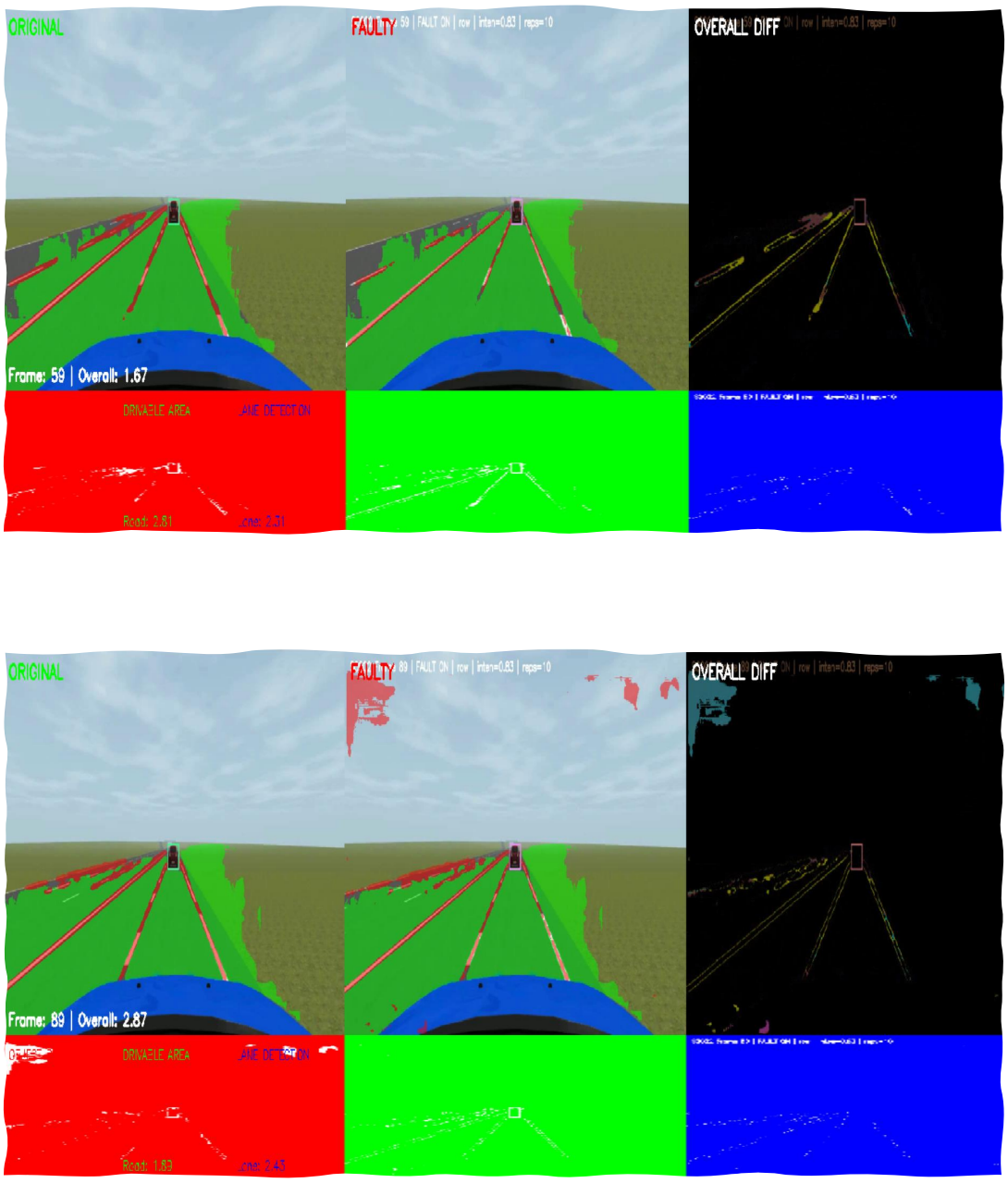


Figure A.19
Visual comparison of the LCH driving scenario under Row fault injection.

A.0.1.3 Column Fault Injection LCH

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0

Figure A.20
LCH Column Stop Criterion

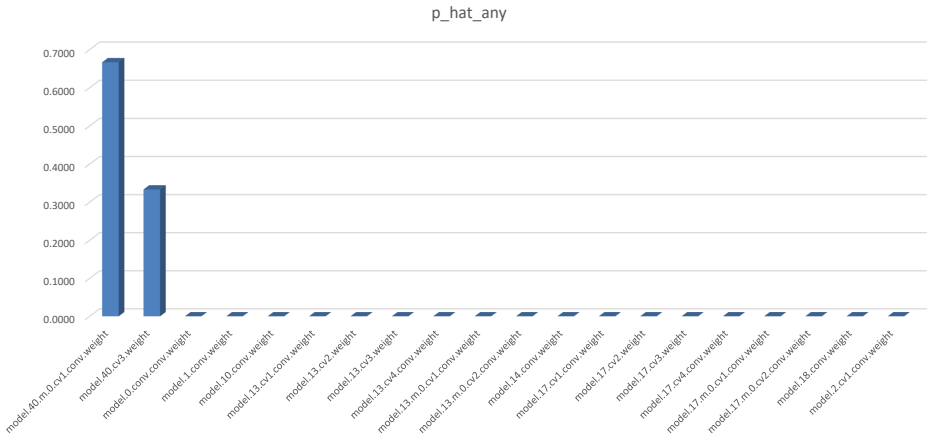
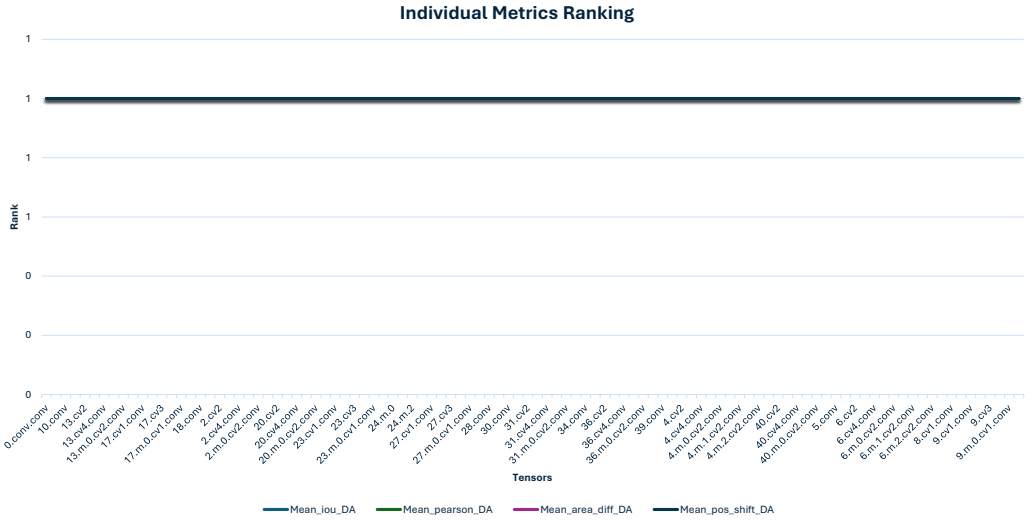


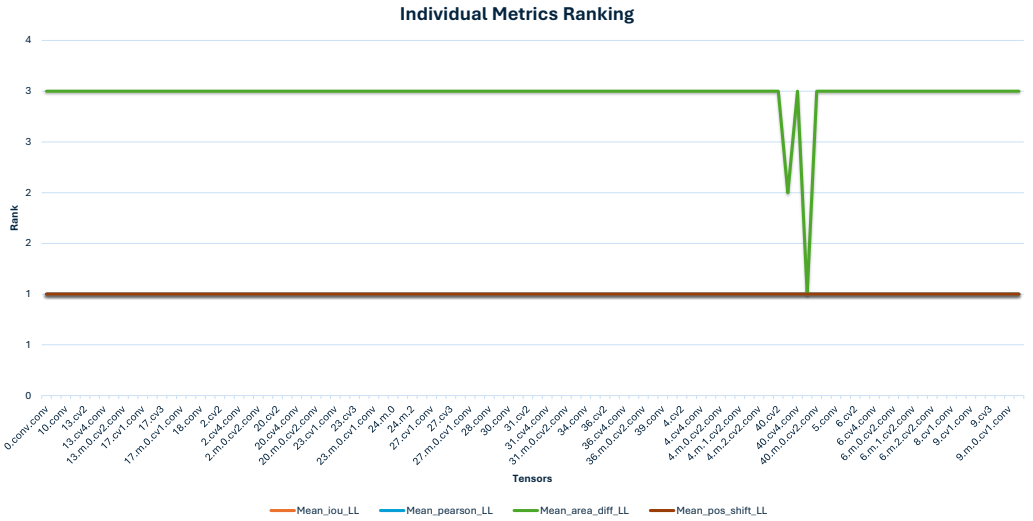
Figure A.21
LCH COL Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.m.0.cv1.conv.weight	3	2	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.cv3.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.2.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

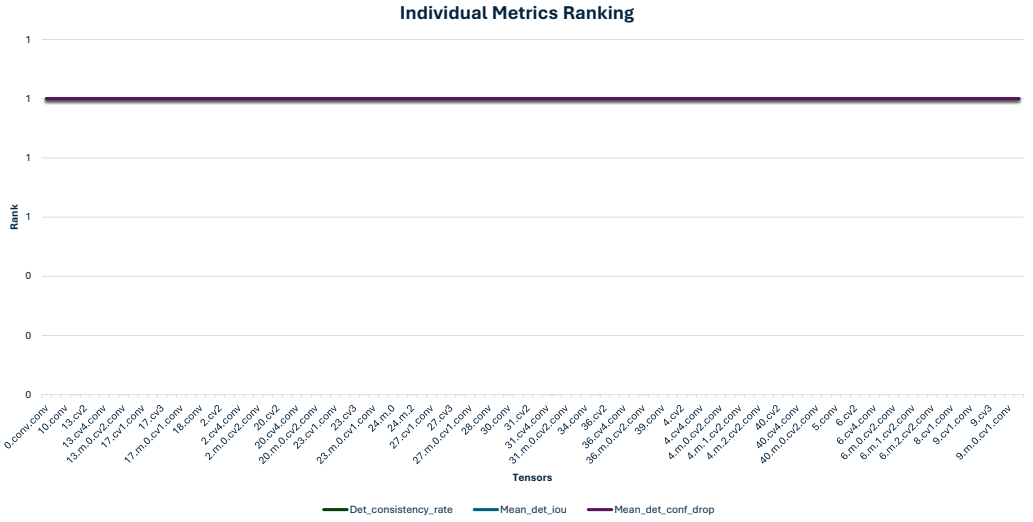
Figure A.22
LCH COL TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.23
LCH COL Task Based Charts

LCH Drive Video Comparison Column Fault Fault

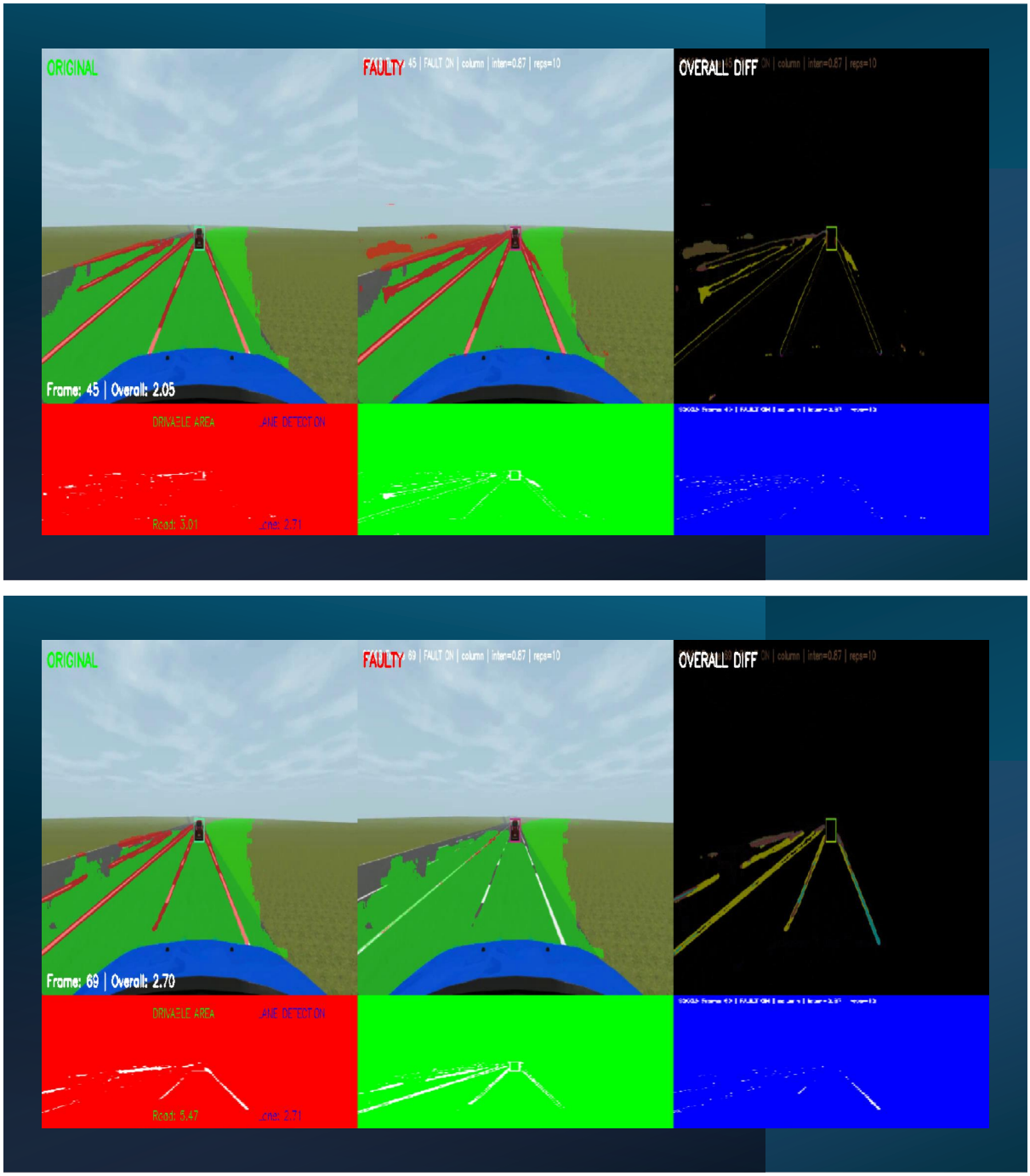


Figure A.24
Visual comparison of the LCH driving scenario under Column fault injection.

A.0.1.4 Bullet Wake Fault Injection LCH

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	16	385	0.0416	0.0199	0.0216	0.0615	STOP	61	0
Seg IoU – Lane Line	41	385	0.1065	0.0308	0.0757	0.1373	STOP	146	0
Pearson – Drivable Area	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pearson – Lane Line	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Area RelDiff – Drivable Area	11	385	0.0286	0.0166	0.0119	0.0452	STOP	43	0
Area RelDiff – Lane Line	14	385	0.0364	0.0187	0.0177	0.0550	STOP	54	0
Pos Shift – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Pos Shift – Lane Line	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	26	385	0.0675	0.0250	0.0425	0.0926	STOP	97	0

Figure A.25
LCH BW Stop Criterion

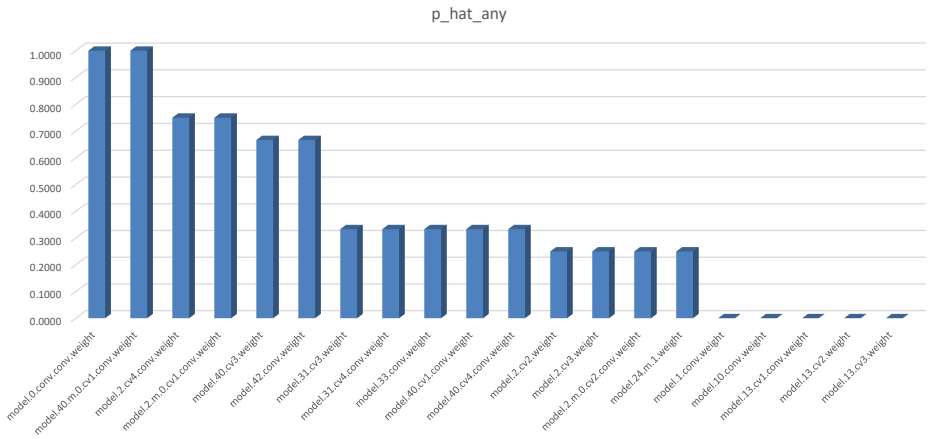
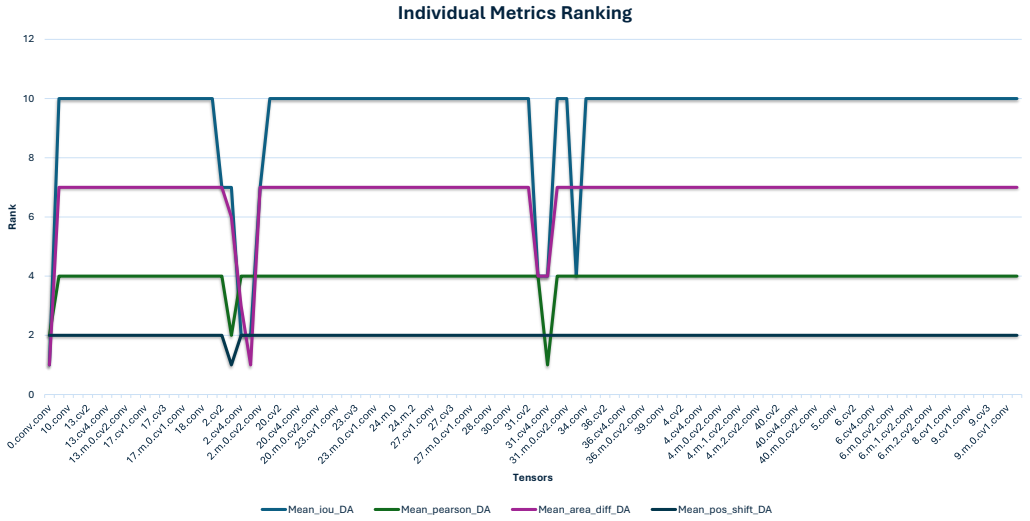


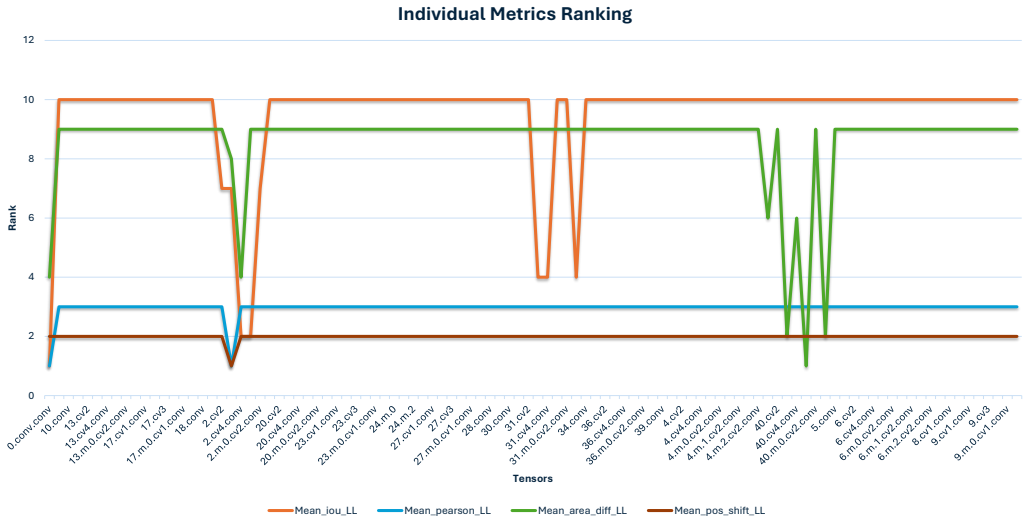
Figure A.26
LCH BW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.0.conv.conv.weight	4	4	1.0000	1.0000	1.0000	0.2500	0.2500	0.7500	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.m.0.cv1.conv.weight	3	3	1	0	0	0	0	0	1	0	0	0	0	0
model.2.cv4.conv.weight	4	3	0.75	0.75	0.75	0	0	0.5	0.5	0	0	0	0	0
model.2.m.0.cv1.conv.weight	4	3	0.75	0.75	0.75	0	0	0.75	0	0	0	0	0	0
model.40.cv3.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.31.cv3.weight	3	1	0.333333333	0.333333333	0.333333	0	0	0.333333	0	0	0	0	0	0
model.31.cv4.conv.weight	3	1	0.333333333	0.333333333	0.333333	0.333333	0	0.333333	0	0	0	0	0	0
model.33.conv.weight	3	1	0.333333333	0.333333333	0.333333	0	0	0	0	0	0	0	0	0
model.40.cv1.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.40.cv4.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.2.cv2.weight	4	1	0.25	0.25	0.25	0	0	0	0	0	0	0	0	0
model.2.cv3.weight	4	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0	0	0
model.2.m.0.cv2.conv.weight	4	1	0.25	0.25	0.25	0	0	0	0	0	0	0	0	0
model.24.m.1.weight	4	1	0.25	0	0	0	0	0	0	0	0	0	0.25	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

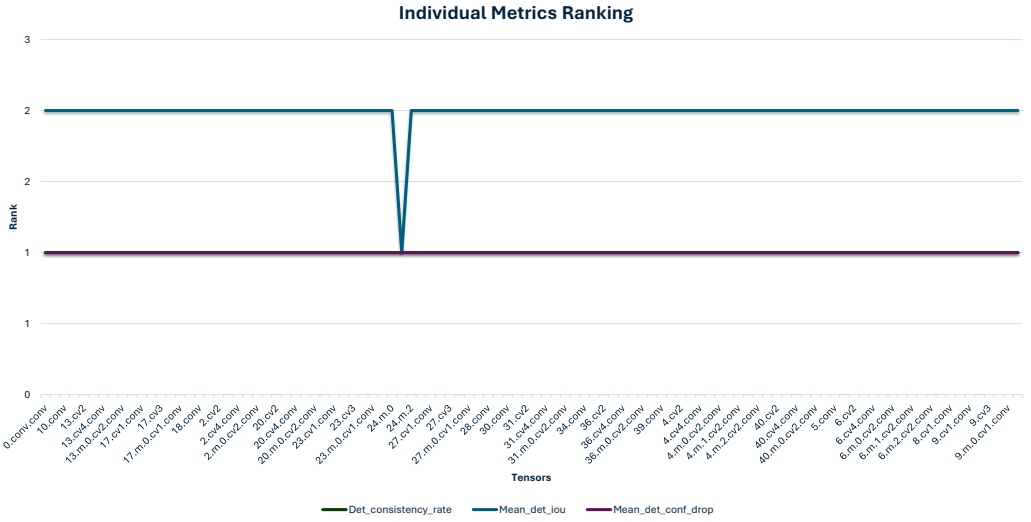
Figure A.27
LCH BW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.28
LCH BW Task Based Charts

LCH Drive Video Comparison BW Fault Fault

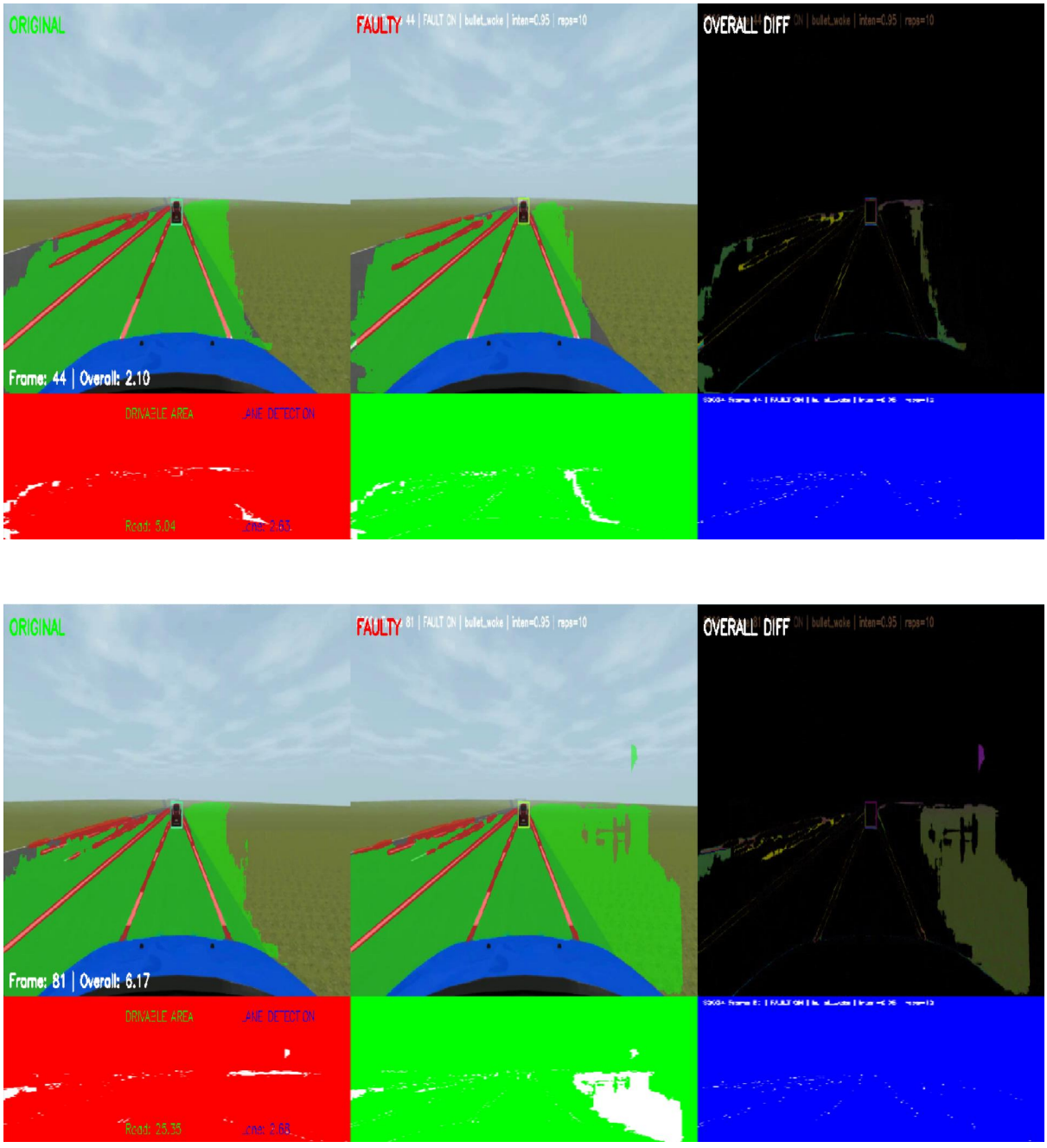


Figure A.29
Visual comparison of the LCH driving scenario under BW fault injection.

