基于ml的自动驾驶汽车故障注入:以贝叶斯故障注入为例

Saurabh Jha*、Subho S. Banerjee*、Timothy Tsai†、Siva K. S. Hari†、Michael B. Sullivan†、Zbigniew T. Kalbarczyk*、Stephen W. Keckler*和Ravishankar K. Iyer*
*伊利诺伊大学厄巴纳-香槟分校，厄巴纳-香槟，IL 61801，美国。†NVIDIA公司，Santa Clara, CA 94086, USA。

*摘要:自动*驾驶汽车(AVs)的安全性和弹性是人们非常关注的问题，例如几起头条新闻的事故。虽然目前的AV开发涉及验证、验证和测试，但在现实驾驶场景中意外故障下AV系统的端到端评估在很大程度上尚未得到探索。本文介绍了DriveFI，一种基于机器学习的故障注入引擎，可以挖掘最大程度影响AV安全的情况和故障，并在两个工业级AV技术堆栈(来自NVIDIA和百度)上进行了演示。例如，DriveFI在不到4小时的时间内发现了561个安全关键故障。相比之下，在数周内执行的随机注入实验无法发现任何安全关键故障。

*索引术语-自动驾驶*车辆;故障注入

I. 介绍

自动驾驶汽车(AVs)是一种复杂的系统，它使用人工智能(AI)和机器学习(ML)来整合机械、电子和计算技术，以做出实时驾驶决策。AI使AVs能够在复杂的环境中导航，同时保持安全范围[1]，[2]，由车载传感器(如摄像头，LiDAR，RADAR)[3] -[5]不断测量和量化。显然，AVs的安全性和弹性是非常值得关注的，例如几起头条新闻的AV碰撞[6]，[7]，以及之前在道路测试中表征AV弹性的工作[8]。因此，迫切需要对AV技术进行全面评估。

今天的AV开发包括验证[9]-[12]、确认[13]和测试[14]，[15]以及整个生命周期中其他形式的评估。然而，在现实的执行环境中对这些系统进行评估，特别是由于随机故障的发生，一直具有挑战性。故障注入(Fault injection, FI)是一种成熟的方法，用于测试计算和网络物理系统[16]在故障下的弹性和错误处理能力。基于fi的AVs评估提出了一个独特的挑战，这不仅是因为AV的复杂性，还因为AI在自由流动的操作环境中的中心地位[17]。此外，AVs代表了软件[18]和硬件技术[19]的复杂集成，这些技术已被证明容易受到硬件和软件错误的影响(例如，SEUs [20]，[21]，*Heisenbugs*[22])。未来不断增加的代码复杂度和缩小特征尺寸的趋势只会加剧这个问题。

本文介绍了*DriveFI*，这是一种用于AVs的智能FI框架，通过识别可能导致碰撞和事故的危险情况来解决上述挑战。DriveFI包括(a)一个FI引擎，它可以修改自动驾驶系统(ADS)的软件和硬件状态来模拟故障的发生，以及(b)一个基于ml的故障选择引擎，我们称之为*贝叶斯故障注入*，它可以找到最可能导致违反安全条件的情况和故障。相比之下，传统的FI技术[16]往往不关注安全违规，并且在实践中表现率低，需要大量的测试时间[23]，[24]。请注意，给定故障模型，DriveFI也可以执行随机FI以获得基线。

的贡献。DriveFI的贝叶斯FI框架能够通过对故障下ADS行为的因果和反事实推理来发现安全关键情况和故障。它通过(a)以车辆运动学和AV架构的形式*整合领域知识*，(b)基于横向和纵向停车距离对*安全性*进行*建模*，以及(c)使用*实际故障模型*来模拟软错误和软件错误。将(a)、(b)和(c)项整合到*贝叶斯网络*(BN)中。bp网络提供了一种有利的形式化方法，可以用可解释的模型对AV系统组件之间的故障传播进行建模。该模型与故障注入结果可用于AVs的安全性设计和评估。此外，bp网络支持快速概率推理，这使得DriveFI能够快速发现安全关键故障。贝叶斯FI框架可以扩展到其他安全关键系统(例如手术机器人)。该框架需要对安全约束和系统软件架构进行规范，以建模系统子组件之间的因果关系。我们在两个工业级4级ADS[25]上展示了这种方法的功能和通用性:DriveAV [3] (NVIDIA的专有ADS)和Apollo 3.0[4](百度的开源ADS)。

结果。我们使用了三种故障模型:(a)非ecc保护的处理器结构中的随机和均匀故障，(b) ADS软件模块输出中的随机和均匀故障(最小或最大值损坏)，以及(c) ADS模块输出被贝叶斯FI损坏的故障。我们注入活动的主要结果包括:

- 使用故障模型(b)，我们编制了一个98,400个故障的列表。对模拟驾驶场景中所有98,400个故障的全面评估需要615天。相比之下，我们的贝叶斯FI能够在不到4小时的时间内找到561个最大程度影响自动驾驶安全性的故障。因此，贝叶斯FI实现了3690倍的加速。贝叶斯FI发现的两种情况在§II-D中描述;其中一个尤其模仿了特斯拉汽车撞车事件[6]。
- 贝叶斯FI能够发现导致安全隐患的关键故障和场景。(a)在561个已确定的故障中，460个表现为安全隐患。(b)这460个故障被发现与68个安全关键场景有关[1](out

[1]A scene is represented by one camera frame.

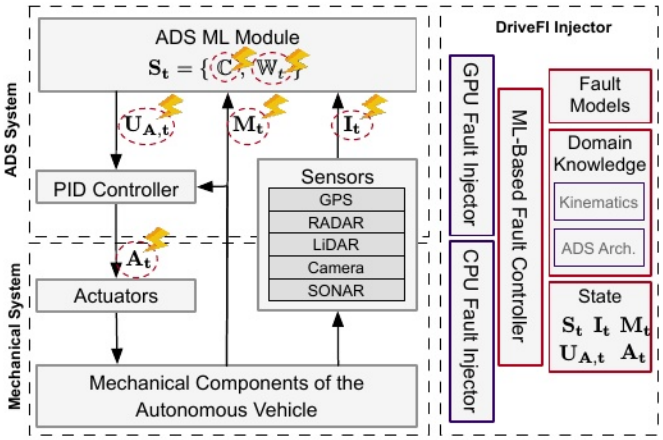


图1:自动驾驶和机械系统的高级概述，以及与DriveFI的互动。

的7200个场景)。

· 相比之下，几周的5000次随机FI实验没有发现任何安全隐患。只有1.93%的单比特注入导致了导致驱动错误的静默数据损坏(SDC)。ADS从所有这些错误中恢复，没有任何安全违规。在7.35%的FIs中，发生了内核恐慌和挂起。预计此类故障的恢复是可以完成的

当前自动驾驶中存在的备份/冗余系统。我们认为，贝叶斯FI对危急情况的挖掘将具有比我们这里的断层注入更广泛的适用性。将一系列故障注入实验的结果结合起来，创建一个情景库，将有助于制造商制定自动驾驶测试和安全驾驶的规则和条件。

正确看待DriveFI。早期工作使用系统理论方法研究自动驾驶汽车的安全性[26]，[27]。最近的研究集中在ADS组成模块的弹性上(见§IV)，例如[24]，[28]-[30]。另一项研究[31]、[32]利用FI研究自动驾驶汽车的传感器相关弹性。与DriveFI相比，之前的方法都没有考虑到现代端到端人工智能驱动系统的弹性，这些系统使用工业级ads来挖掘导致安全隐患的故障。

II. 方法概述

本节概述了本文所提倡的人工智能驱动的贝叶斯FI方法。我们现在介绍本文其余部分中使用的形式主义。

A. 自动驾驶系统

图1展示了自动驾驶汽车(从此以后也被称为*Ego Vehicle*, EV)的基本控制架构。它由机械部件和执行器组成，由*ADS控制*，它代表自动驾驶的计算(硬件和软件)组件。在时间t的每一个瞬间，ADS系统从传感器I_t(例如，摄像头，激光雷达，GPS)输入，从机械部件(例如，速度v_t，加速度a_t))获取惯性测量M_t，并推断驱动命令A_t(例如，油门ζ，制动b，转向角ϕ)。为了清晰起见，我们进一步将ADS细分为两个组件:(a) ML模块(负责感知和规划)，将I_t和M_t作为输入并产生原始驱动命令$U_{A,t}$，以及(b) PID控制器[33]，负责平滑输出$U_{A,t}$以产生

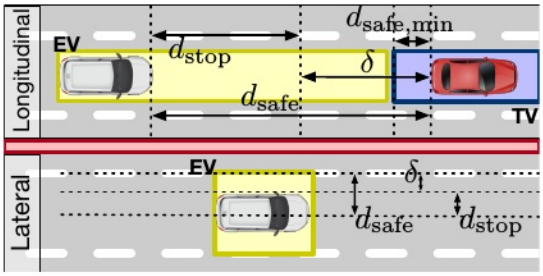


图2:汽车横向和纵向运动的dstop、dsafe和δ的定义。非自动驾驶汽车被标记为*目标车辆*(TV)。

A_t。PID控制器确保自动驾驶不会在A_t中发生任何突然变化。ADS机器学习模块具有瞬时状态S_t，该状态由配置参数C(例如，感知输入相机数据的神经网络权重)和*世界模型*W_t组成，该模型维护并跟踪ADS感知到的所有静态对象(例如，车道标记)和动态对象(例如，其他车辆)的轨迹。

