Enhancing System-level Safety in Autonomous Driving via Feedback Learning

Sin Yong Tan
Department of Mechanical Engineering
Iowa State University
tsyong98@iastate.edu

Weisi Fan
Department of Computer Science
Iowa State University
weisifan@iastate.edu

Qisai Liu
Department of Mechanical Engineering
Iowa State University
supersai@iastate.edu

Tichakorn Wongpiromsarn
Department of Computer Science
Iowa State University
nok@iastate.edu

Soumik Sarkar
Department of Mechanical Engineering
Iowa State University
soumiks@iastate.edu

Abstract

The perception component of autonomous driving systems is often designed and tuned in isolation from the control component based on well-known performance measures such as accuracy, precision, and recall. Commonly used loss functions such as cross-entropy loss and negative log-likelihood only focus on minimizing the loss with respect to misclassification without considering the consequences that follow after the misclassifications. In other words, this approach fails to take into account the difference in the severity of system-level failures due to misclassification and other errors in perception components. Therefore in this work, we proposed a novel feedback learning training framework to build the perception component of an autonomous system that is aware of system-level safety objectives, which in turn, enhances the safety of the vehicle as a whole. The crux of the idea is to utilize the concept of a rulebook to provide feedback on system-level performance as safety scores and leverage them in designing and computing the loss functions for the models in the framework. The framework was trained and tested on an open-sourced dataset, and the experimental results showed that the resulting model had shown superior system-level safety performance over the baseline perception model.

1 Introduction

A typical autonomous driving system usually consists of two main components, perception and control. Perception component will first observe the environment using sensors such as a camera and LiDAR to capture the RGB images and point clouds. Utilizing the captured information, perception component classifies the objects and relevant features and creates a representation of the environment around the system. The control component then evaluates the perceived environment to make decisions and compute the corresponding actuation commands, which change the system's state (for

Machine Learning for Autonomous Driving Workshop at the 36th Conference on Neural Information Processing Systems (NeurIPS 2022), New Orleans, USA.

the simplicity of the presentation, we consider planning and decision-making as part of the control component).

In recent years, deep learning models have thrived in the computer vision domain and have achieved high accuracy in many tasks such as classification [1] and object detection [2]. As these tasks are essential functions of the perception component, deep learning models are widely deployed in the perception component of autonomous systems. However, even the most advanced state-of-the-art deep learning models could not confidently *guarantee* a 100% accuracy in the abovementioned tasks due to various reasons and limitations of the models. As a result, the downstream control component may make incorrect decisions due to inaccuracy in the perceived environment as the control component does not have the ability to evaluate the correctness of the perception results.

In this regard, formal methods can be utilized to precisely define safety and other complex properties, and they have been successfully applied in the verification and synthesis of the control component of autonomous system [3–11]. On top of that, many tools were also developed to formally specify and verify simple machine learning(ML)-based systems [12–15]. Their applications to ML-based perception components, however, are still limited due to the difficulty of formally specifying properties that reflect human-level perception [16]. In recent literature, formal verification was done on simpler functions in neural networks; for example, in [15], the authors verify or specify the property of the activation function's output instead of the neural network itself, and in [12], the formal specification and verification only returns counter-example for system-level safety rule, but the explicit connection between the perception model and counter-example still requires human intervention for analysis. Additionally, as discussed in [16, 17], all misclassifications are not the same; some are more likely to result in system-level failure and more severe consequences. Therefore, it is necessary to develop a training framework capable of instilling system-level specifications and contextual semantics.

To address the issues mentioned, we present a novel method to incorporate system-level safety rules into an autonomous driving system via feedback learning. The contributions are listed as follows:

1. Propose a novel feedback learning training framework that instills system-level safety rules into the perception component of a safe autonomous driving system.
2. Develop a method to incorporate a formally defined rulebook component containing the system-level safety rule in deep learning model training.
3. Show the improvement in autonomous driving system safety on an open-sourced dataset.

2 Preliminary

Before introducing the proposed framework, this section provides the fundamental building block of the framework. There are two terminologies that will be mentioned throughout the paper: rule and rulebook. In [18], rule and rulebook are defined as below:

Definition 1 *(Rule) Given a set of realizations Ξ , rule is defined as a function $f_r : \Xi \mapsto \mathbb{R}_{\geq 0}$.*

In an autonomous driving system, realization Ξ in this definition can be interpreted as the world trajectory that includes the trajectory of all objects in the environment. Given two realizations $\xi_1, \xi_2 \in \Xi$, f_r evaluates each realization and returns a non-negative value corresponding to the severity of rule violation. Numerically, if $f_r(\xi_1) > f_r(\xi_2)$, it means that realization ξ_1 have a more severe violation of the rule than ξ_2. If $f_r(\xi_1) = 0$, it means that realization ξ_1 has no violations with respect to the rule.