A.0.1.5 Shattered Glass Fault Injection LCH

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
eg IoU – Drivable Area	21	385	0.0545	0.0227	0.0319	0.0772	STOP	79	0
eg IoU – Lane Line	38	385	0.0987	0.0298	0.0689	0.1285	STOP	137	0
earson – Drivable Area	6	385	0.0156	0.0124	0.0032	0.0279	STOP	24	0
earson – Lane Line	11	385	0.0286	0.0166	0.0119	0.0452	STOP	43	0
rea RelDiff – Drivable Area	18	385	0.0468	0.0211	0.0257	0.0678	STOP	68	0
rea RelDiff – Lane Line	19	385	0.0494	0.0216	0.0277	0.0710	STOP	72	0
os Shift – Drivable Area	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
os Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
etection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
etection Mean IoU	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
etection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
verall (ANY metric violates its reshold)	34	385	0.0883	0.0283	0.0600	0.1166	STOP	124	0

Figure A.30
LCH SG Stop Criterion

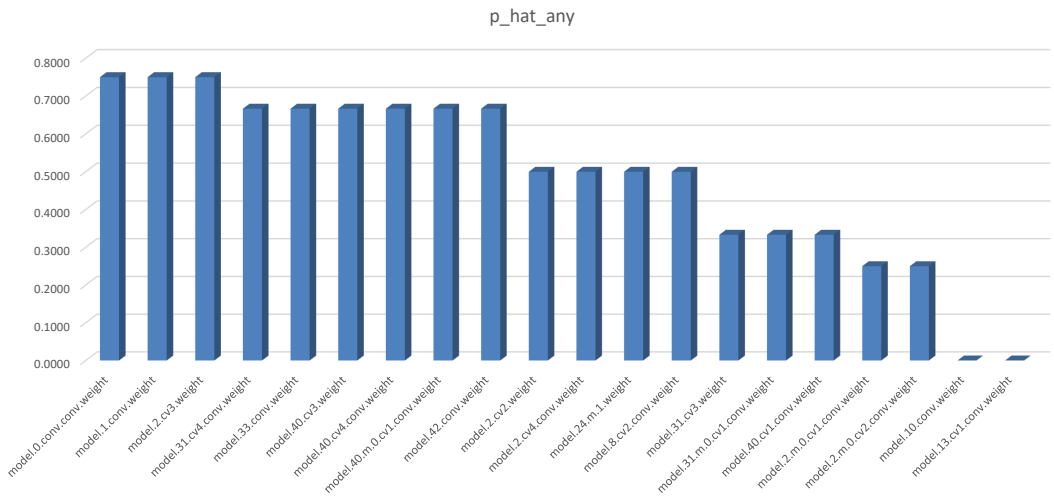
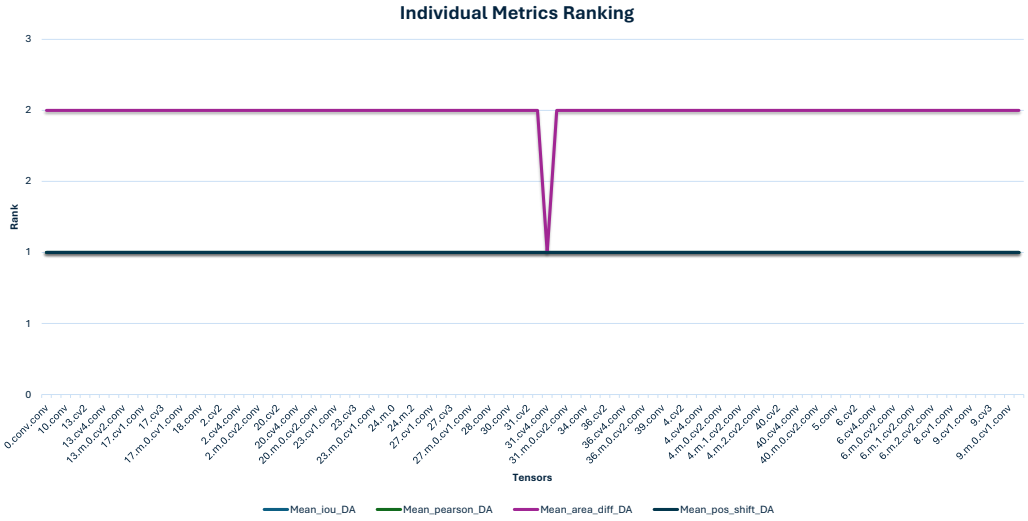


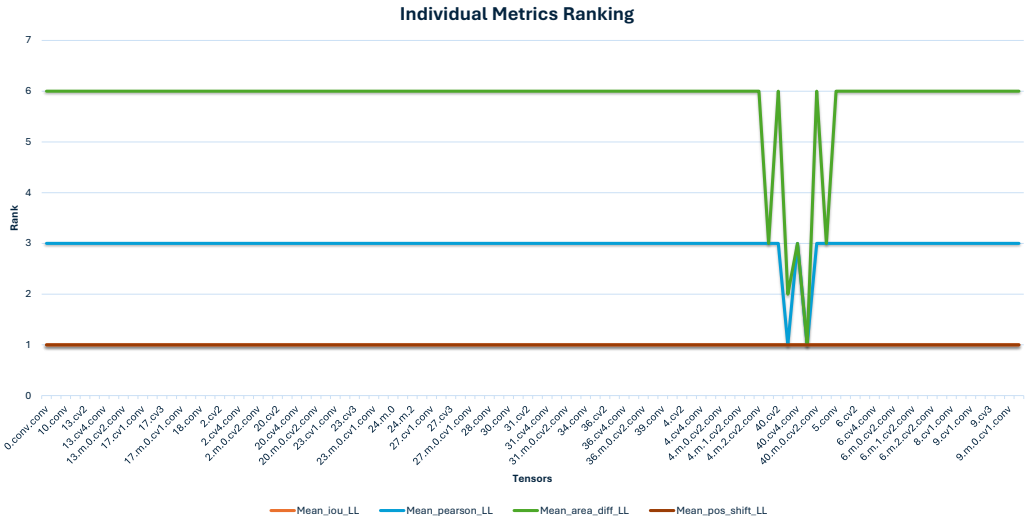
Figure A.31
LCH SG Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.0.conv.conv.weight	4	3	0.7500	0.7500	0.7500	0.2500	0.5000	0.5000	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
model.1.conv.weight	4	3	0.75	0.75	0.75	0.5	0.75	0.75	0.75	0.5	0	0	0	0
model.2.cv3.weight	4	3	0.75	0.75	0.75	0.5	0.5	0.75	0.75	0.5	0	0	0	0
model.31.cv4.conv.weight	3	2	0.666666667	0.333333333	0.333333	0.333333	0	0.666667	0	0	0	0	0	0
model.33.conv.weight	3	2	0.666666667	0.666666667	0.666667	0	0	0.666667	0	0	0	0	0	0
model.40.cv3.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.cv4.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.2.cv2.weight	4	2	0.5	0.5	0.5	0	0	0.5	0	0	0	0	0	0
model.2.cv4.conv.weight	4	2	0.5	0.5	0.5	0	0.25	0.25	0.25	0	0	0	0	0
model.24.m.1.weight	4	2	0.5	0	0	0	0	0	0	0	0	0	0.5	0
model.8.cv2.conv.weight	4	2	0.5	0.5	0.5	0	0	0	0	0	0	0	0	0
model.31.cv3.weight	3	1	0.333333333	0	0	0	0	0.333333	0	0	0	0	0	0
model.31.m.0.cv1.conv.weight	3	1	0.333333333	0.333333333	0.333333	0	0	0	0	0	0	0	0	0
model.40.cv1.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.2.m.0.cv1.conv.weight	4	1	0.25	0.25	0.25	0	0	0.25	0.25	0	0	0	0	0
model.2.m.0.cv2.conv.weight	4	1	0.25	0.25	0.25	0	0	0.25	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

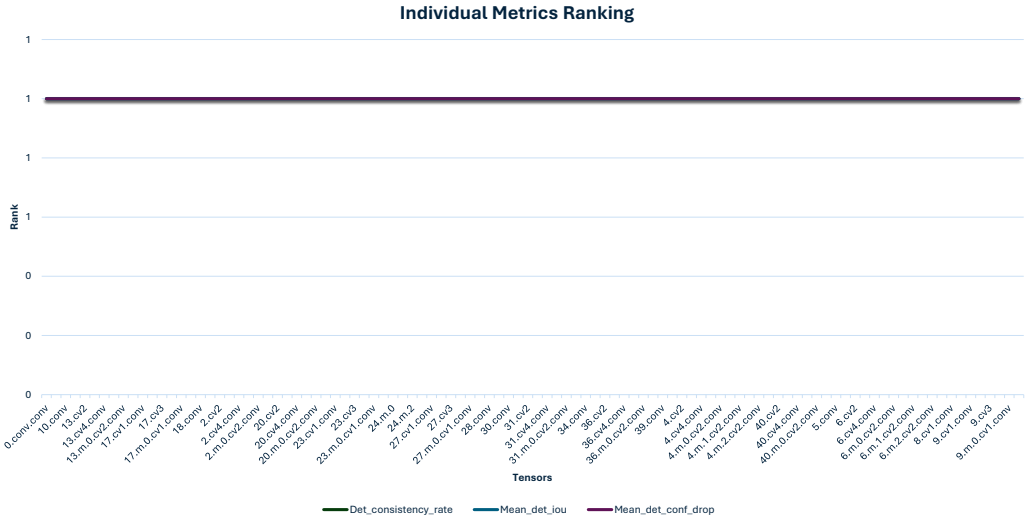
Figure A.32
LCH SG TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.33
LCH SG Task Based Charts

LCH Drive Video Comparison SG Fault Fault

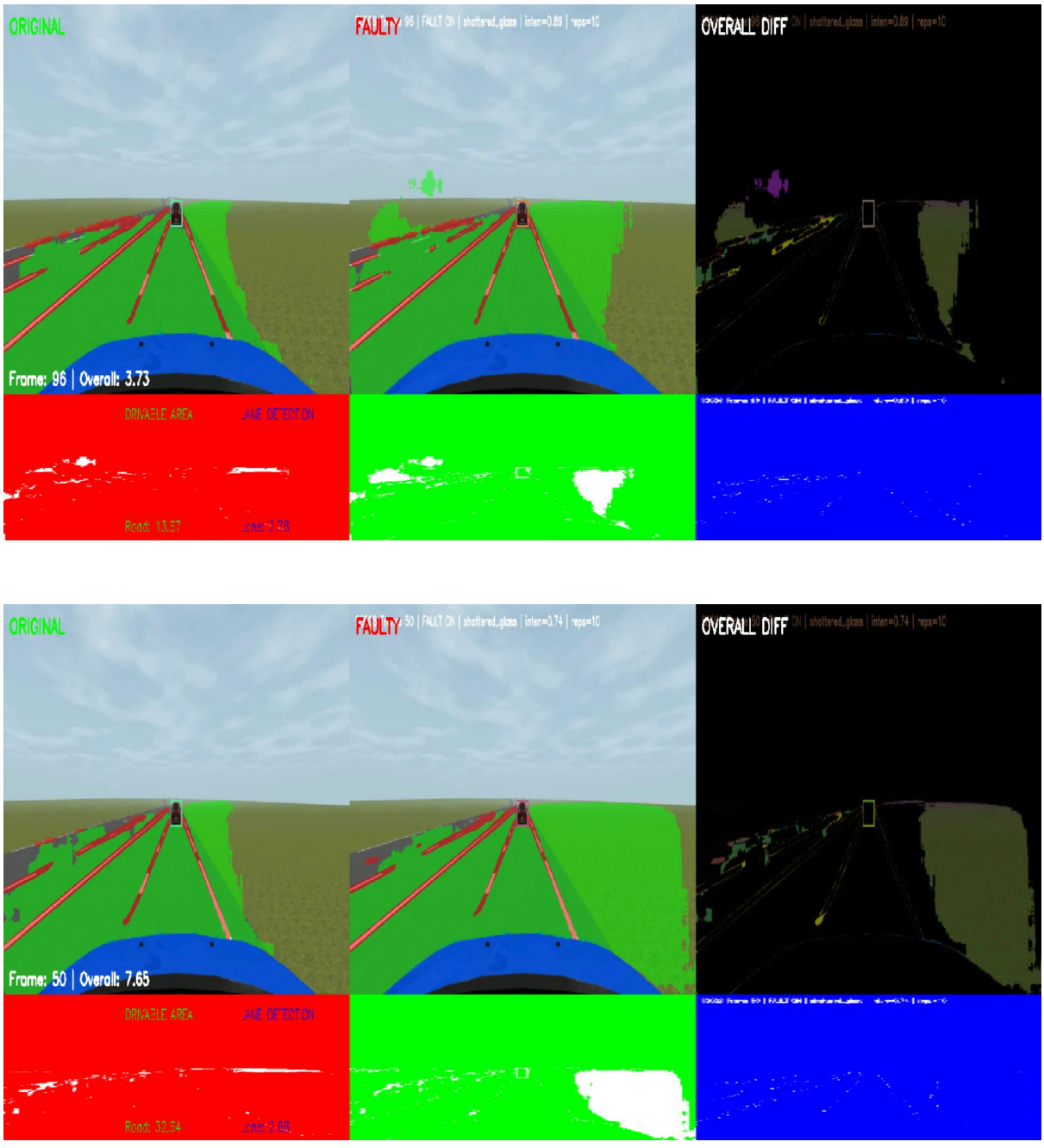


Figure A.34
Visual comparison of the LCH driving scenario under SG fault injection.

A.0.2 CUT OUT Driving Scenario

Figure A.35
CUT-OUT Driving Scenario

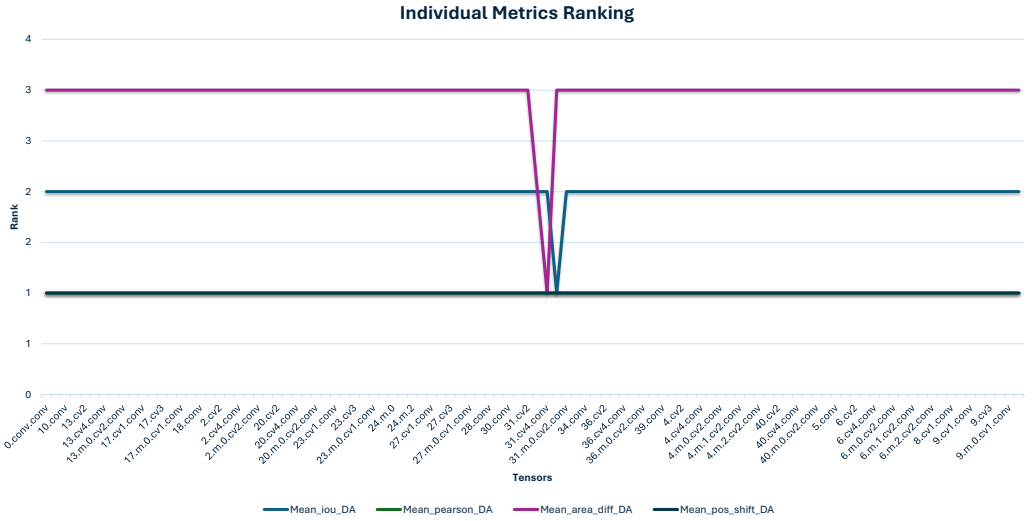
A.0.2.1 Single Point Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Seg IoU – Lane Line	11	385	0.0286	0.0166	0.0119	0.0452	STOP	43	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Area RelDiff – Drivable Area	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Area RelDiff – Lane Line	10	385	0.0260	0.0159	0.0101	0.0418	STOP	39	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	14	385	0.0364	0.0187	0.0177	0.0550	STOP	54	0

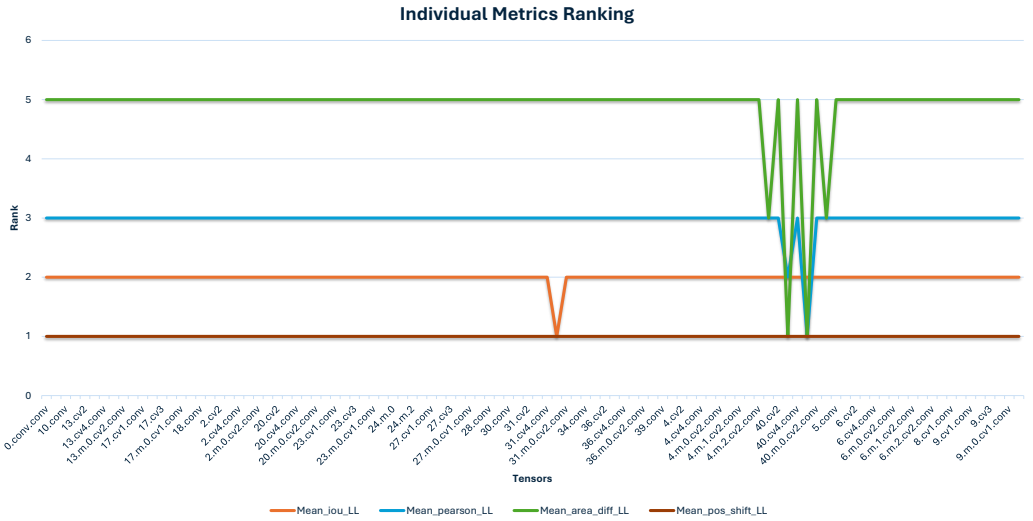
Figure A.36
CUTOUT Single Point Stop Criterion

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.cv3.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.3333	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.m.0.cv1.conv.weight	3	3	1	0	0	0	0.666667	0	1	0	0	0	0	0
model.31.cv4.conv.weight	3	2	0.666666667	0	0	0	0	0.666667	0	0	0	0	0	0
model.40.cv1.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.31.cv3.weight	3	1	0.333333333	0	0	0	0	0.333333	0	0	0	0	0	0
model.31.m.0.cv1.conv.weight	3	1	0.333333333	0.333333333	0.333333	0	0	0	0	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

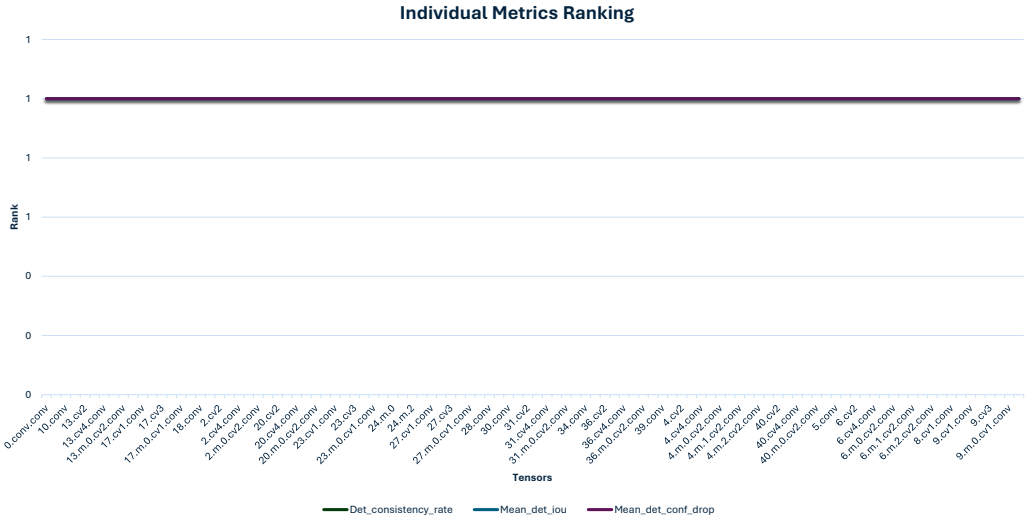
Figure A.37
CUTOUT SP TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.38
CUTOUT SP Task Based Charts

CUTOUT Drive Video Comparison Single Point Fault

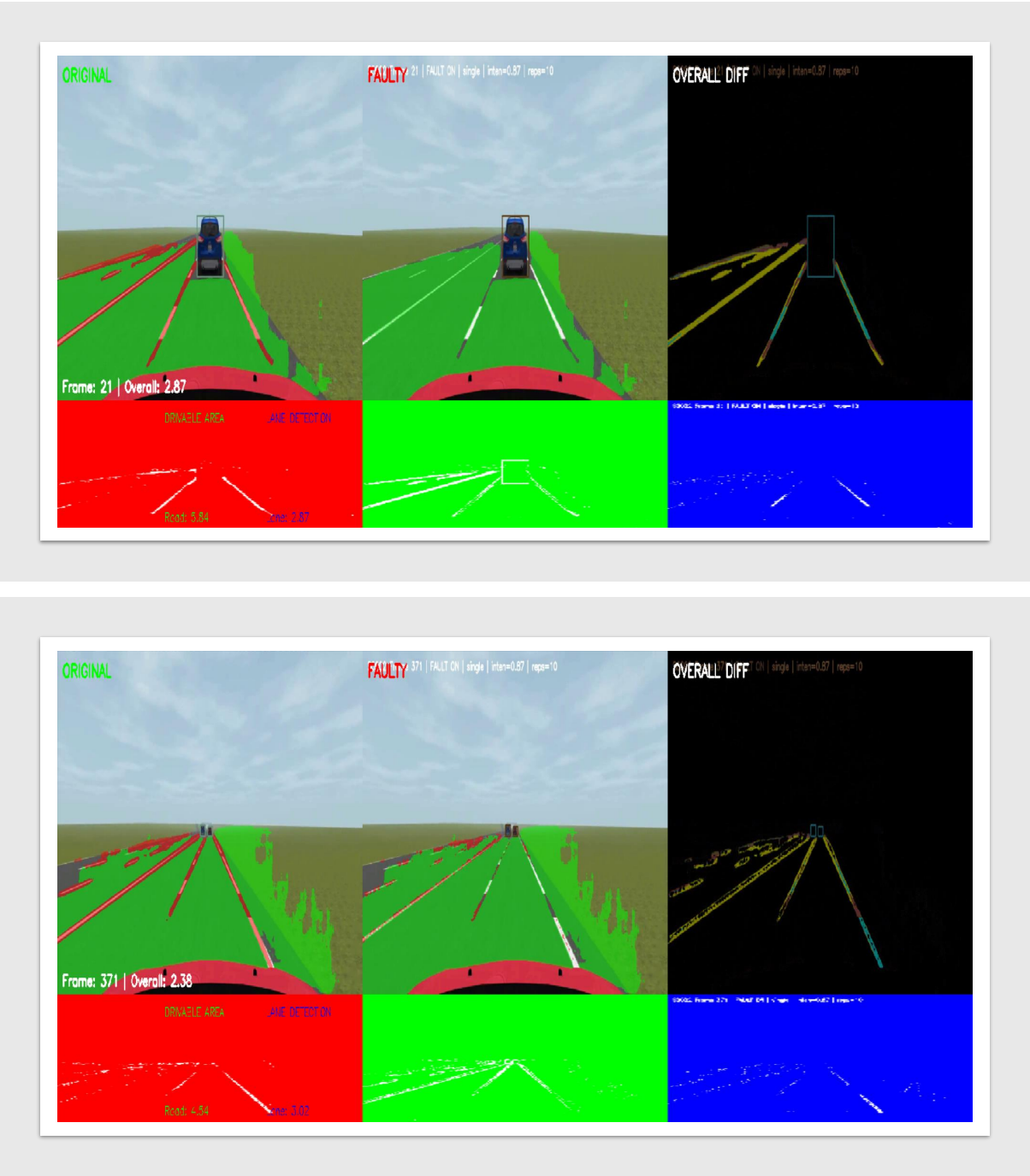


Figure A.39
Visual comparison of the CUTOUT driving scenario under SP fault injection.

A.0.2.2 ROW Fault Injection CUTOUT

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Seg IoU – Lane Line	6	385	0.0156	0.0124	0.0032	0.0279	STOP	24	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	6	385	0.0156	0.0124	0.0032	0.0279	STOP	24	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	8	385	0.0208	0.0142	0.0065	0.0350	STOP	31	0

Figure A.40
CUTOUT ROW Stop Criterion

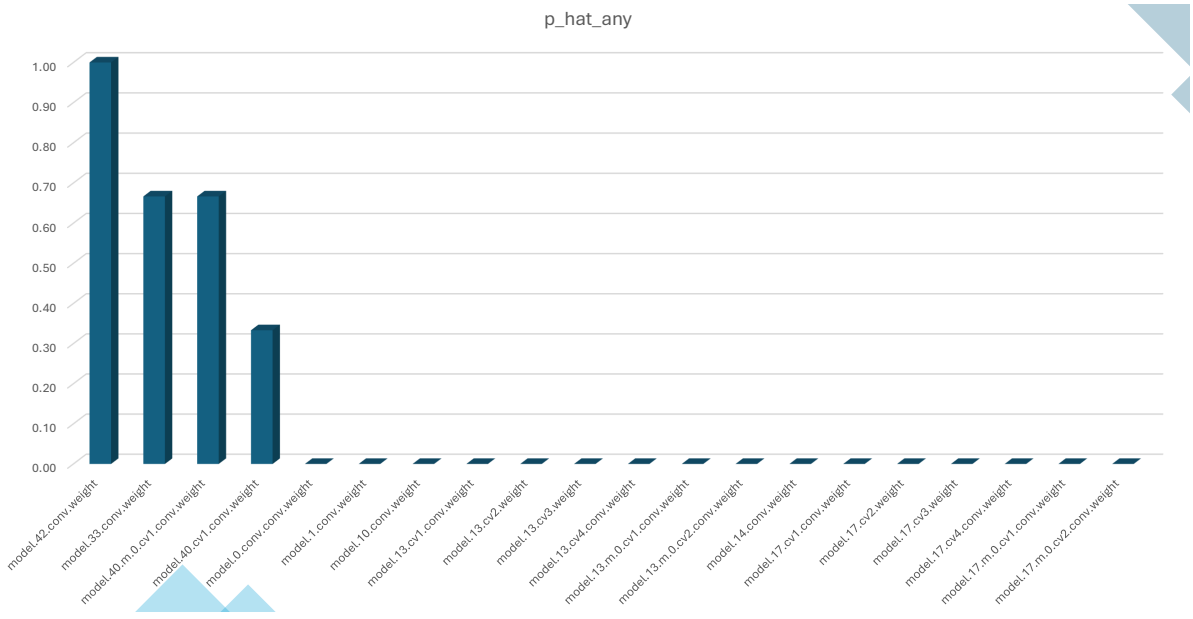
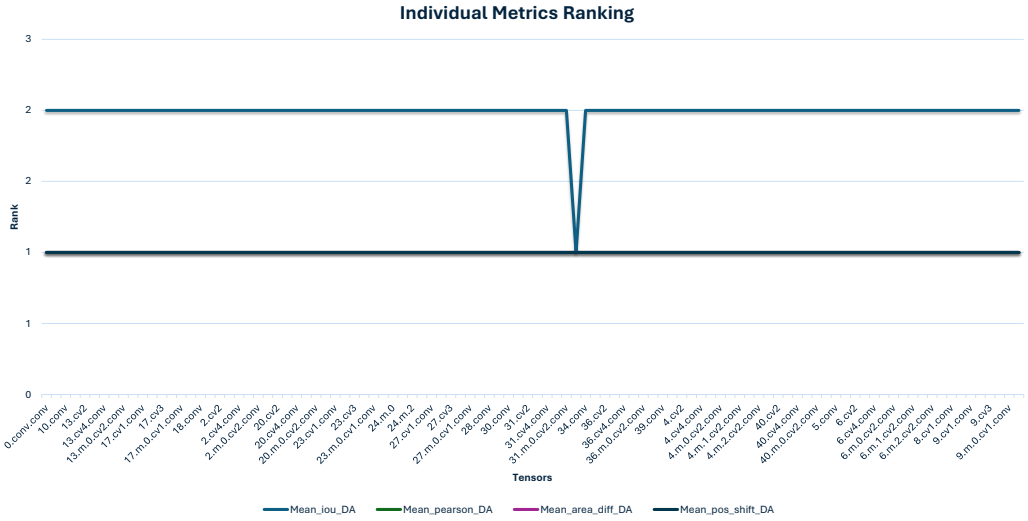


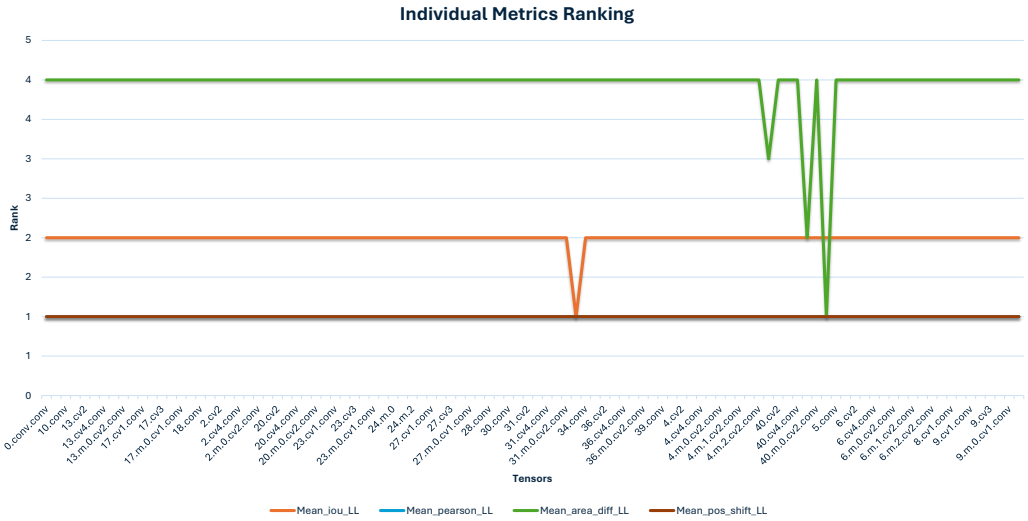
Figure A.41
CUTOUT ROW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.42.conv.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.33.conv.weight	3	2	0.666666667	0.666666667	0.666667	0	0	0	0	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.cv1.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

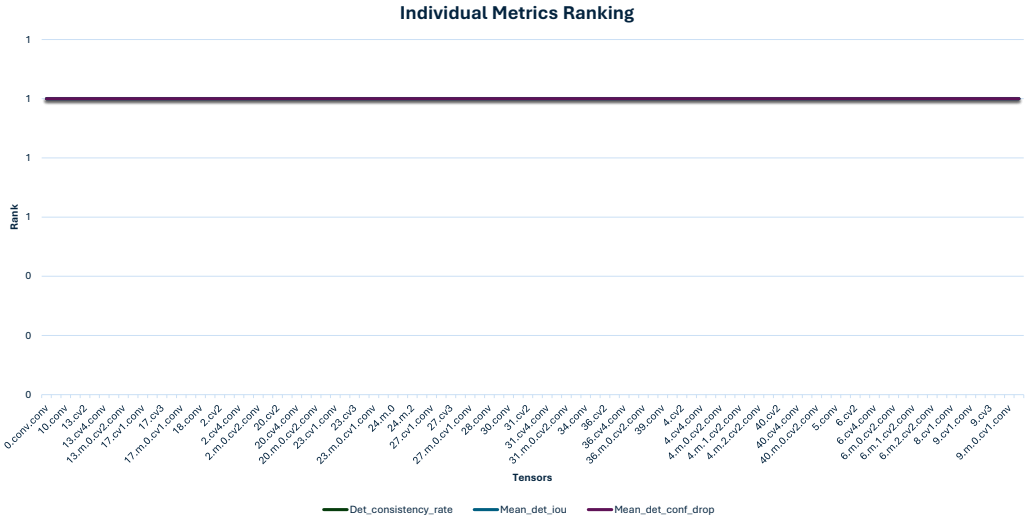
Figure A.42
CUTOUT ROW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.43
CUTOUT ROW Task Based Charts

CUTOUT Drive Video Comparison ROW Fault

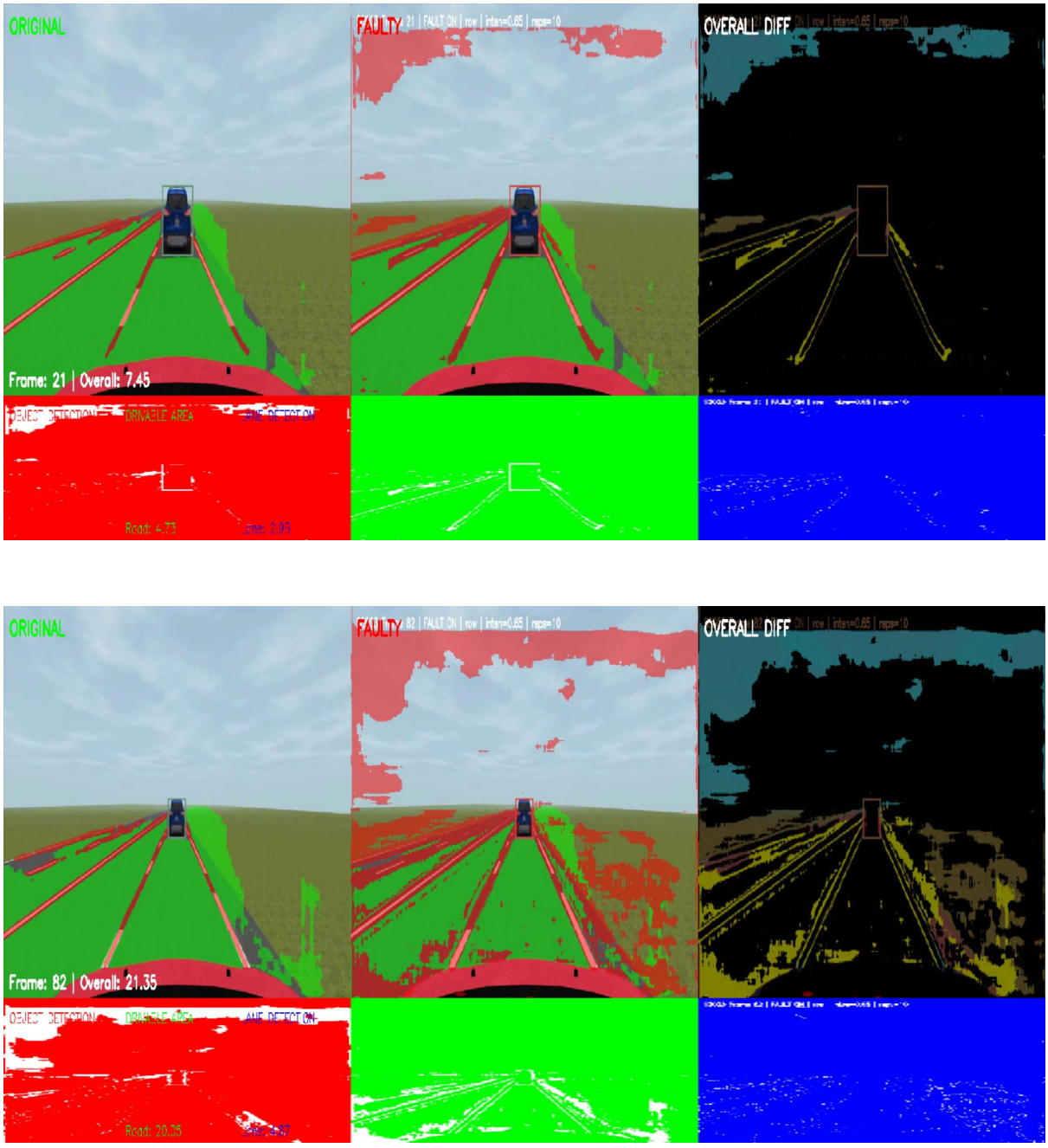


Figure A.44
Visual comparison of the CUTOUT driving scenario under ROW fault injection.