B. 安全

我们根据自动驾驶车辆行驶的纵向(即车辆的运动方向)和横向(即垂直于车辆运动方向)笛卡尔距离定义了自动驾驶车辆的瞬时安全标准(见图2)。这些标准形成了基于避免碰撞的“原始”安全定义，可以扩展到其他安全概念，例如使用交通规则。本文不考虑扩展的安全概念，因为它们可以根据其应用的地理区域的规律进行细致入微的分析。

定义1。*停车距离*d_{stop}被定义为在应用最大舒适减速amax时，车辆在完全停车前行驶的最大距离。

定义2。AV*车辆的安全包络*d_{safe}[1]，[2]定义为AV车辆在不与任何静态或动态物体发生碰撞的情况下所能行驶的最大距离。

安全信封是用来确保(通过对$U_{A,t}$)的约束，车辆轨迹是无碰撞的。生产型ads系统使用[34]、[35]等技术来估计车辆和物体轨迹，从而在向车辆的机械部件发送驱动命令时计算d_{safe}。这些ads通常设定一个最小的dsafe值(即dsafe,min)，以确保人类乘客在接近障碍物时不会感到不舒服。

定义3。*安全潜力*δ定义为$\delta = d_{safe} - d_{stop}$。当横向和纵向δ均> 0时，AV处于*安全状态*。[2]

C. 故障注入

DriveFI的目标是测试存在故障的ads，以识别可能导致事故的危险情况(例如，财产损失或生命损失)。为了实现这一目标，DriveFI包括(a)一个FI引擎，它可以修改ADS的软件和硬件状态来模拟故障的发生，以及(b)一个基于ml的故障选择引擎，它可以找到故障

[2]我们使用简写δ> 0来表示横向和纵向的δs。

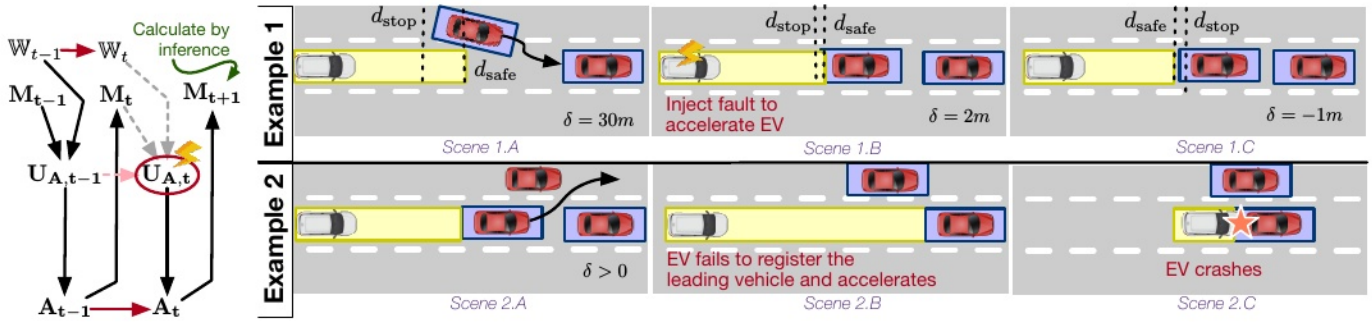


图4:示例场景:(1)目标FI导致危险状况;(2)实际示例图3:贝叶斯FI。特斯拉自动驾驶仪类似于注入故障。

最有可能导致违反安全条件的故障和场景，因此可以用来指导故障注入。综上所述，DriveFI的这些组件可以识别导致类似本节后面描述的特斯拉撞车事故的危险情况。

故障模型。我们假设在DriveFI中注入的错误会破坏GPU的架构状态。(cpu和gpu的)内存和缓存被假定为受SECDED代码保护。每个被注入的故障的特征是它的位置(在这个例子中是它的动态指令计数)和被注入的值。注入到这些处理器的体系结构状态中的*错误*可以表现为上述ADS模块(即I_t、M_t、S_t、$U_{A,t}$和A_t))的输入、输出和内部状态中的错误。DriveFI可以通过破坏存储ADS输出的变量直接将错误注入ADS输出。ADS软件输入/输出变量最终存储在不同的存储层次中，例如寄存器或缓存。在硬件[36]中没有屏蔽时，单位或多位错误会导致变量损坏。因此，故障被注入到这些内存单元中，但变量被损坏以模拟故障。因此，我们的故障注入器针对内部ADS软件状态(S_t)、传感器输入(I_t)、车辆惯性测量(M_t)和驱动命令(U_t、A_t)的每个元素，如图10所示。我们将任何导致AV安全问题的错误定义为*危险*。为了简单明了，在本文的其余部分，我们将注入的故障和错误都称为*故障*。

为了构建基于ml的靶向注入的基线，我们使用DriveFI对来自NVIDIA和百度的两个生产ADS系统的GPU架构状态和ADS模块输出进行随机注入。先前的工作[24]，[30]报道了ADS系统的组成深度学习模型(处理感知:对象识别和跟踪的ConvNets)的显著SDC率(高达20%)，与此相反，我们观察到随机注入很少会导致危险错误。由于ADS堆栈的自然弹性，这些故障被掩盖，即(a)生产ADS系统在60-100 Hz进行实时推断，在重新计算新的系统状态之前，瞬态故障几乎没有机会传播到执行器;(b) ADS系统架构具有固有的弹性，因为它使用了扩展卡尔曼滤波[37](用于传感器融合)和PID控制(用于输出平滑)等算法;(c)并不是所有的驾驶场景/帧即使在故障情况下也是危险的。环境条件，例如街道上其他物体的存在，是定义安全信封的基础。

贝叶斯故障注入。考虑一个错误f，它改变了上述变量之一的值。基于ml的故障注入器的目标是找到一个本质安全(即$\delta>0$)、注入故障f(即$\delta_{do(f)}\leqslant 0$)后变得不安全的临界情况。该条件保持is$_n$的*所有故障*$F_{crit}$的集合定义为

$$\mathbf{F}_{\text{crit}} = \left\{ f : \delta > 0 \wedge \hat{\delta}_{\text{do}(f)} \leq 0 \right\}. \quad (1)$$

要解决这个问题，就需要对过失下ADS的行为进行因果推理和反事实推理。DriveFI通过使用贝*叶斯网络*(BN;如图3所示)，它可以捕获因果关系[38]。BN描述了在t时刻的变量W_t、M_t、$U_{A,t}$和A_t之间的统计关系(用黑色箭头表示)，以及变量之间随时间的关系(用红色箭头表示)。BN的拓扑结构来源于ADS系统的体系结构。例如，图3与图1具有相同的图形结构。DriveFI使用BN计算值$M\boxtimes_{t+1}$的最大似然估计(MLE)[3]，然后使用MLE值基于§III后面描述的自动驾驶汽车的运动学模型计算$\delta\boxtimes_{do(f)}$。我们使用BN后验分布的概率推断来计算

$$\hat{\mathbf{M}}_{\mathbf{t+1}} = \arg\max_{\mathbf{m}} \Pr\left[\mathbf{M}_{\mathbf{t+1}} = \mathbf{m} \mid \text{do}(f)\right]. \quad (2)$$

do(·)表示法基于[38]中定义的*do-演算*。它标志着FI的行动是对BN模式的干预。它将某些概率替换为常数，并删除作为干预目标的统计条件依赖关系(即图3中的虚线)，但保留所有其他统计依赖关系。我们称这种关于断层在进行定向注入时重要性的反事实推理概念为*贝叶斯故障注入*。

D. 案例研究

为了解释对高效FI机制(如基于ml的故障喷油器)的需求，我们讨论了两个因故障导致的车祸的例子。

示例1:危险错误。图4显示了一个示例驾驶场景，其中通过损坏油门命令(从0.2更改为0.6)将故障注入ADS。注入的错误导致了事故。我们假设(a) ADS以30 Hz的频率运行感知、规划和控制推理，(b)所有车辆都在速度为33.5米/秒的高速公路上行驶，这大致是美国高速公路的限速。在场景1A中，Ego车辆(EV)正在加速;然而，由人类操作的目标车辆TV 1号启动了变道程序，变道次数减少了

[3] x的估计值用x⊠表示。

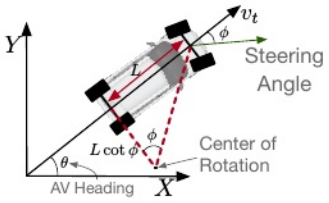


图5:EV运动时的方向。

“场景1B”中所示的从20米到2米的安全潜力三角洲。此时，贝叶斯故障喷射器将故障注入油门命令，导致车辆加速。加速的增加导致EV变得不安全($\delta < 0$)，如图1C所示。EV的速度足够高，即使使用a_{max}，也无法阻止事故的发生。这个例子表明，我们需要一种智能FI机制(比如我们的贝叶斯)，它能够根据安全潜力的运行时测量，在精确的时间瞬间注入故障，从而最大限度地增加ADS上的应力，并导致EV坠毁。正如我们在后面的章节中讨论的那样，使用随机FI来实现相同的目标是不切实际的(或非常困难的)。

例2:真实世界的崩溃。图4展示了一个真实世界的致命事故示例，该事故被证明是由特斯拉自动驾驶[6]的问题引起的。在场景2A中，EV跟随领头车辆(TV1号)。几秒钟后，1号TV换道了(如图2B所示);这时，自动驾驶仪决定加速，以匹配高速公路允许的速度。然而，TV1号在另一辆车(TV2号)后面，EV不知道TV2号;EV认出2号TV并及时减速以避免事故发生时已经太晚了。虽然这次崩溃归因于ADS感知子系统中的设计问题(即延迟识别)，但可以想象运行时故障(延迟对对象的感知)可能导致相同的致命结果。正如我们稍后所展示的，我们的基于贝叶斯的故障注入器能够重新创建这样的场景。