Definition 2 *(Rulebook) A rulebook is a tuple $\langle \mathcal{R}, \leq \rangle$, where $\mathcal{R}$ is a finite set of rules and $\leq$ is a preorder on $\mathcal{R}$.*

In our paper, we only consider a special case of rules preordering known as *total preorder*, where the rules in $\mathcal{R}$ are all comparable and have a clear distinction in rank or level in the rulebook hierarchy. In this case, the set of rules can be organized into a hierarchy of M levels, with the rules in the higher levels indicating greater importance and priority. Given N number of rules, each level in the rulebook hierarchy should have ≥ 1 rules, so $N \geq M$. With the total preorder in place, the rulebook in this paper can be described as a function $f_{rb} : \Xi \mapsto \mathbb{R}^M_{\geq 0}$.

3 Proposed framework

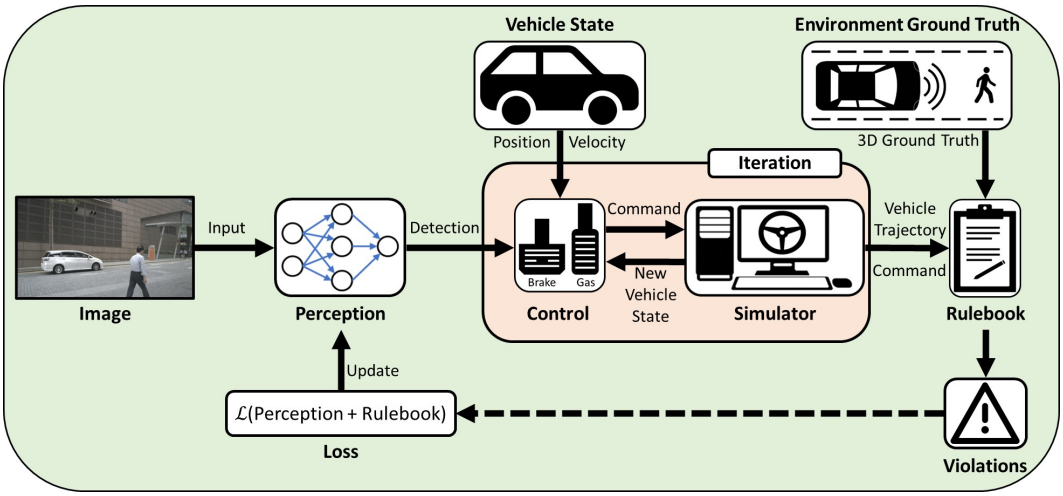

Figure 1: Figure shows the training framework to instill system-level rules into perception model.

In this section, we provided an overview of our proposed framework that could train the perception component of an autonomous driving system with system-level objective awareness via feedback learning. Our proposed framework mainly consists of the following components: (i) perception, (ii) control, (iii) simulator, (iv) rulebook, and (v) loss function, as illustrated in Fig. 1.

Starting from the leftmost of the figure, the perception component (in this case, an object detection deep learning model) takes in images captured by the vehicle camera and returns the detected classes of objects in sight, with their respective bounding boxes that localize the objects, and the confidence of the object predictions. The output from the perception model, along with the current vehicle state information such as position and velocity are then passed into the control component, and the control component evaluates the information received and returns a corresponding command to the actuators (e.g., steering, brake, throttle). After obtaining the command, the next component, simulator, updates the system state (e.g., vehicle position and velocity) and the environment. The new state is again fed back to the control component, and this iteration will repeat until it triggers a terminating condition in the simulator. Upon reaching the terminating condition, the simulator ends the simulation and outputs the trajectory of the vehicle system.

In the next stage, the rulebook component evaluates these trajectories against the environment ground truth and N number of system-level safety objectives in the rulebook for potential violations. Depending on the severity of the violations, the rulebook will provide feedback in the form of a vector of length M with non-negative real numbers, where a larger number indicates a more severe violation. The feedback from the rulebook and the performance metrics of the perception model will be used in the next step of the framework, which is the loss computation, and this loss will be backpropagated to update the weights of the perception model. This process is then repeated until the framework reaches convergence. Ultimately, the training will ensure the autonomous driving system capable of adhering to the system-level safety objectives enlisted in the rulebook.

Perception component The perception component is defined as a function $f_p : D \mapsto E$, where D is the set consisting of all possible input sensor data and E represents the set of all possible perceived environments. For example, D can represent the set of RGB data captured from a camera and point cloud captured from radar and LiDAR, while E represents the set of all possible object detection results and distance between the vehicle and perceived objects.

Control component Control component's main functionality is to return a reasonable command (or action) given the perceived environment from perception component and current vehicle state, as illustrated in Figure 1. Therefore, the control component can be formally defined as a function $f_c : E \times S \mapsto C$, where S represents the vehicle state and C is the action space of the controller.

Simulator Simulator is the component that executes the command returned by the control component through vehicle simulation and returns the updated vehicle state. Thus the simulator can be defined as a function $f_s : C \times S \mapsto S$.

