arXiv:2603.02194v1 [cs.CV] 2 Mar 2026

从排行榜到部署：自动驾驶感知代码仓库中的代码质量挑战

Mateus Karvat Camara[1],Bram Adams[1],和 Sidney Givigi[1]

摘要——**自动驾驶汽车（AV）感知模型通常仅依据基准性能指标进行评估，而对代码质量、生产就绪性和长期可维护性关注有限。这在研究卓越性与受国际安全标准约束的安全关键系统的实际部署之间造成了显著差距。为弥补这一差距，我们提出了首个针对自动驾驶感知代码仓库软件质量的大规模实证研究，系统分析了来自KITTI和NuScenes 3D目标检测排行榜的178个独特模型。借助静态分析工具（Pylint、Bandit和Radon），我们评估了代码错误、安全漏洞、可维护性和开发实践。研究结果表明，在所研究的代码仓库中，仅有**7.3%**满足基本生产就绪性标准，即不存在严重错误且没有高严重性安全漏洞。安全问题高度集中，前五大问题几乎占全部出现次数的**80%**，这促使我们制定了一套可操作的指南以预防这些问题。此外，持续集成/持续部署流水线的采用与更好的代码可维护性相关。我们的研究结果表明，排行榜表现并不能反映生产就绪性，而有针对性的干预措施可以显著提升自动驾驶感知代码的质量与安全性。**

索引词—**自动驾驶车辆、感知模型、自动驾驶车辆安全、软件安全、软件质量。**

I. 引言

感知模型是自动驾驶车辆（AVs）系统流程中的关键组件，使其能够理解周围环境并做出安全关键决策。尽管自动驾驶汽车感知领域的研究发展迅速，模型在 KITTI [1] 和 NuScenes [2], 等基准排行榜上表现出色，但从研究代码过渡到可用于生产的系统仍然面临挑战。生产部署不仅要求高准确率，还要求代码易于维护、安全可靠，并且不含可能危及真实场景安全的严重错误。

最先进的感知模型通常依据其检测准确率进行评估，排行榜排名则是衡量成功与否的主要指标之一。然而，这些排名并不评估代码质量：表现顶尖的模型可能拥有出色的检测指标，但其代码库可能缺乏文档、充斥安全漏洞，或容易出现严重的运行时错误。因此，研究卓越性与生产就绪性之间存在显著差距，阻碍了将高性能研究模型用于生产，往往需要从头重新实现。

[1]女王大学计算机学院。加拿大安大略省金斯顿。mateus.karvat@queensu.ca, bram.adams@queensu.ca, sidney.givigi@queensu.ca

尽管代码质量对自动驾驶汽车系统至关重要，但目前尚无大规模实证研究对自动驾驶感知代码仓库的部署就绪性进行系统分析。鉴于自动驾驶车辆属于安全关键系统，必须满足严格的质量标准 [3]–[5], ，了解感知模型当前的代码质量状况，对于弥合研究与生产之间的差距至关重要。

受此启发，我们对自动驾驶汽车感知模型开展了首项全面的软件质量分析，研究了来自 KITTI 和 NuScenes 三维目标检测排行榜的 178 个独立代码仓库。我们使用静态分析工具 Pylint [6],Bandit [7],和 Radon [8], 从多个质量维度评估各代码仓库：代码错误、安全漏洞、可维护性和开发实践。我们研究了代码度量指标与团队特征之间的关系，以了解哪些因素有助于提升代码库质量，并分析了最常见的问题，以制定预防指南。

我们的分析揭示了当前自动驾驶感知代码面临的重大质量挑战。只有 7.3% 的代码仓库达到生产就绪标准，该标准定义为不存在严重错误和高严重性安全漏洞。此外，安全问题高度集中，排名前五的问题占所有出现次数的近 80%。

我们的主要贡献如下：

- 首项针对自动驾驶感知模型软件质量的大规模实证研究，分析了来自两大三维目标检测排行榜的 178 个代码仓库。
- 制定针对最常见安全问题的指南，这些问题占所有出现问题的 80%。
- 关于开发实践的见解表明，采用持续集成/持续交付（CI/CD）与更高的代码质量相关，尽管研究界很少采用这种做法。

我们的研究结果凸显了自动驾驶感知模型在研究卓越性与生产就绪性之间存在的显著差距。通过识别常见的质量问题及其根本原因，我们为致力于弥合这一差距、开发适用于安全关键型自动驾驶系统部署的感知模型的研究人员和从业者提供了可行的指导。

II. 相关工作

ISO 26262 [4], 等安全标准规定了安全关键型汽车软件安全部署的要求，要求采用编码指南、静态分析和系统化测试。然而，这些标准是为确定性系统设计的，未能充分