III. 贝叶斯故障注入

在本节中，我们详细描述了*贝叶斯故障注入*方法的公式。

A. 基于运动学的安全模型

考虑如图5所示在二维空间中运动的EV。时间t的车辆瞬时位置(x_t，y_t)，速度v_t，航向θ_t，转向角ϕ_t。车辆的运动方程为

$$\dot{x}_t = v_t\cos\theta_t;\ \dot{y}_t = v_t\sin\theta_t;\ \dot{\theta}_t = (v_t\tan\phi_t)/L, \quad (3)$$

其中L为EV车轮间距离[39]。这里v_t和ϕ_t是由EV的控制模型决定的。在我们的例子中，v_t是根据$ADSA_t$的输出定义的，即$v_t = f(\zeta_t, b_t, \phi_t)$。

请注意，更完整的EV运动模型可能包括其他动力学，例如，EV车轮的*滑动*和*打滑*。我们没有在我们的模型中添加这些复杂性，因为这将要求我们做出超出本文范围的额外假设，例如，关于EV的轮胎，道路状况，道路银行和天气。同样，我们没有考虑EV的三维运动，因为这样做需要进一步假设FI活动中地图的拓扑结构(例如，海拔)。我们的方法可以扩展到考虑那些额外的因素。

我们可以通过首先计算车辆完全停止所需的时间t_{stop}来计算最大停止距离d_{stop}从(3)开始，即:

$$\left.\frac{dx_t}{dt}\right|_{t=t_{stop}} = 0 \text{ and } \left.\frac{dy_t}{dt}\right|_{t=t_{stop}} = 0. \quad (4)$$

然后将d_{stop}计算为$[x_{t_{stop}} - x_0,\ y_{t_{stop}} - y_0]^T$，其中(x0, y0)为EV在机动开始时的位置。微分方程组(3)和(4)的闭式解对于任意控制程序(即v_t和ϕ_t)，必须通过迭代数值求解方法，如*龙格-库塔方法*[40]。

紧急停车机动。为了简化我们的分析，我们假设EV执行一个特殊的机动，我们称之为*紧急停车*，使车辆停下来。这个程序的特点是

$$\frac{dv_t}{dt} = -a_{\max} \text{ and } \frac{d\phi_t}{dt} = 0. \quad (5)$$

这对应的是EV的减速速度达到最大减速速度。(5)将(3)降低为

$$d^2x_t/dt^2 = -a_{\max}\sin\theta_t(d\theta_t/dt) \quad (6a)$$

$$d^2y_t/dt^2 = -a_{\max}\cos\theta_t(d\theta_t/dt) \quad (6b)$$

$$\frac{d\theta_t}{dt} = \frac{\left(\sqrt{(dx_t/dt)^2 + (dy_t/dt)^2}\right)}{L}\tan\phi_0, \quad (6c)$$

其中ϕ_0为机动开始时汽车的转向角。DriveFI使用式(4)和式(6)中定义的方程组求d_{stop}。我们使用简写P来表示用于计算的过程(迭代数值积分)

$$d_{stop} = \mathcal{P}(a_{\max}, v_0, \theta_0, \phi_0, x_0, y_0) \quad (7)$$

和EV在机动开始时的初始运动状态(即v_0，θ_0，ϕ_0，x_0，y_0))。

回想第II节，$\delta = d_{safe} - d_{stop}$，$\delta > 0$定义了EV的安全性。假设$d_{safe}$值直接由EV的传感器计算得出。它是在EV的纵向或横向路径上到最近物体(静态或动态)的距离。因此，d_{safe}会随时间变化，并以传感器(例如激光雷达或相机)的刷新率进行更新。我们将EV行驶的车道边界(以下称为自我车*道)作为静*态对象，用于d_{safe}计算，以确保我们将车道违规捕获为安全隐患。

离散化。我们将(7)的求解问题从一个使用连续时间的问题转换为一个使用离散时间概念的问题。离散时间是ADS的自然选择，因为控制决策是在与传感器采样频率相对应的离散步骤上做出的。因此，我们将时间t转换为离散数$k \in N$，使得$t = k\Delta t$，其中Δt是具有最小采样频率的传感器的周期。在DriveWorks和Apollo的DriveFI注入器的情况下，这是7.5 Hz。然而，我们的方法是频率无关的。

B. 毫升模型

目标故障注入器的目标是找到$\delta > 0$，但在向ADS堆栈注入故障f(表现为EV运动状态的变化)的情况下，$\delta_{do(f)} \leq 0$。该问题的解决方案是在故障注入后及时向前推测，在故障下重新计算d_{stop}，然后重新评估EV的安全标准。我们应用ML算法，该算法已被训练为EV运动状态的预测器，作为

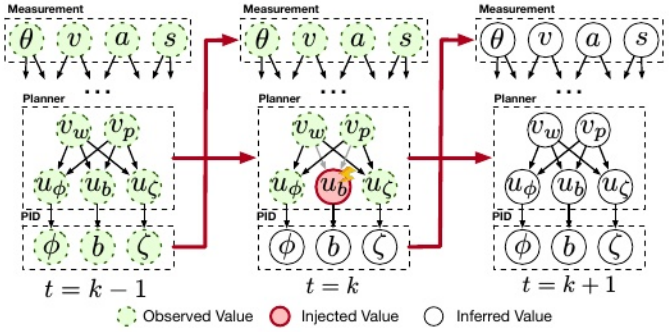


图6:3-时态贝叶斯网络对ADS建模。

推测机制。我们现在描述模型的设计及其训练和推理。

该模型。考虑这样一种情况:在时间点k, EV的ADS中注入了一个故障。当前一个时间步长的(损坏的)驱动命令被执行时，我们想要估计时间k + 1处d_{stop}的值。正如我们在上一节中所展示的那样，我们可以使用(7)来做到这一点。然而，这需要知道x_{k+1}、yk+1、vk+1、θk+1和ϕk+1的值作为启动紧急停止机动的初始条件。DriveFI基于捕获ADS组件的概率模型的后验分布的最大似然估计来估计这些值。

DriveFI使用动态贝叶斯网络(DBN)[41]，特别是3-Temporal贝叶斯网络(TBN)，即展开三次的DBN，对xk+1、yk+1、vk+1、θk+1和ϕ_{k+1}进行建模。这个模型如图6所示。dbn的核心思想是用静态BN对每个时间点建模，并添加从一个时间片到下一个时间片的时间链接(如图6中红色箭头所示)。通常所有时间点都具有相同的BN拓扑和超参数设置。dbns是有向无环图，其中节点代表随机变量，弧线代表变量之间的因果联系[42]。今后我们将BN中的每个随机变量称为一个*节点*，以避免与ADS变量混淆。每个节点x都与一个概率表相关联，该表提供了条件概率分布(CPDs;给定父节点π(x)的值，节点可能值的Pr(x | π(x))。

基于图1所示的拓扑结构构建3-TBN模型(见图6)。阿波罗和DriveFI ads的详细版本见§IV，如图8所示。每个ADS模块中的变量以父子方式连接，反映了图8中的数据流。例如，$u_\zeta$andζ(在图6中)之间的边表示CPD Pr(ζ |u_ζ))。这是ζ的PID控制的近似值。类似地，ADS的其他组件基于其输入和输出变量进行建模。我们假设3-TBN中的节点由具有功能形式的CPD描述

$$\Pr(x \mid \pi(x)) = \mathcal{N}(\mu_{\mathbf{x}}^{\mathbf{T}}\pi(x), \sigma_x)$$

其中N是带有参数μ_x和σ_x的正态分布(对于网络中的每个节点x)。选择这种特殊形式的Pr(x | π(x))是因为(a)它在小概率值下具有数值稳定性，这在处理罕见事件(如故障)时很常见，(b)它简化了训练3-TBN所需的算法。

使用基于3- tbn的建模形式是基于以下隐式假设:(a) EV状态可以完全由其先前状态和观测到的软件变量决定，(b)从一个时间步长到另一个时间步长的过渡参数不随时间变化，即假设马尔可夫动态系统是齐次的。

概率推理。显示故障f(对应于设定模型中某个变量的值)下的最大似然估计值v⨂$_{k+1}$为

$$\hat{v}_{k+1} = \arg\max_{v} \Pr\left(v_{k+1} = v \mid \mathrm{do}(f), \mathbf{O_k^{(f)}}\right). \quad (8)$$

鉴于我们可以在无故障条件下执行EV的模拟，可以观察到所有不是注入变量的子变量具有正确运行的值。这些“黄金”观测值被标记为O_k。(8)解出 (f)

首先通过使用*马尔可夫链蒙特卡洛*方法[41]估计v_{k+1}的后验分布，然后估计v_{k+1}的最有可能值。可以使用类似的程序来计算θ⨂$_{k+1}$和ϕ⨂$_{k+1}$。然后可以使用(3)的时间离散版本计算x⨂k+1和y⨂k+1的值。最后，从(7),我们得到

$$\hat{d_{\text{stop}}} = \mathcal{P}\left(a_{\max}, \hat{x}_{k+1}, \hat{y}_{k+1}, \hat{v}_{k+1}, \hat{\theta}_{k+1}, \hat{\phi}_{k+1}\right). \quad (9)$$