Rulebook component We utilize the concept of rule and rulebook as defined in Section 2 to specify system-level safety objectives. In this framework, the values returned by rulebook are interpreted as the violation scores of the rules at each level of the hierarchy. Specifically, given a sequence ξ of states in $S \times E$, the rulebook component returns $f_{rb}(\xi) = [\gamma_1, \gamma_2, ..., \gamma_M] \in \mathbb{R}^M_{\geq 0}$, where γ_i is the weighted sum of the violation scores of the rules at level i. Note that as will be further discussed in the Remark 1, a state in ξ corresponds to the true state of both the system (obtained, e.g., from the simulator) and the environment (obtained, e.g, from the annotation of the sensor data).

Remark 1 *There are three key differences between the rulebook component and the control component. First, unlike the logic in the controller, the rules in the rulebook do not affect the motion of the vehicle. Instead, the rules defined in the rulebook are system-level rules where the sole purpose is to perform system-level safety evaluation and provide feedback in the form of violation scores. Secondly, besides receiving the vehicle trajectory from simulator, in order to perform evaluation, rulebook component will also have access to the ground truth environment, as illustrated in Figure 1 , while the control component, on the other hand, will only have access to the perceived environment. The detailed information from ground truth allows the rulebook to have a more accurate representation of the environment and be able to evaluate the trajectories correctly. Lastly, rulebook evaluation is a post hoc operation that takes place after the vehicle has moved, while on the other hand, the control component anticipates, plans, and makes decisions before an event (potential collision) happens.*

Loss Function In our proposed framework, the loss function (L) mainly consists of two parts, the perception loss L_P (or in our case, object detection loss) that aims to improve the framework's object detection capability and the rulebook loss L_R that aims to enhance the framework with system-level safety objectives. L_P will vary depending on the object detection model used in the perception component, and it will be discussed in detail in later sections along with the object detection model. L_R on the other hand, is a loss component we proposed in order to provide system-level safety and violations feedback to the perception model. As described in Section 3, the violations feedback is expressed in a length M vector with non-negative real numbers, representing the degree of violation of each safety rule in the rulebook. The violation feedback is translated into rulebook loss by using a weighted mean square error equation, where the weight corresponds to the importance and severity of each rule.

4 Results and discussion

4.1 Datasets

4.1.1 NuScenes dataset

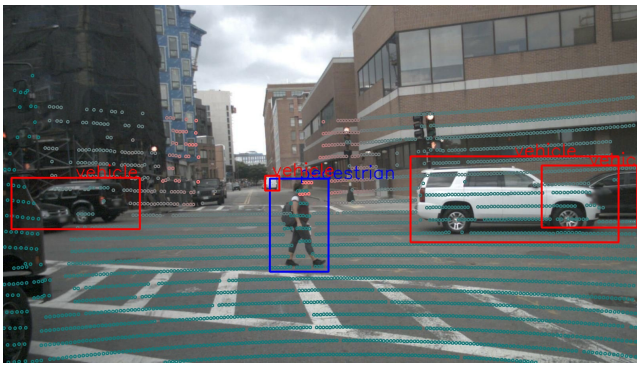

Figure 2: nuScenes camera image with 2D object bounding boxes and LiDAR data overlay.

In this paper, we used a public large-scale autonomous-driving dataset, nuScenes [19], to demonstrate the performance of the proposed framework. The dataset contains consecutive RGB camera images of

dimension (1600×900) captured in numerous driving scenes, such as varying weather, daytime and nighttime, and location, which provides sufficient rich backgrounds to generalize the perception model performance to different road conditions and realistic driving scenarios to validate the framework performance. Besides that, each captured image is also annotated with 3D object bounding boxes and their respective object classes.There are a total of 23 fine object class labels available in the dataset, but for simplicity, we use their respective course object class labels instead, which include animal, pedestrian, movable objects, and vehicle. On top of the bounding boxes and object classes, nuScenes dataset also contains LiDAR data that measures the relative distance between the vehicle and the objects on the camera frames. Figure 2 shows an example of a camera image captured using a camera mounted on the middle front of the vehicle and was superimposed with the ground truth 2D object bounding boxes and LiDAR point clouds data. Together, these annotations not only tell the location of the objects in each camera image but also how far each object is ahead of the vehicle.

4.1.2 Data pre-processing

After introducing the dataset used in this paper, we will now discuss the data pre-processing steps performed in preparation for model training. First of all, for simplicity, we only used the images captured using the middle front camera, and we downselected the scenes with more than 39 frames to ensure frame-to-frame consistency. Each scene in this dataset is captured over a duration of 20 seconds and was annotated with a 2Hz frequency, which results in approximately 40 keyframes per scene. 99.88% of the middle front camera scenes contain more than 39 frames; thus a total of 849 scenes (or 33,960 frames) were downselected for further processing. Next, by default, the camera frames are only annotated with 3D bounding boxes, but for the implementation of our object detection model, 2D bounding boxes are also required. Thus, we first project the 3D bounding boxes on the respective RGB frame to get the cuboid-shaped bounding boxes, then we compute the 2D bounding box vertices by finding the maximum and minimum coordinates of the projected bounding boxes. Finally, along with the computed 2D bounding boxes, the data was split randomly into train and validation sets with a split ratio of 7:3 respectively. In total, the train set contains 594 scenes (23,760 frames), and the validation set contains 255 scenes (10,200 frames).

