A Safety Architecture for Self-Driving Systems

Shai Shalev-Shwartz, Moran Molnar, Ilai Granot, Almog Shany, Amnon Shashua

Mobileye, 2024

Abstract

With the increasing presence of autonomous vehicles on the road and the imminent commercialization of fully autonomous systems at scale, the need for transparency regarding the robust safety architectures underpinning these systems has become paramount. Mobileye is advancing self-driving technologies across a spectrum of autonomy levels, including Driver Assistance, Eyes-off, and fully No-driver systems. This paper describes our architecture for Eyes-off and No-driver Self-Driving Systems (SDSs), with a strong emphasis on adherence to critical safety principles. We advocate two primary safety goals. The first is the elimination of *unreasonable risk*, with an emphasis on *transparency* about the boundaries distinguishing reasonable and unreasonable risks. We provide crisp definitions for these boundaries by addressing lapses of judgment in planning, identifiable hardware failures, the elimination of reproducible errors, and leveraging redundancy to mitigate 'black swan' events. The second goal focuses on the 'greater good': self-driving cars must reduce overall harm compared to the status quo of human-driven vehicles.

Contents

1 Introduction

Towards the large-scale deployment of self-driving cars, one pressing question looms large: how safe is safe enough? This question underscores the complex interplay between technological innovation and societal expectations, especially in safety-critical domains like autonomous driving.

Among the many metrics used to assess safety, the Mean-Time-Between-Failure (MTBF) has emerged as a central benchmark. By focusing on the frequency of accidents or harm, MTBF provides a purely statistical perspective and serves as a 'greater good' measure: self-driving systems must reduce overall harm relative to human-driven vehicles. While a high MTBF is undoubtedly a desirable goal, we argue that it is not sufficient on its own.

One limitation of relying solely on MTBF is its dependency on human driving statistics, which are significantly skewed by illegal or irresponsible behaviors such as driving under the influence, texting while driving, or exceeding legal speed limits. Another limitation is that humans are expected to properly respond to events even when those are extremely rare. For example, a baby lying on the highway is an extremely rare event, way beyond human accident statistics, and yet a human driver is expected to attempt to avoid the collision by swerving or braking. In other words, human drivers are also held to an additional standard: the *'duty of care'*, which mandates avoiding reckless behavior and refraining from taking *unreasonable risks*.

This paper aims to address these gaps by formalizing the concept of 'unreasonable risk' across all elements of the architecture of a Self-Driving System (SDS). By doing so, we propose a framework that goes beyond statistical metrics like MTBF, incorporating principles of transparency, accountability, and adherence to robust safety standards.

We start with a brief description of how our SDS operates, followed by an enumeration of potential failure scenarios. In Section **??**, we formalize our safety requirements, distinguishing between identifiable, reproducible, and 'black swan' failures, and aligning our approach with international standards. The subsequent sections then delve into the methodology for eliminating unreasonable risks across all types of failures.

1.1 How does our SDS work?

The three main hardware components of an SDS are sensors, computer, and actuators.

The SDS relies on various sensors to perceive the environment and understand the vehicle's own state. Environmental sensors include cameras, radars, lidars, and in some configurations, ultrasonic sensors for parking and microphones to detect sirens. Vehicle-specific sensors, such as wheel-tick sensors and gyroscopic yaw-rate sensors, monitor the car's speed and angular velocity. The system also incorporates HD and SD maps, which serve as supplementary data sources for navigation, road structure, and traffic rules. GPS is used for localization (find out where we are on the map) on the SD map (for the HD map, we rely on the cameras for more accurate localization).

The information from all of the sensors arrives to a computer. The computer uses this information in order to figure out where the ego car currently is, where it wants to be, what its kinematic state is, what objects are around it (including their kinematic state), and what semantic information is currently relevant (road markings, traffic signs, and traffic lights that dictate traffic rules). This part of the compute is called sensing or perception, and we call the output of it the "sensing state". Given the "sensing state", the computer should plan what to do next. This planning phase, or driving policy, should consider the current sensing state, but should also deal with uncertainties. Uncertainties stem from limited visibility as well as from the unknown intentions of other road users. The output of planning is speed and curvature commands, determining where the self-driving car should go and at what speed.

Finally, the actuators translate the speed and curvature commands into electric commands to the engine, braking, and steering, in order to follow the commands of the planner.

There are additional important components related to Human-Machine-Interface (HMI), such as display to the driver / passengers in the car, queries to human operators via a tele-operation system (if one exists), and communication with emergency response teams. HMI elements are not covered in this report.

1.2 What can go wrong?

In complex systems like self-driving vehicles, unexpected events and failures may be manifested over time. Similarly to human drivers, failures in self-driving vehicles can be classified by type:

Hardware failures: Hardware failures may occur when a sensor malfunctions, the SDS computer fails, an actuator loses power, we have a flat tire, etc. This is similar to a human driver who could pass out due to a medical issue.

Software bugs: Even if the hardware is operational, we might have a software bug. For example, a buffer overflow in some non-critical component might corrupt safety critical code. This is similar to a human falling asleep or texting while driving.

Perception failures: Interpreting the sensor data and constructing a sufficiently accurate "sensing state" out of it is a complicated task, which involves AI tools such as deep learning. Deep learning is a statistical approach, which may err unexpectedly. Humans might also make perception errors (typically in bad visibility conditions or when driving new areas for the first time).

Planning failures: The decision making of the driving policy may also result in errors, due to mis-interpreting the intentions of other road users, due to errors arising out of limited visibility, or due to butterfly effects (a decision that looks perfect now starts a chain of events that ends up with an unavoidable collision). When humans make planning errors we often call them "lapse of judgement".

Actuation failures: These are cases in which the vehicle didn't follow the planner's commands, similarly to humans who hit the throttle instead of the brake pedal.

Adversarial disruption by other players: These are cases in which an adversarial player deliberately disrupt the normal operation of the system. It can be by a cyber attack or by physically harming some vehicle element. While we take design steps to avoid failures due to adversarial behavior, these issues are out of the scope of this report.

Requiring that an SDS will be perfect is unrealistic. A potentially sufficient requirement is that, on average, the SDS should outperform human drivers. After all, if the SDS outperforms human drivers on average, adopting it will decrease the total number of injuries and casualties from car accidents. However, as mentioned previously, comparing SDS performance with that of human drivers can be misleading since the statistics of human drivers are heavily affected by illegal behavior. Such behavior is irrelevant for an SDS, which illustrates why comparing human drivers to the SDS is not truly apples-to-apples. So, while outperforming human drivers on average is one desired goal, it is not the only requirement to achieve a safe SDS. Automotive standards (such as [**?**] and [**?**]) requires the absence of *unreasonable* risk due to hazards caused by malfunctioning behavior of the system. We argue that a safe SDS should be transparent on its definition of unreasonable risk and the mitigations which are taken to avoid it. In the next section we lay out our view on the safety requirements from an SDS and the methodology used to meet the requirements. The rest of the sections describe our approach for meeting these safety requirements.