培训。上面描述的3-TBN定义了一个概率分布Pr(X_{k-1}, X_k, X_{k+1})，其中X_k=M_k∪S_k∪$U_{A,k}$∪A_k。通过BN形式，将P(X_{k-1}, X_k, X_{k+1})定义为

$$\Pr(\mathbb{X}_{k-1}, \mathbb{X}_k, \mathbb{X}_{k+1}) = \frac{1}{Z} \prod_{x \in \mathbb{X}_k \cup \mathbb{X}_{k+1}} \Pr(x|\pi(x))$$

其中Z是将P归一化为概率分布的配分函数。我们使用*期望最大化算法*[43]进行计算

$$\hat{\mu}, \hat{\sigma} = \arg\max_{\mu,\sigma} E_{\mathbb{X} \mid \mathcal{D},\mu,\sigma}\left[\log P(\mathbb{X} \mid \mu, \sigma)\right] \quad (10)$$

其中D表示在正常运行和FIs期间包含X_{k-1}、X_k和X_{k+1}值的训练数据集。在这里，由于X_{k-1}×X_k×X_{k+1}的组合尺寸很大，Z的计算是难以处理的。然而，(8)不需要Z的计算，因为它是目标函数所有值的公共乘法。

训练数据。X_k中的变量是通过在模拟器中多个驾驶场景中执行ADS来测量的。我们在§IV中描述了这个模拟器的设置。单纯捕获正常操作下的数据不足以捕获由于故障导致的ADS状态下的异常。因此，除了运行没有故障的驾驶场景外，我们在运行驾驶场景的同时注入随机故障(即§II中描述的基线)，一次一个。与训练数据相对应的FI活动在§V中描述。为每个故障重新创建了向均匀随机选择的场景中注入故障20到50次的过程。改变故障数量的原因是，一些变量(如ζ、b和ϕ)在没有注入的模拟运行中显示出所有可能的值，而其他变量，如有状态变量，根本不会自然变化。

故障注入。F_{crit}(from(1))的计算在每个驾驶场景的每一帧都是离线完成的。FI过程执行如下(见图7):

· 对于每个驾驶场景，执行一个非故障注入的模拟“黄金”执行。在每个瞬间k,

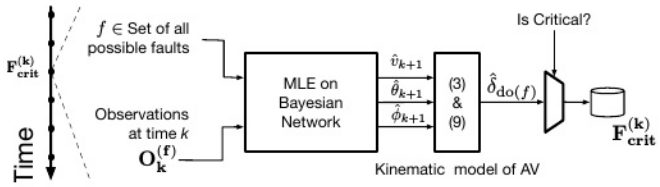


图7:离线执行每个模拟时间点的BN MLE推断，以找到关键故障集。

X_k中的变量被测量和存储。 · X_k的这些“黄金”值通过(9)建立Fcrit (k)为每个场景/帧，基于(1)。

· 在模拟EV上执行FI运动，在Sk Fcrit (k)中执行一帧故障，每次执行一个故障。

IV. ADS架构与仿真

A. AI平台

AV汽车使用ADS技术来支持和取代人类驾驶员来控制车辆的转向、加速和监测周围环境(例如其他车辆/行人)[44]。ADS架构由五个基本层组成[4]，讨论如下:

传感器抽象层(图8中的1):传感器抽象层负责对输入进行预处理
数据，噪声过滤，增益控制[45]，色调映射[46]，去马赛克[47]，以及感兴趣区域的提取，取决于传感器类型。ADS支持范围广泛的传感器，如全球定位系统(GPS)、惯性测量单元(IMU)、声纳、雷达、激光雷达和摄像传感器。我们的实验只使用两个摄像头(安装在车辆的顶部和前部)和一个激光雷达。

感知层(图8中的2):传感器抽象层将数据输入感知层，感知层使用计算机
视觉技术(包括深度学习[48])检测驾驶场景中出现的静态物体(例如车道、交通标志、障碍物)和动态物体(例如乘用车、卡车、骑自行车的人、行人)。

目标检测算法执行几个任务(例如，分割、分类和聚类)。它单独使用所有传感器数据，然后使用传感器融合算法(例如，扩展卡尔曼滤波[37]，[49])合并数据。融合算法为目标检测提供了软件级的数据冗余。利用高清地图和定位模块，ADS可以预先确定特定静态物体(如交通灯)的位置，进一步提高障碍物检测任务的信心。

感知层还负责对物体和车道的时间跟踪。跟踪是必要的，以确保一个物体不会因为分类错误或没有检测到任何东西而突然从一帧中消失。因此，传感器融合和跟踪在软件的感知层提供了空间和时间上的冗余。在精确确定和跟踪物体和车道后，感知层计算各种有用的指标，如“最近路径障碍”(CIPO)和每个物体的“尾随距离”。这种物体与测量或推断的度量(例如，CIPO和尾随距离)的关联被定义为AV的*世界模型*。

定位层(图8中的3):定位模块负责聚合来自各种来源的数据，以定

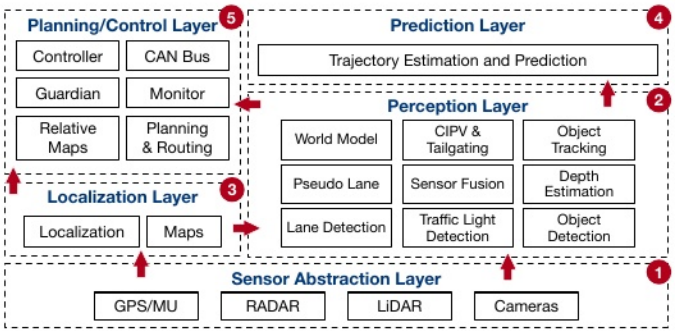


图8:ADS架构。

位世界模型中的自动驾驶车辆。世界模型中的定位可以使用GPS传感器或使用相机/激光雷达输入来完成。本文中描述的工作仅使用相机/激光雷达以及地图来实现定位(即，它不使用GPS)。

预测层(图8中的4):预测层负责为us-检测到的物体生成轨迹
Ing信息来自世界模型(例如，位置，标题，速度，加速度)。因此，它可以概率地识别AV路径上的障碍物[50]。

规划与控制层(图8中的5):规划与控制层负责生成导航规划
根据EV的出发地和目的地，并向AV发送控制信号(驱动、制动、转向)。“路由模块”根据请求生成高级导航信息。路由模块需要知道路由起点和路由终点，以便计算通道车道和道路。“规划模块”通过使用定位输出、预测输出和路由输出，规划出安全无碰撞的轨迹。“控制模块”以规划的轨迹为输入，生成控制命令传递给CAN总线，CAN总线将信息传递给AV的机械部件。监控系统监控车辆内的所有模块，包括硬件。“监控模块”接收来自不同模块的数据，并将其传递给人机界面，供人类驾驶员查看，以确保所有模块正常运行。当出现模块或硬件故障时，监视器会在“监护模块”中触发警报，然后由“监护模块”选择要采取的行动，以防止发生事故。

B. 仿真平台

本文使用基于虚幻引擎(UE)的仿真平台(卡拉[51]和DriveSim[52])，这些平台能够通过使用城市布局、建筑物、行人、车辆和天气条件(如晴天、雨天和雾天)的库来模拟复杂的城市和高速公路驾驶场景。模拟平台能够定期生成传感器数据(来自摄像头和激光雷达)，这些数据可以馈送到ADS平台。一个驾驶场景由DriveAV中的500个场景或阿波罗中的2400个场景组成，其中EV从道路上的固定起点行驶到固定终点。驾驶场景中的场景是模拟纪元的物理世界的表征，对应于摄像机框架。图9展示了本研究中使用的三个高速公路(DS1-DS3)和三个城市(DS4-DS6)驾驶场景的场景。DS1-3由DriveSim中的DriveAV控制，DS4-6由卡拉中的阿波罗控制。在这些场景中，一辆EV和一些ue控制的电视/行人

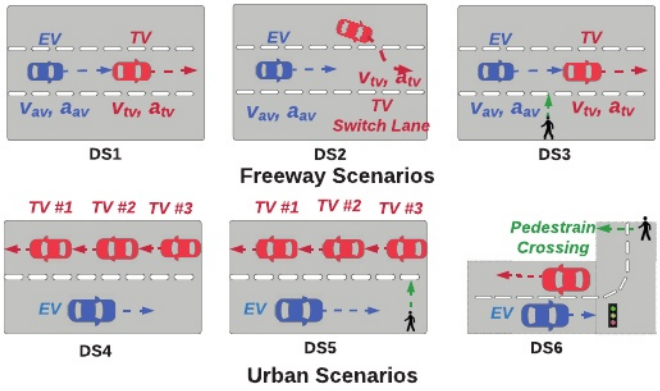


图9:仿真引擎支持的驾驶场景。

被放置在城市和高速公路上，以不同的速度/加速度行驶，并相隔一定距离。预计EV将在这些设置中执行驾驶操作。这些场景代表了人类日常遇到的最常见的驾驶案例。在DS1-DS6中，Ego车辆不换道，没有其他车辆跟在Ego车辆后面，Ego车辆处于安全状态。

C. 硬件平台

NVIDIA DriveAV ADS是为NVIDIA AGX Pegasus平台设计的[53]，该平台由两个Xavier soc和两个分立gpu组成，但也支持基于x86 CPU和GPU的开发平台。在我们的实验中，我们使用开发平台及其实用程序来促进DriveFI工具的创建。阿波罗 ADS在Nuvo-6108GC上得到支持[54]，该处理器由Intel Xeon cpu和NVIDIA gpu组成。我们在带有两个NVIDIA Titan Xp gpu的x86工作站上使用阿波罗。