A.0.2.3 Column Fault Injection CUTOUT

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0

Figure A.45
CUTOUT COL Stop Criterion

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.m.0.cv1.conv.weight	3	1	0.3333	0.0000	0.0000	0.0000	0.0000	0.0000	0.3333	0.0000	0.0000	0.0000	0.0000	0.0000
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.2.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.2.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.46
CUTOUT COL TOP 20 Vulnerable Tensors

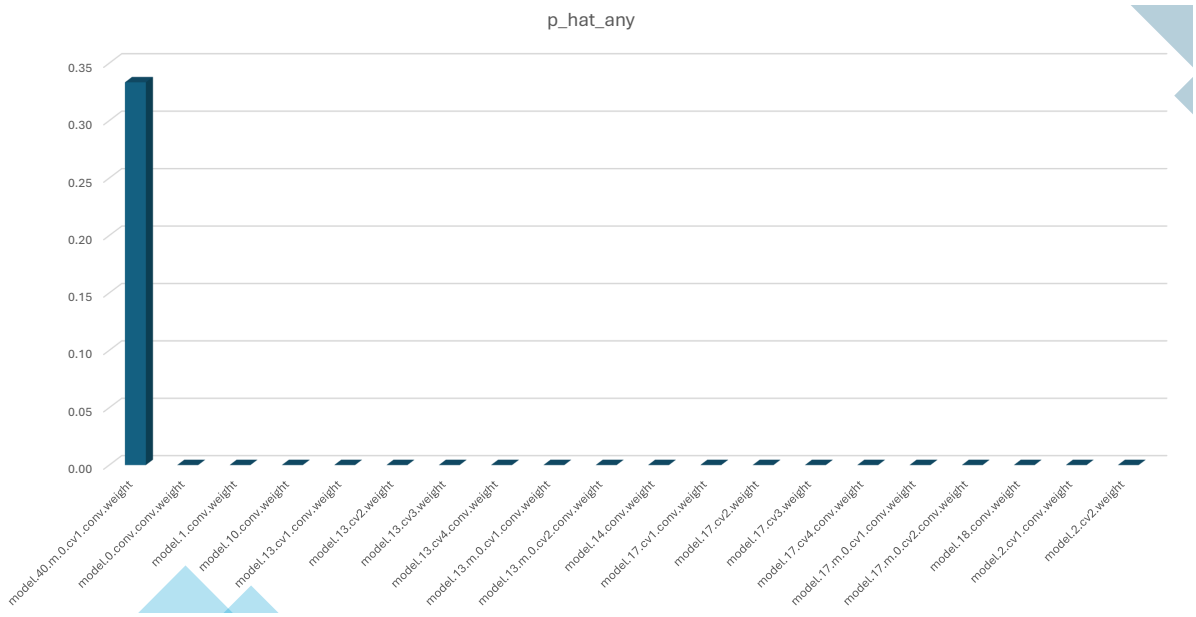
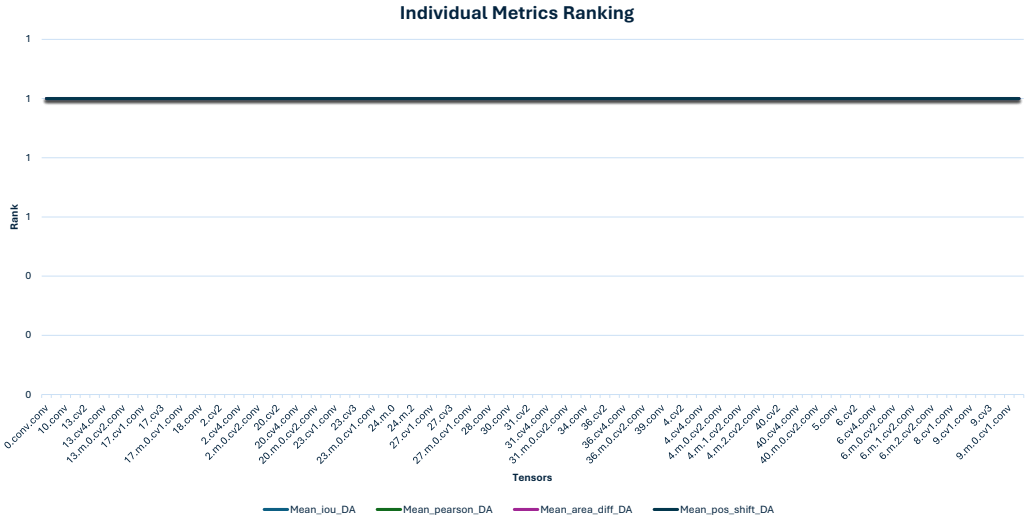
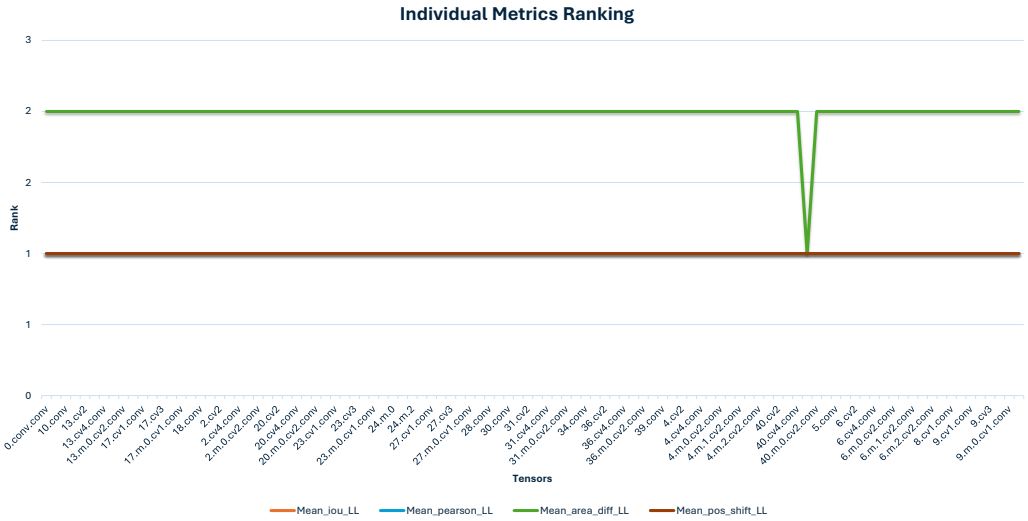


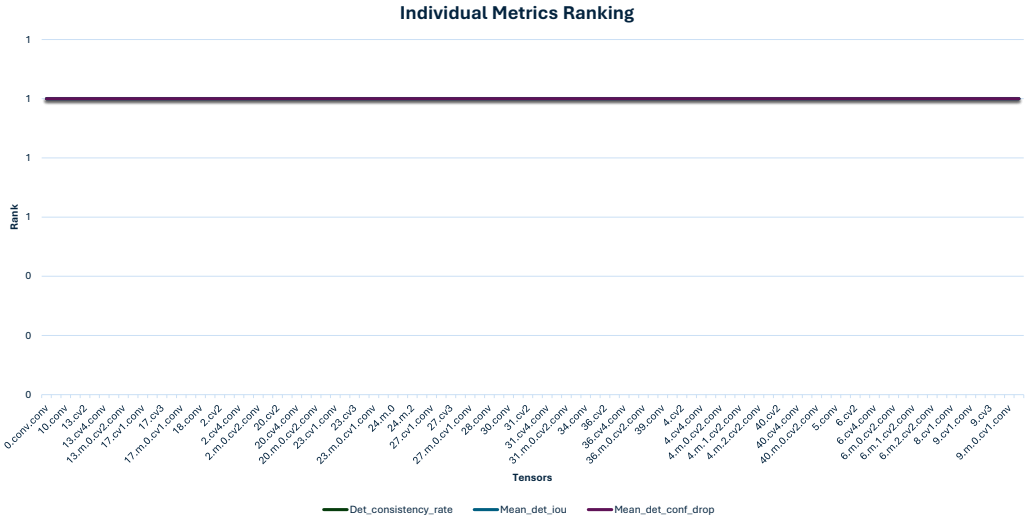
Figure A.47
CUTOUT COL Bar Graph



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.48
CUTOUT COL Task Based Charts

CUTOUT Drive Video Comparison COL Fault

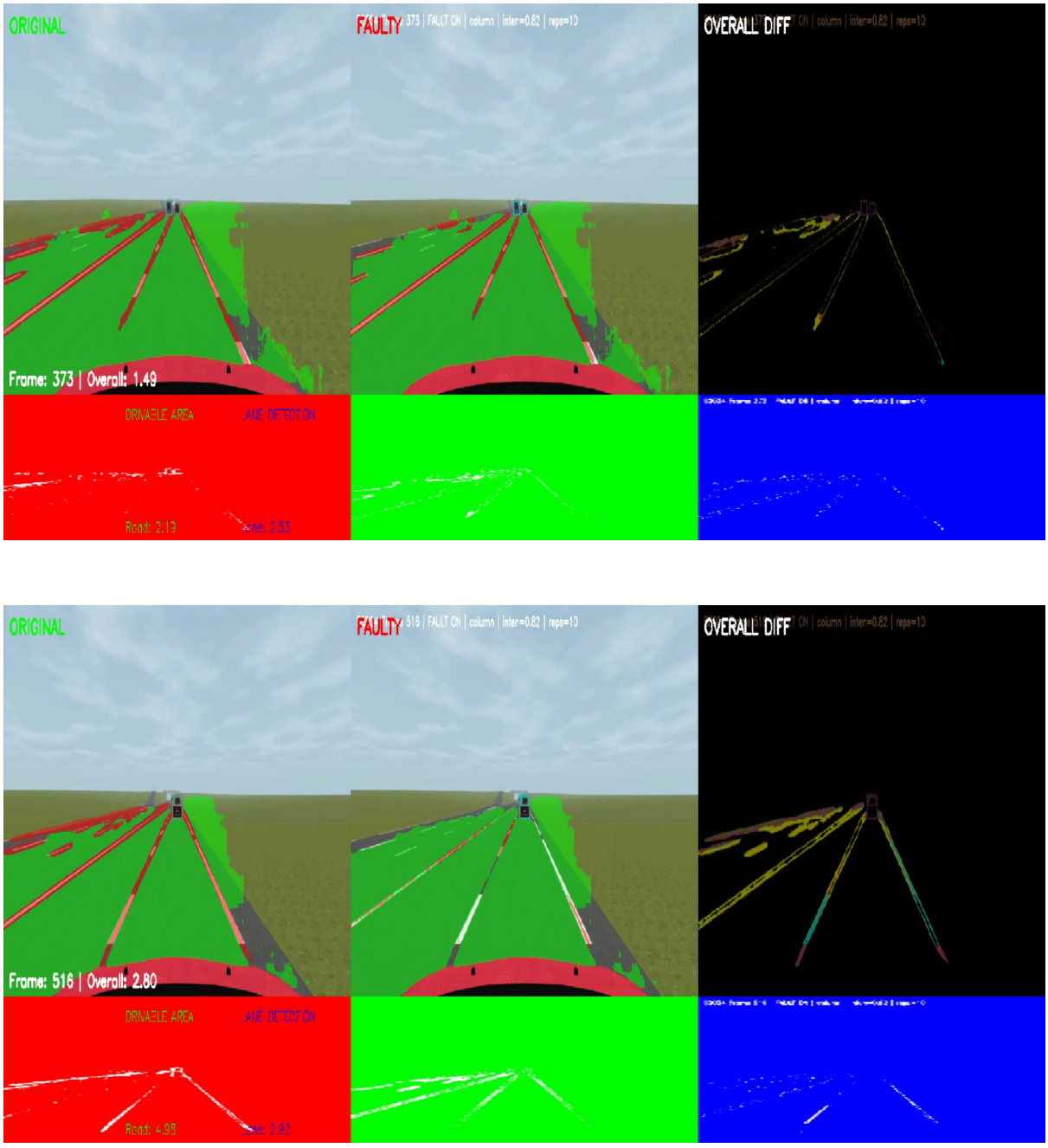


Figure A.49
Visual comparison of the CUTOUT driving scenario under ROW fault injection.

A.0.2.4 Bullet Wake Fault Injection CUTOUT

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	21	385	0.0545	0.0227	0.0319	0.0772	STOP	79	0
Seg IoU – Lane Line	38	385	0.0987	0.0298	0.0689	0.1285	STOP	137	0
Pearson – Drivable Area	7	385	0.0182	0.0133	0.0048	0.0315	STOP	27	0
Pearson – Lane Line	9	385	0.0234	0.0151	0.0083	0.0385	STOP	35	0
Area RelDiff – Drivable Area	15	385	0.0390	0.0193	0.0197	0.0583	STOP	58	0
Area RelDiff – Lane Line	19	385	0.0494	0.0216	0.0277	0.0710	STOP	72	0
Pos Shift – Drivable Area	5	385	0.0130	0.0113	0.0017	0.0243	STOP	20	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	32	385	0.0831	0.0275	0.0556	0.1107	STOP	117	0

Figure A.50
CUTOUT BW Stop Criterion

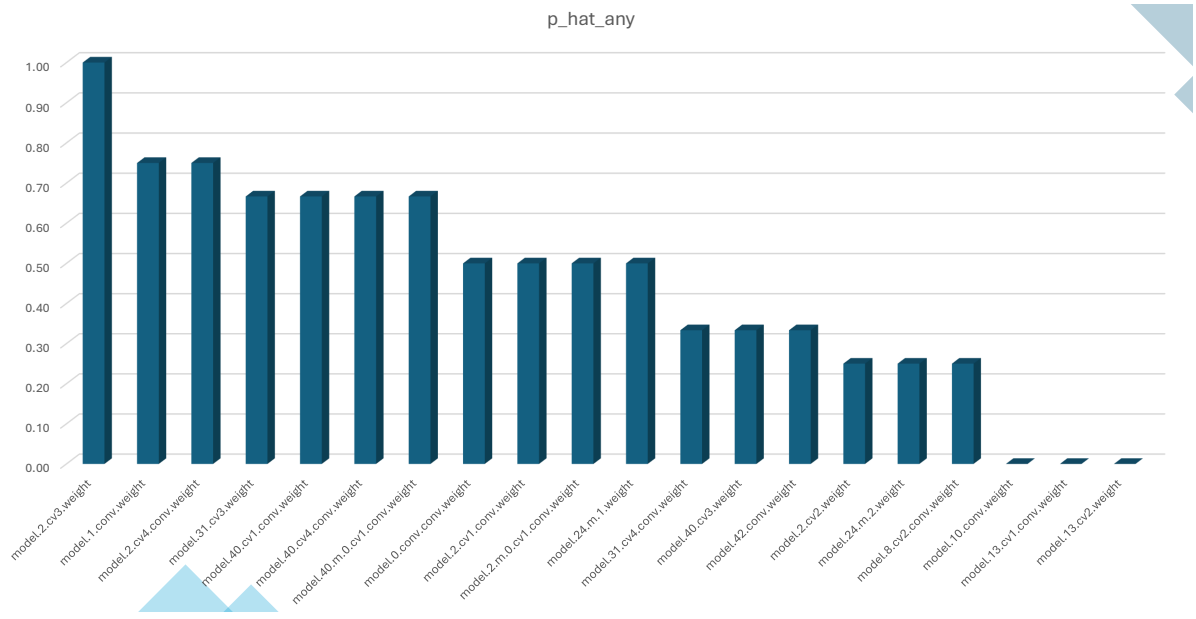
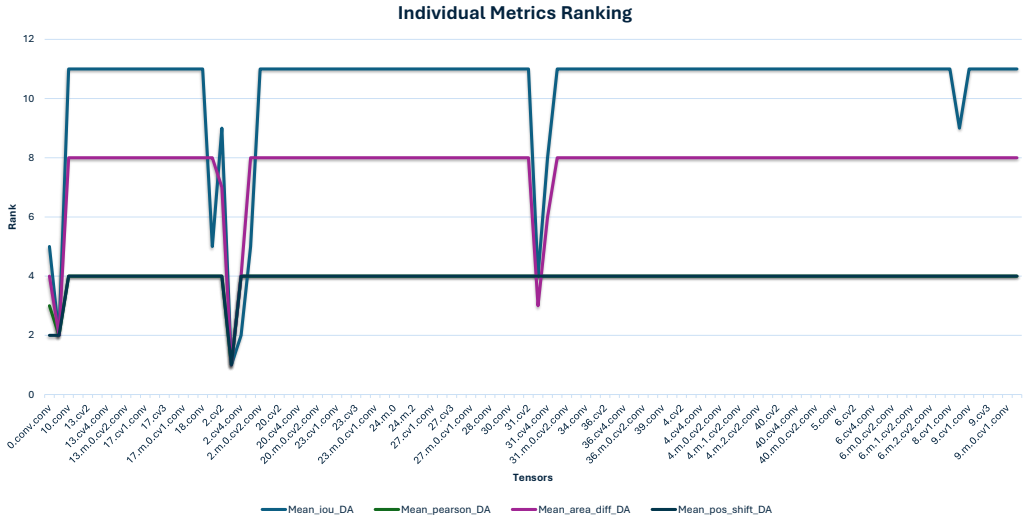


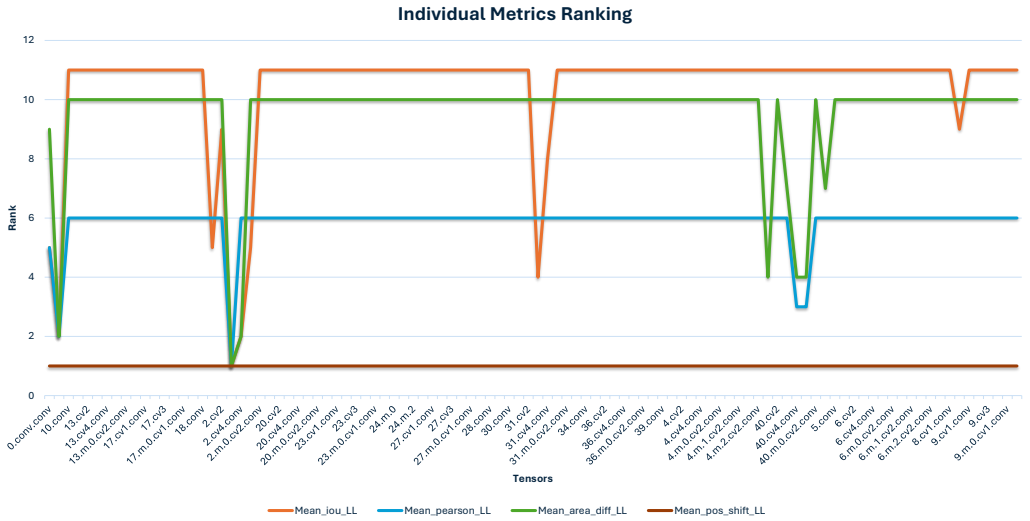
Figure A.51
CUTOUT BW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.2.cv3.weight	4	4	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.7500	0.0000	0.0000	0.0000	0.0000
model.1.conv.weight	4	3	0.75	0.75	0.75	0.5	0.5	0.75	0.75	0.25	0	0	0.25	0
model.2.cv4.conv.weight	4	3	0.75	0.75	0.75	0	0	0.5	0.75	0	0	0	0	0
model.31.cv3.weight	3	2	0.666666667	0.666666667	0.666667	0	0	0.666667	0	0	0	0	0	0
model.40.cv1.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.cv4.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.0.conv.conv.weight	4	2	0.5	0.5	0.5	0.25	0.25	0.5	0.25	0.25	0	0	0	0
model.2.cv1.conv.weight	4	2	0.5	0.5	0.5	0	0	0	0	0	0	0	0	0
model.2.m.0.cv1.conv.weight	4	2	0.5	0.5	0.5	0	0	0	0	0	0	0	0	0
model.24.m.1.weight	4	2	0.5	0	0	0	0	0	0	0	0	0	0.5	0
model.31.cv4.conv.weight	3	1	0.333333333	0.333333333	0.333333	0	0	0.333333	0	0	0	0	0	0
model.40.cv3.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.42.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.2.cv2.weight	4	1	0.25	0.25	0.25	0	0	0.25	0	0	0	0	0	0
model.24.m.2.weight	4	1	0.25	0	0	0	0	0	0	0	0	0	0.25	0
model.8.cv2.conv.weight	4	1	0.25	0.25	0.25	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

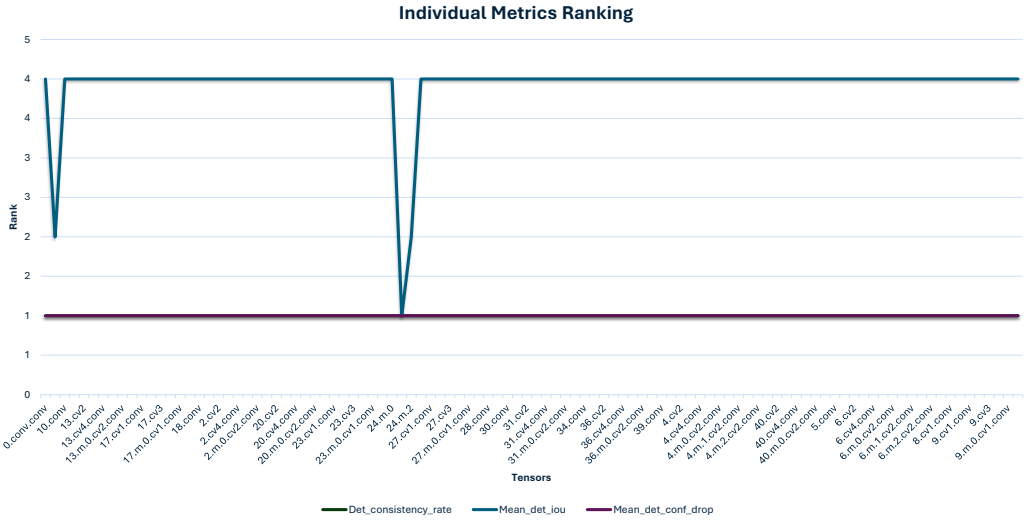
Figure A.52
CUTOUT BW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.53
CUTOUT BW Task Based Charts

CUTOUT Drive Video Comparison BW Fault

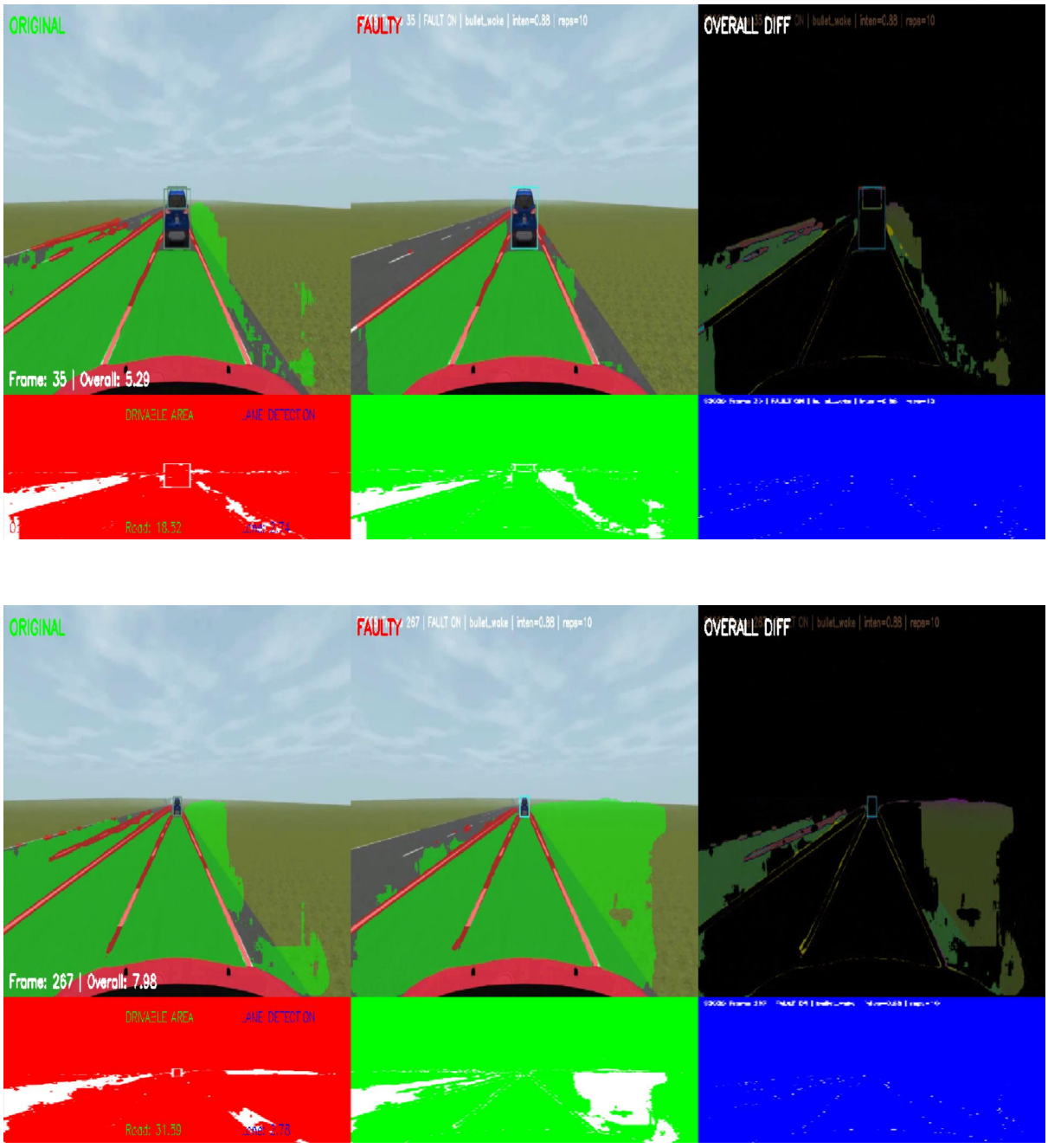


Figure A.54
Visual comparison of the CUTOUT driving scenario under BW fault injection.

A.0.2.5 Shattered Glass Wake Fault Injection CUTOUT

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	25	385	0.0649	0.0246	0.0403	0.0895	STOP	93	0
Seg IoU – Lane Line	44	385	0.1143	0.0318	0.0825	0.1460	STOP	155	0
Pearson – Drivable Area	6	385	0.0156	0.0124	0.0032	0.0279	STOP	24	0
Pearson – Lane Line	10	385	0.0260	0.0159	0.0101	0.0418	STOP	39	0
Area RelDiff – Drivable Area	17	385	0.0442	0.0205	0.0237	0.0647	STOP	65	0
Area RelDiff – Lane Line	22	385	0.0571	0.0232	0.0340	0.0803	STOP	83	0
Pos Shift – Drivable Area	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pos Shift – Lane Line	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	43	385	0.1117	0.0314	0.0803	0.1431	STOP	152	0

Figure A.55
CUTOUT SG Stop Criterion

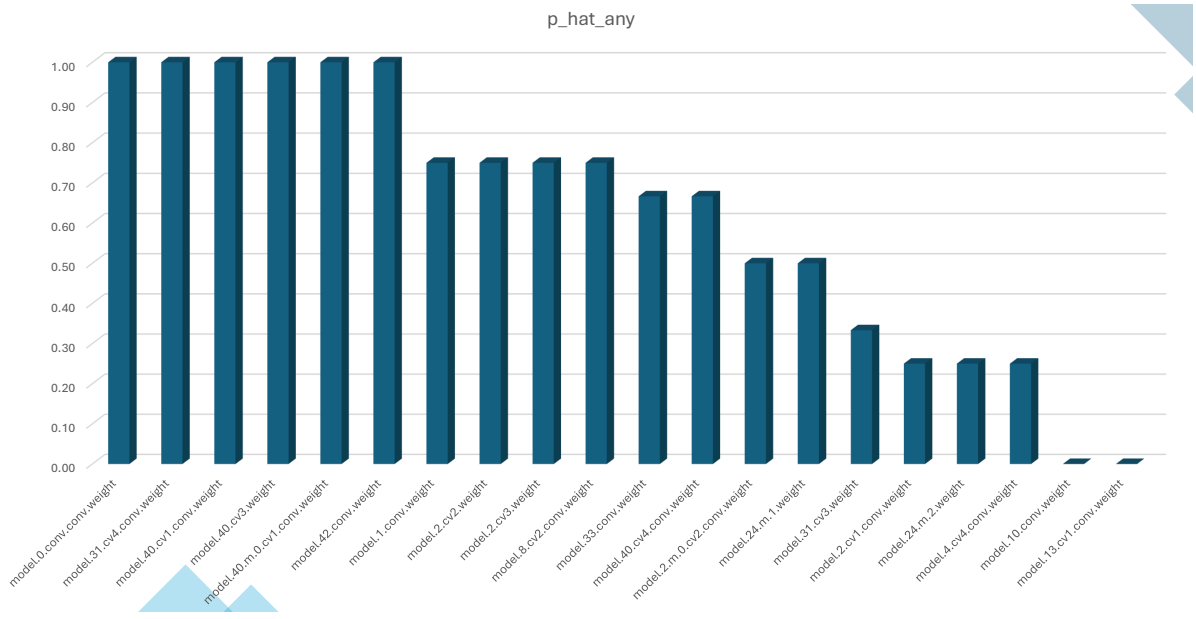
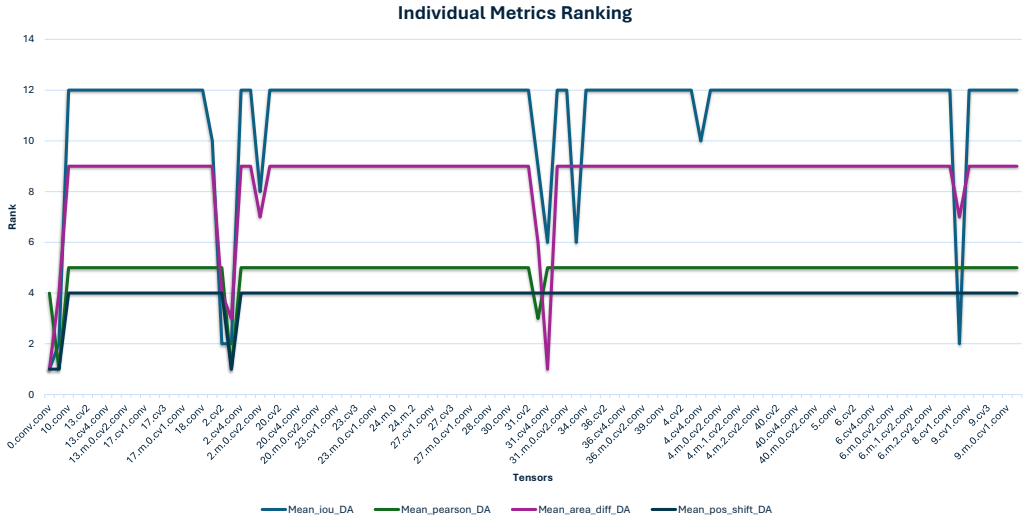