2 Safety requirements

While we ultimately would like to minimize the number of collisions, as illustrated in Figure **??** not all collisions are avoidable. Therefore, our first design principle is:

- Do not cause a collision. This is the foundational principle of SDS safety, ensuring that the SDS behavior does not instigate incidents.
- If someone else is going to cause a collision and the SDS can avoid it without causing another collision, it should try.

To formalize this design principle, we rely on the Responsibility-Sensitive-Safety (RSS) model [**?**]. RSS is an open, transparent framework that was proposed by Mobileye in 2017 and its principles have been embedded in worldwide standards [**?**, **?**, **?**, **?**]. It is designed to guide SDS decision-making toward safety by defining responses to potentially

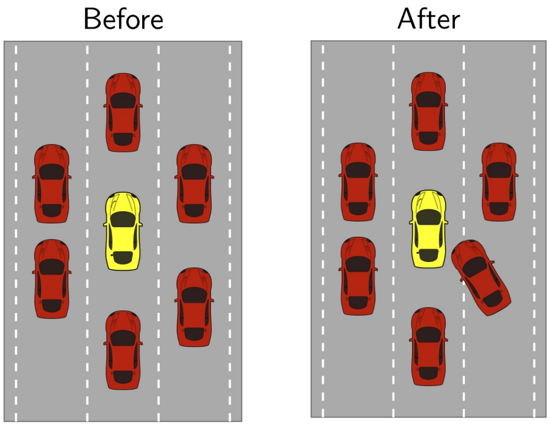

Figure 1: Car accidents are asymmetric. This diagram is intended to demonstrate a situation where the yellow car can do nothing to prevent the collision, so if the yellow car is a self-driving car, even if there is a collision it is operating correctly. On the other hand, the red car that swerves into the yellow car is clearly at fault, so if a self-driving car behaves like this then it is clearly failing.

hazardous situations. RSS provides guidelines such that, absent other system failures, a driving policy which adheres to its rules is formally guaranteed not to cause a collision. Furthermore, RSS defines parameters through which one can draw the line between reasonable driving decisions and lapse of judgement. The RSS model is particularly useful for handling uncertainties and therefore balancing the tradeoff between safety and usefulness. Indeed, an overly-cautious SDS that fears unreasonable behavior of other drivers will drive super slow and impede traffic, while an overly-optimistic SDS could drive recklessly and compromise safety.

For full formal details of RSS we refer the reader to [**?**], and here we only briefly outline the main ingredients. At any point in time, RSS dictates if the state of the self-driving vehicle relative to its surroundings is dangerous or not. If the state is not dangerous, the driving policy may take any decision it likes (with some limitations that make the decision consistent with the RSS rules, see [**?**] for details). If the SDS is in a potentially dangerous state, RSS prescribes a "proper response," such as reducing speed or stopping a lateral movement to avoid a potential collision. Adherence to these responses ensures that the SDS minimizes accident risk. One of the formal guarantees given by the RSS model is that if the driving policy adheres to the proper response prescribed by RSS at dangerous situations, then it is guaranteed not to cause an accident.

The integration of RSS into the SDS architecture eliminates the risk of planning failures. This isolates potential failures to occurrences when we are not following the RSS proper response in dangerous situations [1], which can happen due to hardware failures, software bugs, perception errors, or actuation errors. We next classify failures into three categories, each with tailored mitigation strategies.

2.1 Types of Failures: Identified, Reproducible, and Black Swans

While adherence to RSS minimizes accident risk, different types of system failures could cause us not to follow RSS rules. The root cause of these failures can be due to hardware malfunctioning, software bugs, perception errors, or actuation errors. Regardless of the root cause, we classify potential failures into the following three categories:

[1] While the most important requirement from our SDS is to not *cause* an accident, if another road user might cause an accident, we should try to avoid it. To do so, we do not allow the vehicle to cause a different accident, but if we can avoid the accident without causing another one, we should do so. The formal guidelines regarding when the system should avoid an accident and how are outlined in Definition 12 in the RSS paper [**?**]. In terms of practicality, our SDS contains an Automatic Emergency Braking and Swerving layer, which is heavily tested on simulators as well as in dedicated test tracks.

An identified issue: These are cases in which the system itself identifies a problem with a crucial element of the SDS. For example, we identify that one of our sensor is broken or blocked, that one of our computing boards is burnt, or that we have a flat tire. Ultimately, we strive to maximize the system's ability to detect its own failures and to respond with a Minimal Risk Maneuver (MRM), leveraging built-in redundancies where possible. For example, our SDS has two computing boards, and if we detect that one of them is burnt, the other is desgined to bring the vehicle to a safe stop on the road shoulder.

A reproducible failure: We define reproducible failures as scenarios in which the system fails, without understanding that it fails, and this failure can be easily reproduced on a test track. For example, suppose that a system has a sensor set that is insufficient for emergency swerving maneuvers to avoid debris that might fall from a truck. We can easily reproduce this failure on a test track (while using non-harmful debris). As a sharper example, suppose that a system consistently doesn't detect babies lying on the road. This is an incredibly rare event, and it is likely that even in an extremely large dataset we will see zero examples of this failure. However, using a baby doll instead of a real one, it is easy to reproduce this failure on a test track. The development methodology for collecting reproducible errors involves offline probing of data, analysis of human interventions, as well as recreation and simulation of potentially hazardous scenarios. Naturally, we might have unknown reproducible errors. Using Carl Sagan's words, "Absence of Evidence does not mean Evidence of Absence". We refer to an unknown reproducible failure as a "black swan" (see next paragraph) until we find a single instance of it, and then it again becomes a reproducible error.

Black swans: The term "black swans" has been used by [**?**] to represent unexpected events. In our case, we use it for rare, unexpected failures that cannot be easily predicted or reproduced, often due to complex environmental factors, unforeseeable hardware anomalies, or the statistical nature of modern AI tools. These are cases in which the system fails, it doesn't understand that it failed, and this type of failure is either not reproducible or wasn't known to be a reproducible error. There are various reasons for such failures but the most common one stems from the reliance on statistical learning tools (such as deep learning). While deep learning is a powerful tool, in almost all cases there is a tail of anecdotal examples on which the model fails. For those examples, if we manage to "understand" the source of the failure in a way that allows reproducibility, then we have a mechanism for provably improving the model. However, in many cases we do not have enough understanding to reproduce the problem and so, even if we continue to train the model with the problematic examples as additional training examples, we cannot know if the model will truly solve the problem or rather overfit to these examples.

2.2 Safety Goal

Our safety goal is comprised of two principles:

- The overall Mean-Time-Between-Failure (MTBF) of the system should be at least as good as human statistics.
- Absence of *unreasonable risk* where the SDS provider should be *transparent* regarding the boundary between reasonable and unreasonable risk. It is important to note that this boundary is not solely based on an MTBF argument.

The first requirement can be seen as a "greater good" argument: using self-driving cars shouldn't cause more harm overall relative to the current situation with human driven cars. As mentioned before, the MTBF of humans is heavily affected (reduced) by illegal, irresponsible, or careless behavior, so an unadjusted human-level MTBF is insufficient. Hence, the second requirement complements the first one by demanding that the SDS eliminates unreasonable risks. This is the essence of ISO 26262 [**?**], ISO 21448 [**?**], and the RSS framework.