4.2 Perception component - implementation

In this section, we discuss the object detection model used in the perception component. Note that the model proposed is not restricted, and it can be replaced with other object detection models if desired.

4.2.1 YOLOv5 Network

In our application, we selected one of the state-of-the-art object detection models known as YOLOv5 [20], that have numerous successful implementation cases. In the past two years, YOLOv5 was implemented in various object detection tasks, for example, in detecting people [21–24], road defects [25, 26], street trees [27] and vehicles [28]. YOLOv5 models are introduced in five different sizes; nano, small, medium, large, and extra-large, with the model sizes sorted in ascending order. Out of these different sizes, we selected the small model (YOLOv5**s**) for our perception component. The YOLOv5s model is configured with a 0.33 height multiplier and 0.5 width multiplier and will generate a model weight with only 14 MB, 7.25 M model parameters, and 16.8 GFLOPs. There are several reasons for this decision. Despite the small model size, YOLOV5s achieve great performance in the Common Objects in Context (COCO) [29] dataset. The fast inferencing speed also allows for a quick simulation in the feedback loops, and it is beneficial in an autonomous driving setting where real-time responses and decision-making are crucial criteria.

Regardless of the model size, YOLOv5 architecture can be broken down into three parts, including the model backbone, model neck, and model head. In object detection models, the model backbone plays a heavy role in image feature extraction, and in YOLOv5, the architecture used is Cross Stage Partial Network (CSPNet) [30] developed by Wang et al. (2019). Following the model backbone is the model neck, where the main role is to generate feature pyramids that consist of feature maps of different scales. In YOLOv5, the Path Aggregation Network (PANet) [31] developed by Liu et al. (2018) was used as the model neck, and the feature pyramid learned is a significant component in training the model to identify similar objects with various sizes. The last part of the model architecture is the model head, where the main purpose is to output the prediction of object classes with their corresponding bounding boxes and confidence.

4.2.2 Training

In this proposed framework, the training is performed in two consecutive phases. In the first phase, the YOLOv5 model will go through the original object detection training pipeline without the feedback loop. This phase aims to train the YOLOv5 model to obtain decent object detection performance before enhancing it with the system-level safety objectives in the later phase. In our experiment, the model is trained with an image input dimension of (416×416), with zero padding to preserve the original aspect ratio. There are several augmentation techniques that was used during the training process, including image scaling, translation, image flipping, Hue-Saturation-Value(HSV) augmentation, mixup [32] and mosaic augmentation to provide a greater and richer variation in the training images. In each training, the model weight was initialized using the YOLOv5s weights pre-trained on the Common Objects in Context (COCO) [29] dataset. There are several reasons to start the training with the pre-trained weights. First of all, the COCO dataset contains 80 object classes, in which many of the classes are similar to the object classes in nuScenes dataset. Given the great performance of the pre-trained weights on the COCO dataset, the model has already learned some meaningful features to detect and classify objects in an image. Essentially, these learned features could be transferrable to the training on nuScenes dataset to provide a good starting point and even facilitate a faster convergence in the training process.

In the second training phase, the main purpose shift from solely learning to perform object detection to simultaneously learning the system-level safety objectives as well. Two main aspects differ between the first and second training phases. First and foremost, using the best model weights from phase 1 training, the model resumes training with the addition of the feedback loop that includes the control, simulator, and rulebook components as shown in Figure 1. This feedback loop imitates a simple autonomous driving setting with the rulebook reflecting the degree of safety of the driving behavior. Secondly, most of the augmentations performed in first phase training such as the translation, scaling and mosaic augmentations are suppressed in the second phase training. The main reason to suppress these augmentations is to preserve the position of the ground truth bounding boxes with their corresponding LiDAR distances in order to create realistic and plausible simulation scenarios in the feedback loop.

4.2.3 Training hyperparameters and settings

For the results presented in this section, the experiments are conducted using the following hyperparameters and settings. The best performance was achieved using an input batch size of 64, and using stochastic gradient descent (SGD) optimizer with a momentum of 0.937, weight decay of 0.0005, and initial learning rate of 0.01. Using the YOLOv5s pre-trained weights as weight initialization, the model is able to converge and reach the best mAP at epoch 29 with a training time of 4 min per epoch. In phase 2 training, the batch size is increased to 80 (to 2 scenes with 40 frames per scene), and with data augmentation suppressed as explained in Section 4.2.2. All experiments were conducted using NVIDIA Titan RTX GPU, and a fixed random seed 0 was used in the training arguments to ensure results reproducibility. To provide a thorough study, we execute phase 2 training under two different loss settings, where the first model will be trained using the combination of perception loss and rulebook loss, and the second model will be trained using only the rulebook loss. After each training completes, the model weights and architectures are saved for evaluation on object detection performance and compliance with the safety rules in the rulebook.