V. DRIVEFI架构

DriveFI软件架构如图10所示。DriveFI利用现有工具来模拟驾驶场景，并通过使用AI代理(由阿波罗或DriveAV提供)在模拟中控制EV。场景管理器协调模拟器和AI代理运行驾驶场景，监控软件状态以及EV的安全性。DriveFI与一个活动管理器捆绑在一起，该活动管理器将XML配置文件作为输入，以选择故障模型、FI的软件或硬件模块站点、故障数量和驾驶场景。活动经理使用指定的配置来(a)配置ADS工作负载，(b)生成故障计划[4]，以及(c)每次运行将一个或多个临时故障注入ADS系统。基于配置文件中的值，活动管理器运行指定数量的黄金模拟，在运行驾驶场景时配置ADS，并根据生成的故障计划运行指定数量的实验，每次注入一个或多个故障。“事件驱动同步”模块有助于在所有工具包(基于ui的驾驶场景模拟器、监控代理、活动管理器、故障注入器和AI代理)之间进行协调。

我们构建了DriveFI来表征误差传播和掩蔽(a)在计算元素中，(b)在ADS中，(c)在车辆动力学和交通中。低级电路、微架构和RTL故障表现为架构状态

故障计划指定要损坏的指令/变量、损坏时间和损坏值。

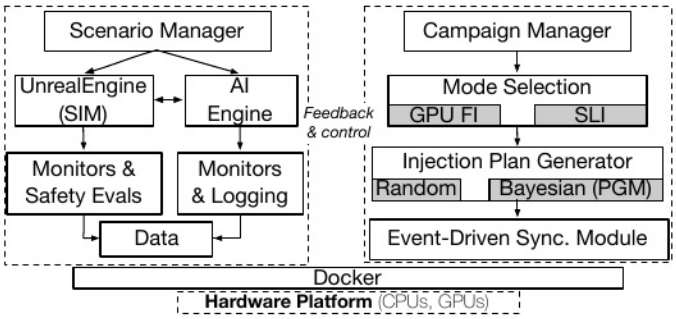


图10:DriveFI架构。

计算元素的错误。未被屏蔽的体系结构状态错误表现为ADS模块内部状态的错误，未在模块中被屏蔽的错误传播到模块的输出。最后，在任何模块中未被掩盖的错误都表现为发送给AV的驱动命令错误。因此，为了模拟故障和错误，我们构建了两个故障注入器:(a) GPU故障注入器(GI;参见章节V-A)能够将故障注入GPU架构状态，以揭示GPU故障向ADS状态的传播，以及(b)一个源级故障注入器(SLI，参见章节V-B)能够注入故障以破坏ADS软件变量。ADS的最终输出(驱动值ζ， b， θ)的损坏有助于我们测量与车辆动力学和交通相关的弹性。因此，我们的方法有助于测量不同级别的故障和错误掩蔽/传播以及相应的对AV安全性的影响。

DriveFI与一个活动管理器捆绑在一起，该管理器以XML配置文件作为输入，为FI选择故障模型、软件或硬件模块站点、故障数量和驾驶场景。活动经理使用指定的配置(a)来分析ADS工作负载，(b)生成故障计划，以及(c)在每次运行时向ADS系统注入一个或多个临时故障。对于本文，我们开发了一个“事件驱动同步”模块，在所有工具包(基于ue的驾驶场景模拟器、监控代理、活动管理器、故障注入器和AI代理)之间进行协调。

A. 注入计算元素:GPU故障模型

我们考虑了GPU处理器的功能单元(例如，算术和逻辑单元以及负载存储单元)、锁存器和未受保护的SRAM结构中的瞬态故障。这种瞬态故障是通过在执行指令的输出中注入位翻转(单位和双位)来建模的。如果目标寄存器是通用寄存器或条件码，则随机选择一个或两个比特进行翻转。对于存储指令，我们在存储值中翻转一个随机选择的位(或位)。由于我们直接将故障注入到活动状态(目标寄存器)中，我们的故障模型没有考虑硬件堆栈较低层中的各种屏蔽因素，例如电路、门和微架构级屏蔽，以及由于架构未触及值中的故障而导致的屏蔽。GI采用了类似于SASSIFI[23]的方法，包括剖析传递和故障注入计划生成。我们不考虑缓存、内存和寄存器文件中的故障，因为它们是由ECC保护的。

B. 将故障注入ADS模块输出变量

SLI(源级注入)的目标是通过修改ADS模块输出来破坏ADS的内部状态

表1:sil支持ADS模块输出示例。

FI Target (Output Variables)
Path Perception Module
lane_type, lane_width
Object Perception Module
camera_object_distance, camera_object_class, lidar_object_distance, lidar_object_class, sensor_fused_obstacle_distance, sensor_fused_obstacle_class
Planning & Control Module
vehicle_state_measurements (pos, v, a), obstacle_state_measurements (pos, v, a), actuator_values (ζ, b, ϕ), pid_measured_value, pid_output

变量(因此，是另一个模块的输入变量)的ADS组件。SLI是作为静态链接到ADS软件的库实现的;但是，它的使用需要修改源代码并重新编译ADS软件。在sli链接的ADS和非sli ADS之间，我们没有观察到任何明显的运行时间差异。在这项工作中，我们手动识别存储ADS模块输出的软件变量，这些模块在推断EV的驱动命令中起着关键作用。为了标记输出变量并调用相应的模块注入器，通过使用XML配置文件中提供的故障模型来获得损坏的值，需要进行源代码修改。在表1中，我们显示了使用SLI作为目标的每个ADS模块(见图8)中的一些变量。

SLI支持的损坏第k个场景(在驾驶场景的所有场景中均匀随机选择)中的一个或多个软件输出变量的故障模型由(a)多个故障(即要注入的多个连续场景)和(b)故障位置指定。在基于*sli的实验中,一个单一的错误*是ADS模块的单个输出变量的损坏。在下文中，我们定义了这些sil支持的故障模型。

1-固定。在给定ADS软件模块输出的第k个场景注入单个故障。在实验中，使用一个恒定值来破坏给定的ADS软件模块输出。总共有41种“1-固定”故障类型，每种类型由(a) ADS模块输出和(b)损坏值定义。有界连续输出被损坏为这些输出的最大或最小可能值，例如，注入到制动驱动输出中，SLI使用最大制动值1.0或最小制动值0.0。无界连续输出值(例如，v, a和pos)被损坏为当前输出值的两倍或一半5。对于分类输出变量，输出值被损坏为其中一个分类值;例如，对象/障碍类可以被损坏为“不关心/消失”，“行人”，“车辆”和“骑自行车的人”。

m固定。从场景k开始，将m个故障注入给定的一组ADS软件模块输出，并继续将故障注入ADS软件模块输出，直到场景k + m。m在10 ~ 100之间均匀随机选择。m所选择的范围足够大，足以支持对一定数量的连续帧/场景进行阈值研究，这些连续帧/场景必须注入，以造成危险的情况。同样，有41种“m固定”故障类型。

1-Random。在均匀随机选择的一组ADS模块输出中，在第k个场景注入单个故障。注入的故障值也是均匀随机选择的

我们将输出的损坏限制为两倍或一半，否则ADS可能会将注入的故障检测为错误。

表2:错误注入实验。

Campaign	Target module	#Faults/Experiment
1-GPU-all	All GPU kernels	1
1-RANDOM	All software module outputs	1
1-Fixed_throttle_max	Actuator - throttle	1
1-Fixed_brake_max	Actuator - brake	1
1-Fixed_Steer_max	Actuator - steer	1
1-Fixed_obstacle_rem	Perception - obstacle disappear	1
1-Fixed_obstacle_dist	Perception - obstacle distance	1
1-Fixed_lane_rem	Perception - lane disappear	1
M-Random	All software module outputs	10–100
M-Fixed_throttle_max	Actuator - throttle	10–100
M-Fixed_brake_max	Actuator - brake	10–100
M-Fixed_Steer_max	Actuator - steer	10–100
M-Fixed_obstacle_rem	Perception - obstacle disappear	10–100
M-Fixed_obstacle_dist	Perception - obstacle distance	10–100
M-Fixed_lane_rem	Perception - lane disappear	10–100
1-PGM	All software modules	1

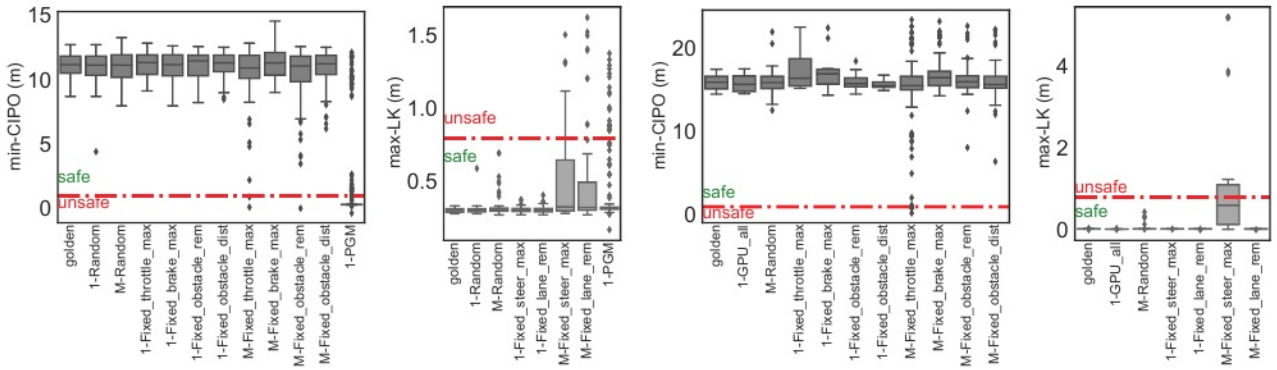
从所选ADS模块输出的值范围中。

M-Random。从场景k开始，在一组随机选择的ADS软件模块输出中注入m个故障，并继续在ADS软件模块输出中注入故障，直到场景k + m。m在10 ~ 100之间均匀随机地选择。在这种情况下，ADS模块和损坏值都是均匀随机选择的。