The RSS framework deals with the multi-agent nature of driving and provides a transparent model for setting the boundary between reasonable and unreasonable assumptions about the behavior of other road agents. As such, it determines what is considered a reasonable risk for the driving policy. It also provides a framework to guarantee **zero** unreasonable planning errors. In the MTBF language we are expecting an **infinite** MTBF for "lapse of judgements" errors.

The Functional-Safety (FuSa) ISO 26262 standard [**?**] deals with failures of electric/electronic systems. Its focus is on systematic failures (e.g. design defects) and random hardware failures (e.g., electronic component wear or faults).

FuSa mainly tackles risks due to hardware faults (e.g. a microcontroller fails) or classical software bugs (e.g. memory corruption). The main goal of FuSa is to ensure that the system is designed to behave safely even when faults occur. FuSa requires that the MTBF of failing to identify certain hardware and software bugs should exceed 10^8 hours of driving which is way beyond Human accident statistics. This is the methodology we employ for identifiable errors.

The Safety-Of-The-Intended-Functionality (SOTIF) ISO 21448 standard [**?**] deals with "functional insufficiencies". These are risks which do not stem from malfunctioning hardware or a classical software bug, but instead, risks due to insufficient accuracy level of the system. For example, a camera-based system that misinterprets a glare as an obstacle. SOTIF refers to such failures as "functional insufficiency". To separate classical software bugs from functional insufficiencies, we sometime call such errors "AI bugs". The goal of SOTIF is to ensure that the system performs as intended under all *foreseeable* operational conditions. To do so, SOTIF guides the development process, shifting failures from being unknown to being known and from being hazardous to being non-hazardous. SOTIF requires a hazard analysis process involving the creation of all sorts of scenarios, including edge cases, and reasoning about potential faults in each scenario. This is the methodology we employ for reproducible errors. By and large, we aim to eliminate all reproducible errors independent of their MTBF. There are some exceptions where reproducible failures are to be classified as reasonable risks and the important thing for this classification is to be transparent about it. For example, consider a situation where an individual deliberately places nails on the road, resulting in the simultaneous puncture of two tires. In such a scenario, the self-driving system may lose control of the vehicle due to the severe mechanical instability caused by the dual tire failure. While this poses a safety risk, and is a reproducible failure, addressing this type of edge case could require disproportionate effort, cost, or technological complexity relative to its likelihood of occurrence.

Combining RSS, FuSa, and SOTIF, we end up with the following methodology for achieving the safety goal:

- Our starting point is the RSS model, which formalizes the notion of "not causing accidents". In addition, RSS provides full coverage for the driving policy element of the SDS. Equipped with RSS, the validation process is isolated to verifying that we would not fail to respond properly (according to RSS definitions) at potentially hazardous conditions.

- The next step is to follow the FuSa standard by defining a large set of potential failures. The system should identify each one and define the Minimal Risk Maneuver (MRM) to perform as a mitigation. This yields a set of *identifiable* failures. We also analyze the probability of failing to properly identify such failures and keep the residual risk within target rates according to the standard.

- The last step follows the SOTIF standard as follows. We record potential failures of the system during the development phase by offline probing, interventions and complaints of human drivers, and recreation and simulation of potentially hazardous and edge-case scenarios. For each such failure, we determine whether it is a reproducible failure or a black swan (based on the ability to reproduce the failure on a test track). The treatment of reproducible errors follows the ASPICE methodology [**?**]:

  - When a bug or error is identified and fixed, the fix should be documented. This ensures that there is an auditable record of what went wrong and how it was resolved. This is critical for ensuring process transparency and enabling root cause analysis in the future.
  - Adding a test to prevent recurrence: to ensure monotonically increasing quality, a test case is added in order to validate the fix and to ensure that the error does not recur in subsequent versions of the software.
  - The fix may involve change of Operational Design Domain (ODD). For example, the SDS design may exclude operating during a snow storm. In this case, the ODD restriction as well as the method for detection of out of ODD mode should be documented.
  - In the design of the SDS, certain hazardous scenarios may be identified where system failure is acknowledged but deemed as an acceptable risk (such as the example of nails on the road mentioned above). Consequently, such risks may be categorized as reasonable risks and documented as such.

- For black swan failures, we design the system with sufficient redundancies so that black swan failures will always involve a failure of at least two sub-systems.

- Finally, the overall MTBF of the system should be estimated, with the goal of matching or exceeding human-level statistics.

Redundancy is a central tool for reaching our safety goal. It is key to increasing the overall MTBF of the system as well as for identifying failures. Embedding redundancy in our architecture ensures it remains resilient, adaptable, and responsive to a range of potential hazards. The next section is devoted to the benefits of redundancies and to fusion techniques between several sources.

3 Redundancy and Fusion

Redundancy is a cornerstone of designing systems that demand extremely high accuracy and reliability, especially in safety-critical domains such as autonomous driving. By incorporating multiple independent components or subsystems that perform the same function, redundancy ensures that the system can continue to operate correctly even if one component fails. This approach not only enhances fault tolerance but also enables cross-validation among redundant elements, improving the ability to identify failures and to adjust the behavior of the system accordingly. For instance, redundant sensors can provide overlapping data streams, allowing the system to corroborate information and detect inconsistencies. Similarly, redundant computing units can execute identical algorithms in parallel to identify and rectify computational discrepancies.

Intuitively, the probability that two redundant sub-systems that perform the same function will fail at the same time should be very low. We start with formalizing this intuition in Section **??**. The next question is what to do when two sub-systems disagree. This is discussed in Section **??**.

3.1 MTBF of Independent Systems

Consider two systems A_1, A_2. The MTBF of a system is the average time it can be activated consecutively without causing a failure.

Lemma 1 *Assume that a system failure starts no more than 10 seconds before the actual collision it causes. Suppose that the failures of A_1, A_2 are independent random variables and each one has an MTBF of T_1, T_2 hours, respectively. A joint failure is a situation in which both A_1 and A_2 fail, and let us denote the mean time between joint failures by T hours. Then, $T \geq 180\, T_1\, T_2$.*

Proof Let us take a large set of drives, and divide them into segments of 10 seconds, while leaving "gaps" of 10 seconds after each piece. Let us throw all pieces into a virtual bag. Let p_i be the probability to retrieve a random piece from the bag on which A_i fails. If the system failures are independent, then the probability that both of them fail on the same piece is $p = p_1 p_2$. So, the average number of pieces before a failure is $1/p$. Translating to hours, we get $T = (20/3600)/p = (1/180)/p$, which gives

$$T = \frac{(1/180)}{p_1 p_2} = T_1\, T_2\, 180^2\, (1/180) = 180\, T_1\, T_2$$

■

The above lemma relies on the assumption that the two sub-systems have independent failure modes. Naturally, this will not always be the case. For example, if two sub-systems rely on the same sensor, then both would fail if that sensor is broken. However, if we have a different element in the SDS that detects if a sensor is malfunctioning or not, the conditional failure probability (conditioning on a well-functioning joint sensor) may be independent. In particular, this is helpful in assessing 'black swan' failures. Likewise, if two sub-systems are being trained on the same data, then a common failure in the training data might lead to a common failure of the two sub-systems. However, such a failure is likely to be 'reproducible'. By carefully designing the system, we can increase the likelihood that redundant systems do not have common un-identified and un-reproducible failures.