4.3 Control Component - implementation

In this section, we will provide the detail of the control component as mentioned in Section 3. In Section 4.2.1, a YOLOv5 model is used as the perception component, and given an image, the model will output the bounding boxes center coordinate (x, y), width (w) and height (h), object classes (cls), and prediction confidence $(conf)$. On top of that, by incorporating the LiDAR data, the estimated distance (d_{obj}) for each object can be obtained by finding the minimum value in each enclosed predicted bounding box region. Given the above, we can express the perceived environment as $e = \{[x, y, w, h, cls, conf, d_{obj}]\} \in E$. Next, the vehicle state S describes the state of the vehicle throughout the simulation. It consist of the vehicle's position (p), velocity (v), and acceleration (a), thus it can be expressed as $s = \{[p, v, a]\} \in S$. Lastly, the command return by the controller can be expressed as $c = \{-1, 0, 1\} \in C$, with each discrete value indicating the deceleration, maintain velocity, and acceleration command, respectively.

4.4 Simulator - implementation

At the beginning of each simulation, the system state was initialized with the vehicle position $p = 0m$, velocity $v = 0ms^{-1}$, and a fixed acceleration value of $a = 3ms^{-2}$. In our simulation, each frame from a scene was treated as a static image (or observation), and the perceived objects in a frame remain the same from timestep to timestep, but each object's distance will be updated as the vehicle moves forward during the simulation. Due to the restriction of a static image in simulation, the vehicle will only move in the forward (z) direction. However, the vehicle motion can be extended into 3D by performing the same computation in the other axis of directions. Next, the time interval (τ) of each simulation step was set as a small value of $\tau = 0.1s$. Although a small time interval increases the simulation's computation load, it creates a more realistic simulation and provides finer control for the vehicle.

4.4.1 Update vehicle state

As described in Section 4.3, the system state consists of vehicle position, velocity, and acceleration. Besides the fixed acceleration value, the updated position, and velocity of the vehicle can again be estimated using two kinematic equations. First, the updated velocity v' can be estimated using the equation $v' = max((v + ca\tau), 0)$, which clips all negative velocity values to zero to ensure the vehicle does not move backward. After computing the updated velocity v', the value can be used to compute the vehicle displacement $d = \frac{1}{2}\tau(v' + v)$, and finally the updated position to $p' = p + d$.

4.5 Rulebook Component - implementation

As mentioned above in Section 2, in our paper, rulebook is a set of *total preordered* rules, and in our implementation of the autonomous driving system, there are three rules that we imposed on the vehicle. The details of the rules are described below with Rule 1,2 and 3 arranged in descending levels of the rulebook hierarchy.

Rule 1: Avoid collision with pedestrians. This rule will evaluate the vehicle's trajectory to verify if there is any collision with pedestrians during the simulation. If a collision occurs, a violation score γ_1 will be returned, and the higher the impact velocity, the larger the magnitude of the score will be. In implementation, γ_1 can be computed as $\gamma_1 = n_{ped} \times v$, where n_{ped} is the number of pedestrians in the collision, and v is the vehicle velocity during collision.

Rule 2: Avoid collision with other objects. Rule 2 is similar to Rule 1 but will evaluate the collision with any other objects instead.

Rule 3: Maintain velocity if vehicle is within $[d_1, d_2]$ range of an object. d_1 and d_2 are the predefined distances measured from the objects of interest with $d_2 > d_1$. Rule 3 will evaluate the vehicle's velocity as it approaches within the range $[d_1, d_2]$ to an object and verifies if the vehicle is maintaining a constant non-negative velocity. If the vehicle violates this rule, a violation score γ_3 will be returned.

4.6 Loss function

As discussed in Section 4.2.1, we selected the YOLOv5 model in our perception component, so the perception loss will correspond to the losses used in YOLOv5 model training. The training of the YOLOv5 model uses three losses, including the bounding boxes regression loss, objectness loss, and classification loss, where each loss contributes to different training objectives. Bounding boxes regression loss is computed using mean square error loss and it was used to train a model to predict the coordinates of the bounding boxes for objects in an image. The second part of the perception loss is objectness loss, that was computed using binary cross-entropy loss, and it trains the model to distinguish if a bounding box contains an object of interest or simply the background. The third part of perception loss is the classification loss, computed using the cross-entropy loss, and it trains the model to classify the objects captured in the bounding boxes to their corresponding classes.

Our proposed rulebook loss, on the other hand, is used to instill the system-level objectives into the model, and it is computed using a weighted mean square error equation with the length M vector input from the rulebook component. Recall that the non-negative real numbers in the vector represent the degree of violation of each safety rule in the rulebook.

Table 1: Table shows the object detection evaluation results.