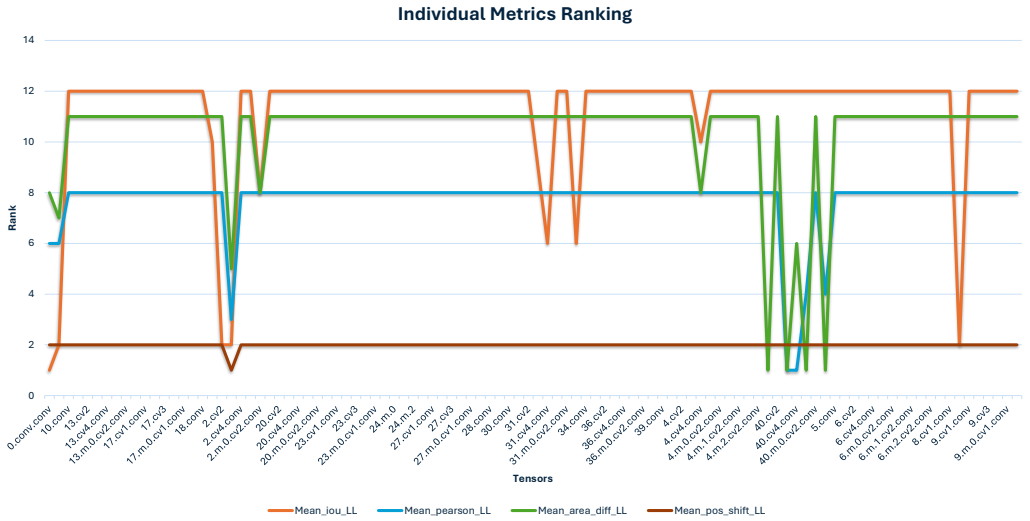
Figure A.56
CUTOUT SG Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.0.conv.conv.weight	4	4	1.0000	1.0000	1.0000	0.2500	0.2500	1.0000	0.2500	0.2500	0.0000	0.0000	0.0000	0.0000
model.31.cv4.conv.weight	3	3	1	0.666666667	0.666667	0	0	1	0	0	0	0	0	0
model.40.cv1.conv.weight	3	3	1	0	0	0	0	0	1	0	0	0	0	0
model.40.cv3.weight	3	3	1	0	0	0	0.666667	0	1	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	3	1	0	0	0	0.333333	0	1	0	0	0	0	0
model.42.conv.weight	3	3	1	0	0	0	0.333333	0	1	0	0	0	0	0
model.1.conv.weight	4	3	0.75	0.75	0.75	0.5	0.25	0.5	0.5	0.25	0	0	0	0
model.2.cv2.weight	4	3	0.75	0.75	0.75	0	0	0.5	0	0	0	0	0	0
model.2.cv3.weight	4	3	0.75	0.75	0.75	0.5	0.5	0.75	0.75	0.25	0.25	0	0	0
model.8.cv2.conv.weight	4	3	0.75	0.75	0.75	0	0	0.25	0	0	0	0	0	0
model.33.conv.weight	3	2	0.666666667	0.666666667	0.666667	0	0	0	0	0	0	0	0	0
model.40.cv4.conv.weight	3	2	0.666666667	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.2.m.0.cv2.conv.weight	4	2	0.5	0.5	0.5	0	0	0.25	0.25	0	0	0	0	0
model.24.m.1.weight	4	2	0.5	0	0	0	0	0	0	0	0	0	0.5	0
model.31.cv3.weight	3	1	0.333333333	0.333333333	0.333333	0.333333	0	0.333333	0	0	0	0	0	0
model.2.cv1.conv.weight	4	1	0.25	0.25	0.25	0	0	0	0	0	0	0	0	0
model.24.m.2.weight	4	1	0.25	0	0	0	0	0	0	0	0	0	0.25	0
model.4.cv4.conv.weight	4	1	0.25	0.25	0.25	0	0	0	0.25	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

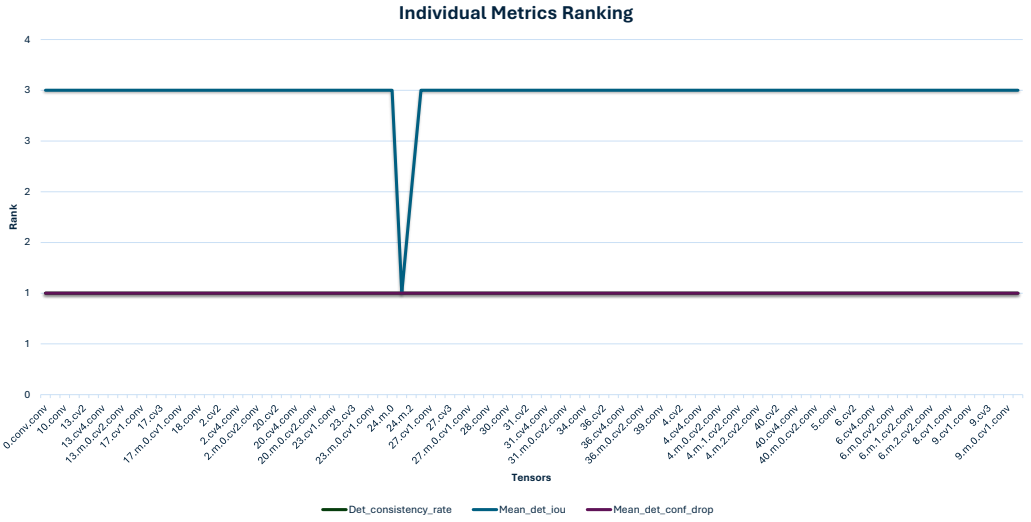
Figure A.57
CUTOUT SG TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.58
CUTOUT SG Task Based Charts

CUTOUT Drive Video Comparison SG Fault

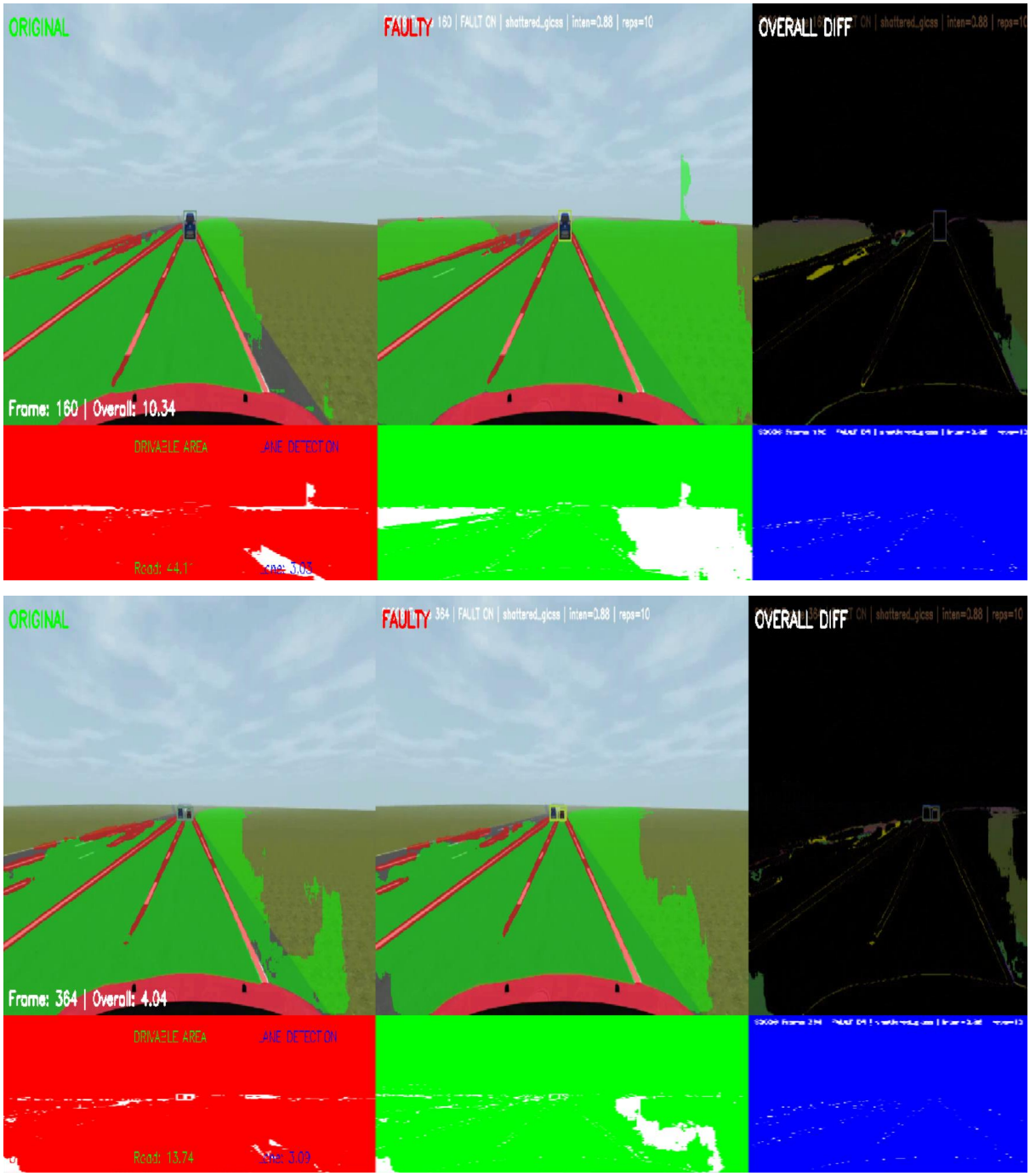


Figure A.59
Visual comparison of the CUTOUT driving scenario under SG fault injection.

A.0.3 CUT IN Driving Scenario

Figure A.60
CUT IN Driving Scenario

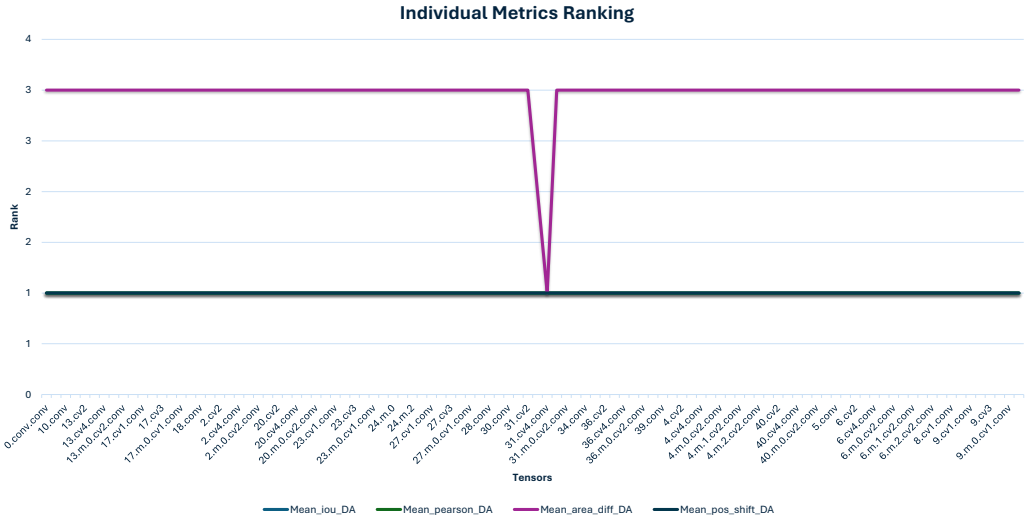
A.0.3.1 CUT-IN Single Point Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	10	385	0.0260	0.0159	0.0101	0.0418	STOP	39	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Area RelDiff – Drivable Area	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Area RelDiff – Lane Line	10	385	0.0260	0.0159	0.0101	0.0418	STOP	39	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	13	385	0.0338	0.0180	0.0157	0.0518	STOP	50	0

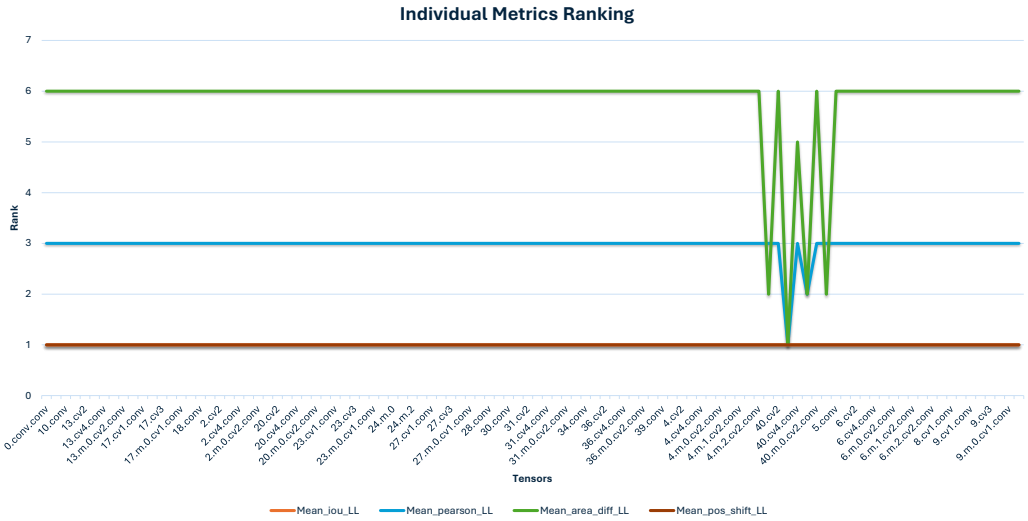
Figure A.61
CUTIN Single Point Stop Criterion

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.cv3.weight	3	3	1.00	0.0000	0.0000	0.0000	0.6667	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.31.cv4.conv.weight	3	2	0.67	0	0	0	0	0.666667	0	0	0	0	0	0
model.40.cv1.conv.weight	3	2	0.67	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.67	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.67	0	0	0	0	0	0.666667	0	0	0	0	0
model.31.cv3.weight	3	1	0.33	0	0	0	0	0.333333	0	0	0	0	0	0
model.40.cv4.conv.weight	3	1	0.33	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0

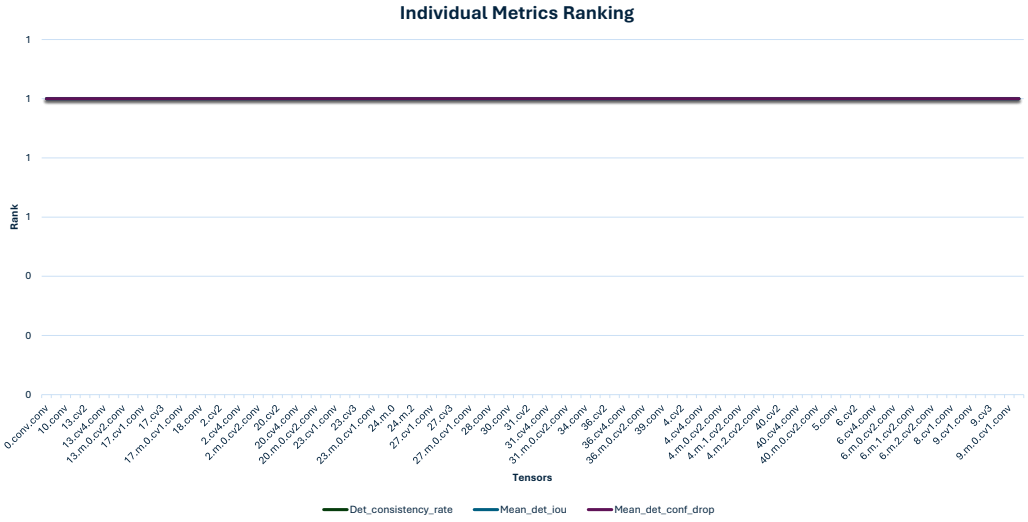
Figure A.62
CUTIN SP TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.63
CUTIN SP Task Based Charts

CUTIN Drive Video Comparison Single Point Fault

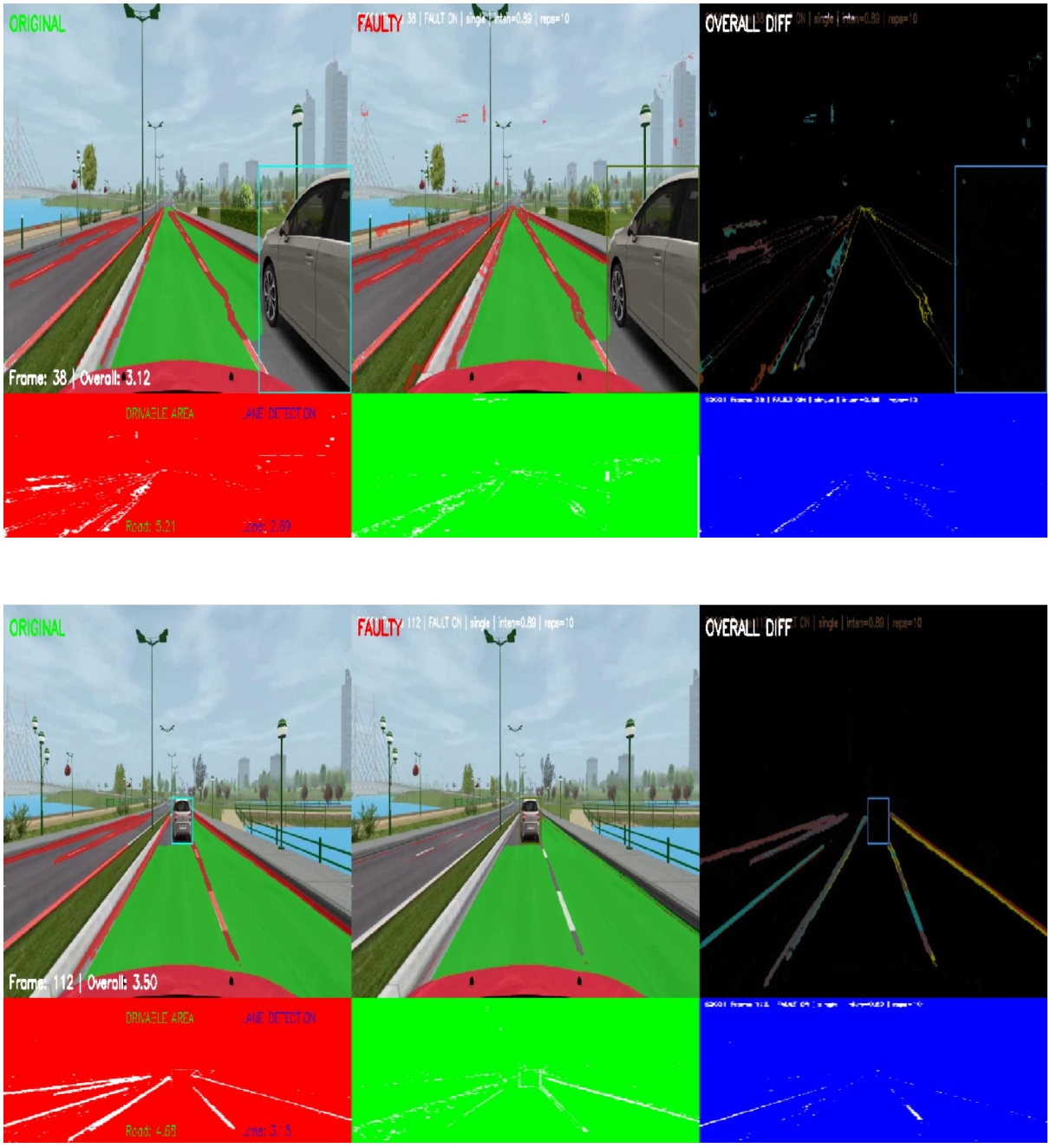


Figure A.64
Visual comparison of the CUT IN driving scenario under SP fault injection.

A.0.3.2 CUT-IN ROW Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	5	385	0.0130	0.0113	0.0017	0.0243	STOP	20	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0

Figure A.65
CUTIN ROW Stop Criterion

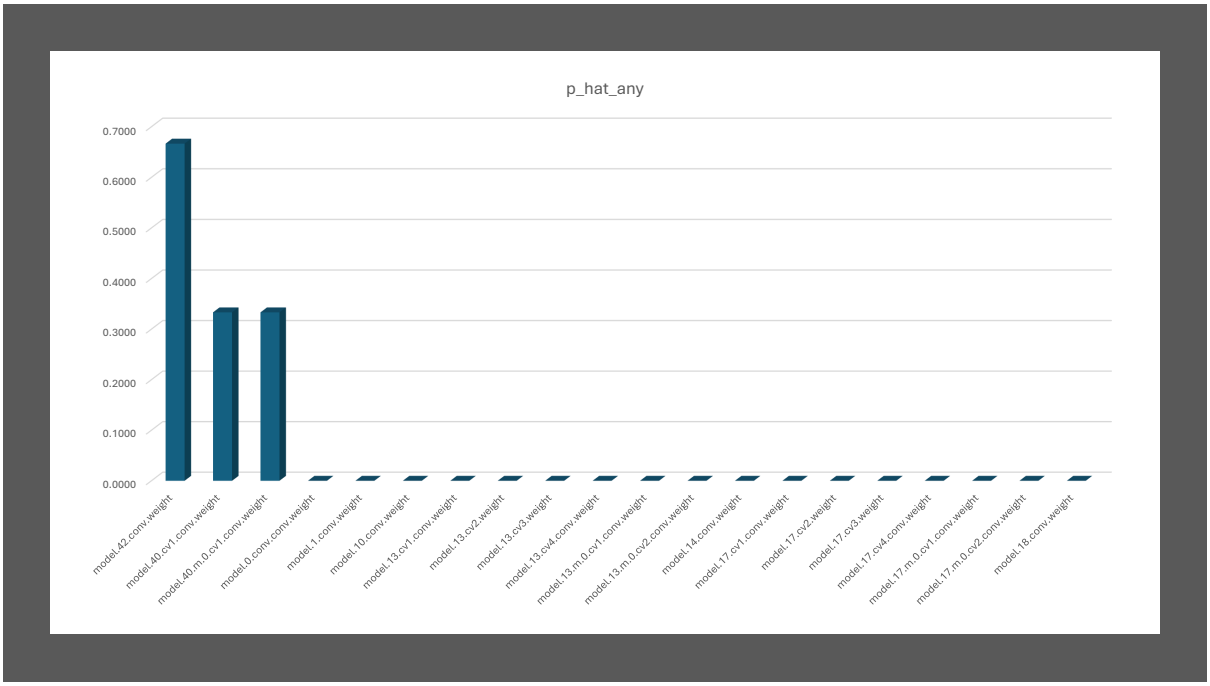
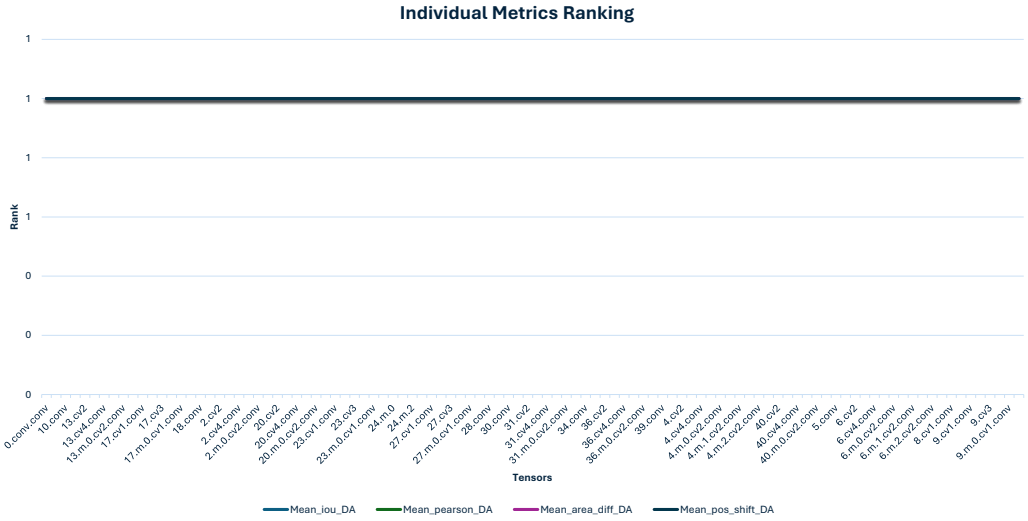


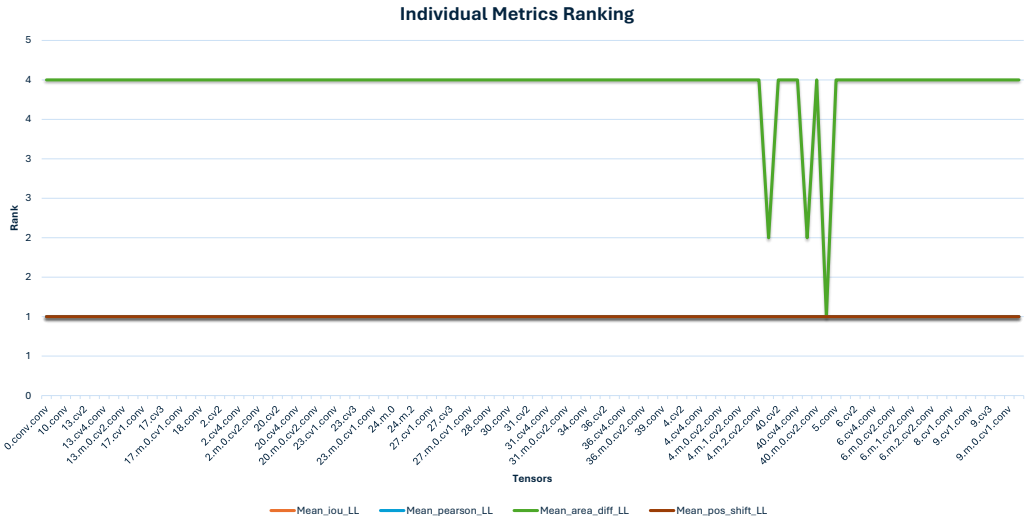
Figure A.66
CUTIN ROW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.42.conv.weight	3	2	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.cv1.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

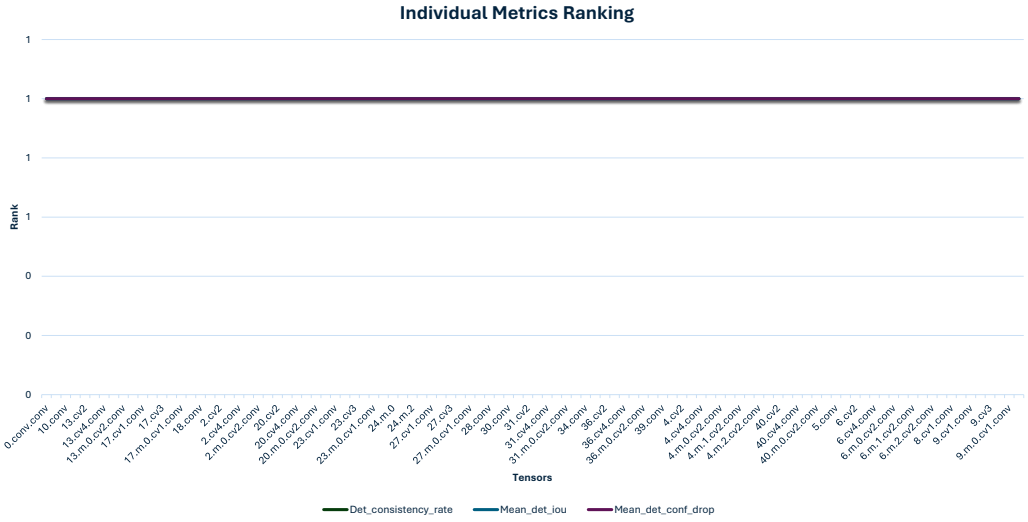
Figure A.67
CUTIN ROW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.68
CUTIN ROW Task Based Charts

CUTIN Drive Video Comparison ROW Fault

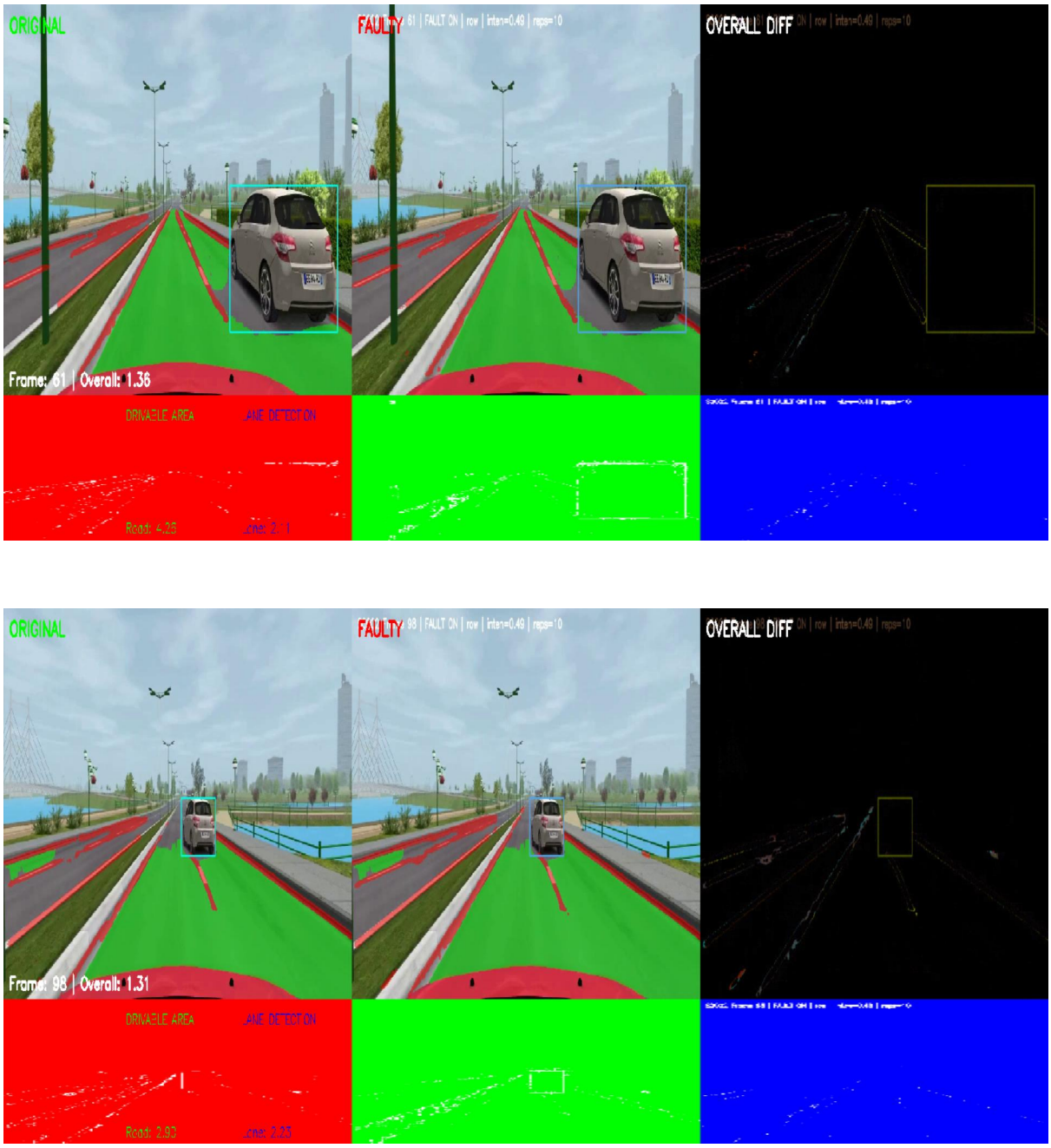


Figure A.69
Visual comparison of the CUT IN driving scenario under ROW fault injection.