Developing multiple redundant systems is an established development methodology. Sometimes, it makes sense to rely on an independent assumption also for the sake of system validation. However, it is impossible to prove independence, and whenever flat statistics of the system performance can be done, it is a more advisable validation approach.

3.2 Fusion: Worst-case, Majority, and PGF

Suppose we have two sub-systems, A_1, A_2. Lemma **??** tells us that the mean time between a joint failure of two independent systems is much higher than the MTBF of each individual system. In other words, with high probability at least one of these systems does not fail. But how can we know which one is the non-failing system? To which sub-system should we listen? Deciding how to combine the outputs of several sub-systems is called 'fusion'.

In some cases, each sub-system has an additional fault-monitoring component. For example, communication channels detect potential corrupted messages by sending checksum messages. In such cases, whenever A_1 and A_2 disagree, and one of them self-detect a potential failure, we should listen to the other sub-system. We will rely on this technique for tackling some identifiable failures. However, this additional fault-monitoring information is not always available.

When working with deep learning tools, a tempting approach is to let the machine learn by itself how to combine the inputs of all sources. This is not always possible due to the lack of failure data, for example, when we want to use redundancy in order to solve power supply issues in one of the computers. But, even when tackling perception problems, an end-to-end fusion approach is not always practical. One such example is outlined in Appendix **??**, where we describe the *shortcut learning problem*.

The next approach is called *worst-case fusion*. For concreteness, consider the simple decision of whether we have a car in front of us for which we are at a dangerous state according to RSS and we need to apply braking (we will refer to it below as "RSS violation"). The worst-case fusion approach is as follows: if either A_1 or A_2 states that we need to brake, then we shall brake, and only if both A_1 and A_2 tell us that braking is not required we will not brake. Given Lemma **??**, we conclude that this "worst-case fusion" approach will have a high probability of not missing a required braking (MTBF of mis-detecting an RSS violation is at least $180T_1T_2$ where T_1, T_2 are the MTBFs of mis-detection by each individual system). However, it is not hard to see that the probability of an unnecessary braking (a "false alarm") increases by a factor of 2.

To deal with both mis-detections and false alarms we can require three sub-systems, A_1, A_2, A_3, and follow a *majority vote*. Concretely, let's say that the three systems are based on camera, radar, and a lidar. If at least 2-out-of-3 (2oo3) systems think we are at an RSS violation, then we should brake. Otherwise, we shouldn't brake. What is the probability that the majority vote fails? Let $\epsilon(A_i)$ be a Boolean variable indicating whether system A_i fails. The majority vote would fail if A_1 agrees with A_2 and both fail, or if A_1 agrees with A_3 and both fail, or if A_2 agrees with A_3 and both fail. This implies that,

$$\mathbb{P}[\epsilon(\text{majority}(A_1, A_2, A_3))] \leq \mathbb{P}[\epsilon(A_1) \wedge \epsilon(A_2)] + \mathbb{P}[\epsilon(A_1) \wedge \epsilon(A_3)] + \mathbb{P}[\epsilon(A_2) \wedge \epsilon(A_3)]$$

Similarly to Lemma **??** (and with the same assumptions), it is easy to verify that this implies an MTBF of at least $\frac{60\, T_1 T_2 T_3}{\max\{T_1, T_2, T_3\}}$, where T_i is the MTBF of system A_i. That is, we again obtain a quadratic improvement in the MTBF of the fused system. And, unlike the worst-case fusion approach, we now get a quadratic improvement for both mis-detections and false-alarms.

The notion of "majority over three sub-systems" is well-defined only for binary decisions. However, many of the decisions we need to make while driving are not binary decisions. Most notably, the geometry of the lane we are driving in is not a binary decision, and this geometry has profound implications on RSS decisions—see Figure **??**.

We therefore propose a generalization of the majority vote, which we call the Primary-Guardian-Fallback (PGF) fusion system. Consider three systems, P, G, F, standing for Primary, Guardian, and Fallback. The Primary and Fallback systems are arbitrary systems (not necessarily with binary outcomes). The Guardian system is a binary system (its output is a Boolean value), whose goal is to output 1 if the Primary system is better than the Fallback system and 0 otherwise. The PGF fusion system is:

$$\phi(P, G, F) = \begin{cases} P & \text{if } G = 1 \\ F & \text{if } G = 0 \end{cases}$$

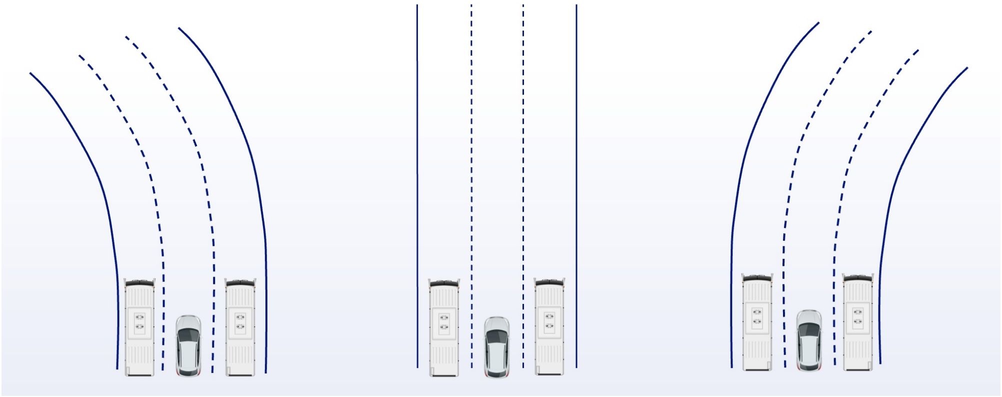

Figure 2: Majority vote is not well-defined for the geometry of the lanes system: according to RSS rules, we must continue left/straight/right depending on the geometry on the left/center/right pictures above.

That is, the fused system follows the Primary or the Fallback systems depending on the output of the Guardian system. We note that recently, [**?**] discussed asymmetric fusion techniques and proposed a similar idea to PGF, where the three systems are called Doer/Checker/Fallback.

We next analyze the probability that the PGF fusing system will err.

$$\begin{aligned}&\mathbb{P}[\epsilon(\phi(P,G,F))] = \mathbb{P}[\epsilon(P) \,\wedge\, G] + \mathbb{P}[\epsilon(F) \wedge !G]\\&= \mathbb{P}[\epsilon(P) \,\wedge\, \epsilon(F) \,\wedge\, G] + \mathbb{P}[\epsilon(P) \wedge !\epsilon(F) \,\wedge\, G] + \mathbb{P}[\epsilon(P) \,\wedge\, \epsilon(F) \wedge !G] + \mathbb{P}[!\epsilon(P) \,\wedge\, \epsilon(F) \wedge !G]\\&= \mathbb{P}[\epsilon(P) \,\wedge\, \epsilon(F)] + \mathbb{P}[\epsilon(P) \wedge !\epsilon(F) \,\wedge\, G] + \mathbb{P}[!\epsilon(P) \,\wedge\, \epsilon(F) \wedge !G]\end{aligned}$$

The error is decomposed into three summands:

1. This summand indicates that both the Primary and Fallback systems fail.
2. This summand indicates that the Primary system fails, the Fallback system doesn't fail, and the Guardian system fails (because it should prefer the Fallback system in such cases).
3. This summand indicates that the Primary system doesn't fail, the Fallback system fails, and the Guardian system also fails (because it should prefer the Primary system in such cases).