Models	Metrics			
	Precision	Recall	mAP@0.5	mAP@0.5:0.95
$M_{1,P}$: Best Phase 1 Model	**0.675**	0.418	**0.574**	**0.297**
$M_{2,PR}$: Perception[a]& Rulebook Loss	0.634	**0.426**	0.533	0.260
$M_{2,R}$: Rulebook Loss Only	0.656	**0.426**	0.511	0.247

[a] **Perception Loss**: Bounding boxes regression loss + objectness loss + classification loss

However, on top of knowing the degree of violation of each rule, the severity of the violation might also be different between the rules themselves, and this information should also be reflected in the loss computation. For example, violating the rule to avoid collision with pedestrians (Rule 1) might have worse consequences that violating the rule of maintaining speed (Rule 3). In order to address that, each rule is assigned a weight w, where a rule with more severe consequences upon violation will be assigned a higher weight to scale up the loss incurred. Therefore, the rulebook loss L_R can be expressed as a weighted MSE loss of $L_R = \frac{1}{N}\sum_{i=1}^{N} w_i(t_i - \gamma_i)^2$, where w_i is the weight assigned to the i^{th} rule, t_i is the target value, and γ_i is the safety violation score for i^{th} rule. In our experiment, for simplicity, we defined $t_i = 0$ for all i, but this value can be updated to different optimal target values under different scenarios respectively.

In the implementation of the control, simulator, and rulebook component, the internal computations are implemented using PyTorch math functions with gradient computation enabled for PyTorch AutoGrad computations [33], and each component is structured as PyTorch modules to allow forward and backward propagation during the training. This allows the gradients to backpropagate through the three components to the perception model after loss computation and essentially completes the feedback learning cycle.

4.7 Experiment results

Table 2: Table shows a summary of the changes in violation scores using model $M_{2,PR}$ and $M_{2,R}$.

Scenes Scores	$M_{2,PR} - M_{1,P}$			$M_{2,R} - M_{1,P}$		
	Num. Scene / Total Scene	Percentage	Avg. ΔScore[a]	Num. Scene / Total Scene	Percentage	Avg. ΔScore[a]
Improvement[b]	**19 / 50**	**38%**	**-299.3**	**24 / 50**	**48%**	**-305.3**
Deterioration	5 / 50	10%	223.6	0 / 50	0%	-
No changes	26 / 50	52%	-	26 / 50	52%	-
Always zero	14 / 26	54%	-	15 / 26	58%	-

[a] **Avg. ΔScore**: Average *changes* in violation scores. Smaller value indicates better performance.
[b] **Improvement**: Violation scores reduced compared to model $M_{1,P}$.

There are two research questions that we are addressing with the following results; with the addition of a safety-rule-based feedback loop, (i) How does it affect the performance of the perception component (object detection)? (ii) Did the vehicle become safer (violation reduced) after training?

First, we will answer the first question, i.e., the object detection performance by looking at the results tabulated in Table 1. There are three models that we are examining, including (i) $M_{1,P}$: the best model from the first training phase, and (ii) $M_{2,PR}$: the second phase model trained using both perception loss and rulebook loss, and (iii) $M_{2,R}$: the second phase model trained using only the rulebook loss. From Table 1, the model from first phase training ($M_{1,P}$) shows the best detection results by having the highest precision, mAP@0.5, and mAP@0.5:0.95 values. This result is expected as the main focus of phase 1 training is solely trained using the objectness loss, classification loss, and bounding box loss, for which the main purpose is to focus on object detection. Moving on to the next two rows in the table, we observe that all the evaluation metrics values (except recall) for phase 2 models ($M_{2,PR}$ and $M_{2,R}$) were slightly lowered compared to $M_{1,P}$ model. In general, t here are

two reasons that could contribute to the decrease in detection performance for phase 2 models. First, phase 2 training is initialized using $M_{1,P}$, the best performing detection model, but the additional rulebook loss shifted the objective of the model to consider not only the detection but also the safety rules. Thus through the phase 2 training process, part of the detection performance achieved by $M_{1,P}$ from the first training phase deteriorates but compensated with the improvement in safety. The second reason that could contribute to the decrease in detection performance is the suppression of data augmentation in phase 2 training. In order to recreate a realistic simulation, most of the data augmentation that could alter an object's position in an image such as translation and mosaic augmentations are suppressed as discussed in Section 4.2.2. This essentially reduces the variation and diversity of the training images, causing a decrease in robustness and detection performance. Next, comparing the performance between model $M_{2,PR}$ and $M_{2,R}$, $M_{2,PR}$ is having a higher mAP@0.5 and mAP@0.5:0.95, and similar recall value compare to $M_{2,R}$. This is because unlike model $M_{2,R}$, model $M_{2,PR}$ still pertains the perception loss on top of the rulebook loss, so although the detection performance for model $M_{2,PR}$ is not as great as model $M_{1,P}$, it still outperforms model $M_{2,R}$.