A.0.3.3 CUT-IN Column Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Lane Line	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0

Figure A.70
CUTIN COL Stop Criterion

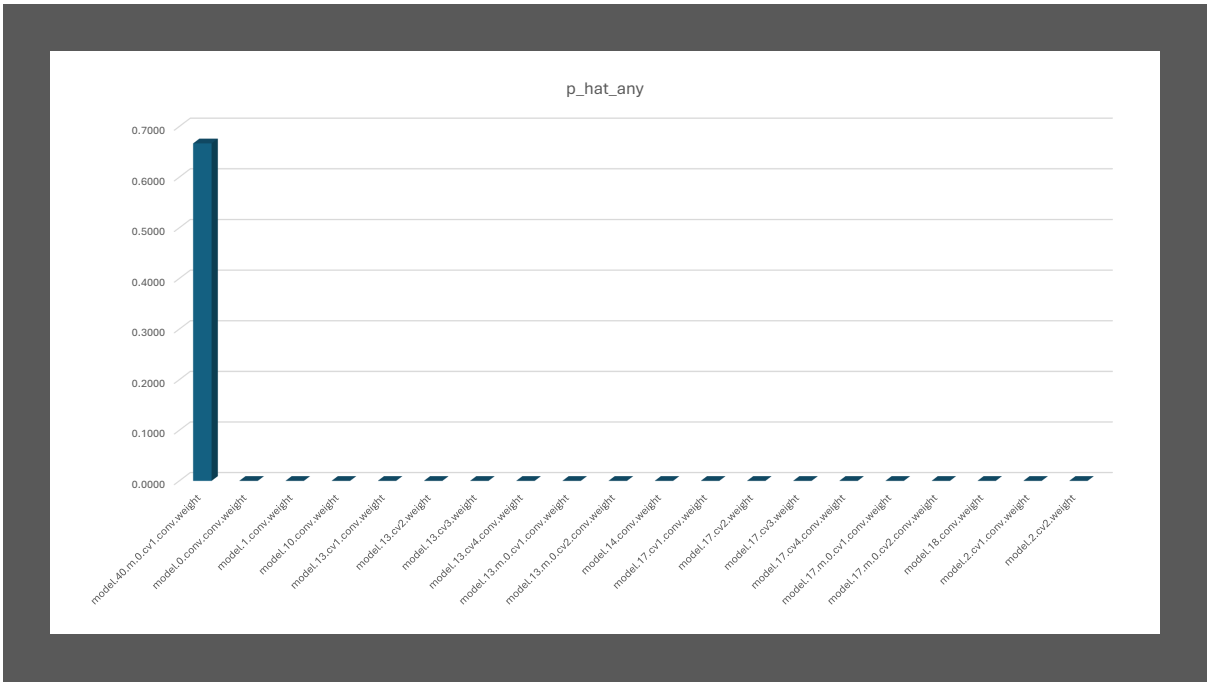
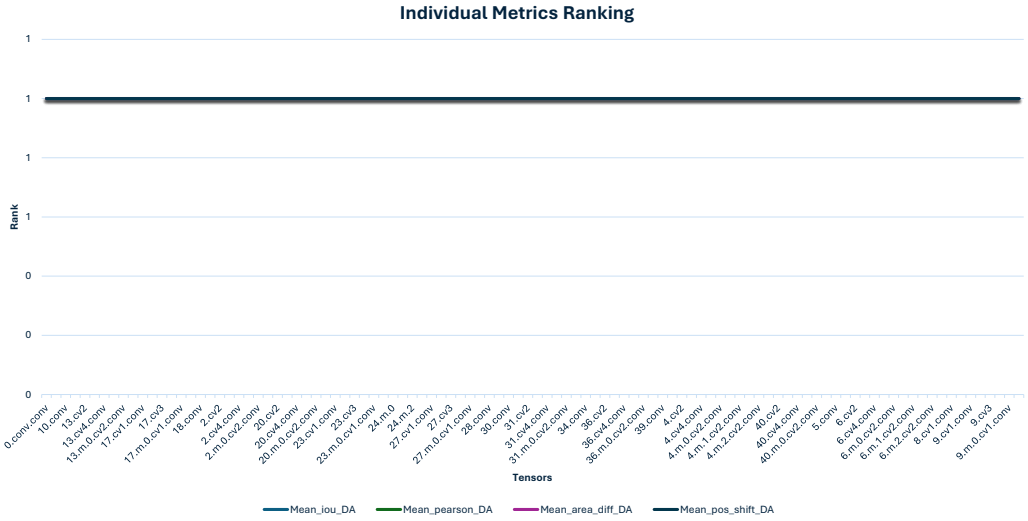


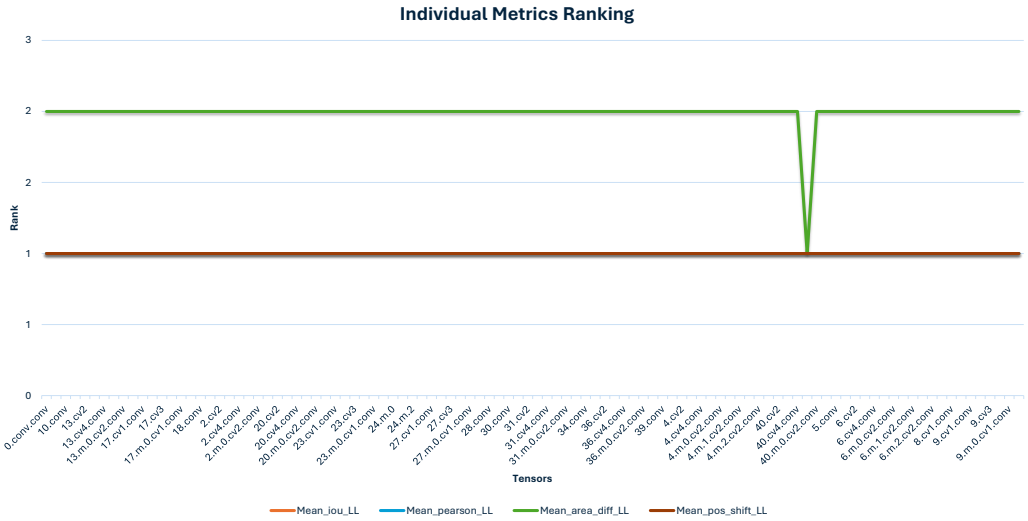
Figure A.71
CUTIN COL Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.m.0.cv1.conv.weight	3	2	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000
model.0.conv.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.2.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.2.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

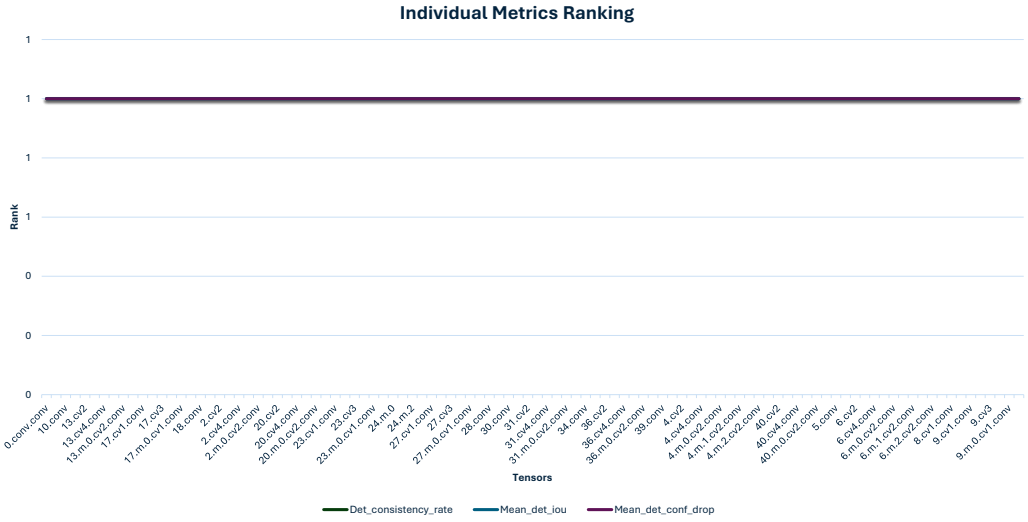
Figure A.72
CUTIN COL TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.73
CUTIN COL Task Based Charts

CUTIN Drive Video Comparison COL Fault

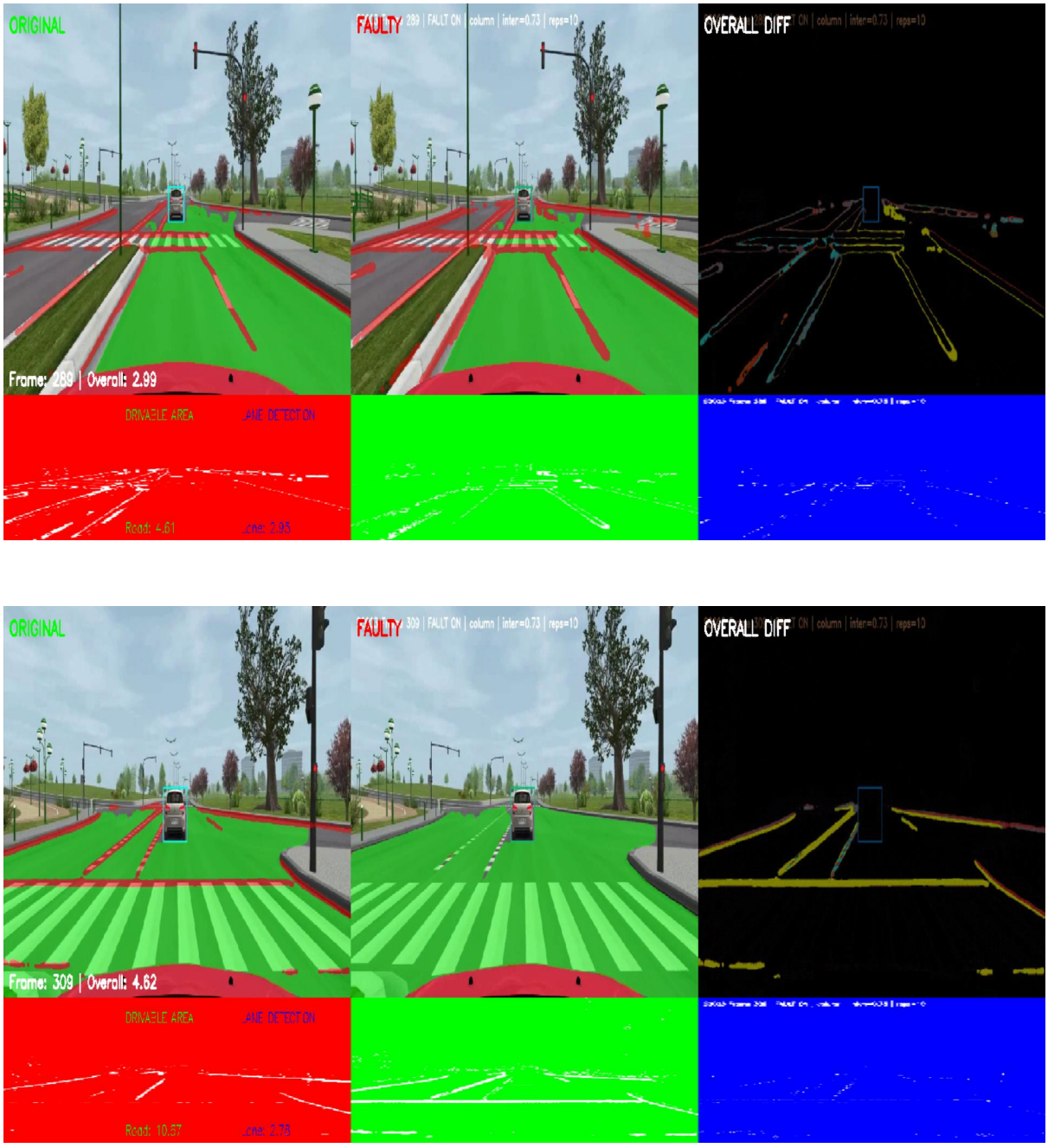


Figure A.74
Visual comparison of the CUT IN driving scenario under COL fault injection.

A.0.3.4 CUT-IN Column Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Seg IoU – Lane Line	37	385	0.0961	0.0294	0.0667	0.1255	STOP	133	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	11	385	0.0286	0.0166	0.0119	0.0452	STOP	43	0
Area RelDiff – Drivable Area	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Area RelDiff – Lane Line	17	385	0.0442	0.0205	0.0237	0.0647	STOP	65	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	20	385	0.0519	0.0221	0.0298	0.0741	STOP	76	0

Figure A.75
CUTIN BW Stop Criterion

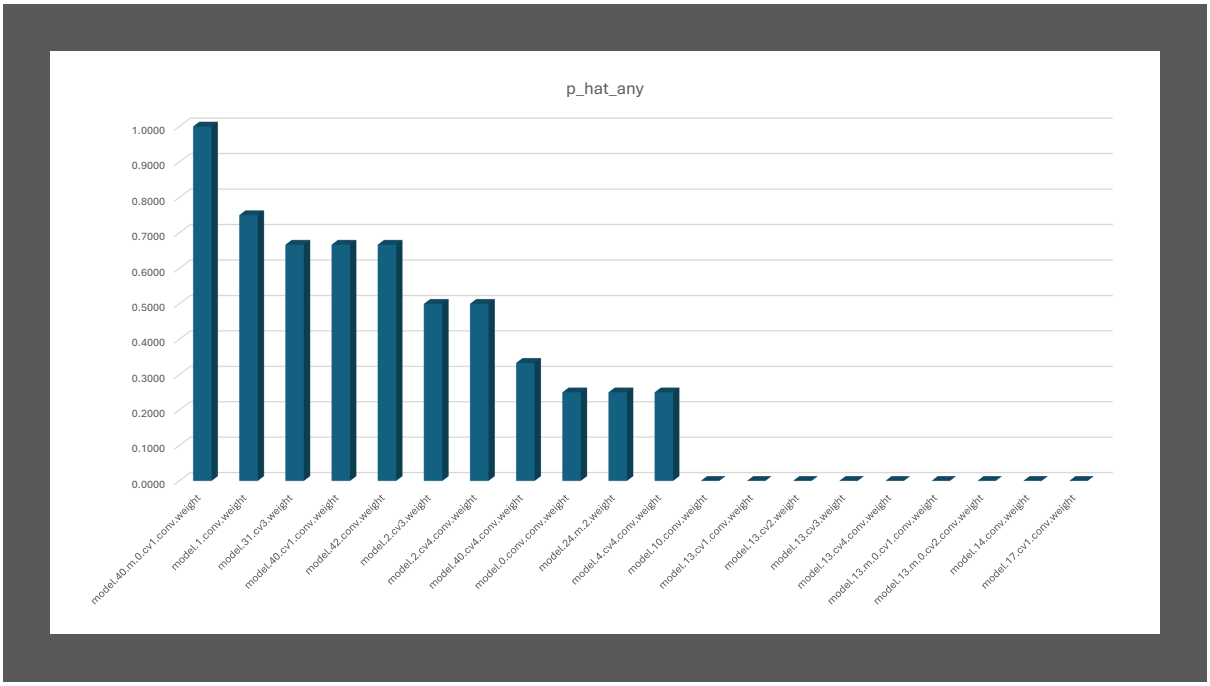
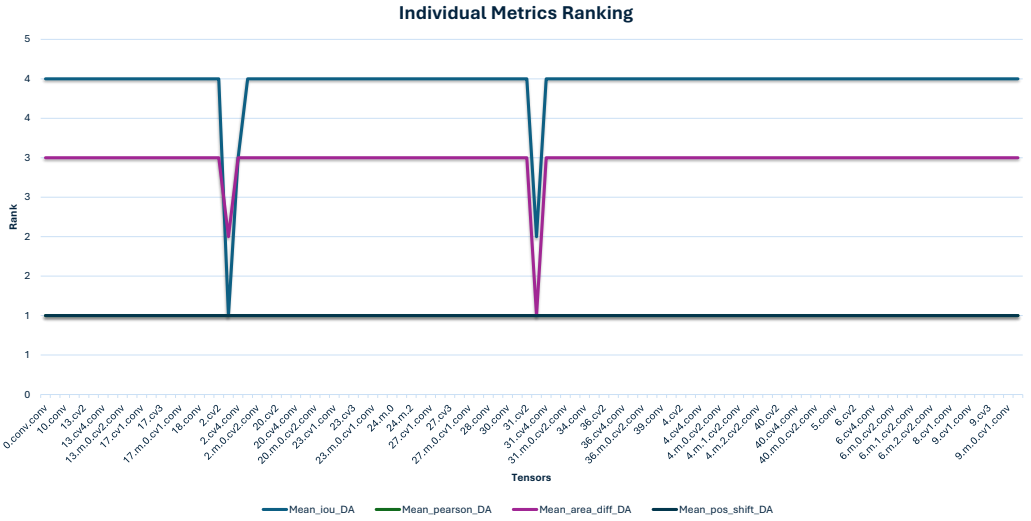


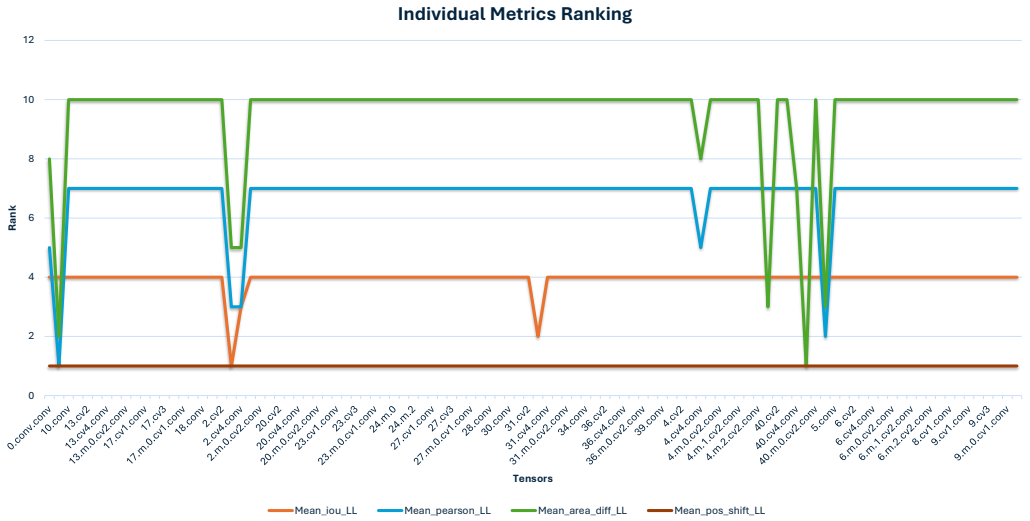
Figure A.76
CUTIN BW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.m.0.cv1.conv.weight	3	3	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.1.conv.weight	4	3	0.75	0	0	0	0.75	0	0.75	0	0	0	0.25	0
model.31.cv3.weight	3	2	0.666666667	0.333333333	0.333333	0	0	0.666667	0	0	0	0	0	0
model.40.cv1.conv.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.666666667	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.2.cv3.weight	4	2	0.5	0.5	0.5	0	0.5	0.5	0.5	0	0	0	0	0
model.2.cv4.conv.weight	4	2	0.5	0.25	0.25	0	0.5	0	0.5	0	0	0	0	0
model.40.cv4.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	1	0.25	0	0	0	0.25	0	0.25	0	0	0	0	0
model.24.m.2.weight	4	1	0.25	0	0	0	0	0	0	0	0	0	0.25	0
model.4.cv4.conv.weight	4	1	0.25	0	0	0	0.25	0	0.25	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

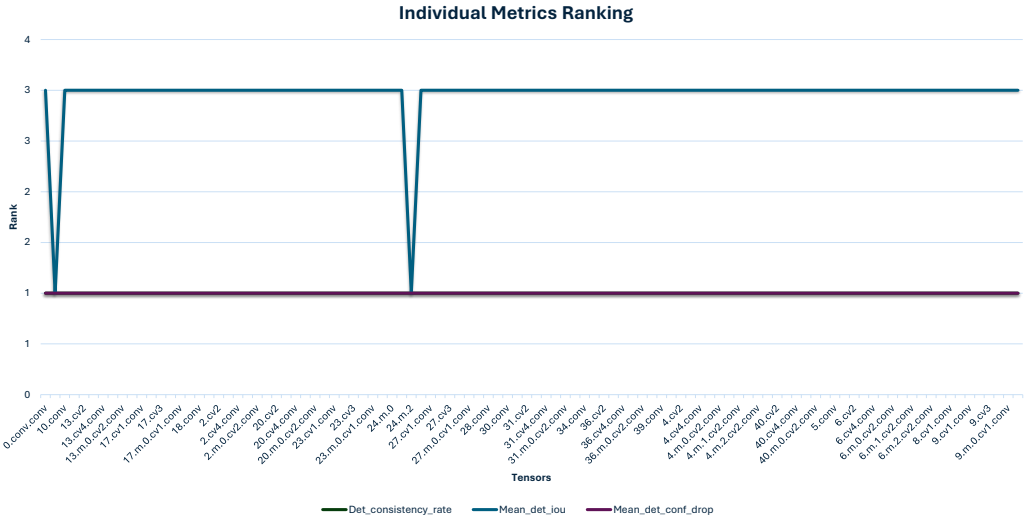
Figure A.77
CUTIN BW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.78
CUTIN BW Task Based Charts

CUTIN Drive Video Comparison BW Fault

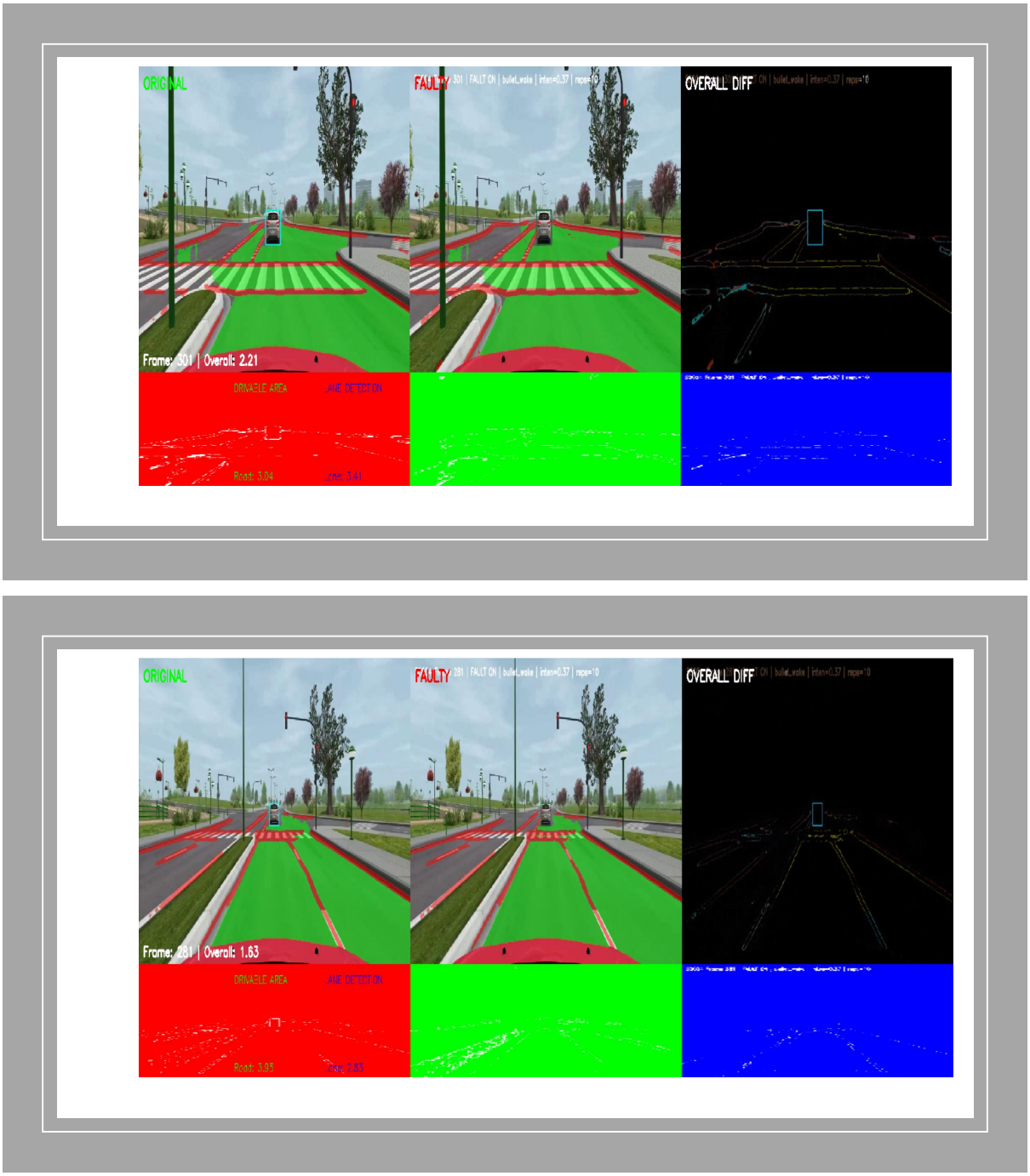


Figure A.79
Visual comparison of the CUT IN driving scenario under BW fault injection.

A.0.3.5 CUT-IN SHattered Glass Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Seg IoU – Lane Line	40	385	0.1039	0.0305	0.0734	0.1343	STOP	143	0
Pearson – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Pearson – Lane Line	8	385	0.0208	0.0142	0.0065	0.0350	STOP	31	0
Area RelDiff – Drivable Area	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Area RelDiff – Lane Line	16	385	0.0416	0.0199	0.0216	0.0615	STOP	61	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	21	385	0.0545	0.0227	0.0319	0.0772	STOP	79	0

Figure A.80
CUTIN SG Stop Criterion

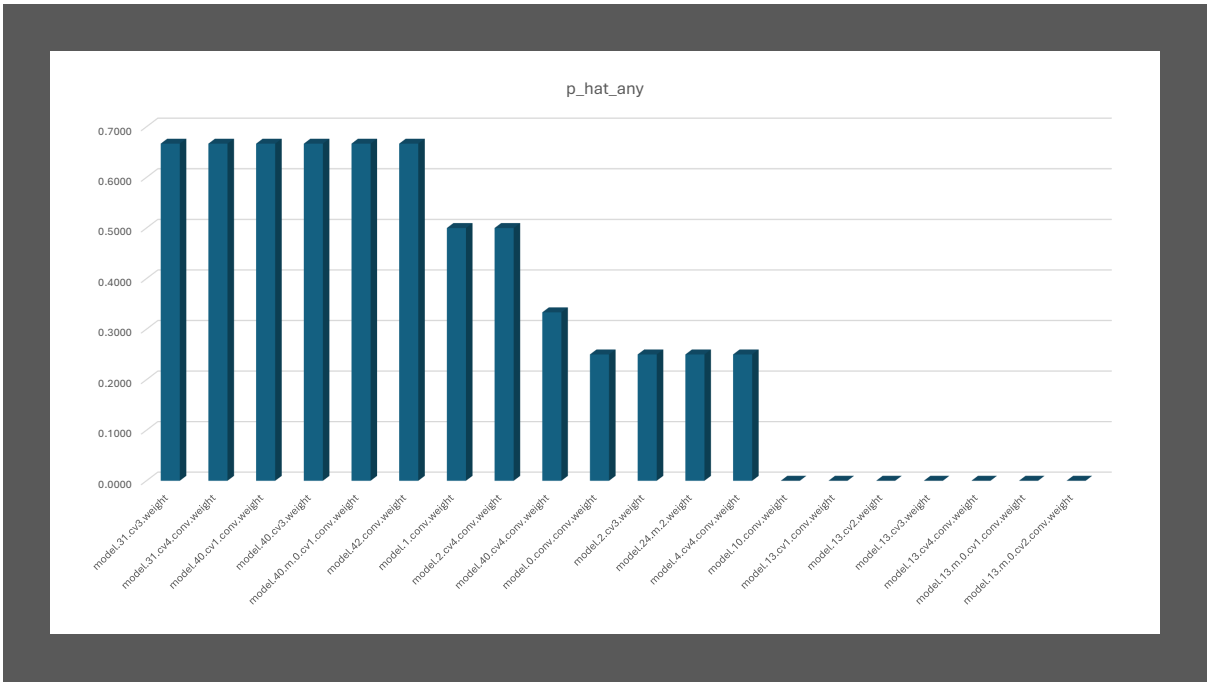
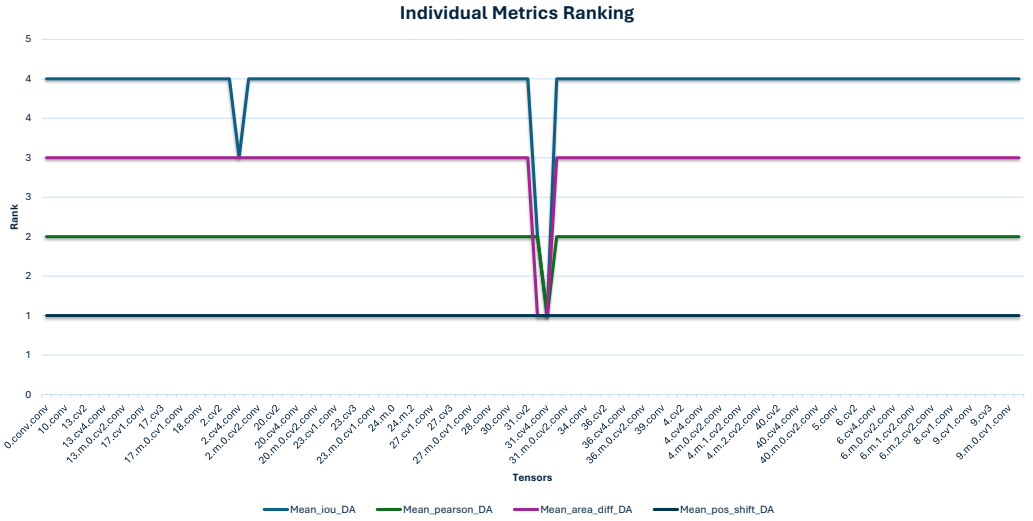