In all three cases, at least two of the sub-systems fail. Therefore, similarly to Lemma **??** (and with the same assumptions), it is easy to verify that this implies an MTBF of at least $\frac{60\,T_p T_g T_f}{\max\{T_p,T_g,T_f\}}$, where T_p, T_g, T_f are the MTBFs of the P,G,F systems, respectively. We again obtain a quadratic improvement in the MTBF of the fused system, in the same manner as we got for the majority vote.

In fact, PGF can express the majority vote as a special case, as stated in the following:

Lemma 2 *Consider a binary decision, when we have three sub-systems, each giving the binary prediction b_1, b_2, b_3. Consider PGF with Primary being b_1, Guardian being $1[b_1 \neq b_2]$, and Fallback being b_3. Then, this PGF setting is equivalent to the 2oo3 majority vote.*

Proof If $b_1 = b_2$, then PGF will output b_1 which must also be the majority vote. If $b_1 \neq b_2$, then PGF outputs b_3. It must hold that either $b_3 = b_1$ or $b_3 = b_2$. In both cases, b_3 equals to the majority vote. ■

While PGF can express the majority vote as a special case, it can do much more, and we show its expressivity power for elements of our SDS in Section **??**.

4 Identified Failures

Our SDS is designed to identify a large set of potential failures. Each such potential failure comes with a Minimal Risk Maneuver (MRM). By identifying the failure and reacting with an appropriate MRM we obtain a robust SDS that eliminates unreasonable risks according to the FuSa standard. That is, the system behaves safely even when faults occur.

FuSa defines a risk classification scheme for potential hazards called Automotive Safety Integrity Level (ASIL), which includes the severity of the scenario (4 levels, ranging from "no injuries" to "life threatening") and its exposure (5 levels, from "incredibly unlikely" to "high probability") [2] This yields 5 ASIL levels: ASIL D, ASIL C, ASIL B, ASIL A, and QM. ASIL D is the highest risk level. An example of a potential dangerous hazard that warrants the ASIL D level is complete loss of the braking system. The ASIL level affects the required MTBF of the component, where the standard sets the required MTBF of ASIL D to be at least 10^8 hours.

How can we reach such a high MTBF? One approach that the standard suggests is to use formal verification tools or semi-formal modeling languages in the design. This is not always applicable for complex software or hardware. An alternative approach is to rely on redundancy. In particular, designing two independent components, A_1, A_2, that perform the same task and a Listener entity, L. In the nominal case, the Listener receives two identical messages from the A_1 and A_2. When the Listener receives different messages, it knows that there is a fault. According to Lemma **??**, the probability that the Listener will get the same message from A_1 and A_2 while both of them fail is much lower than the probability that each individual system fails. For example, our SDS contains two computers, and safety critical calculations are performed on the two computers in parallel. So, if the MTBF of each individual computer to fail due to hardware or software issues is at least 10^3, then the MTBF that both computers will fail at the same time is greater than 10^8. This approach is often referred to as ASIL B(D).

Remark 1 *The Listener itself needs to comply with an ASIL D rating, otherwise it breaks the integrity of the above design. This can be achieved in an analogue way - we can either design the Listener to be a simple enough component and make it comply with ASIL D using formal or semi-formal verification techniques, or we could replicate the Listener itself and send the messages from A_1, A_2 to two Listeners, rolling-up the ASIL D requirements to a higher entity.*

The above approach enables us to identify that we have a failure, with a very high MTBF. But, we also would like to understand the source of the failure in order to respond with an appropriate MRM. For example, in the case of a computer failure, we would like to know which of the two computers failed. Without this knowledge, the only MRM we can perform is an immediate stop. With the knowledge of which computer failed we can continue driving while comfortably changing lanes until we can stop safely on the shoulder.

We employ two approaches for knowing which of the two sources failed. One approach is embedding self-diagnostics in each individual source, so that when the two sources disagree, and one of them reports an issue, we listen to the one who didn't report an issue. The second approach is applicable in cases when we have three views of the same phenomena, and by comparing the three views we could understand if one of the views fails from which we can infer a failure in some system element. This is particularly useful for fault-monitoring sensor failures. For example, cameras, radars, and lidars all detect objects in the scene. So, if we observe that the detections of one of these sensors is significantly different than the other two, we can infer that this sensor is out of nominal behavior.

The subsections below list the failures we aim to identify by grouping them into the following:

1. Computer failures: these include issues with the power supplier, hardware failures, and software failures.

2. Communication failures: the SDS involves communication channels between different computing elements and between the SDS computers and the vehicle actuators. Failures can be corrupted messages, lost messages, and messages that didn't arrive sufficiently fast.

3. Vehicle failures: these includes for example failures in the braking/steering systems or a flat tire.

4. Sensor failures: for example, one of the cameras is blocked or broken.

[2] A third element that affects the ASIL rating is called controllability, which is the ability of a human driver to control the car when faults occur. This is less relevant in our context as we deal with fully autonomous driving, and therefore there is no human driver who can take over immediately when faults occur.