However, besides object detection results, a greater goal of this paper lies in the second research question, which is the improvement in vehicle safety after the training with a feedback loop and rulebook loss. For this second part of the results, the vehicle safety is evaluated by comparing the violation scores in simulation between the models ($M_{2,PR} - M_{1,P}$) and ($M_{2,R} - M_{1,P}$). In other words, the violation score computed using model $M_{1,P}$ will act as the baseline, and by taking a score difference for each scene between, for example, $M_{2,PR}$ and $M_{1,P}$, we can evaluate if $M_{2,PR}$ had shown any improvement (violation score reduced) or deterioration (violation score increased) after going through the second phase training. Table 2 tabulates a summary of the changes in violation scores using model $M_{2,PR}$ and $M_{2,R}$, where a score improvement is indicated by the negative average score *difference*. From the table, we observe that *both* models $M_{2,PR}$ and $M_{2,R}$ had successfully reduced the violation scores in some of the test scenes, i.e., improving vehicle safety. Starting from model $M_{2,PR}$ in the table, we observe that out of 50 test scenes, there are 19 scenes (equivalent to 38% of total test scenes) that show an improvement in the violation score, which is a positive result but there is also 5 scenes that show deterioration and 26 scenes with no changes in violation scores. This result, however, is outperformed by model $M_{2,R}$ which has 24 scenes (48% of total test scenes) that show improvements, and none of the test scenes shows a deterioration in violation scores. Besides the superior number of scenes with score improvements, the average violation score changes in $M_{2,R}$ also shows a smaller value of -305.3, indicating that the magnitude of score improvement is also greater for model $M_{2,R}$ compared to model $M_{2,PR}$. In short, model $M_{2,R}$ that was trained only using rulebook loss in the second phase of training is showing a greater improvement in vehicle safety as compared to model $M_{2,PR}$ that was trained using both perception and rulebook losses. Finally, the last row of the table shows that, among the scenes that shows no changes in the violation scores, more than 50% of them have *no* safety violations (zero violation scores) to begin with, so essentially these scenes cannot be improved further, but on the other hand, this also means that the remaining scenes *does* have safety violations and there is still room for improvements.

5 Conclusion and future work

We proposed a novel framework capable of learning system-level objectives via feedback learning and ultimately enhancing the safety of autonomous driving systems. We also utilized a rulebook component that contains system-level safety objectives and provides a violation score that reflects the severity of rule violations. Finally, we demonstrated the object detection and safety performance of the models before and after going through the proposed feedback learning training. There are currently two directions for our future research. First, we will attempt a different violation score aggregation strategy to allow the aggregation of rules with different hierarchies of importance and to find a balance between object detection and vehicle safety performance. Secondly, we will also utilize autonomous driving simulation tools such as CARLA [34] to gain a greater degree of freedom in controlling the simulation and scenario generation.

Acknowledgments and Disclosure of Funding

This work was partly supported by the National Science Foundation under grants CAREER-1845969, CNS-2141153, and Iowa State Translational AI Center (TrAC) seed grant.

References

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

[2] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[3] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 3116–3121, April 2007.

[4] D. Conner, H. Kress-Gazit, H. Choset, A. Rizzi, and G. Pappas, "Valet parking without a valet," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 572–577, 2007.

[5] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[6] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pp. 101–110, 2010.

[7] P. Tabuada and G. J. Pappas, "Linear time logic control of linear systems," *IEEE Transactions on Automatic Control*, vol. 51, no. 12, pp. 1862–1877, 2006.

[8] A. Girard and G. J. Pappas, "Hierarchical control system design using approximate simulation," *Automatica*, vol. 45, no. 2, pp. 566–571, 2009.

[9] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2009.

[10] L. I. R. Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion planning," in *52nd IEEE Conference on Decision and Control*, pp. 3217–3224, Dec 2013.

[11] T. Wongpiromsarn, K. Slutsky, E. Frazzoli, and U. Topcu, "Minimum-violation planning for autonomous systems: Theoretical and practical considerations," in *2021 American Control Conference*, 2021. submitted.

[12] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *International Conference on Computer Aided Verification*, pp. 432–442, Springer, 2019.

[13] D. J. Fremont, J. Chiu, D. D. Margineantu, D. Osipychev, and S. A. Seshia, "Formal analysis and redesign of a neural network-based aircraft taxiing system with verifai," in *International Conference on Computer Aided Verification*, pp. 122–134, Springer, 2020.

[14] P. Schmitt, N. Britten, J. Jeong, A. Coffey, K. Clark, S. S. Kothawade, E. C. Grigore, A. Khaw, C. Konopka, L. Pham, *et al.*, "nureality: A vr environment for research of pedestrian and autonomous vehicle interactions," *arXiv preprint arXiv:2201.04742*, 2022.

[15] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 169–178, 2019.

[16] T. Dreossi, S. Jha, and S. A. Seshia, "Semantic adversarial deep learning," in *International Conference on Computer Aided Verification*, pp. 3–26, Springer, 2018.

[17] A. Badithela, T. Wongpiromsarn, and R. M. Murray, "Leveraging classification metrics for quantitative system-level analysis with temporal logic specifications," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 564–571, IEEE, 2021.

[18] A. Censi, K. Slutsky, T. Wongpiromsarn, D. Yershov, S. Pendleton, J. Fu, and E. Frazzoli, "Liability, ethics, and culture-aware behavior specification using rulebooks," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8536–8542, 2019.

[19] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.

[20] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, K. Michael, J. Fang, imyhxy, Lorna, C. Wong, Z. Yifu, A. V, D. Montes, Z. Wang, C. Fati, J. Nadar, Laughing, UnglvKitDe, tkianai, yxNONG, P. Skalski, A. Hogan, M. Strobel, M. Jain, L. Mammana, and xylieong, "ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations," Aug. 2022.