应对机器学习组件带来的挑战 [9]。ISO 21448（预期功能安全） [5] 旨在作为补充，通过应对感知和决策功能不足所引发的安全风险，而非硬件或软件故障所致的风险，完善相关标准。这些标准共同确立了这样的要求：自动驾驶汽车感知系统的软件质量不仅是开发便利性问题，更是一项安全要求。

Koopman 和 Wagner [10]指出，仅进行系统级测试不足以确保自动驾驶汽车的安全，严格的软件开发流程至关重要。Tabani 等人 [11], 的研究也体现了这一关切，他们评估了百度Apollo 自动驾驶框架对 ISO 26262 软件指南的遵循情况。分析揭示了显著的合规性差距，尤其是在依赖 GPU 加速深度学习库的模块中；这些库并非为安全关键型应用而设计。这凸显了一个更广泛的问题：自动驾驶汽车软件经常基于并非为汽车领域设计的组件构建。尽管他们的研究聚焦于单一量产框架，我们的工作则研究了这类系统通常以之为基础的研究代码仓库。

近来，Cheng 等人 [12] 使用 CodeQL 分析了自动驾驶平台代码仓库中的安全漏洞，发现漏洞严重性与系统性能之间存在直接关联。Kochanthara 等人 [13]依据 ISO 26262 和 ISO 21448 为自动驾驶汽车感知系统推导了安全需求，表明基于机器学习的感知流水线会引入特定风险，需要采取有针对性的缓解策略。

这些研究证实，感知系统中的代码级质量问题会带来实际的安全影响，但都没有对感知模型研究代码仓库进行大规模定量评估，而这些代码仓库往往是生产实现的起点。据我们所知，尚无先前研究对公开可用的自动驾驶汽车感知模型代码仓库进行系统性的大规模代码质量分析，考察其中的错误、安全漏洞和可维护性；本研究正是为了弥补这一空白。

III. 数据集

我们从 KITTI [1] 和 NuScenes [2] 3D 目标检测排行榜中选取了代码仓库。由于 KITTI 排行榜按三种目标类别（汽车、行人和骑行者）分别排名，我们纳入了任一类别中列出的所有模型，并剔除了在多个类别中重复出现的模型。我们于 2025年11月收集数据，初始样本包括 KITTI 排行榜上的 389 个模型和 NuScenes 排行榜上的 330 个模型。

随后，我们对这些模型进行了筛选，首先移除了没有代码仓库链接的模型，最终保留 KITTI 的 164 个模型和 NuScenes 的 134 个模型。鉴于这两个排行榜是面向自动驾驶车辆规模最大的三维目标检测排行榜，我们发现，竟有 421 个模型没有任何代码仓库链接，这令人惊讶。这意味着，近 60% 的模型无法轻易复现其结果，且其复用受到严重阻碍。

两个模型池被合并，并移除了同时出现在两个排行榜上的两个重复项。随后我们验-证了所有代码仓库，使用我们的数据抓取脚本，因无效链接（404 错误或已删除的代码仓库）排除了 11 个模型。接下来，我们移除了 72 个模型变体（例如 UVTR-Camera 和 UVTR-LiDAR），因为这些模型变体共享同一代码库。此后，我们又移除了四个链接有效但数据无法访问的代码仓库，原因是代码仓库为空、已归档或设为私有。

经过这些步骤后，剩下 209 个代码仓库。随后，我们根据其中的 Python 文件评估了这些代码仓库的规模。尽管剔除了初始集合中约 70% 的代码仓库，但剩余代码仓库中仍有 13.4% 不含 Python 代码，且仅由 Markdown 文件组成（这些文件在 GitHub 上用作 README/文档）。因此，我们剔除了 28 个仅包含 Markdown 文件的代码仓库，另有 3 个因链接重复或文件名无效而被剔除。

最终数据集包含 178 个代码仓库，规模从 600 到 184.9k 源代码行数（SLOC）不等（平均值为 23.1k，中位数为 14k，四分位距: 7.4k–23.3k）。

IV. 方法

为评估自动驾驶感知代码仓库的代码质量与安全性，我们采用了静态分析工具 Pylint [6](v2.16.2)、Bandit [7] (v1.8.6) 和 Radon [8](v6.0.1)。

Pylint 是一款流行的 Python 代码检查工具，用于分析代码以识别错误、落实编码规范并提供重构建议。我们使用它来识别代码仓库中的代码错误，并特别关注那些会阻止代码执行或导致运行时崩溃的错误，我们将这类错误定义为严重错误。这些错误包括导入错误、名称错误、语法错误、类型错误和逻辑错误。

Bandit 用于识别安全问题；这些问题对生产环境软件至关重要，但在研究代码中却常常被忽视。对于每个代码仓库，我们收集了安全问题的总数以及被归类为高严重性的数量。我们还分析了 Bandit 的报告，以提取每种问题类型的代表性示例，并收集严重性分类。