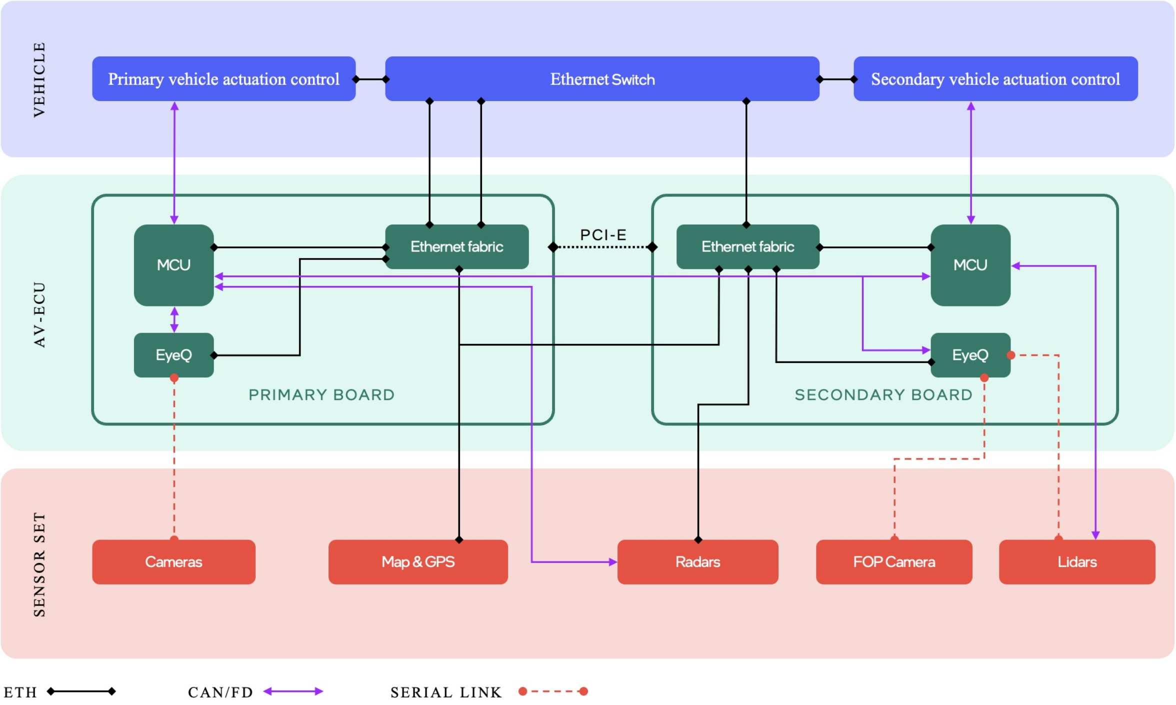

Figure 3: A schematic illustration of our SDS.

We rely on several sensor modalities, two computers each with its own power supply, two different communication channels between critical components, and redundant vehicle actuators. A schematic view of the system is given in Figure **??**.

4.1 Potential Computer Failures

Each of the two computing boards contains a Microcontroller unit (MCU) as well as one or more EyeQ chips. The MCU is a simpler computing element with a simpler software structure, and therefore we often use it as a Listener element to more complex sources (such as the EyeQ chip). The EyeQ chip is a much more sophisticated compute element, involving several types of accelerators (most notably, for high utilization implementation of deep neural networks and computer vision algorithms). The potential computer failures that we monitor are listed below.

- **Correct operating state of the Computing unit**: dedicated SW and HW safety elements run BIST (Built In Self Tests), monitor and handle the unit faults, and ensure correct configuration of the unit registers.
- **Fault in power supply**: The MCU compute unit is divided into two electrically independent and physically isolated domains, each unit is monitored for over/under voltage.
- **Computing unit availability**: A Windowed Challenger watchdog is constantly running on each safety element. In addition, heartbeats are being sent between different computing units to ensure their availability.
- **ECU temperature management**: Safety elements monitor the correct configuration of thermal sensors and ensure that the computing unit operates within its intended operating range.
- **Lockstep meachnism**: A lockstep mechanism is implemented on the computing unit safety host using diverse and separated cores, and critical signals are encoded and checked in two redundant components.

- **Corrupted memory**: The main measures to ensure non-corrupt memory are an Error Correction Circuit (ECC) mechanism, a lockstep mode that reduce the risk of memory corruption in the Direct Memory Access (DMA), and a Memory Protection Unit (MPU) which protects memory by allocating different sections to different software components.

4.2 Potential Communication Failures

The SDS is designed with dual communication channels of two different types for all critical communications. The safety critical communication paths are:

- Control commands from the computer (ECU) to the vehicle using the MCU of each of the two boards. For each MCU we communicate with the vehicle actuators through two channels: CAN and Ethernet.
- Car signals coming from the vehicle to the ECU are also communicated over both CAN and Ethernet.
- Board-to-board communication is through two different channels: PCIe and Ethernet.
- MCU-to-MCU communication is through two different channels: CAN and Ethernet.
- Data from sensors (including the map, we consider the map as an additional sensor) arrive with a single communication channel, since we have redundant sensors and so the protection against failures of communication between the ECU and the sensors happen by the same manner as we protect against sensor failures.

For each individual communication channel, we perform the following self-diagnostic to verify the integrity of the message:

- We apply Cyclic Redundancy Check (CRC) as an error detection code to detect accidental bit changes in the message.
- Time synch: each message comes with a time stamp and we check if the time stamp matches what we expect. In particular, for some components we expect a “heartbeat”, namely, the sender should send a message at a constant frequency indicating that it is “alive”. We also use time stamp of messages to make sure that they are not old.

4.3 Potential Vehicle Failures

The vehicle platform is responsible for monitoring the health of critical elements such as the braking and steering systems. The SDS monitors the following status indicators from different vehicle modules and performs the appropriate MRM as follows:

- **Immediate stop**: brakes system diagnostics, steering diagnostics, vehicle collision detection.
- **Stop on shoulder if possible**: vehicle speed indication is faulty, steering wheel angle sensor, IMU fault, error in control unit, engine diagnostics (including battery status), transmission gear status, disruptive cleaning signal, headlights or blinking diagnostics, suspension system fault, flat tire indication, wipers status, friction detection, open door / hood /trunk status, seat belt (per seat), ABS status, ESP status.

4.4 Potential Sensor Failures

Each sensor has its own self-diagnostic module that reports to the SDS computer. In addition, the computer monitors the performance level of each sensor in order to detect systematic degradation in the sensor quality. There are various reasons for performance degradation such as blockage due to dirt or dust (for cameras and lidars), multi-path interference due to clutter (for radars), lighting conditions (for cameras), reflective/dark surfaces (for lidars), and more. While such conditions might cause degradation in the detection quality, it is not always the case, and therefore trying to detect each potential environmental condition without the actual effect on the system performance is maybe not the best approach. We therefore take another route and rely on a comparison method instead of an absolute one.

In particular, we aim to detect out of nominal relative performance with respect to (at least) two sensors. For static objects the SDS compares camera detections of various landmarks (traffic lights and traffic signs, poles, lane marks, road edges and road surface) to their expected position according to the HD map. Similarly, we can compare the positions of these landmarks as detected by the camera to their positions according to the lidar. The advantage of this approach is that it is strongly correlated with degradation in the sensor quality that affects the SDS performance. In addition, since we have three sensor modalities here (camera, map, and lidar), once we determine that there is a significant difference in the detections, we can also know which one of them is degraded (according to the majority vote).

For dynamic objects the SDS compares the camera objects to lidar and radar detections, and can again determine if one of them is degraded.

5 Perception System Design

In the previous section we discussed how our SDS is properly designed to identify potential failures and mitigate them. In Section **??** we also discussed our methodology for handling reproducible failures of the system. The remaining failures are *black swans*, and in this section we discuss the system design so that black swans will always involve failure of at least two sub-systems.

We consider five critical elements of our SDS: RSS response with respect to physical objects, ego-motion (determining the movement of the self-driving car itself over time), lanes semantics (where are the lanes, including relevant traffic rules), traffic lights, and view range.

5.1 Physical Objects and Ego Motion

Let's start with the RSS response with respect to physical objects. We need to determine if we are in an RSS violation with respect to some physical object. This is a binary decision. Suppose we have three sub-systems for physical object detection: camera, radar, and lidar. The obvious option is to follow the 2oo3 majority vote. The disadvantage of this simplistic approach is that most of the decisions we have to make are not binary: for example, given a car in front of us, we are not braking only when we reach an RSS violation, but we sometimes slow down in order to avoid having an RSS violation to begin with. Furthermore, this 2oo3 fusion approach doesn't allow us to enjoy the benefits of each individual sensor. Cameras are best in semantic understanding (is it a car, a bike, or a pedestrian), lidar is the best in 3D position, and radar is the best in velocity estimation. The PGF architecture allows us to be more nuanced. Here is another option, in which only the Guardian system is "binarized" but the Primary and Fallback systems are continuous.