VI. 结果

在本节中，我们描述了故障和错误注入对EV安全性的影响。在我们的工作中，我们使用基于ue的模拟器研究了三种高速公路驾驶场景(DS1-DS3)和三种城市驾驶场景(DS4-DS6)。DS1 - DS3由DriveAV控制，DS4-DS6由Apollo控制。通过计算CIPO(路径中最近的障碍物)和LK距离(距离车道中心的横向距离)来验证EV在任何给定场景下的安全性。当纵向d_{min}< 1.0 m时，与CIPO的最小距离小于1.0 m，或EV穿过Ego车道时，与车道中心的距离为0.80 m，发生安全隐患。因此，所有场景中的最小CIPO距离(min-CIPO)和最大LK距离(max-LK)表征了整个模拟的安全隐患。

由于篇幅限制，在不失一般性的前提下，我们将讨论范围限制在EV由DriveAV控制的DS1和EV由Apollo控制的DS6。图11a-11d分别显示了Apollo (DS6)和DriveAV (DS1)在所有断层注入实验和黄金运行中的min-CIPO和max-LK箱形图。这些实验总结在表II中。箱线图显示了定量数据的分布，便于变量之间或分类变量的跨水平比较。箱线图显示了数据集的四分位数，而胡须扩展显示了分布的其余部分(最大和最小样本)，除了被确定为异常值[55]的点。为了了解驾驶场景的模拟和安全特性，我们在没有任何注入的情况下，对每个场景进行了50次端到端的模拟。这些运行被称为*黄金运行*。黄金跑作为参考，我们在论文的其余部分将注入的模拟跑进行比较。DriveAV的min-CIPO和max-LK距离中位数分别为16 m (见图11c中的“golden”)和0.019 m(见图11d中的“golden”)，Apollo的最小cipo和最大lk距离中位数分别为11.19 m(见图11a中的“golden”)和0.31 m(见图11b中的“golden”)。这些“黄金”跑都没有造成安全隐患。



(a) Apollo min-CIPO (b) Apollo max-LK (c) DriveAV min-CIPO (d) DriveAV max-LK

图11:使用FI活动的故障/错误影响表征。(a)和(b)使用DS6;(c)和(d)使用DS1。

A. gpu级故障注入

我们在DriveAV中对每个驾驶场景(DS1、DS2、DS3)进行了800次gpu级FI实验。在DriveAV中模拟的DS1的min-CIPO和最大lk分别在图11c和图11d中标记为“1-GPU”。每个场景我们只进行了800个gpu级FI实验，因为我们在运行过程中没有观察到任何安全违规行为，并且运行更多的实验将会非常昂贵(每个驾驶场景2.7天，800*5分钟/FI)。在标记为“1-GPU_all”的FI实验中，从ADS中的所有动态指令中均匀随机选择故障。由于GI和阿波罗之间的CUDA驱动程序版本不匹配，我们没有在阿波罗上进行任何GPU FI实验。解决这个问题需要厂商的支持和修复。从图11c和图11d可以看出，即使在FI之后，EV始终是安全的，其分布与黄金情况下的分布相似。

gpu中的故障传播和掩蔽。在DS1-DS3驱动场景下的所有GPU-FI实验中，总共有2400个FI实验，其中1.9%的注入故障导致无声数据损坏(即导致ADS模块的最终输出驱动输出损坏)，0.02%导致对象误分类错误6。物体误分类错误都没有导致致动输出损坏。我们的研究结果表明，感知模块(负责对象检测和分类)比其他ADS模块更具弹性。原因是感知软件利用了传感器融合(即传感设备中的冗余可以补偿单个传感器的故障)。在所有驾驶场景中，sdc没有导致任何EV安全漏洞。

7.35%的故障导致可检测到的不可纠正错误(会费)，导致ADS软件崩溃(61%)或挂起(39%)。ADS系统是用来处理可检测到的错误，并采取相应的纠正或安全措施的。虽然会费比sdc更常见，但期望系统可以通过备份/冗余系统从此类故障中恢复。

错误持续存在于多个帧。在2%的误分类错误案例中(回想一下0.02%的gpu级fi导致误分类错误)，ADS感知模块输出被错误地分类了不止一帧，即注入故障的影响持续了不止一帧。在我们的数据中，我们观察到多达8个连续帧的对象分类错误。在这些情况下，由于ADS平台的时间性质，错误最终确实被掩盖了。例如，ADS以固定的间隔输入新的传感器数据，例如在我们的研究中每秒7.5次。这一观察结果表明，需要在软件级别对ADSs中的故障屏蔽和传播进行更深入的研究，以处理故障持续一帧以上的情况。

B. 源级故障注入

在上一节中，我们观察到ADS能够补偿注入的瞬态故障。为了进一步了解ADS平台对故障的敏感性及其在持续错误情况下的鲁棒性，我们使用SLI进行了有针对性的FI，将一个或多个故障直接注入ADS模块输出中。我们针对每个驾驶场景(DriveAV的场景1-3和阿波罗的场景4-6)进行了84次基于si的FI活动。在43个活动中，1个对应“1-随机”，1个对应“M-Random”，41个对应“m -固定”下的41种故障类型，41个对应“M-Random”下的41种故障类型。标签如图11所示。

ADS对单故障和多故障的鲁棒性。发现ADS平台对单个故障注入具有鲁棒性(“1-随机”活动)。为了了解故障产生的多重随机误差对持久性的鲁棒性，我们使用“1-随机”和“M-Random”故障模型对驾驶场景进行了FI运动。“M-Random”的min-CIPO和最大lk分布与阿波罗(见图11a和图11b中的“M-Random”)和DriveAV(见图11c和图11d中的“M-Random”)的黄金运行中的分布有统计学差异。在“1-随机”和“M-Random”活动中，注入的故障都没有导致危险的驾驶情况;然而，ADS的安全性更容易受到“M-Random”故障模型的影响(特别是车道保持功能)。例如，阿波罗注射剂的最小min-CIPO从8.7 m下降到8.0 m，最大lk从0.34 m增加到0.7 m。DriveAV的min-CIPO从15.2 m增加到12.6 m, 最大lk从0.024 m减少到0.43 m。

ADS模块对单个和多个故障的鲁棒性。ADS模块组件内部的持续性故障，会持续给对应模块产生错误。我们测试了ADS对故障的鲁棒性

[6] *Object misclassification* refers to incorrect classification of an object, e.g., a pedestrian may be recognized as a vehicle.

[7] The AV came closer to the other vehicle/pedestrian compared to when no fault was injected.

模块通过使所选模块中的一个输出产生多个故障。在这些活动中，我们使用了“1固定”和“M-Fixed”故障模型。“M-Fixed”和“1固定”故障类型共有41种故障类型(例如，“油门最大”，“障碍物移除”和“车道移除”)。由于空间不足，我们讨论了仅选择的活动的结果。所选择的活动(如图11所示)包括(a)致动模块输出损坏(其中刹车、油门和转向都被更改为允许的“最大”值);(b)传感器融合输出损坏(其中障碍类改为“消失”，轨迹规划中可考虑的距离改为“最大”);以及(c)车道输出损坏(其中车道类型改为“消失”)。导致安全漏洞的FI实验显示为最小cipo红线以下和最大lk红线上方的数据点。显然，在“1固定”故障模型下进行的FI活动都不会导致安全隐患，但很少观察到“m -随机”FI活动。我们根据模块脆弱性因子(MVF)对ADS模块进行排名，我们通过查找导致(a)最小cipo距离小于黄金运行中的最小最小cipo距离的模拟百分比来计算，或者(b)最大lk距离最大值大于黄金模拟运行中的最大lk距离。使用该方法，我们发现“转向角”(MVF=46%)，“车道分类”(MVF=43%)，“障碍物分类”(MVF=10%)和“油门”(MVF=7%)是阿波罗最脆弱的组件，而对于DriveAV，我们发现除了“车道分类”和“障碍物清除”之外，相同的组件是脆弱的。

DriveAV中“车道分类”和“障碍物清除”的高弹性可归因于自由空间检测模块(阿波罗中不存在)和场景属性。自由空间检测模块帮助DriveAV EV检测可驾驶空间(使用专门的DNN网络来寻找可驾驶空间)，即使对象被错误分类或其属性(如距离和速度等)损坏。free-space检测模块在不需要完全复制障碍物检测和分类模块的情况下确保了安全性。这两个模块的缺陷掩盖也可以归因于世界模型中的障碍注册和跟踪，该模型有助于随着时间的推移跟踪障碍物。