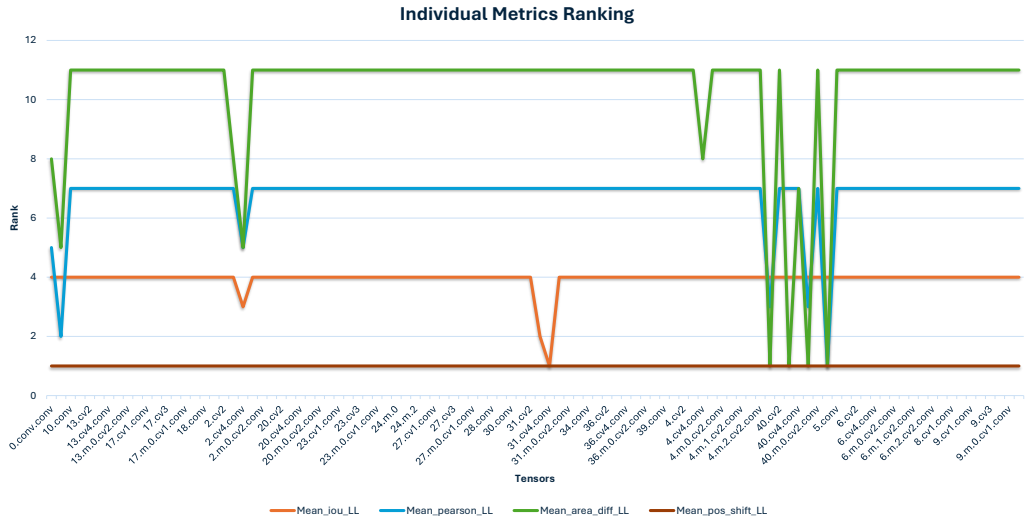
Figure A.81
CUTIN SG Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.31.cv3.weight	3	2	0.6667	0.3333	0.3333	0.0000	0.0000	0.6667	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.31.cv4.conv.weight	3	2	0.666666667	0.666666667	0.666667	0.333333	0	0.666667	0	0	0	0	0	0
model.40.cv1.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.40.cv3.weight	3	2	0.666666667	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.666666667	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.666666667	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.1.conv.weight	4	2	0.5	0	0	0	0.5	0	0.5	0	0	0	0.25	0
model.2.cv4.conv.weight	4	2	0.5	0.25	0.25	0	0.25	0	0.5	0	0	0	0	0
model.40.cv4.conv.weight	3	1	0.333333333	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	1	0.25	0	0	0	0.25	0	0.25	0	0	0	0	0
model.2.cv3.weight	4	1	0.25	0	0	0	0	0	0.25	0	0	0	0	0
model.24.m.2.weight	4	1	0.25	0	0	0	0	0	0	0	0	0	0.25	0
model.4.cv4.conv.weight	4	1	0.25	0	0	0	0	0	0.25	0	0	0	0	0
model.10.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0	0	0	0	0	0	0	0	0	0	0	0

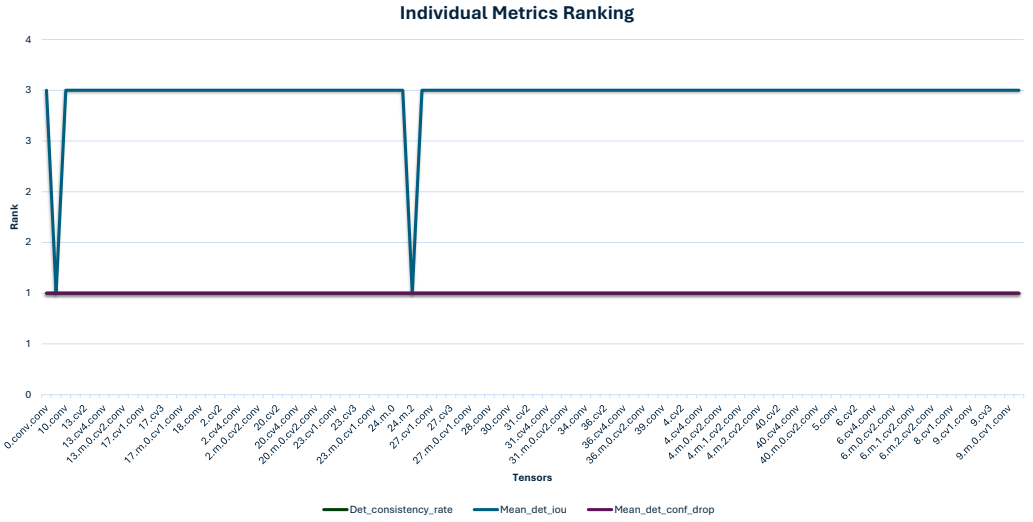
Figure A.82
CUTIN SG TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.83
CUTIN SG Task Based Charts

CUTIN Drive Video Comparison SG Fault

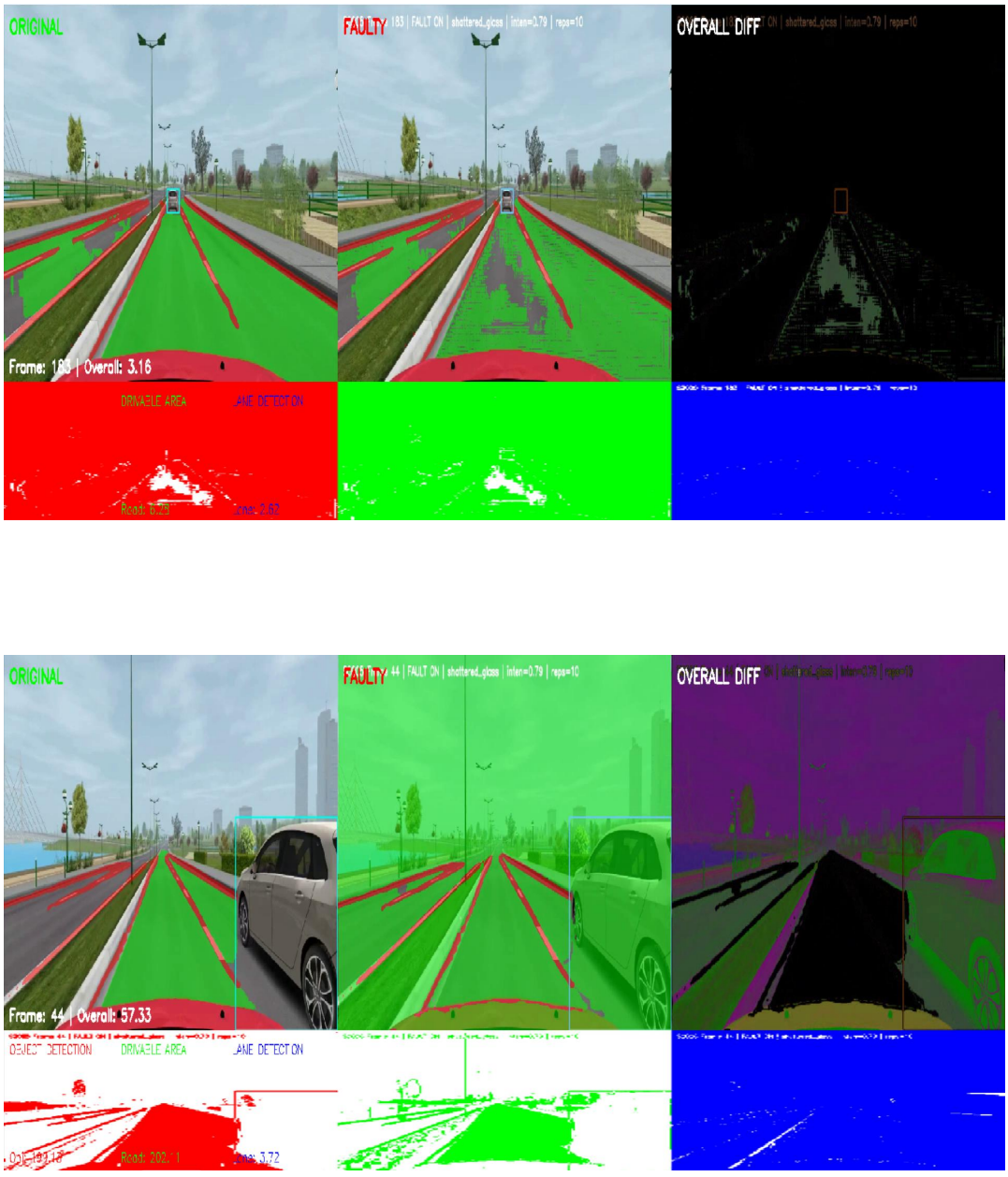


Figure A.84
Visual comparison of the CUT IN driving scenario under SG fault injection.

A.0.4 Accelerate and Lane Change Driving Scenario

Figure A.85
Enter Caption

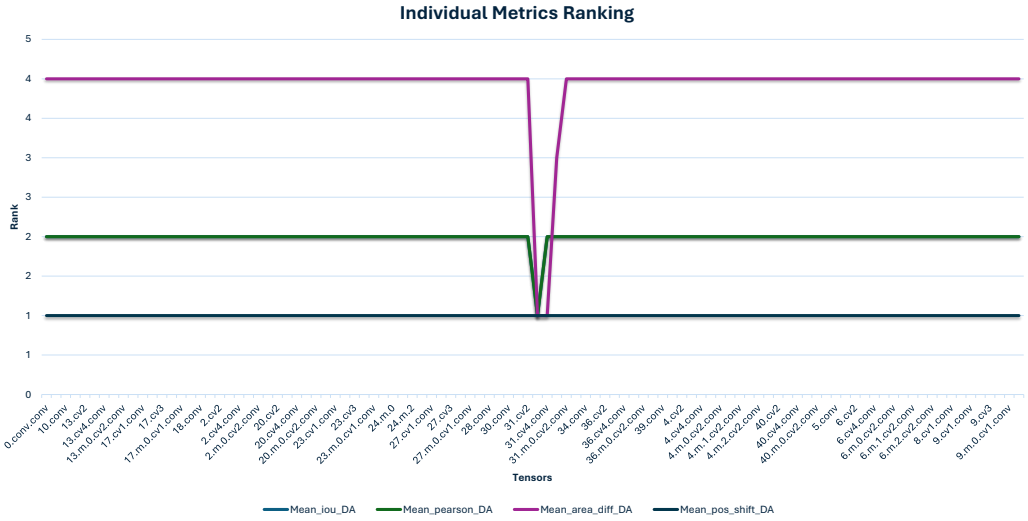
A.0.4.1 ACC Single Point Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Seg IoU – Lane Line	14	385	0.0364	0.0187	0.0177	0.0550	STOP	54	0
Pearson – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Pearson – Lane Line	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Area RelDiff – Drivable Area	5	385	0.0130	0.0113	0.0017	0.0243	STOP	20	0
Area RelDiff – Lane Line	11	385	0.0286	0.0166	0.0119	0.0452	STOP	43	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	16	385	0.0416	0.0199	0.0216	0.0615	STOP	61	0

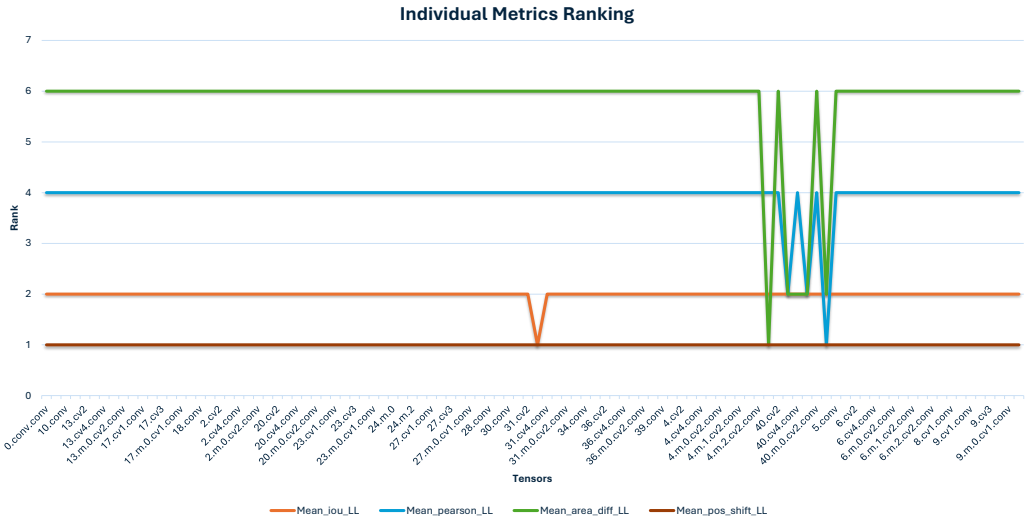
Figure A.86
ACC Single Point Stop Criterion

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.cv1.conv.weight	3	3	1.00	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.31.cv3.weight	3	2	0.67	0.666666667	0.666667	0.333333	0	0.666667	0	0	0	0	0	0
model.31.cv4.conv.weight	3	2	0.67	0	0	0	0	0.666667	0	0	0	0	0	0
model.40.cv3.weight	3	2	0.67	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.40.cv4.conv.weight	3	2	0.67	0	0	0	0	0	0.666667	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.67	0	0	0	0.333333	0	0.666667	0	0	0	0	0
model.42.conv.weight	3	2	0.67	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.31.m.0.cv1.conv.weight	3	1	0.33	0	0	0	0	0.333333	0	0	0	0	0	0
model.0.conv.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0

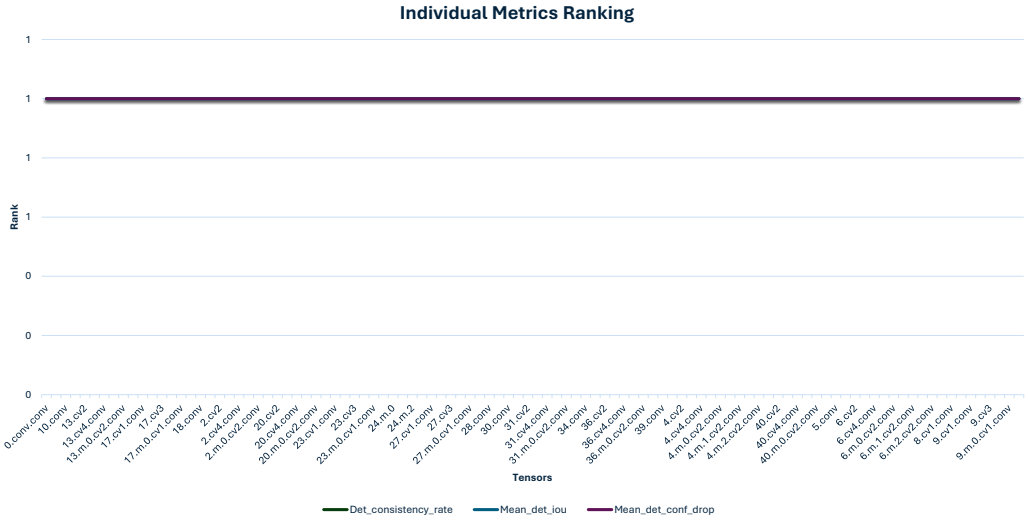
Figure A.87
ACC SP TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.88
ACC SP Task Based Charts

ACC Drive Video Comparison Single Point Fault

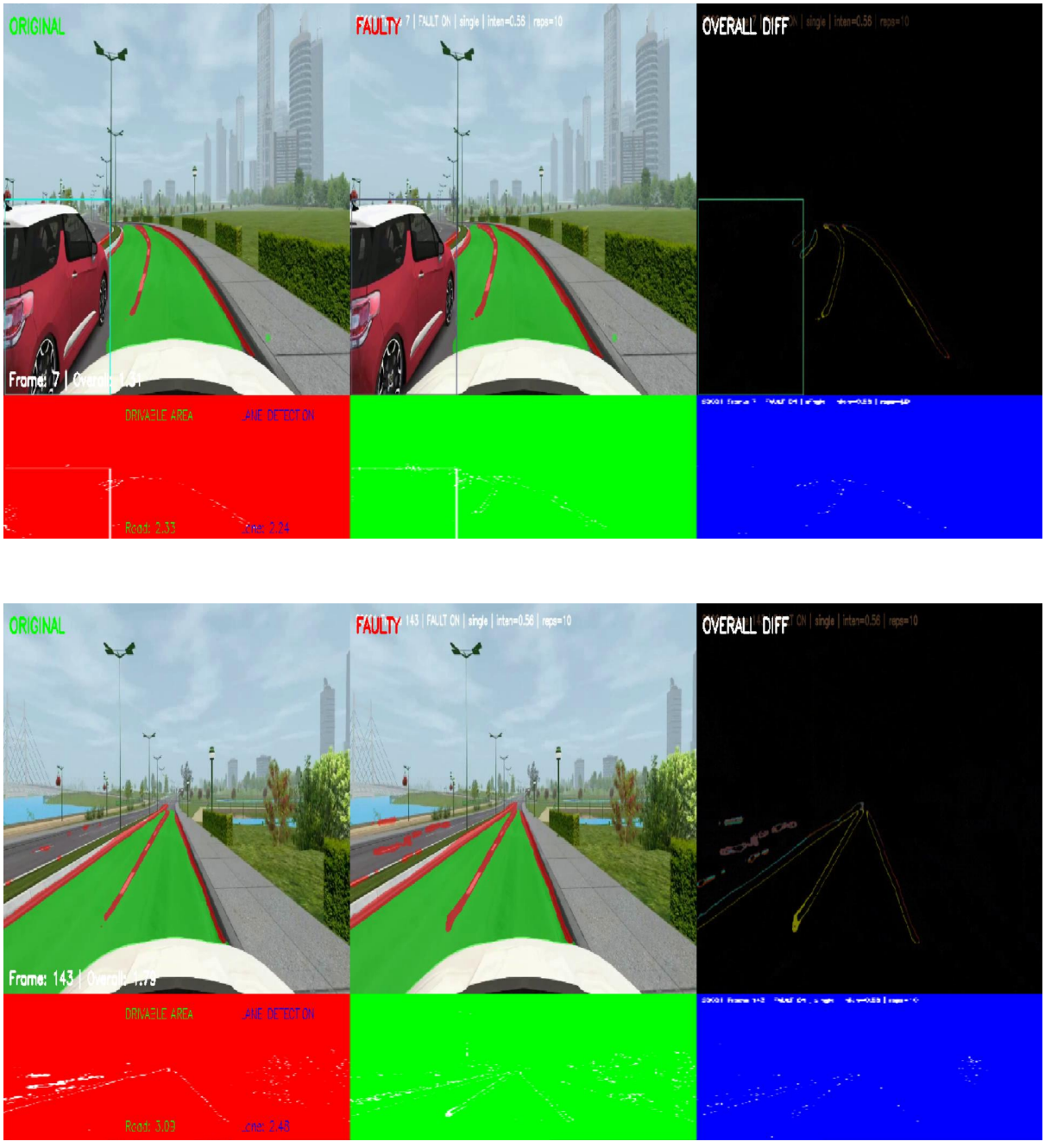


Figure A.89
Visual comparison of the ACC driving scenario under SP fault injection.

A.0.4.2 ACC ROW Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	6	385	0.0156	0.0124	0.0032	0.0279	STOP	24	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Area RelDiff – Drivable Area	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Area RelDiff – Lane Line	6	385	0.0156	0.0124	0.0032	0.0279	STOP	24	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	8	385	0.0208	0.0142	0.0065	0.0350	STOP	31	0

Figure A.90
ACC ROW Stop Criterion

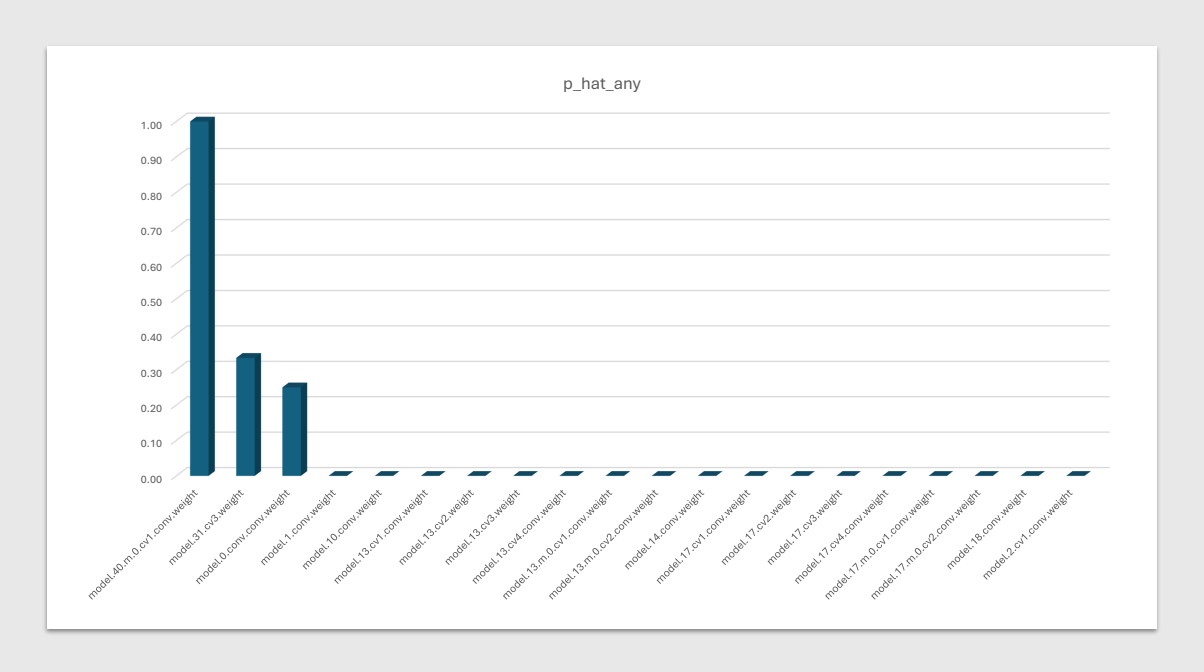
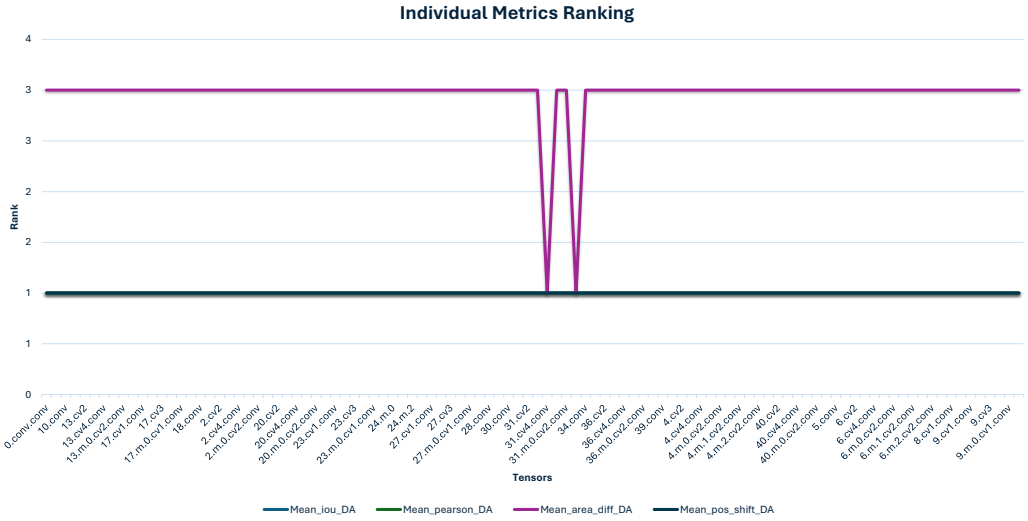


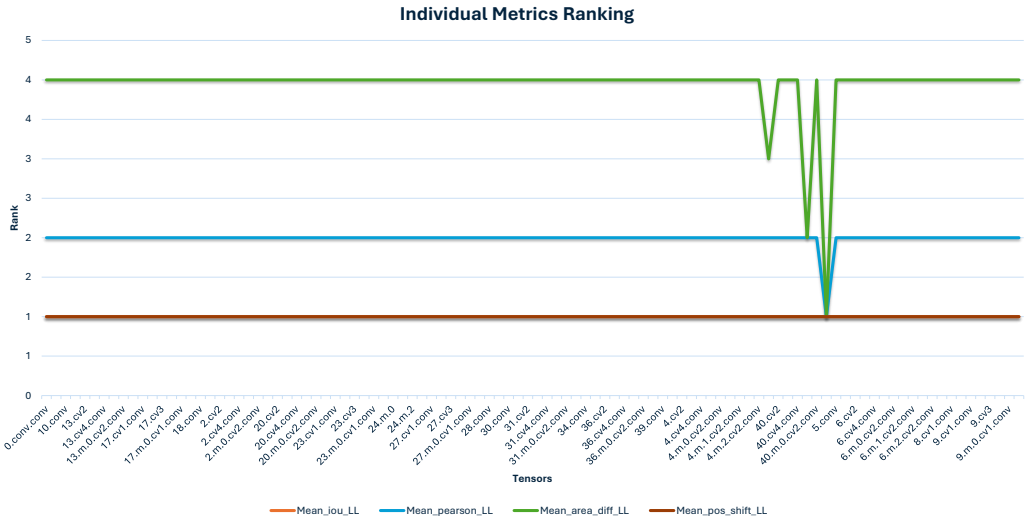
Figure A.91
ACC ROW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.42.conv.weight	3	3	1.00	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.40.m.0.cv1.conv.weight	3	2	0.67	0	0	0	0	0	0.666667	0	0	0	0	0
model.31.cv4.conv.weight	3	1	0.33	0	0	0	0	0.333333	0	0	0	0	0	0
model.33.conv.weight	3	1	0.33	0	0	0	0	0.333333	0	0	0	0	0	0
model.40.cv1.conv.weight	3	1	0.33	0	0	0	0	0	0.333333	0	0	0	0	0
model.0.conv.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0

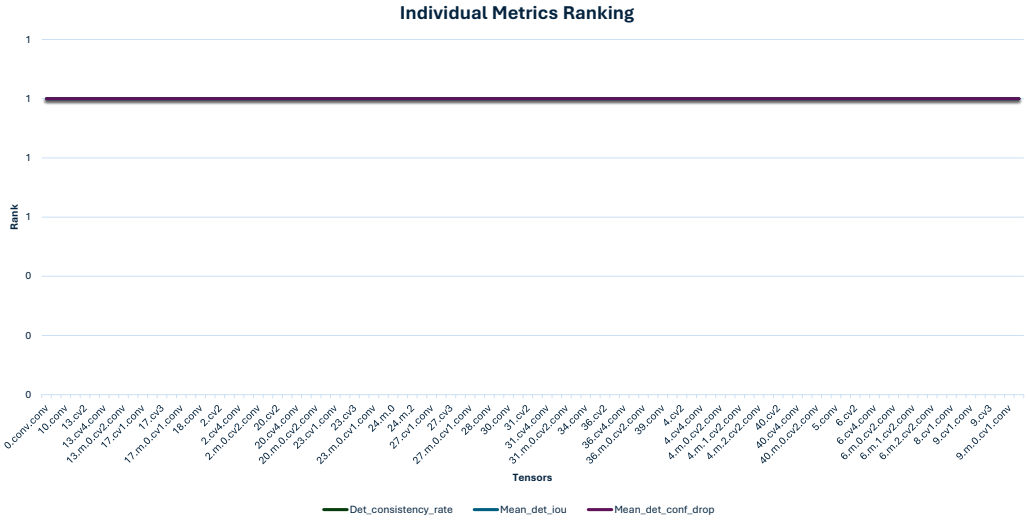
Figure A.92
ACC ROW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.93
ACC ROW Task Based Charts

ACC Drive Video Comparison ROW Fault

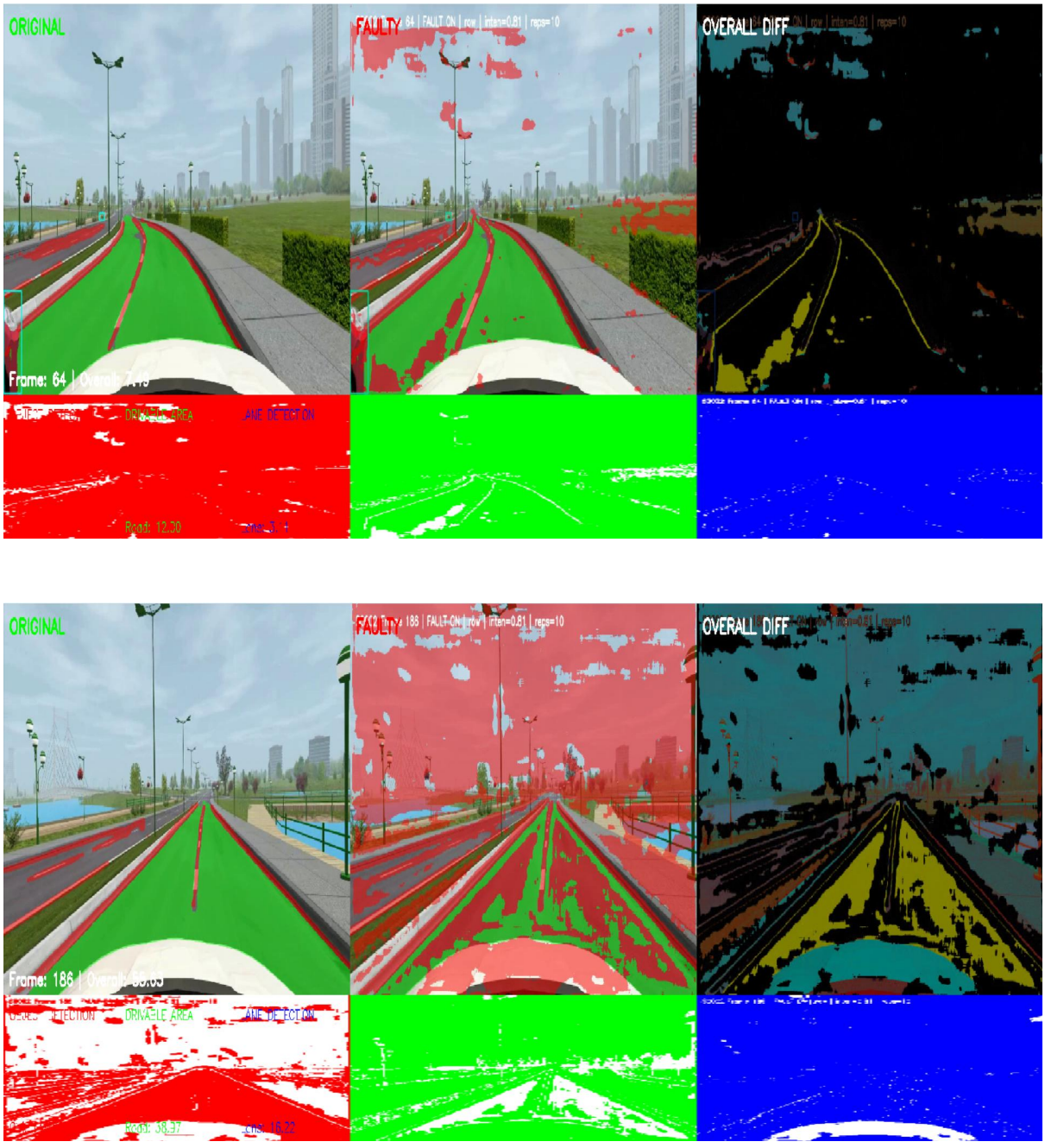


Figure A.94
Visual comparison of the ACC driving scenario under ROW fault injection.