- The Primary system is a Compound AI System (CAIS). This CAIS starts with a perception system which is built by separate training of several sub-systems, followed by a trainable graph-neural-network for fusing the outputs of the sub-systems as well as traditional computer vision and filtering techniques. The output of the perception system is "sensing-state", which is fed into an RSS-based driving policy module. The driving policy module outputs driving commands.

- The Guardian system first checks if the command of the Primary system adheres with RSS according to each sensor *individually*. In case it adheres with at least 2-out-of-3 sensors, the Guardian system approves the Primary system. Otherwise, it rejects the Primary system and picks the Fallback system.

- The Fallback system is as follows: if according to 2oo3 sensors we are at RSS violation, apply minimal braking. Else, apply the output of an end-to-end policy network, while limiting its output to only allow mild braking.

Let's analyze the error of this PGF setting, starting with the case in which the Primary system errs and the Guardian system doesn't catch it. This means that 2oo3 sensors agree with the Primary system and yet it still errs. This type of error of PGF is similar to the vanilla majority vote. Next, consider the case in which the Guardian rejects the Primary system. It means that 2oo3 sensors thought that the Primary system is bad. If these 2 sensors think we are at an RSS violation, we will apply minimal braking. This will be an error if we don't actually need minimal braking. But again, this type of error of PGF is similar to the vanilla majority vote. Last, if there are no 2oo3 sensors that think we are at an

RSS violation, then we will apply the output of the end-to-end policy network, while only allowing comfort braking. If strong braking is required in reality, then again we err like the majority vote (because there were no 2oo3 sensors that thought we must brake). All in all, this PGF system will make a severe error with the same likelihood as a 2oo3 system, but it has the benefit that most of the time it can benefit from the best of each of the sensors, and even in edge cases, it listens to some extent to an end-to-end policy network.

The above applies to all types of RSS proper responses (longitudinal response, lateral response, and multi-geometry response). It assumes that the underlying lane semantics is valid. PGF for ego-motion can be derived in the same manner (camera/radar/lidar individual sub-systems), except that the camera sub-system requires additional vehicle sensors (most importantly, estimation of the vehicle's speed from a "wheel ticks" sensor).

5.2 Lane Semantic System

We next tackle the lanes semantic system. Here, lidars and radars have a small contribution, and the main sensors are cameras as well as an HD map. We propose the following PGF architecture.

- The Primary system is based on a deep network we call "RoadX". The input to this network is both the map and the camera data, and the outputs are the geometry of the lanes and their semantic meaning. In other words, the deep network performs a low-level fusion between the map and the camera. Given the output of this low-level fusion, the Primary system drives the car according to our driving policy (while taking into account the objects, as described previously).

- The Fallback system is to drive the car according to an end-to-end deep network, that takes camera data as input, and output the driving commands. We require that the end-to-end network also outputs the geometry of the most relevant lane according to which the car should drive.

- The Guardian system is based on a discriminative deep network that we call "lane validator". Its input is a proposed lane and driving commands how to drive in this lane, projected onto the camera image data, as well as physical boundaries detected by lidars and imaging radars, also projected as additional channels of the image space. The output of the network is a binary decision, whether to approve or disapprove the lane/trajectory. The Guardian system applies the discriminative network both on the output of the Primary and Fallback systems to get two bits b_p, b_f. If $b_p = 1$ we apply the Primary system. If $b_p = -1$ and $b_f = 1$ we apply the Fallback system. If $b_p = b_f = -1$ we drive according to the geometry of the Fallback system, while in parallel apply MRM braking.

To analyze the potential failures of this PGF system, let's start with the case in which in reality the map is valid. In this case, we expect that the Primary system will agree with the map. Suppose that this happens, and the Guardian approves the Primary system, then we won't have an error. If the Primary system suggests a different lane, and the Guardian approves it, but this different lane is wrong, we have two different errors - both the Primary system failed to approve the map and the Guardian system failed to recognize the error of the alternative lane. If the Primary system approves the map, and the Guardian wrongly rejected it, we go to the end-to-end trajectory. If it is wrong, then again we had two different errors - disapproving the map by the Guardian and a wrong trajectory by the end-to-end network. Next, suppose that the map is outdated (even though the map is designed to be updated frequently). If the Primary system approves it, and the Guardian approves it, we again have two different networks that are failing ("generative and discriminative"). If the Primary proposes an alternative lane, the Guardian approves it, but it is wrong, then again we have a failure of two different networks. Finally, if the Guardian rejects the Primary system, and the Fallback is wrong, then either both the Fallback and Primary are wrong, or both the Fallback and Guardian are wrong. In all failure cases, we have two rather different systems that err.

5.3 Traffic Lights

For traffic-light recognition, we also propose three sub-systems. The primary one is a compound AI system which searches for all traffic lights in the image data (while also using prior locations from the map), detect the colors and association to lanes, and decide how to react by our driving policy. The Guardian system is based on a "stop or go" discriminative network, aimed at giving a bottom line decision. The Fallback system is based on our end-to-end deep network, that takes camera data as input, and output the driving commands.

5.4 View Range

Finally, the last essential component of the system according to RSS is view range. View range measures what we don't see. In particular, the system adjusts its speed if there is no sufficient view range ahead. Similarly, the system will drive more carefully at an unprotected turn if there is another merging or crossing lane with a limited view range. With a 360 degree surround sensing of three sensor modalities we could tackle view range in the same manner as we tackled physical objects. In some configurations, however, we do not have three sensor modalities in all sectors, and then we can adjust our speed due to view range by following the worst-out-of-2 sensors. Since braking for view range is always a mild braking, the risk from increasing unnecessary braking due to false alarms of lack of view range is not severe.

To summarize, each critical component of our SDS relies on a PGF fusion, where the choice of the Primary-Guardian-Fallback systems is done in a manner that will ensure that a system failure always involve a failure of at least two sub-systems.

6 Conclusion

The development of safe and reliable Self-Driving Systems (SDSs) is a multi-faceted challenge that requires a robust and systematic approach. Through this paper, we have outlined a safety architecture built on foundational principles and standards like RSS, FuSa, and SOTIF. These frameworks collectively address the different dimensions of risk: avoiding collisions, mitigating identifiable and reproducible failures, and managing the uncertainty posed by black swans.

A cornerstone of the architecture is the integration of redundancy across hardware, software, and AI processes. Redundancy not only enhances resilience but also contributes to failure identification and mitigation, ensuring that the SDS operates within acceptable risk boundaries. By adhering to the principles of transparency and continuous improvement, we address not just the immediate performance of the SDS but its long-term evolution in response to emerging challenges and discoveries.

A The Shortcut Learning Problem

We describe a simple synthetic example whose goal is to demonstrate how end-to-end Stochastic Gradient Descent (SGD) training might have an extremely slow convergence, relative to a high-level fusion.