[21] M. Jacoby, S. Y. Tan, M. Katanbaf, A. Saffari, H. Saha, Z. Kapetanovic, J. Garland, A. Florita, G. Henze, S. Sarkar, *et al.*, "Whisper: Wireless home identification and sensing platform for energy reduction," *Journal of Sensor and Actuator Networks*, vol. 10, no. 4, p. 71, 2021.

[22] A. Saffari, S. Y. Tan, M. Katanbaf, H. Saha, J. R. Smith, and S. Sarkar, "Battery-free camera occupancy detection system," in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, pp. 13–18, 2021.

[23] M. M. Islam, A. A. R. Newaz, and A. Karimoddini, "A pedestrian detection and tracking framework for autonomous cars: Efficient fusion of camera and lidar data," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1287–1292, IEEE, 2021.

[24] M. Jacoby, S. Y. Tan, G. Henze, and S. Sarkar, "A high-fidelity residential building occupancy detection dataset," *Scientific Data*, vol. 8, no. 1, pp. 1–14, 2021.

[25] Z. Chen, Y. Zhang, Y. Luo, Z. Wang, J. Zhong, and A. Southon, "Roadatlas: Intelligent platform for automated road defect detection and asset management," in *ACM Multimedia Asia*, pp. 1–3, 2021.

[26] R. Vishwakarma and R. Vennelakanti, "Cnn model & tuning for global road damage detection," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 5609–5615, IEEE, 2020.

[27] A. Bahety, R. Saluja, R. K. Sarvadevabhatla, A. Subramanian, and C. Jawahar, "Automatic quantification and visualization of street trees," in *Proceedings of the Twelfth Indian Conference on Computer Vision, Graphics and Image Processing*, pp. 1–9, 2021.

[28] Y. Zhang, W. Shi, M. Zhang, and L. Peng, "Electric and fuel car identification based on uav thermal infrared images using deep convolutional neural networks," *International Journal of Remote Sensing*, vol. 42, no. 22, pp. 8526–8541, 2021.

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[30] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "Cspnet: A new backbone that can enhance learning capability of cnn," 2019.

[31] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," 2018.

[32] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[34] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*, pp. 1–16, PMLR, 2017.

A Control mechanism

Internally, the command returned by the controller is governed by the vehicle stopping distance (d_{stop}) and the control logic, which are listed as follows. First of all, the vehicle stopping distance (d_{stop}) can be computed as $d_{stop} = v^2/2a$ where v and a are the current velocity and acceleration, respectively. Using the computed d_{stop}, the controller can estimate the projected vehicle position and the distance between the vehicle and the perceived objects (d_{obj}) at that projected position. Then the controller will return a command obeying the following logic:

Logic (1): Is $d_{obj} < d_{ped}$? In order to avoid collision with pedestrians, the vehicle should maintain sufficient clearance distance (d_{ped}) with pedestrians and should decelerate if the vehicle gets too close to a pedestrian.

Logic (2): Is $d_{obj} < d_{oth}$? Besides keeping distance from pedestrians, the vehicle should also maintain sufficient clearance distance (d_{oth}) with other objects.

Logic (3): Is $d_{safe_1} < d_{obj} < d_{safe_2}$? $[d_{safe_1}, d_{safe_2}]$ is specified as a safe object's distance range where the vehicle should slowly transition from acceleration to maintain velocity (and eventually decelerate) as it comes within this proximity of the perceived objects. In our case, $d_{safe_1} = d_{ped}$ if the object is pedestrian or $d_{safe_1} = d_{oth}$ if the object is other objects, and $d_{safe_2} = d_{safe_1} + 2$ (meters).

With the logic in place, the controller's command is returned as:

$$c = \begin{cases} -1 & \text{if (1) or (2) is True} \\ 0 & \text{else if (3) is True} \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

B Simulation terminating condition

As mentioned in Section 3, the simulator will return the newly updated vehicle state to the controller, and the interaction cycle between controller and simulator repeats as the simulation progresses. Within this interaction cycle, there are three terminating conditions that will stop the simulation and proceed to the next step in the feedback loop.

Cond. 1: Number of iteration steps = 200. During the simulation, an iteration counter keeps track of the number of updates performed by the simulator, and the simulation will terminate when this counter reaches 200.

Cond. 2 Vehicle velocity $v = 0$ and controller command $c = 0$ or $c = -1$. If the vehicle fully stopped during the simulator (e.g., stopped in front of an object), and the controller command returns the "maintain speed" or "decelerate" command, the updated vehicle state will be the same as the initial state (at the timepoint of the simulation), and thus the simulation should be terminated.

Cond. 3 Vehicle distance traveled $\geq$ 70 m. The LiDAR data used to estimate the object distances have a usable return of up to 70 m; thus 70 m was used as the maximum vehicle travel distance that terminates the simulation when exceeded.

As the simulation terminates, the simulator will return the vehicle trajectory information (Ψ), including a series of position, velocity, and acceleration throughout the simulation, and this information will be passed to the next component, the rulebook, for system-level safety evaluation.