A.0.4.3 ACC Column Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Seg IoU – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pearson – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pearson – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Area RelDiff – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Area RelDiff – Lane Line	3	385	0.0078	0.0088	0.0000	0.0166	STOP	12	0
Pos Shift – Drivable Area	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Pos Shift – Lane Line	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Consistency Rate	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Detection Mean IoU	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	5	385	0.0130	0.0113	0.0017	0.0243	STOP	20	0

Figure A.95
ACC COL Stop Criterion

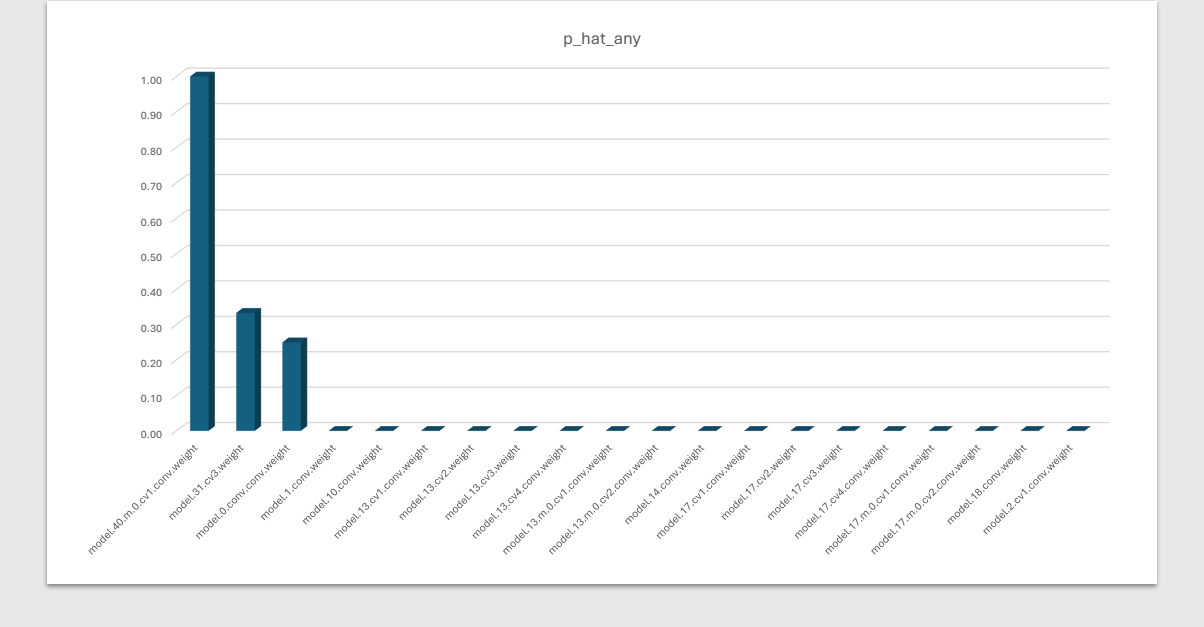
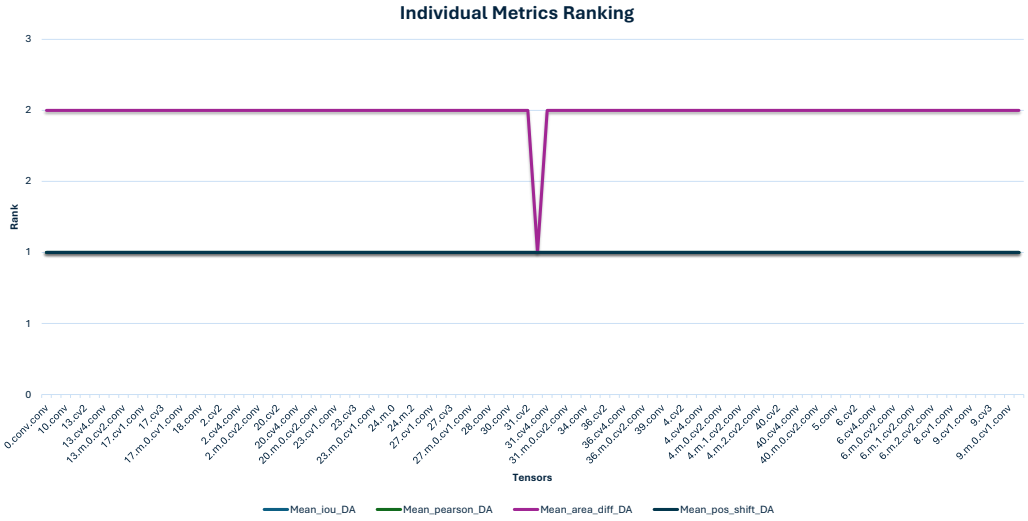


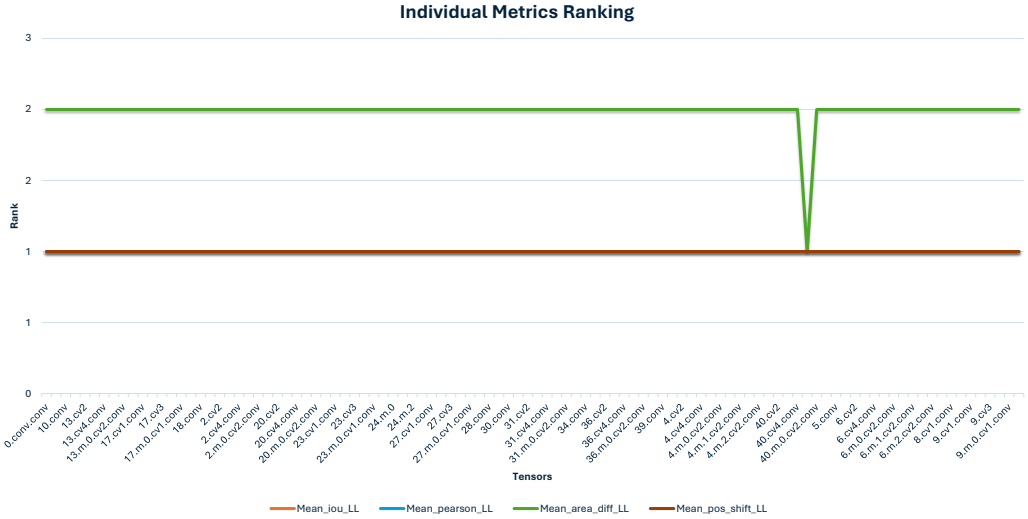
Figure A.96
ACC COL Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.40.m.0.cv1.conv.weight	3	3	1.00	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
model.31.cv3.weight	3	1	0.33	0	0	0	0	0.333333	0	0	0	0	0	0
model.0.conv.conv.weight	4	1	0.25	0	0	0	0	0	0	0	0	0.25	0	0
model.1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.10.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv3.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.cv4.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.13.m.0.cv2.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.14.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv2.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv3.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.cv4.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.17.m.0.cv2.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.18.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0
model.2.cv1.conv.weight	4	0	0.00	0	0	0	0	0	0	0	0	0	0	0

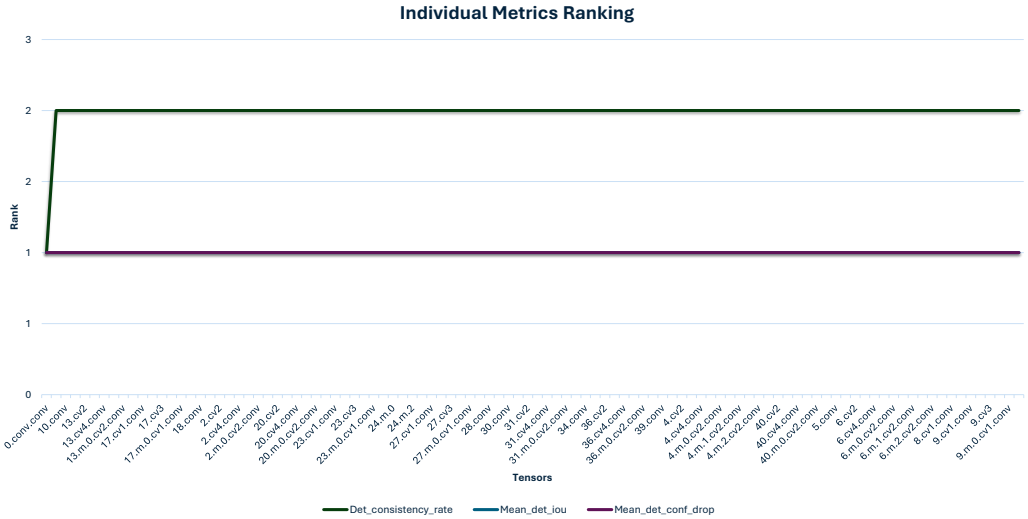
Figure A.97
ACC COL TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.98
ACC COL Task Based Charts

ACC Drive Video Comparison COL Fault

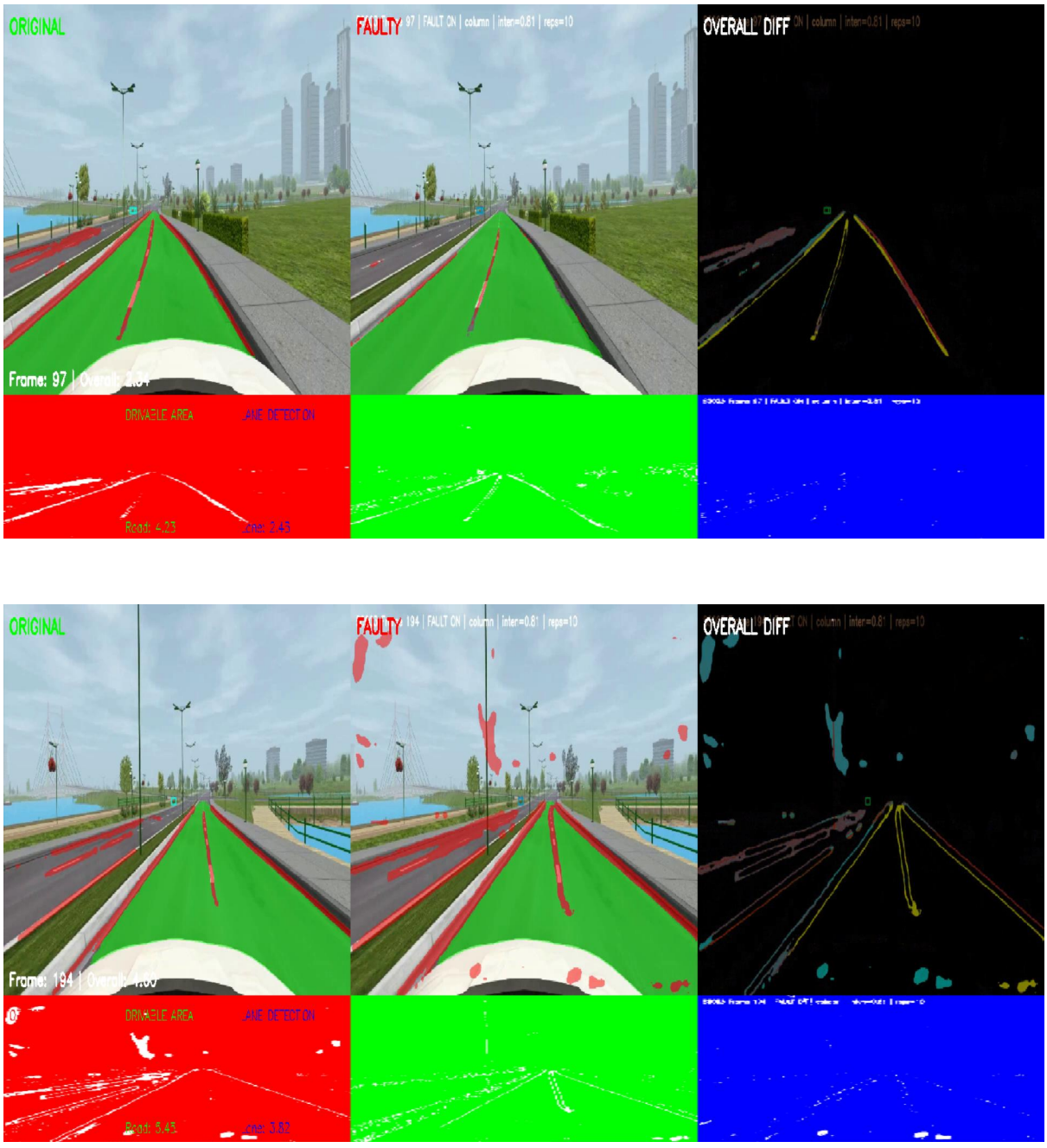


Figure A.99
Visual comparison of the ACC driving scenario under COL fault injection.

A.0.4.4 ACC Bullet Wake Fault Injection

Metric_Label	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	16	385	0.0416	0.0199	0.0216	0.0615	STOP	61	0
Seg IoU – Lane Line	82	385	0.2130	0.0409	0.1721	0.2538	STOP	257	0
Pearson – Drivable Area	4	385	0.0104	0.0101	0.0003	0.0205	STOP	16	0
Pearson – Lane Line	18	385	0.0468	0.0211	0.0257	0.0678	STOP	68	0
Area RelDiff – Drivable Area	25	385	0.0649	0.0246	0.0403	0.0895	STOP	93	0
Area RelDiff – Lane Line	27	385	0.0701	0.0255	0.0446	0.0956	STOP	100	0
Pos Shift – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Pos Shift – Lane Line	2	385	0.0052	0.0072	0.0000	0.0124	STOP	8	0
Detection Consistency Rate	49	385	0.1273	0.0333	0.0940	0.1605	STOP	171	0
Detection Mean IoU	8	385	0.0208	0.0142	0.0065	0.0350	STOP	31	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	70	385	0.1818	0.0385	0.1433	0.2203	STOP	228	0

Figure A.100
ACC BW Stop Criterion

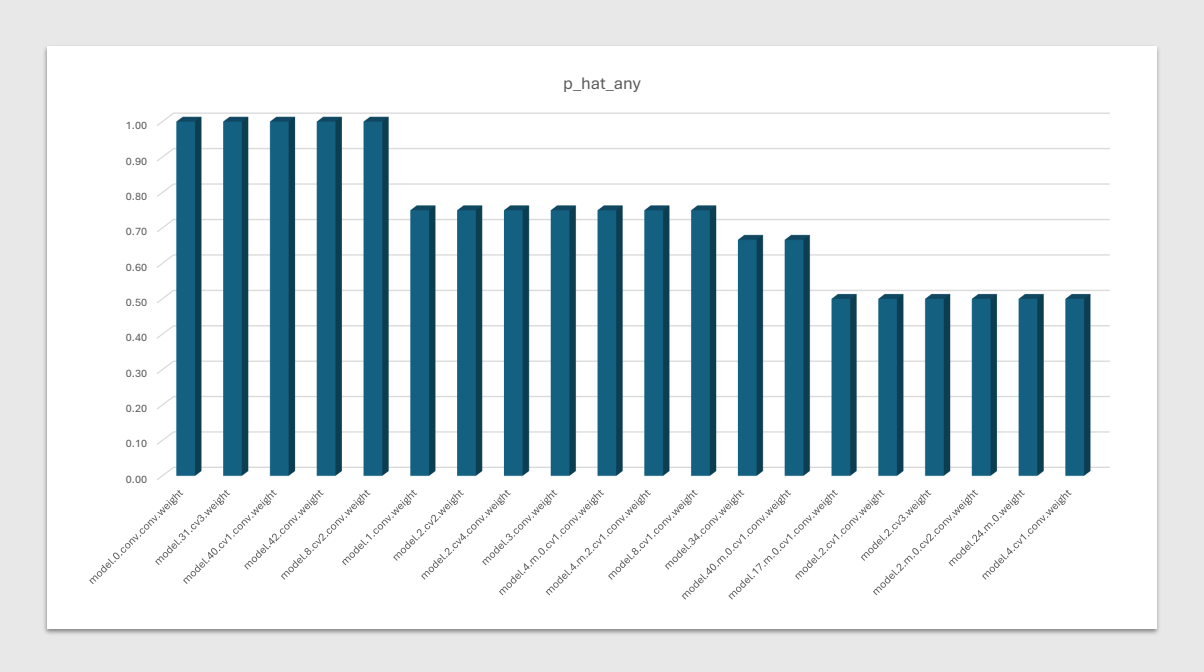
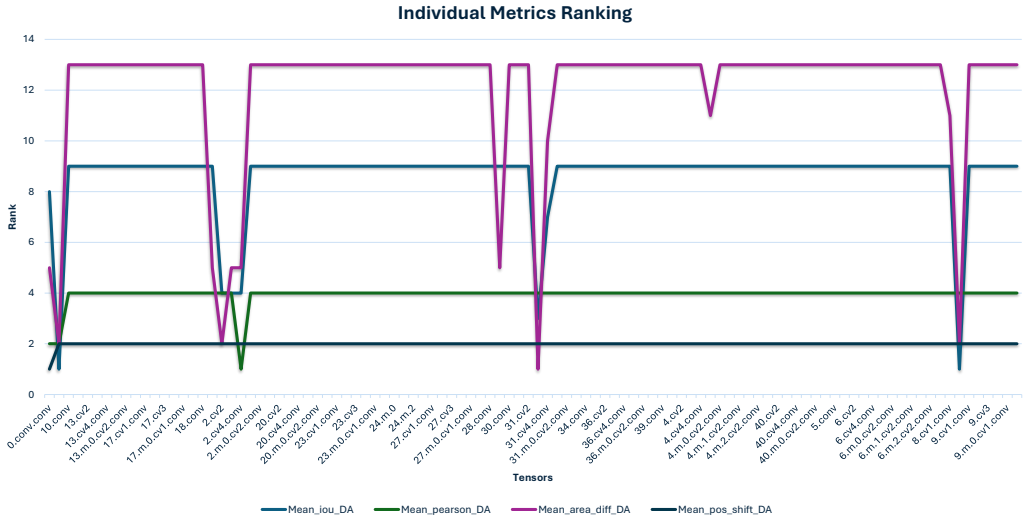


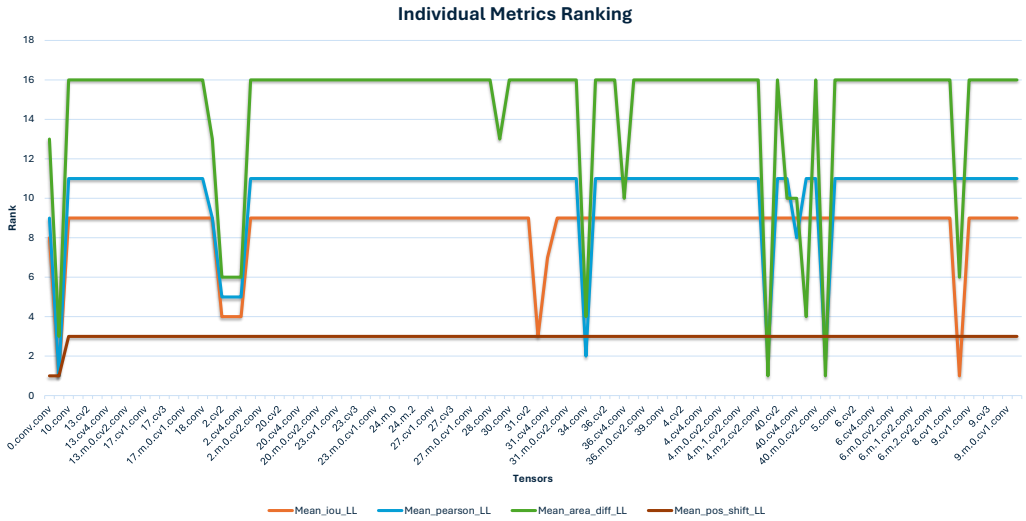
Figure A.101
ACC BW Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.0.conv.conv.weight	4	4	1.00	0.2500	0.2500	0.2500	0.2500	0.5000	0.2500	0.2500	0.2500	1.0000	0.2500	0.0000
model.31.cv3.weight	3	3	1.00	0.666666667	0.666667	0	0	1	0	0	0	0	0	0
model.40.cv1.conv.weight	3	3	1.00	0	0	0	0.666667	0	1	0	0	0	0	0
model.42.conv.weight	3	3	1.00	0	0	0	0.666667	0	1	0	0	0	0	0
model.8.cv2.conv.weight	4	4	1.00	0.75	0.75	0	0	0.75	0.5	0	0	1	0	0
model.1.conv.weight	4	3	0.75	0.75	0.75	0.25	0.75	0.75	0.75	0	0.25	0.75	0.5	0
model.2.cv2.weight	4	3	0.75	0.5	0.5	0	0.5	0.75	0.5	0	0	0.75	0	0
model.2.cv4.conv.weight	4	3	0.75	0.5	0.5	0.5	0.5	0.5	0.5	0	0	0.75	0.25	0
model.3.conv.weight	4	3	0.75	0	0	0	0	0.5	0.25	0	0	0.75	0	0
model.4.m.0.cv1.conv.weight	4	3	0.75	0	0	0	0	0.25	0	0	0	0.75	0	0
model.4.m.2.cv1.conv.weight	4	3	0.75	0	0	0	0	0	0	0	0	0.75	0	0
model.8.cv1.conv.weight	4	3	0.75	0	0	0	0	0.25	0	0	0	0.75	0	0
model.34.conv.weight	3	2	0.67	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	2	0.67	0	0	0	0	0	0.666667	0	0	0	0	0
model.17.m.0.cv1.conv.weight	4	2	0.50	0	0	0	0	0	0	0	0	0.5	0	0
model.2.cv1.conv.weight	4	2	0.50	0	0	0	0.25	0.5	0.25	0	0	0.5	0	0
model.2.cv3.weight	4	2	0.50	0.5	0.5	0	0.5	0.5	0.5	0	0	0.5	0	0
model.2.m.0.cv2.conv.weight	4	2	0.50	0	0	0	0	0	0	0	0	0.5	0	0
model.24.m.0.weight	4	2	0.50	0	0	0	0	0	0	0	0	0	0.5	0
model.4.cv1.conv.weight	4	2	0.50	0	0	0	0	0	0	0	0	0.5	0	0

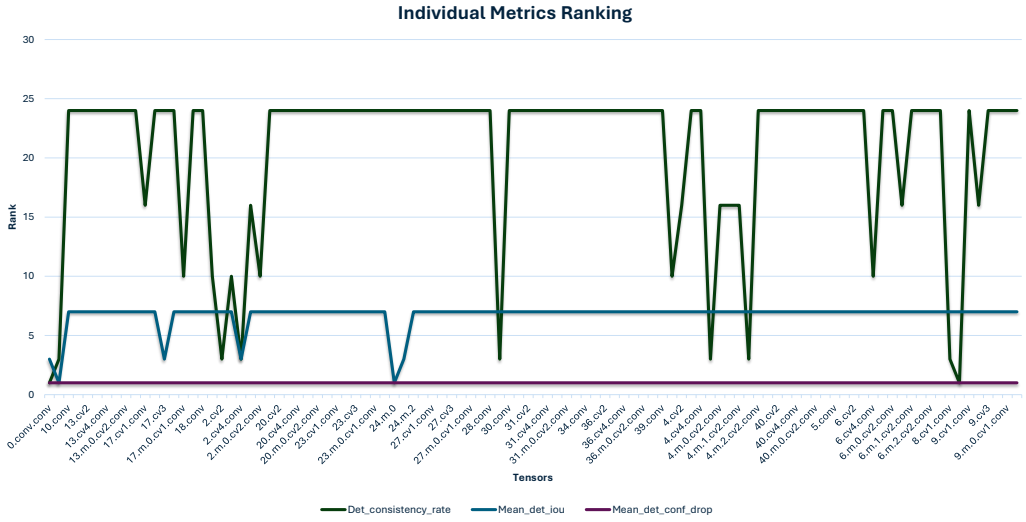
Figure A.102
ACC BW TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



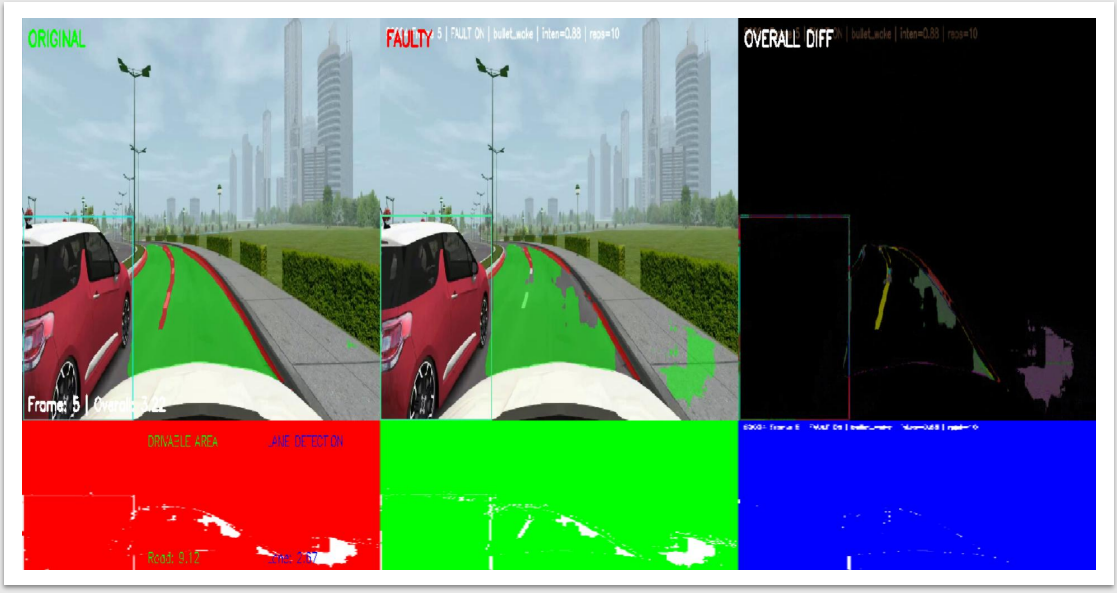
(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.103
ACC BW Task Based Charts

ACC Drive Video Comparison BW Fault



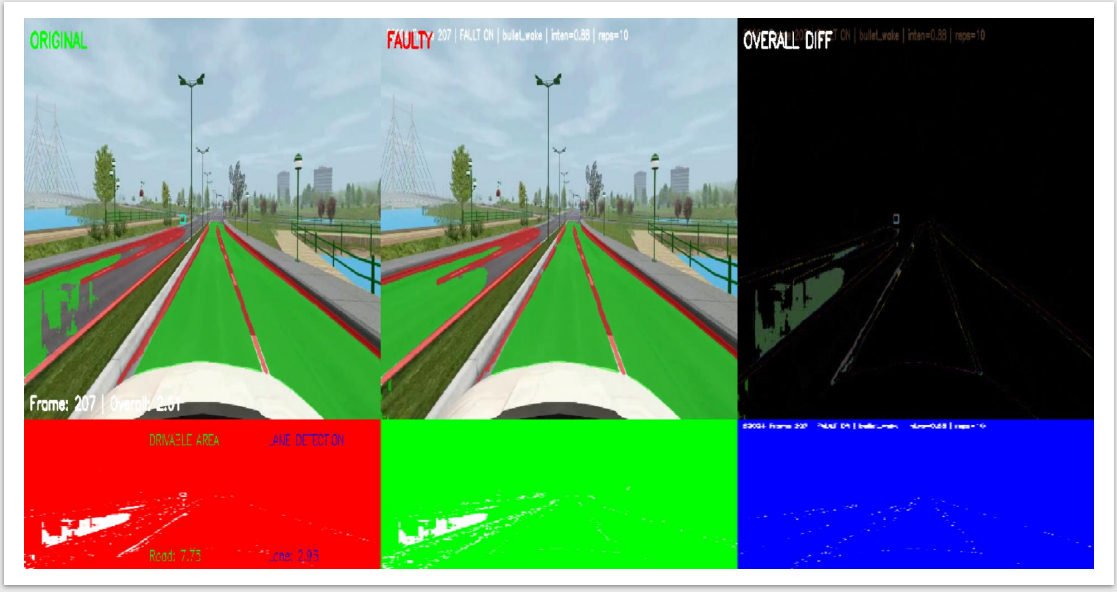


Figure A.104
Visual comparison of the ACC driving scenario under BW fault injection.