Consider the following problem: the input is $x \in \{\pm 1\}^5$ and the output is $y \in \{\pm 1\}$. The joint distribution over $x \times y$ is as follows. First we sample $y \sim B(1/2)$ (that is, $\mathbb{P}[y = 1] = \mathbb{P}[y = -1] = 1/2$). Then we sample three hidden variables r_1, r_2, r_3 i.i.d. according to $B(1 - \epsilon)$, namely $\mathbb{P}[r_i = 1] = 1 - \epsilon, \mathbb{P}[r_i = -1] = \epsilon$. Then we sample x_4, x_5 i.i.d. according to $B(1/2)$. Finally, we set $x_1 = yr_1, x_2 = yr_2, x_3 = yr_3x_4x_5$.

We consider learning data from the above distribution by a standard one hidden layer ResNet network as follows:

$$o(x) = u^\top x + w^\top [Wx + b]_+$$

where $W \in \mathbb{R}^{h,5}, b \in \mathbb{R}^h, w \in \mathbb{R}^h$ are the weights of the one-hidden layer network (with h neurons in the hidden layer), and $u \in \mathbb{R}^5$ is the weight vector of the ResNet skip connection. We also denote $\theta = (u, w, W, b)$.

We refer to $o(x)$ as a "logit" that defines the estimated probability, for $a \in \{\pm 1\}$, $p[\hat{y}(x) = a] = 1/(1 + \exp(-a\, o(x)))$. The log-loss becomes:

$$L(\theta) = \mathop{\mathbb{E}}_{x}[-\log(p[\hat{y}(x) = y(x)])] = \mathop{\mathbb{E}}_{x}[\log(1 + \exp(-y\, o(x)))]$$

The zero-one loss is

$$L_{01}(\theta) = \mathop{\mathbb{E}}_{x} 1[p[\hat{y}(x) = y(x)] > 1/2] = \mathop{\mathbb{E}}_{x} 1[y(x)o(x) \leq 0]$$

We start with showing that the optimal solution has error of $O(\epsilon^2)$.

Lemma 3 *If $h \geq 8$, then there exists θ such that $L_{01}(\theta) \leq 3\epsilon^2$.*

Proof For simplicity assume that $h = 8$. Let $i(x_3, x_4, x_5) = 4\frac{x_3+1}{2} + 2\frac{x_4+1}{2} + \frac{x_5+1}{2} \in \{0, 1, \ldots, 7\}$ be the decimal value corresponding to the bit sequence (x_3, x_4, x_5). For every possible x_3, x_4, x_5 out of the 8 options, set row $i(x_3, x_4, x_5)$ of W to be $[0, 0, x_3, x_4, x_5]$ and the corresponding element of b to be -2. It follows that $[Wx + b]_+$ is all zeros except 1 in the index $i(x_3, x_4, x_5)$. Next, set the $i(x_3, x_4, x_5)$'th element of w to be $x_3x_4x_5$. Finally, set $u = [1, 1, 0, 0, 0]$. All of this implies that for these weights,

$$o(x) = x_1 + x_2 + x_3x_4x_5 \ .$$

Therefore,

$$yo(x) = yx_1 + yx_2 + yx_3x_4x_5 = r_1 + r_2 + r_3 \ .$$

The probability that $yo(x) \leq 0$ is

$$3\epsilon^2(1 - \epsilon) + \epsilon^3 \leq 3\epsilon^2$$

and the claim follows. ■

Our main argument regards the dynamic of Gradient Descent (and Stochastic Gradient Descent) when applied to this problem. Let T be the iteration complexity of reaching error of $O(\epsilon)$ had we always zero x_1 and x_2. We argue that when $\epsilon \ll 1$, GD converges quickly to a sub-optimal solution, $\hat{\theta}$, for which $L_{01}(\hat{\theta})$ is order of ϵ, and it takes $\Omega(T/\epsilon)$ iterations to converge to the optimal solution of $O(\epsilon^2)$.

A full proof is too technical for this manuscript. Here is a sketch of the main idea. Fix some (x, y) and denote $\delta_\ell = \frac{y}{1+\exp(y\,o(x))}$. Then, a SGD update of u w.r.t. this (x, y) is $u = u + \eta\,\delta_\ell\,x$. As long as $y\,o(x)$ is not a large positive number, we effectively sum $y\,x$ with positive weights. Since x_3, x_4, x_5 has zero correlation with y, the last three elements of u are 0 in expectation, while for the first 2 elements, the value of yx_1 and yx_2 is almost always 1, so u will converge to a vector for which the last 3 elements are close to zero while the first 2 are large positive numbers. This will happen quickly, and gives a model with error of $\approx \epsilon$.

As to the other components, let $o_l = Wx + b, o_+ = [o_l]_+$, then w is updated by $w = w + \eta\,\delta_\ell\,o_+$, and W is updated by $W = W + \eta\,\delta_\ell\,(w * o_+)x^\top$, and b is updated by $b = b + \eta\,\delta_\ell\,(w * o_+)$, where $*$ denotes element wise multiplication. Now, with a random initialization of w, W, b, the h-dimensional vector $(w * o_+)$ looks like a random vector. So, it will take some time to converge, and the vector u will converge as explained above much faster to $u \approx (B, B, 0, 0, 0)$.

The second phase of learning is when $u \approx (B, B, 0, 0, 0)$ for some large positive B. From this point on, with probability of $1 - O(\epsilon)$, the value of $\frac{1}{1+\exp(y\,o(x))}$ is close to zero. We only learn something when $r_1 = -1$ or $r_2 = -1$, which happens once in $O(1/\epsilon)$ iterations on average. So, in the second phase, the learning of the optimal solution becomes factor $O(1/\epsilon)$ slower relatively to a case in which x_1, x_2 are absent from the problem. So, the convergence will happen only after T/ϵ iterations.

Finally, we can make T as large as we want by enlarging x_3, x_4, x_5 to $x_3, x_4, \ldots, x_n$, and we can make ϵ very small in the context of self-driving problems.

References

[1] SIG Automotive. Automotive spice®. *Prozess Assessment Model*, 2:5, 2010.

[2] ChinaITS. Technical requirement of safety assurance of av decision making 0116-2019. ChinaITS, 2019.

[3] IEEE. Ieee p2846a standard for assumptions in safety-related models for automated driving systems. IEEE Standard Association, 2022.

[4] ISO. Iso 26262: Road vehicles – functional safety. International Standard ISO/FDIS 26262, 2011. Second edition, 2018.

[5] ISO. Iso 4804: Road vehicles — safety and cybersecurity for automated driving systems — design, verification and validation. International Standard ISO/FDIS 21448, 2020.

[6] ISO. Iso 21448: Safety of the intended functionality. International Standard ISO/FDIS 21448, 2022.

[7] NASSIM Nicholas Taleb. The black swan: The impact of the highly improbable. *Victoria*, 250:595–7955, 2015.

[8] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv:1708.06374*, 2017.

[9] safety The Autonomous working group and architecture. Safe automated driving: Requirements and architectures. The AUTONOMOUS, 2024.