ADS中的补偿:ADS自动补偿EV状态(即θ， v, a, s)的任何变化，这些变化导致由一个或多个故障/错误引起的不安全状态。它通过发出使EV进入安全状态的驱动命令来实现这一点。例如，EV可以通过制动(b)来补偿增加的v，通过节流(ζ)来补偿减小的v，或者通过转向(φ)来补偿航向角的变化。图12显示了黄金和注入运行的油门(ζ)值(在左子图中)和通过制动实现的补偿(在右子图中)，其中ζ在30个连续帧/场景中损坏了FI实验。时间步K的补偿被计算为注入道和黄金道中在时间步K观测到的“制动”值的累积和之间的差。注入会导致车辆速度的增加，通过制动来补偿。在图12的右侧子图中，我们显示补偿增加，直到时间步K = 232来撤销多个故障的影响，然后随着黄金跑道和故障跑道(即注入故障的跑道)中的制动值变得相等，补偿变平。我们

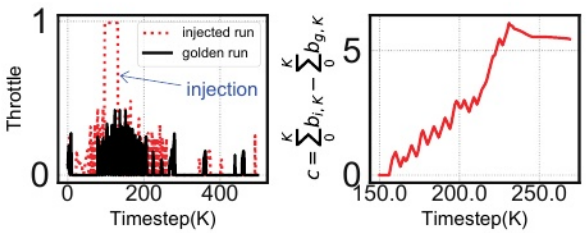


图12:DriveAV中30个连续故障对ζ的影响。左子图显示金色模拟(黑色)和注入模拟(红色)的ζ。右子图显示补偿c。

对注入制动和转向值的故障观察到类似的补偿行为。

ADS补偿注入故障的能力取决于故障的数量和注入的时间。图11a中“M-Fixed_throttle_max”红线下方的离群数据点对应于将故障注入到ζ值中的30个连续帧/场景。在本次FI实验中(未见图12)，车辆无法对注入的故障进行补偿，因为故障是在K = 400时注入的，并且车辆没有足够的时间停车，即EV在注入结束时达到不安全状态。在阿波罗中，只有20个注入到ζ值中的错误导致了不安全状态。*持续误差对EV状态有显著影响，ADS补偿误差影响的能力取决于fi发生的时间和位置。*

C. 基于贝叶斯fi的注射结果

到目前为止，在我们的FI活动中，只有当多个故障被注入ADS(即注入多个连续帧/场景)时，才会产生危险的驾驶条件(事故和车道违规)。然而，在现实世界中，更有可能发生单一故障，因此找到单一故障可能导致危险驾驶条件的条件是很重要的。解决找到所有此类单一故障(即关键故障)问题的一种方法是在模拟器中运行驾驶场景时注入每一个单一故障。然而，这种方法的成本太高，在实践中是不可行的。例如，在我们的模拟平台中，详尽地搜索“1-固定”故障模型下41种故障类型中的哪一种会导致安全危害，将花费272天[8][9]。另一种寻找关键故障的方法是均匀随机地注入故障。然而，GPU硬件级FI(见§ VI-A)和ADS软件模块级FI(见§ VI-B)的结果表明，我们需要一种智能FI方法，能够识别驾驶场景中的危险情况，并利用它们来指导FI实验。基于这种方法的故障注入器将在ADS最脆弱的时候注入故障(即故障很可能传播到执行器)，并且以ADS无法补偿故障的方式注入故障。贝叶斯故障注入器能够找到一个本质安全(即δ> 0)但注入故障f(即$\delta_{do(f)} \leqslant 0$)后变得不安全的临界情况。我们通过将故障*注入到由*Apollo控制的DS4-DS6驾驶场景中，证明了贝叶斯FI的有效性。

贝叶斯FI的有效性。当我们使用贝叶斯FI时，82%的注入断层导致危险。(95%的危险是涉及行人的事故，5%是

8 615 days/DS = 9 min/DS *41 fault types *2400 scenes.

9 Note that traditional FI is sampling-based, so 615 days represents the worst case of enumeration of all faults.

莱恩侵犯)。贝叶斯FI从“1-固定”故障模型的41种故障类型中选择一种，使用SLI将单个故障注入ADS模块输出变量。回想一下，在“1-固定”故障模型中，故障位置(即ADS模块输出变量)和损坏值由故障类型定义。相比之下，随机的单个FI都没有导致安全隐患。贝叶斯FI结果在图11a和图11b中标记为“1-PGM”。图11a红线以下的所有数据点对应碰撞，图11b红线以上的所有数据点对应车道违章。min-CIPO距离的中位数为0.32 m，明显小于金线11.19 m的中位数。虽然与黄金跑相比，“1-PGM”运动的最大lk值中位数没有变化，但5%的危险是由于车道违规造成的。

挖掘关键断层和关键场景。如前所述，在SLI的“1-固定”故障模型下注入所有类型的故障将是非常昂贵的。贝叶斯FI帮助我们找到了每个场景的所有关键故障$|F_{crit}|$以及更容易发生故障的矿井驾驶场景。贝叶斯FI挖掘的关键故障可以帮助设计人员了解系统的弱点和可能导致危险的边缘情况，而关键场景可以被设计人员使用，仅在这些场景中注入随机故障(使用GI或SLI)，以帮助他们了解架构脆弱性因素(AVF)。我们相信，贝叶斯FI对关键场景的挖掘将具有比我们这里的FI更广泛的适用性。将一系列FI实验的结果结合起来，创建一个场景库，将有助于制造商制定自动驾驶测试和安全驾驶的规则和条件。表III给出了在驱动场景(DS4-6)中开采的关键断层和场景的汇总统计。在DS4-6中总共发现了561个关键故障。通过对所挖掘的关键故障进行检查，我们发现ADS模块对车辆碰撞最敏感的前3个输出分别是油门值(561个关键故障中的24%)、PID控制器输入(18%)和传感器融合障碍等级值(561个关键故障中的15%)。贝叶斯FI用于创建车道违规的ADS模块输出为(a)车道类型值(561的2%)，(b)油门(1.4%)和(c)转向(1.4%)。56%的故障类型未被贝叶斯FI利用;例如，贝叶斯FI从未被注入到相机-传感器目标分类模块的输出中。

对于DS4，我们没有发现任何关键场景或错误。这是意料之中的，因为在我们的驾驶场景中，EV周围没有尾随或领先的车辆。所有车辆都在另一条车道上，遵循完全不同的轨迹，在这种情况下，一个故障不足以使EV进入相邻车道。对于DS5，发现0.88%的场景和0.20%的故障是关键的。在这种情况下，关键场景对应于(a)物体(即行人)首先被注册到世界模型中，(b)EV随后开始制动的场景。在(a)的情况下，贝叶斯FI选择移除障碍物(例如，通过移除障碍物，或错误分类对象)，在(b)的情况下，贝叶斯FI选择加速车辆(例如，通过破坏PID输出或规划器输出)。对于DS6，我们观察到1.96%的场景和0.36%的故障是关键的。我们对DS5做了类似的观察。然而，除此之外，我们发现EV在转弯时容易发生故障。在这些情况下，贝叶斯FI选择与消失相对应的故障

表III:基于pgm的故障注入总结。

Driving scenario	Crit. scenes %	Crit. faults %	Hazard rate
DS4 (2400 scenes)	0	0.0	0.0
DS5 (2400 scenes)	0.88	0.20	0.36
DS6 (2400 scenes)	1.96	0.36	0.20

[1] “FIXED”故障模型中的故障总数(TF) = #scenes/DS * #error types = 98400/DS

[2] 紧急场景% = #按#scenes/DS查找到紧急故障的场景

[3] Critical fault % =(贝叶斯FI挖掘的Critical fault)/TF

车道或转向值损坏。当车道标志缺失时，EV倾向于跟随前车。然而，在没有前导车辆跟随的轮转中，这种错误就变得至关重要。*值得注意的是，贝叶斯FI能够在4小时内挖掘关键故障和场景，并在模拟器中模拟所有提取的故障大约需要54小时*。

VII. 相关工作

传统上，AV研究的重点是改进ML/AI技术。然而，随着模型在计算平台上的大规模部署，重点转变为评估驱动AV的计算堆栈的弹性和安全特性，评估自动驾驶的安全性和弹性需要强大的测试技术，这些技术可扩展，并直接适用于现实驾驶场景。单纯基于统计指标(如道路行驶10亿英里)或在CARLA[51]或Open Pilot[5]、[56]、[57]等平台上进行的模拟来进行安全性论证，是不可扩展的，也是不实用的。事实证明，测试ADS的稳健性具有挑战性，而且大多是临时的或基于经验的[17]。特别是，为了测试ADS硬件和软件组件的功能和设计，目前的方法依赖于将无效或受干扰的输入[28]，[31]，[32]或故障和错误[24]，[31]，[58]注入模拟ADS或ADS组件中，并在道路上累积数百万英里[59]。

然而，这些方法是不可扩展的，因为(a)它们缺乏模拟或真实的数据集，这些数据集将代表各种驾驶场景[56];(b)需要数十亿英里的驾驶里程来添加功能或进行bug修复，才能驱动统计测量[60];(c)它们仅限于深度神经网络[24]，[57]，[61]-[63]和传感器[31]，[32]，即使深度神经网络仅占整个生态系统的一小部分;(d)一旦简单的bug被修复，发现罕见的危险事件的成本将呈指数级增长，因为故障可能只在特定条件下(例如，某一软件状态)才会出现。

VIII. 结论

在这项工作中，我们介绍了故障注入工具DriveFI，以及经验评估ADS故障传播、弹性和安全特性的方法，以及生成和测试角落案例故障条件的方法。DriveFI结合了贝叶斯和传统的FI框架，它们协同工作以加速发现安全关键故障。

致谢

本材料是基于美国国家科学基金会(NSF)资助的工作。中枢神经系统15 - 45069。我们感谢K. Atchley、J. Applequist、K. Saboo和S. Cui。我们也要感谢NVIDIA公司的软件访问和设备捐赠。本材料中表达的任何观点、发现、结论或建议均为作者的观点，并不一定反映NSF和NVIDIA的观点。

参考文献

[1] S. M. Erlien, "Shared vehicle control using safe driving envelopes for obstacle avoidance and stability," Ph.D. dissertation, Stanford University, 2015.