A.0.4.5 ACC Shattered Glass Fault Injection

c	Failures_x	n	p_hat	ê (MOE)	CI_low	CI_high	STOP	n_required_for_Egoal	Additional_needed
Seg IoU – Drivable Area	16	385	0.0416	0.0199	0.0216	0.0615	STOP	61	0
Seg IoU – Lane Line	79	385	0.2052	0.0403	0.1649	0.2455	STOP	250	0
Pearson – Drivable Area	8	385	0.0208	0.0142	0.0065	0.0350	STOP	31	0
Pearson – Lane Line	25	385	0.0649	0.0246	0.0403	0.0895	STOP	93	0
Area RelDiff – Drivable Area	22	385	0.0571	0.0232	0.0340	0.0803	STOP	83	0
Area RelDiff – Lane Line	31	385	0.0805	0.0272	0.0534	0.1077	STOP	114	0
Pos Shift – Drivable Area	1	385	0.0026	0.0051	0.0000	0.0077	STOP	4	0
Pos Shift – Lane Line	9	385	0.0234	0.0151	0.0083	0.0385	STOP	35	0
Detection Consistency Rate	41	385	0.1065	0.0308	0.0757	0.1373	STOP	146	0
Detection Mean IoU	9	385	0.0234	0.0151	0.0083	0.0385	STOP	35	0
Detection Mean Conf Drop	0	385	0.0000	0.0000	0.0000	0.0000	STOP	385	0
Overall (ANY metric violates its threshold)	61	385	0.1584	0.0364	0.1220	0.1949	STOP	205	0

Figure A.105
ACC SG Stop Criterion

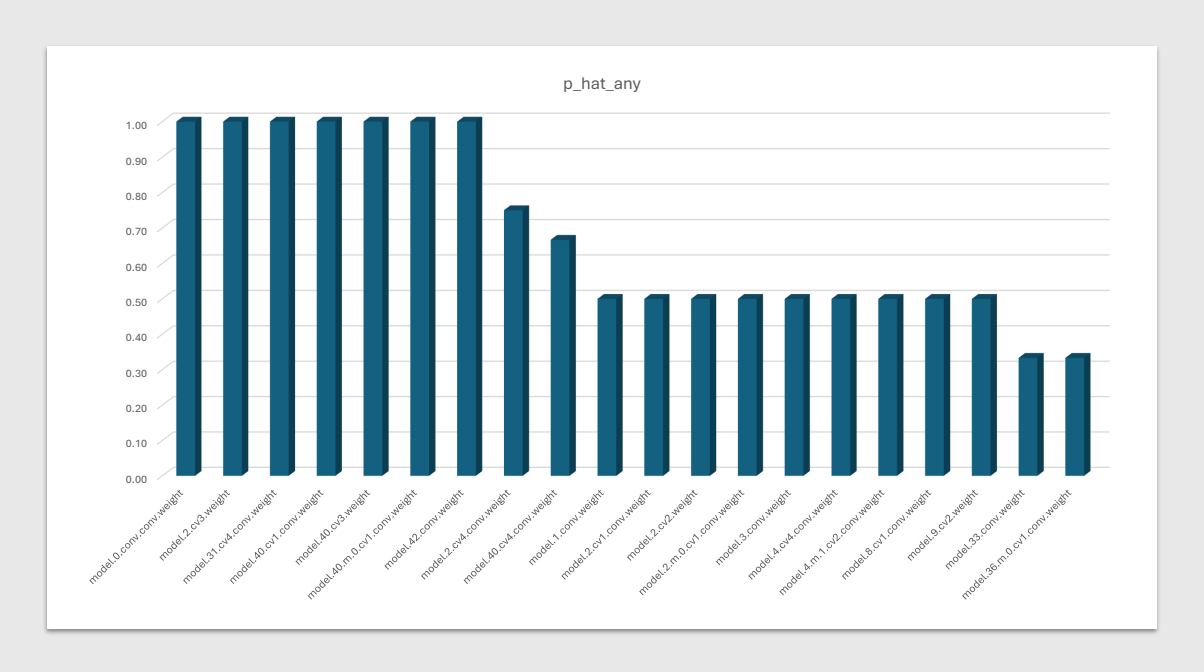
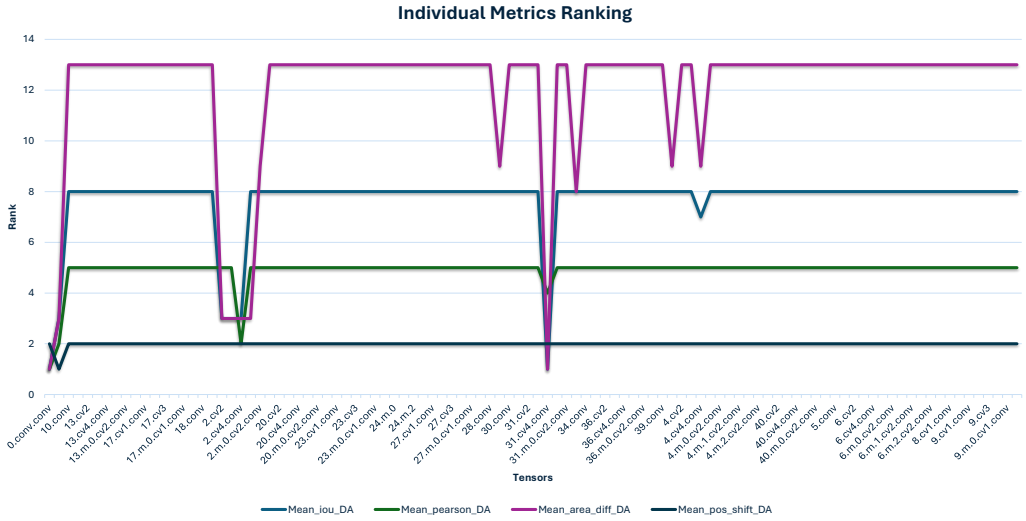


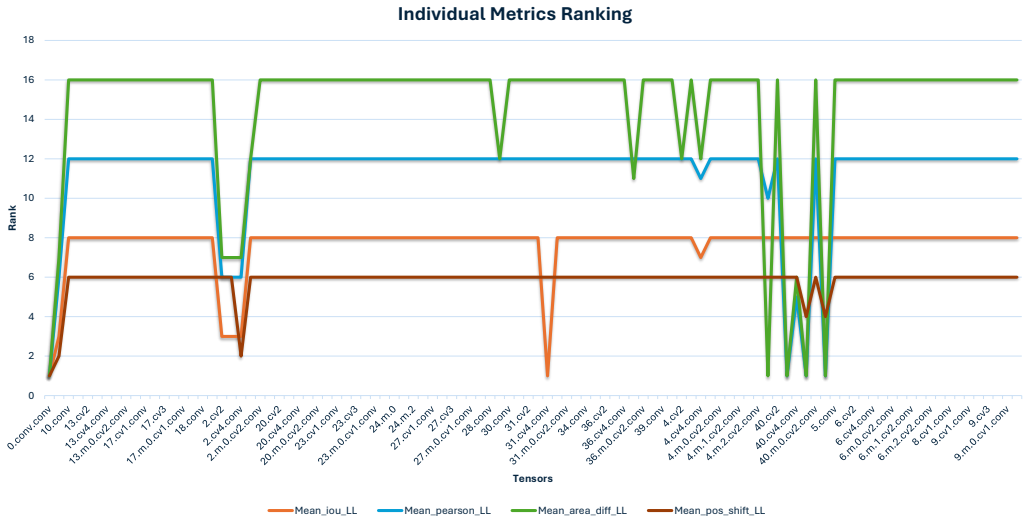
Figure A.106
ACC SG Bar Graph

target_tensor	n_t	x_any	p_hat_any	p_mean_iou_da	p_mean_iou_ll	p_mean_pearson_da	p_mean_pearson_ll	p_mean_area_diff_da	p_mean_area_diff_ll	p_mean_pos_shift_da	p_mean_pos_shift_ll	p_det_consistency_rate	p_mean_det_iou	p_mean_det_conf_drop
model.0.conv.conv.weight	4	4	1.00	1.0000	1.0000	0.7500	1.0000	1.0000	1.0000	0.0000	0.7500	1.0000	0.7500	0.0000
model.2.cv3.weight	4	4	1.00	0.5	0.5	0	0.5	0.5	0.5	0	0	1	0.25	0
model.31.cv4.conv.weight	3	3	1.00	1	1	0.333333	0	1	0	0	0	0	0	0
model.40.cv1.conv.weight	3	3	1.00	0	0	0	0.333333	0	1	0	0	0	0	0
model.40.cv3.weight	3	3	1.00	0	0	0	1	0	1	0	0	0	0	0
model.40.m.0.cv1.conv.weight	3	3	1.00	0	0	0	1	0	1	0	0.333333	0	0	0
model.42.conv.weight	3	3	1.00	0	0	0	1	0	1	0	0.333333	0	0	0
model.2.cv4.conv.weight	4	3	0.75	0.5	0.5	0.5	0.5	0.5	0.5	0	0.5	0.75	0.5	0
model.40.cv4.conv.weight	3	2	0.67	0	0	0	0.666667	0	0.666667	0	0	0	0	0
model.1.conv.weight	4	2	0.50	0.5	0.5	0.5	0.5	0.5	0.5	0.25	0.5	0.5	0.5	0
model.2.cv1.conv.weight	4	2	0.50	0	0	0	0	0	0	0	0	0.5	0	0
model.2.cv2.weight	4	2	0.50	0.5	0.5	0	0.5	0.5	0.5	0	0	0.5	0	0
model.2.m.0.cv1.conv.weight	4	2	0.50	0	0	0	0	0.5	0.25	0	0	0.5	0	0
model.3.conv.weight	4	2	0.50	0	0	0	0	0.25	0.25	0	0	0.5	0	0
model.4.cv4.conv.weight	4	2	0.50	0.25	0.25	0	0.25	0.25	0.25	0	0	0.5	0	0
model.4.m.1.cv2.conv.weight	4	2	0.50	0	0	0	0	0	0	0	0	0.5	0	0
model.8.cv1.conv.weight	4	2	0.50	0	0	0	0	0	0	0	0	0.5	0	0
model.9.cv2.weight	4	2	0.50	0	0	0	0	0	0	0	0	0.5	0	0
model.33.conv.weight	3	1	0.33	0	0	0	0	0.333333	0	0	0	0	0	0
model.36.m.0.cv1.conv.weight	3	1	0.33	0	0	0	0	0	0.333333	0	0	0	0	0

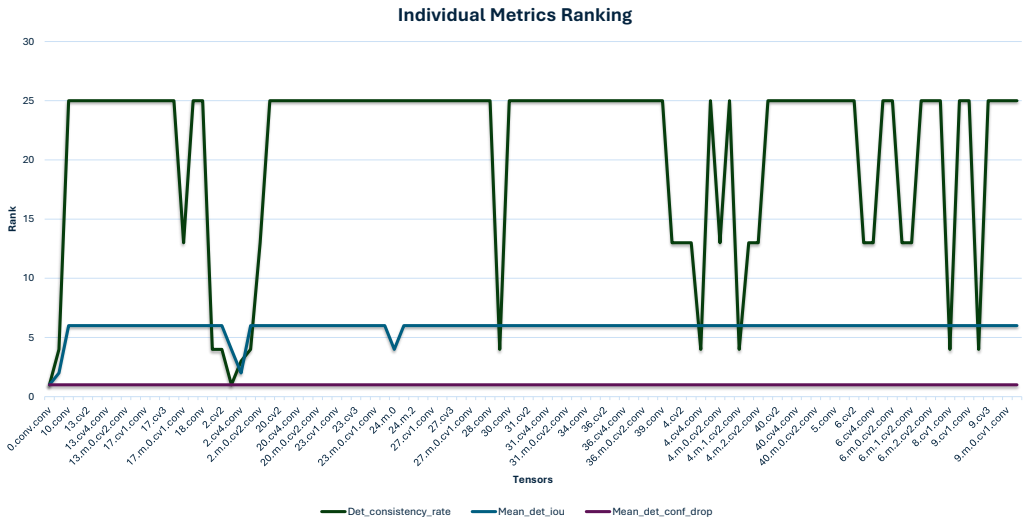
Figure A.107
ACC SG TOP 20 Vulnerable Tensors



(a) Individual Metrics Ranking - DA Metrics



(b) Individual Metrics Ranking - LL Metrics



(c) Individual Metrics Ranking - Detection Metrics

Figure A.108
ACC SG Task Based Charts

ACC Drive Video Comparison SG Fault

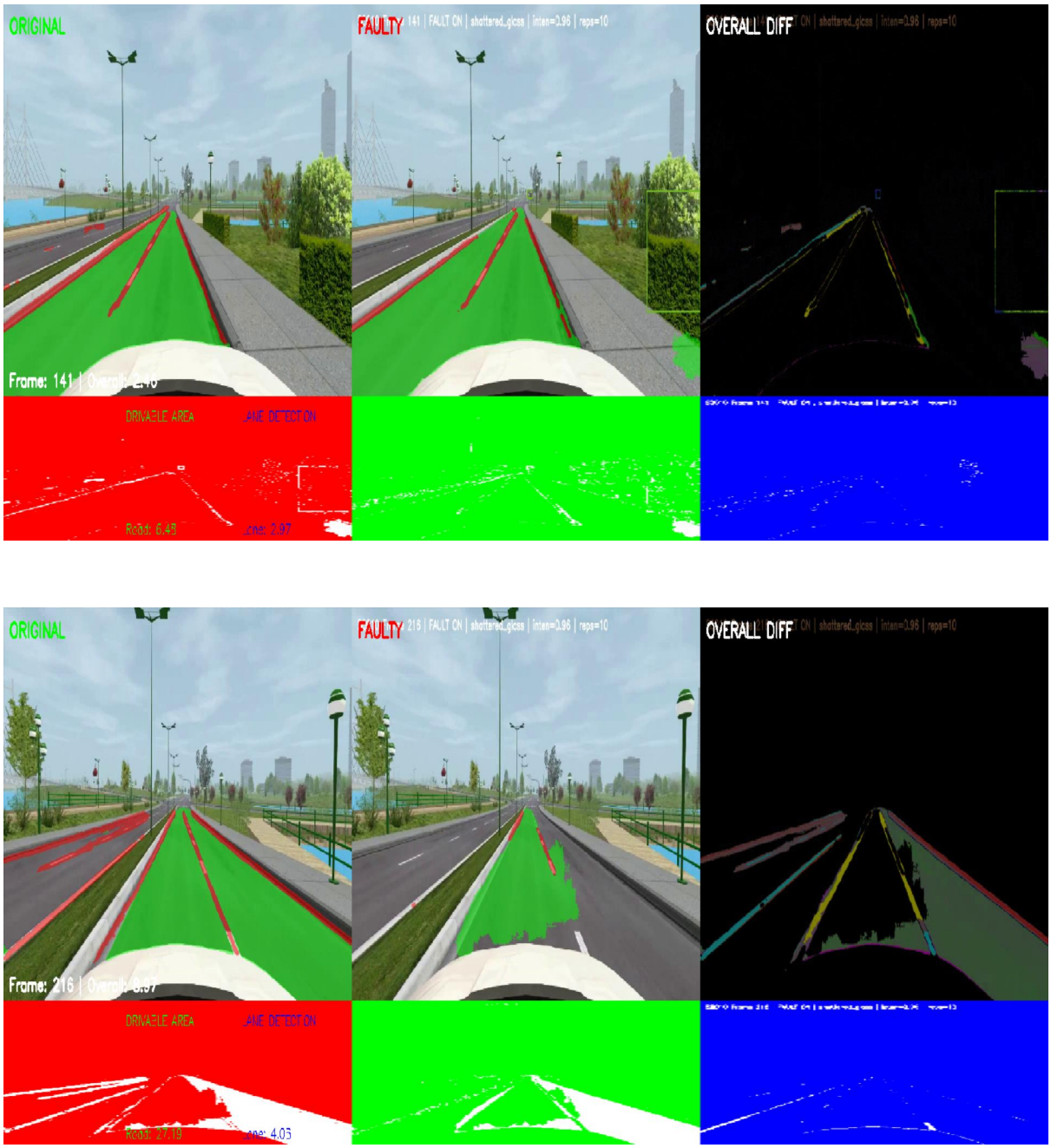


Figure A.109
Visual comparison of the ACC driving scenario under SG fault injection.

2) ROW Fault

Metric_Label	p_hat (CTD)	p_hat (LCH)	p_hat (CUTOUT)	p_hat (CUTIN)	p_hat (ACLH)	p_hat mean	Rank
Seg IoU – Drivable Area	0.0000	0.0000	0.0052	0.0000	0.0000	0.0010	4
Seg IoU – Lane Line	0.0286	0.0182	0.0156	0.0130	0.0156	0.0182	1
Pearson – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Pearson – Lane Line	0.0104	0.0000	0.0000	0.0000	0.0078	0.0036	3
Area RelDiff – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0052	0.0010	4
Area RelDiff – Lane Line	0.0234	0.0182	0.0156	0.0104	0.0156	0.0166	2
Pos Shift – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Pos Shift – Lane Line	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Detection Consistency Rate	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Detection Mean IoU	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Detection Mean Conf Drop	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6

Figure A.110
ROW Fault Vulnerable Metrics

2) ROW Fault

Tasks	Mean	Rank
Drivable Area (DA)	0.0005	2
Lane Line (LL)	0.0096	1
Object Detection (OD)	0.0000	3

Figure A.111
ROW Task Based Ranking

3) Column Fault

Metric_Label	p_hat (CTD)	p_hat (LCH)	p_hat (CUTOUT)	p_hat (CUTIN)	p_hat (ACLH)	p_hat mean	Rank
Seg IoU – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Seg IoU – Lane Line	0.0078	0.0078	0.0026	0.0052	0.0078	0.0062	1
Pearson – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Pearson – Lane Line	0.0026	0.0000	0.0000	0.0000	0.0000	0.0005	3
Area RelDiff – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0026	0.0005	3
Area RelDiff – Lane Line	0.0078	0.0078	0.0026	0.0052	0.0078	0.0062	1
Pos Shift – Drivable Area	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Pos Shift – Lane Line	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Detection Consistency Rate	0.0000	0.0000	0.0000	0.0000	0.0026	0.0005	3
Detection Mean IoU	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6
Detection Mean Conf Drop	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	6

Figure A.112
COL Fault Vulnerable Metrics

3) Column Fault

Tasks	Mean	Rank
Drivable Area (DA)	0.0001	3
Lane Line (LL)	0.0032	1
Object Detection (OD)	0.0009	2

Figure A.113
COL Task Based Ranking

4) Bullet Wake Fault

Metric_Label	p_hat (CTD)	p_hat (LCH)	p_hat (CUTOUT)	p_hat (CUTIN)	p_hat (ACLH)	p_hat mean	Rank
Seg IoU – Drivable Area	0.0442	0.0416	0.0545	0.0104	0.0416	0.0384	3
Seg IoU – Lane Line	0.1896	0.1065	0.0987	0.0961	0.2130	0.1408	1
Pearson – Drivable Area	0.0104	0.0078	0.0182	0.0000	0.0104	0.0094	8
Pearson – Lane Line	0.0597	0.0052	0.0234	0.0286	0.0468	0.0327	5
Area RelDiff – Drivable Area	0.0156	0.0286	0.0390	0.0104	0.0649	0.0317	6
Area RelDiff – Lane Line	0.0961	0.0364	0.0494	0.0442	0.0701	0.0592	2
Pos Shift – Drivable Area	0.0078	0.0026	0.0130	0.0000	0.0026	0.0052	9
Pos Shift – Lane Line	0.0078	0.0026	0.0000	0.0000	0.0052	0.0031	10
Detection Consistency Rate	0.0571	0.0000	0.0000	0.0000	0.1273	0.0369	4
Detection Mean IoU	0.0312	0.0026	0.0104	0.0052	0.0208	0.0140	7
Detection Mean Conf Drop	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	11

Figure A.114
BW Fault Vulnerable Metrics

4) Bullet Wake Fault

Tasks	Mean	Rank
Drivable Area (DA)	0.0212	3
Lane Line (LL)	0.0590	1
Object Detection (OD)	0.0494	2

Figure A.115
BW Task Based Ranking

Bibliography

[1] K. Rana and N. Khatri, "Automotive intelligence: Unleashing the potential of ai beyond advance driver assisting system, a comprehensive review," *Computers and Electrical Engineering*, vol. 117, p. 109 237, 2024, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2024.109237. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790624001654.

[2] S. Suhaib Kamran, A. Haleem, S. Bahl, M. Javaid, C. Prakash, and D. Budhhi, "Artificial intelligence and advanced materials in automotive industry: Potential applications and perspectives," *Materials Today: Proceedings*, vol. 62, pp. 4207–4214, 2022, International Conference on Materials, Processing Characterization (13th ICMPC), ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2022.04.727. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785322028954.

[3] M. Hossain, M. Rahman, and D. Ramasamy, "Artificial intelligence-driven vehicle fault diagnosis to revolutionize automotive maintenance: A review," *Computer Modeling in Engineering Sciences*, vol. 141, no. 2, pp. 951–996, 2024.

[4] A. Pandharipande et al., "Sensing and machine learning for automotive perception: A review," *IEEE Sensors Journal*, vol. 23, no. 11, pp. 11 097–11 115, 2023.

[5] B. Okumura et al., "Challenges in perception and decision making for intelligent automotive vehicles: A case study," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 20–32, 2016.

[6] T. Goelles, B. Schlager, and S. Muckenhuber, "Fault detection, isolation, identification and recovery (fdiir) methods for automotive perception sensors including a

detailed literature survey for lidar," *Sensors*, vol. 20, no. 13, 2020, ISSN: 1424-8220. [Online]. Available: https://www.mdpi.com/1424-8220/20/13/3662.

[7] B. Dally, " Hardware for Deep Learning," in *2023 IEEE Hot Chips 35 Symposium (HCS)*, IEEE Computer Society, 2023, pp. 1–58. DOI: 10.1109/HCS59251.2023.10254716. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HCS59251.2023.10254716.

[8] D. Gnad et al., "Reliability and security of ai hardware," in *2024 IEEE European Test Symposium (ETS)*, 2024, pp. 1–10.

[9] E. Talpes et al., "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020. DOI: 10.1109/MM.2020.2975764.

[10] P. K. Datla Jagannadha et al., "Special session: In-system-test (ist) architecture for nvidia drive-agx platforms," in *2019 IEEE 37th VLSI Test Symposium (VTS)*, 2019, pp. 1–8. DOI: 10.1109/VTS.2019.8758636.

[11] Z. Qiao et al., "Av4ev: Open-source modular autonomous electric vehicle platform for making mobility research accessible," in *2024 IEEE Intelligent Vehicles Symposium (IV)*, 2024, pp. 2942–2947. DOI: 10.1109/IV55156.2024.10588611.

[12] Y. Sun, X. Yang, H. Xiao, and H. Feng, "An intelligent driving simulation platform: Architecture, implementation and application," in *2020 International Conference on Connected and Autonomous Driving (MetroCAD)*, 2020, pp. 71–75. DOI: 10.1109/MetroCAD48866.2020.00019.

[13] P.-h. Hong, M. Qiu, and Y. Wang, "Autonomous driving and control: Case studies with self-driving platforms," in *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2020, pp. 17–22. DOI: 10.1109/CSCloud-EdgeCom49738.2020.00013.

[14] K. Bok, Y. Kim, A. Kim, S.-S. Lee, and K. Kim, "An optimized edge platform for 3d object detection in autonomous driving," in *2025 International Conference on*

Electronics, Information, and Communication (ICEIC), 2025, pp. 1–3. DOI: 10.1109/ICEIC64972.2025.10879728.

[15] J. Levinson et al., "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 163–168. DOI: 10.1109/IVS.2011.5940562.

[16] X. Weng, B. Ivanovic, Y. Wang, Y. Wang, and M. Pavone, "Para-drive: Parallelized architecture for real-time autonomous driving," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 15 449–15 458. DOI: 10.1109/CVPR52733.2024.01463.

[17] H.-H. Nguyen, D. N.-N. Tran, and J. W. Jeon, "Real-time semantic segmentation on edge devices with nvidia jetson agx xavier," in *2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, 2022, pp. 1–4. DOI: 10.1109/ICCE-Asia57006.2022.9954835.

[18] S. Alcaide, L. Kosmidis, H. Tabani, C. Hernandez, J. Abella, and F. J. Cazorla, "Safety-related challenges and opportunities for gpus in the automotive domain," *IEEE Micro*, vol. 38, no. 6, pp. 46–55, 2018. DOI: 10.1109/MM.2018.2873870.

[19] M. H. Ahmadilivani et al., "Special session: Reliability assessment recipes for dnn accelerators," in *2024 IEEE 42nd VLSI Test Symposium (VTS)*, 2024, pp. 1–11. DOI: 10.1109/VTS60656.2024.10538707.

[20] J.-G. Ahn, R. Nathanael, I.-R. Chen, P.-C. Yeh, and J. Chang, "Product lifetime estimation in 7nm with large data of failure rate and si-based thermal coupling model," in *2021 IEEE International Reliability Physics Symposium (IRPS)*, 2021, pp. 1–6. DOI: 10.1109/IRPS46558.2021.9405193.

[21] H. Amrouch and J. Henkel, "Reliability degradation in the scope of aging — from physical to system level," in *2015 10th International Design Test Symposium (IDT)*, 2015, pp. 9–12. DOI: 10.1109/IDT.2015.7396727.

[22] R. Balasubramanian and K. Sankaralingam, "Understanding the impact of gate-level physical reliability effects on whole program execution," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 60–71. DOI: 10.1109/HPCA.2014.6835976.

[23] H. Cho, C.-Y. Cher, T. Shepherd, and S. Mitra, "Understanding soft errors in uncore components," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6. DOI: 10.1145/2744769.2744923.

[24] S. Hamdioui, D. Gizopoulos, G. Guido, M. Nicolaidis, A. Grasset, and P. Bonnot, "Reliability challenges of real-time systems in forthcoming technology nodes," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 129–134. DOI: 10.7873/DATE.2013.040.

[25] W. Tian, W. Zha, H. Lei, and X. Yang, "Research and analysis of reliability evaluation methods for automotive electronic components," in *2024 25th International Conference on Electronic Packaging Technology (ICEPT)*, 2024, pp. 1–4. DOI: 10.1109/ICEPT63120.2024.10668438.

[26] C. M. Tan, U. Narula, and D. Kapoor, "Reliability paradox for worldwide automotive electronics," in *2017 Annual Reliability and Maintainability Symposium (RAMS)*, 2017, pp. 1–7. DOI: 10.1109/RAM.2017.7889654.

[27] L. Daotan, W. Yong, X. Zhenfeng, W. Qin, and H. Shengzong, "Study on the effect of thin resistive film's structure on the reliability of automotive electronics," in *2019 20th International Conference on Electronic Packaging Technology(ICEPT)*, 2019, pp. 1–4. DOI: 10.1109/ICEPT47577.2019.245146.

[28] G. Xie, Y. Chen, Y. Liu, R. Li, and K. Li, "Minimizing development cost with reliability goal for automotive functional safety during design phase," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 196–211, 2018. DOI: 10.1109/TR.2017.2778070.

[29] P. Mehune and S. Dorle, "Robust and adaptive neural networks for self-driving cars in challenging road conditions," in *2024 2nd International Conference on Computer,*

Communication and Control (IC4), 2024, pp. 1–5. DOI: 10.1109/IC457434.2024.10486747.

[30] P. Rech, "Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions," *IEEE Transactions on Nuclear Science*, vol. 71, no. 4, pp. 377–404, 2024. DOI: 10.1109/TNS.2024.3349956.

[31] International Organization for Standardization, "Iso 26262 road vehicles – functional safety," International Organization for Standardization, ISO Standard, 2018.

[32] International Organization for Standardization, "Iso 21448 road vehicles – safety of the intended functionality," International Organization for Standardization, ISO Standard, 2022.

[33] R. Baumann, "Soft errors in advanced semiconductor devices," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[34] E.-W. Caro-Anzola et al., "Iot and edge computing: Applications and reliability implications," in *2025 IEEE 26th Latin American Test Symposium (LATS)*, 2025, pp. 1–10. DOI: 10.1109/LATS65346.2025.10963944.

[35] J. E. R. Condia, J.-D. Guerrero-Balaguera, R. L. Sierra, and M. S. Reorda, "Investigating and mitigating critical faults in floating-point and posit arithmetic hardware," *IEEE Transactions on Emerging Topics in Computing*, vol. 13, no. 4, pp. 1605–1617, 2025. DOI: 10.1109/TETC.2025.3615827.

[36] M. M. Goncalves, J. R. Azambuja, J. E. R. Condia, M. S. Reorda, and L. Sterpone, "Evaluating software-based hardening techniques for general-purpose registers on a gpgpu," in *2020 IEEE Latin-American Test Symposium (LATS)*, 2020, pp. 1–6. DOI: 10.1109/LATS49555.2020.9093682.

[37] J. Chen et al., "Reliability assessment of large dnn models: Trading off performance and accuracy," in *2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*, 2024, pp. 1–10. DOI: 10.1109/VLSI-SoC62099.2024.10767814.

[38] M. L. Pinto-Salamanca, J. E. R. Condia, J. A. Hidalgo-López, and W. J. Pérez-Holguín, "Analyzing the reliability of stream sparse matrix-vector multiplication accelerators: A high-level approach," in *2024 IEEE 25th Latin American Test Symposium (LATS)*, 2024, pp. 1–4. DOI: 10.1109/LATS62223.2024.10534624.

[39] J. E. Rodriguez Condia, J.-D. Guerrero-Balaguera, E. J. Patiño Núñez, R. Limas, and M. Sonza Reorda, "Investigating and reducing the architectural impact of transient faults in special function units for gpus," *Journal of Electronic Testing*, vol. 40, no. 2, pp. 215–228, 2024.

[40] J. E. R. Condia and M. S. Reorda, "Evaluating the impact of transition delay faults in gpus," in *2023 36th International Conference on VLSI Design and 2023 22nd International Conference on Embedded Systems (VLSID)*, 2023, pp. 353–358. DOI: 10.1109/VLSID57277.2023.00077.

[41] J. E. Rodriguez Condia, P. Rech, F. F. d. Santos, L. Carro, and M. S. Reorda, "An effective method to identify microarchitectural vulnerabilities in gpus," *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 2, pp. 129–141, 2022. DOI: 10.1109/TDMR.2022.3166260.

[42] A. Ruospo, M. S. Reorda, R. Mariani, and E. Sanchez, "An effective iterative statistical fault injection methodology for deep neural networks," *IEEE Transactions on Computers*, vol. 74, no. 7, pp. 2431–2444, Jul. 2025.

[43] F. F. d. Santos, J. E. R. Condia, L. Carro, M. S. Reorda, and P. Rech, "Revealing gpus vulnerabilities by combining register-transfer and software-level fault injection," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Taipei, Taiwan, 2021, pp. 292–304.

[44] G. Esposito, J.-D. Guerrero-Balaguera, J. E. R. Condia, and M. S. Reorda, "Evaluating different fault injection abstractions on the assessment of dnn sw hardening strategies," in *2024 IEEE 33rd Asian Test Symposium (ATS)*, 2024, pp. 1–6. DOI: 10.1109/ATS64447.2024.10915284.

[45] G. Esposito, J.-D. Guerrero-Balaguera, J. E. R. Condia, M. Levorato, and M. S. Reorda, "Enhancing the reliability of split computing deep neural networks," in *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2024, pp. 1–7. DOI: 10.1109/IOLTS60994.2024.10616071.

[46] J.-D. Guerrero-Balaguera, I. A. Harshbarger, J. E. R. Condia, M. Levorato, and M. S. Reorda, "Reliability estimation of split dnn models for distributed computing in iot systems," in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, 2023, pp. 1–4. DOI: 10.1109/ISIE51358.2023.10227928.

[47] S. S. Hegde et al., "Estimating the impact of soft errors on ai-based perception in automotive," in *2025 IEEE 26th Latin American Test Symposium (LATS)*, 2025, pp. 1–6. DOI: 10.1109/LATS65346.2025.10963959.

[48] S. Hegde et al., "Early reliability assessment of ai-based automotive systems," *ACM Trans. Internet Things*, Aug. 2025, Just Accepted. DOI: 10.1145/3758327. [Online]. Available: https://doi.org/10.1145/3758327.

[49] J.-D. Guerrero-Balaguera et al., "Shadowfi: An open-source framework for fault evaluation of complex ic designs using hyperscale computing," *IEEE Access*, vol. 13, pp. 211 382–211 406, 2025. DOI: 10.1109/ACCESS.2025.3641762.

[50] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "Nvbitfi: Dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291. DOI: 10.1109/DSN48987.2021.00041.

[51] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 249–258. DOI: 10.1109/ISPASS.2017.7975296.

[52] S. Tselonis and D. Gizopoulos, "Gufi: A framework for gpus reliability assessment," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 90–100. DOI: 10.1109/ISPASS.2016.7482077.

[53] J. E. R. Condia, J.-D. Guerrero-Balaguera, F. F. Dos Santos, M. S. Reorda, and P. Rech, "A multi-level approach to evaluate the impact of gpu permanent faults on cnn's reliability," in *2022 IEEE International Test Conference (ITC)*, 2022, pp. 278–287. DOI: 10.1109/ITC50671.2022.00036.

[54] J.-C. Laprie, "From dependability to resilience," *38th IEEE/IFIP International Conference on Dependable Systems and Networks*, G8–G9, 2008.

[55] C. Bolchini, L. Cassano, A. Miele, and A. Toschi, "Fast and accurate error simulation for cnns against soft errors," *IEEE Transactions on Computers*, vol. 72, no. 4, Apr. 2023.

[56] D. Wu et al., "Yolop: You only look once for panoptic driving perception," *Machine Intelligence Research*, vol. 19, no. 6, pp. 550–562, 2022. DOI: 10.1007/s11633-022-1339-y.

[57] D. Wu et al., *Yolop: You only look once for panoptic driving perception*, https://github.com/hustvl/YOLOP, 2021.

[58] A. Paszke et al., "Pytorch: An imperative style, high performance deep learning library," *Advances in Neural Information Processing Systems*, 2019.

[59] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 2002.

[60] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995.

[61] G. Li, S. K. S. Hari, M. Sullivan, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators," in *International Conference for High Performance Computing*, 2017.

[62] J. Chen, G. Li, and K. Pattabiraman, "Understanding the impact of hardware faults on deep neural networks," in *International Conference on Dependable Systems and Networks (DSN)*, 2018.

[63] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, 2005.

[64] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09, Nice, France: European Design and Automation Association, 2009, pp. 502–506, ISBN: 9783981080155.

[65] L. Chen, Y. Wu, J. Stegmaier, and D. Merhof, "Sortedap: Rethinking evaluation metrics for instance segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct. 2023, pp. 3923–3929.

[66] V. Turco, L. Fezza, A. Ruospo, E. Sanchez, and M. Sonza Reorda, "Apss metrics for fault detection: Area, position, symmetry, and shape in image segmentation," in *2025 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2025, pp. 1–6. DOI: 10.1109/DFT66274.2025.11257474.

[67] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988. DOI: 10.1080/00031305.1988.10475524.

[68] K. Oksuz, B. C. Cam, E. Akbas, and S. Kalkan, "Localization recall precision (lrp): A new performance metric for object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 505–521.

[69] F. Kuppers, J. Kronenberger, A. Shantia, and A. Haselhoff, "Multivariate confidence calibration for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2020.