[2] J. Suh, B. Kim, and K. Yi, "Design and evaluation of a driving mode decision algorithm for automated driving vehicle on a motorway," *IFAC-PapersOnLine*, vol. 49, no. 11, pp. 115–120, 2016.

[3] Nvidia, "nvidia drive | nvidia developer," https://developer. nvidia. com/driveworks.

[4] Baidu, "Apollo Open Platform," http://apollo.auto, Accessed: 2018-09-02.

[5] CommaAI, "OpenPilot: Open source driving agent," https://github. com/commaai/openpilot, Accessed: 2018-09-12.

[6] S. Alvarez, "Research group demos why Tesla Autopilot could crash into a stationary vehicle," https://www. teslarati. com/tesla-research-group-autopilot-crash-demo/, June 2018.

[7] T.S., "Why Uber's self-driving car killed a pedestrian," The Economist May 29, 2018 https://www. economist. com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian.

[8] S. S. Banerjee *et al.*, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *Proc. 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.

[9] C. Fan *et al.*, "DryVR: Data-Driven Verification and Compositional Reasoning for Automotive systems," in *Computer Aided Verification*. Springer International Publishing, 2017, pp. 441–461.

[10] E. M. Clarke and O. Grumberg, "The model checking problem for concurrent systems with many similar processes," in *Proc. Temporal Logic in Specification*, 1987, pp. 188–201. [Online]. Available: https://doi.org/10.1007/3-540-51803-7_26

[11] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach," in *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*, 1983, pp. 117–126. [Online]. Available: https://doi.org/10.1145/567067.567080

[12] J. R. Bitner *et al.*, "Efficient algorithmic circuit verification using indexed bdds," in *Digest of Papers: 24th Symposium on Fault-Tolerant Computing*, 1994, pp. 266–275. [Online]. Available: https://doi. org/10. 1109/FTCS. 1994. 315633

[13] J. Shen and J. A. Abraham, "Native mode functional test generation for processors with applications to self test and design validation," in *Int. Proc. Test Conference*. IEEE, 1998, pp. 990–999.

[14] R. K. Roy *et al.*, "Compaction of atpg-generated test sequences for sequential circuits," in *Digest of Technical Papers, 1988 IEEE International Conference on Computer-Aided Design*, 1988, pp. 382–385. [Online]. Available: https://doi.org/10.1109/ICCAD.1988.122533

[15] I. Hamzaoglu and J. H. Patel, "Deterministic test pattern generation techniques for sequential circuits," in *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design*, 2000, pp. 538–543. [Online]. Available: https://doi.org/10.1109/ICCAD.2000.896528

[16] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, April 1997.

[17] L. Fraade-Blanar *et al.*, "Measuring automated vehicle safety," 2018.

[18] M. T. Review, "many cars have a hundred million lines of code," https://www. technologyreview. com/s/508231/many-cars-have-a-hundred-million-lines-of-code/.

[19] A. Hawkins, "Nvidia says its new supercomputer will enable the highest level of automated driving," The Verge Oct. 10, 2017 https://www. theverge. com/2017/10/10/16449416/nvidia-pegasus-self-driving-car-ai-robotaxi.

[20] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011, pp. 365–376.

[21] T. Karnik and P. Hazucha, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[22] M. Musuvathi *et al.*, "Finding and reproducing heisenbugs in concurrent programs." in *OSDI*, vol. 8, 2008, pp. 267–280.

[23] S. K. S. Hari *et al.*, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 249–258.

[24] G. Li *et al.*, "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *Proc. International Conf. for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 8:1–8:12.

[25] NHTSA, "Automated Driving Systems: A Vision for Safety," https://www. nhtsa. gov/sites/nhtsa. dot. gov/files/documents/13069a-ads2. 0_090617_v9a_tag. pdf, 2017.

[26] A. Abdulkhaleq *et al.*, "A Systematic Approach Based on STPA for Developing a Dependable Architecture for Fully Automated Driving Vehicles," *Procedia Engineering*, vol. 179, pp. 41–51, 2017.

[27] N. Leveson, "A new accident model for engineering safer systems," *Safety science*, vol. 42, no. 4, pp. 237–270, 2004.

[28] K. Pei *et al.*, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[29] B. Salami, O. Unsal, and A. Cristal, "On the Resilience of RTL NN Accelerators: Fault Characterization and Mitigation," *arXiv preprint arXiv: 1806.09679*, 2018.

[30] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 17.

[31] S. Jha *et al.*, "AVFI: Fault Injection for Autonomous Vehicles," in *Proc. 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 55–56.

[32] A. H. M. Rubaiyat, Y. Qin, and H. Alemzadeh, "Experimental re-silience assessment of an open-source driving agent," *arXiv preprint arXiv:1807.06172*, 2018.

[33] K. J. Åström and T. Hägglund, *PID controllers: theory, design, and tuning*. Instrument Society of America Research Triangle Park, NC, 1995, vol. 2.

[34] S. M. Erlien, S. Fujita, and J. C. Gerdes, "Safe driving envelopes for shared control of ground vehicles," *IFAC Proceedings Volumes*, vol. 46, no. 21, pp. 831–836, 2013.

[35] S. J. Anderson, S. B. Karumanchi, and K. Iagnemma, "Constraint-based planning and control for safe, semi-autonomous operation of vehicles," in *Proc. 2012 IEEE Intelligent Vehicles Symposium (IV), 2012 IEEE*, pp. 383–388.

[36] A. Avizienis *et al.*, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[37] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal pfrocessing, sensor fusion, and target recognition VI*, vol. 3068. International Society for Optics and Photonics, 1997, pp. 182–194.

[38] J. Pearl, "Theoretical impediments to machine learning with seven sparks from the causal revolution," 2018.

[39] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[40] P. L. DeVries and P. Hamill, "A first course in computational physics," 1995.

[41] D. Koller *et al.*, "Towards robust automatic traffic scene analysis in real-time," in *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, vol. 1. IEEE, 1994, pp. 126–131.

[42] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

[43] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (methodological)*, pp. 1–38, 1977.

[44] SAE International, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, Sep 2016.

[45] A. B. Watson and J. A. Solomon, "Model of visual contrast gain control and pattern masking," *JOSA A*, vol. 14, no. 9, pp. 2379–2391, 1997.

[46] P. Debevec and S. Gibson, "A tone mapping algorithm for high contrast images," in *13th Eurographics Workshop on Rendering: Pisa, Italy*. Citeseer, 2002.

[47] D. Menon and G. Calvagno, "Color image demosaicking: An overview," *Signal Processing: Image Communication*, vol. 26, no. 8-9, pp. 518–533, 2011.

[48] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[49] D. Reid *et al.*, "An algorithm for tracking multiple targets," *IEEE Transactions on Automatic Control*, vol. 24, no. 6, pp. 843–854, 1979.

[50] A. Houenou *et al.*, "Vehicle trajectory prediction based on motion model and maneuver recognition," in *Intelligent Robots and Systems, Proc. 2013 IEEE/RSJ International Conference*, pp. 4363–4369.

[51] A. Dosovitskiy *et al.*, "CARLA: An open urban driving simulator," in *Proc. of the 1st Annual Conf. on Robot Learning*, 2017, pp. 1–16.

[52] NVIDIA, "NVIDIA Drive Simulation," https://www. nvidia. com/en-us/self-driving-cars/drive-constellation/, Accessed: 2018-09-02.

[53] Nvidia, "Drive Pegasus," https://www. nvidia. com/en-us/self-driving-cars/drive-platform/, Accessed: 2018-09-12.

[54] NEOUSYS, "nuvo-6108gc gpu computing platform | nvidia rtx 2080-gtx 1080ti-titanx," https://www. neousys-tech. com/en/product/application/gpu-computing/nuvo-6108gc-gpu-computing, Accessed: 2018-11-28.

[55] M. Waskom, "seaborn.boxplot," https://seaborn.pydata.org/generated/seaborn.boxplot.html.

[56] J. M. Anderson *et al.*, "Autonomous vehicle technology: A guide for policymakers," RAND Corp., Tech. Rep. RR-443-2-RC, 2016.

[57] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182– 193, 2016.

[58] S. Jha *et al.*, "Kayotee: A Fault Injection-based System to Assess the Safety and Reliability of Autonomous Vehicles to Faults and Errors," in *Third IEEE International Workshop on Automotive Reliability & Test*. IEEE, 2018.

[59] Waymo, "On the Road to Fully Self-Driving," Waymo Safety Report https://assets.documentcloud.org/documents/4107762/Waymo-Safety-Report-2017.pdf, Accessed: 2017-11-27.

[60] P. Koopman and M. Wagner, "Toward a framework for highly automated vehicle safety validation," SAE Technical Paper, Tech. Rep., 2018.

[61] J. Lu *et al.*, "No need to worry about adversarial examples in object detection in autonomous vehicles," *arXiv preprint arXiv:1707. 03501*, 2017.

[62] K. Pei *et al.*, "Towards practical verification of machine learning: The case of computer vision systems," *arXiv preprint arXiv:1712.01785*, 2017.

[63] H. Lakkaraju *et al.*, "Identifying unknown unknowns in the open world: Representations and policies for guided exploration." in *AAAI*, vol. 1, 2017, p. 2.