Radon 用于评估每个代码仓库的源代码行数总量，以及可维护性指数（可维护性指数） [14], 这一表明代码维护与修改难易程度的指标。

由于我们的数据集呈现非正态分布且包含异常值，我们采用斯皮尔曼相关系数来量化连续变量之间的关系，因为该方法对此类数据具有稳健性。为比较独立组别之间的分布，我们使用了曼-惠特尼 U 检验（$\alpha = 0.05$），这是独立样本 t 检验的一种非参数替代方法，不假定数据服从正态分布。

最后，为了更好地了解团队特征和开发实践如何影响代码质量，我们使用 GitHub API 收集了代码仓库指标、贡献者数量、问题统计数据和受欢迎度指标（星标和复刻数）。我们还评估了持续集成/持续交付流水线的采用情况（ GitHub Actions 或 Travis CI）以及测试基础设施等开发实践，因为这些代表着可能影响错误检测和代码可维护性的系统性质量保证机制。

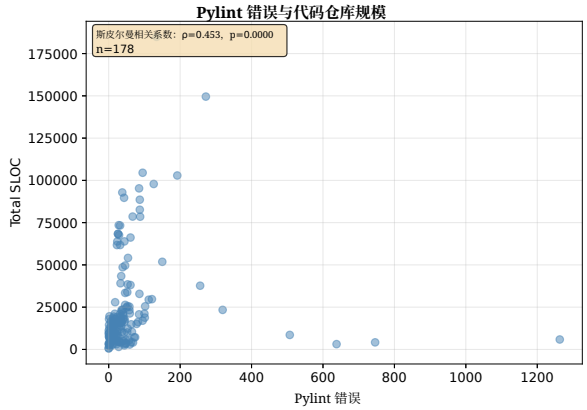

图 1。错误与代码仓库规模之间的关系（以 S LOC 衡量）。

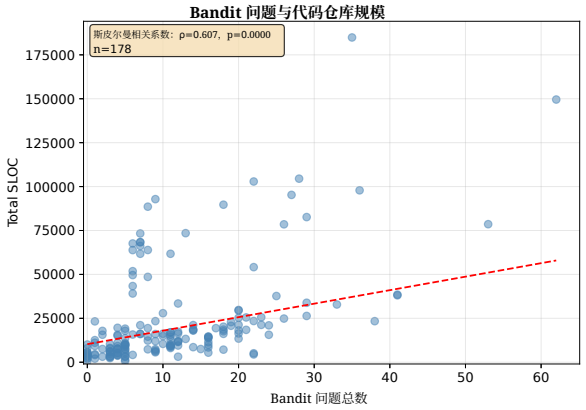

Fig. 2. 代码仓库规模与安全问题数量的相关性 LOC）。 。

V. 结果

A. 生产就绪性

鉴于自动驾驶车辆属于安全关键系统，我们参考了关键系统的软件指南 [3] 和机器学习生产就绪框架 [15] 来定义生产就绪度标准，并将其设定为不存在严重错误且没有高严重性的安全问题。根据这些标准，**仅7.3% 的代码仓库符合可用于生产的标准**（178 个代码仓库中有 13 个）。大多数符合生产就绪标准的代码仓库的源代码行数低于数据集的中位数（14k），仅有三个例外（14.5k、17.9k 和 51.8k）。

关于错误，97.2% 的代码仓库至少存在一个错误，只有五个代码仓库没有错误。错误数量的中位数为 29 个（平均值：55.7 个），范围为 0 到 1,263 个（四分位距：10–52）。比较代码仓库规模后，我们发现无错误代码仓库明显较小。我们的数据集中，源代码行数的总体平均值为 23.1k，而无错误代码仓库平均仅有 5k 行，相比之下，存在错误的代码仓库平均有 23.6k 行。

图 1 证实了代码仓库规模与错误数量之间的关系，显示两者之间存在正向且具有统计显著性的相关性。因此，错误预防机制必须随代码库规模而扩展，但正如 V-B 节所示，即使是基本的安全实践也很少得到采用。

关于安全问题，93.3% 的代码仓库至少存在一个漏洞，只有 12 个代码仓库没有任何问题。安全漏洞在 Python 生态系统中普遍存在，46% 的 PyPI 软件包至少存在一个问题 [16],；不过，自动驾驶感知代码仓库中观察到的较高比例，可能反映了研究代码实践更重视功能而非安全性。每个代码仓库的安全问题数量为 0 至 62 个，中位数为 9 个（四分位距：4–16）。在确认代码错误会随代码仓库规模增大而相应增加后，我们进一步验证了安全漏洞是否也呈现类似模式。图 2 显示，安全问题数量同样与代码仓库规模相关。我们发现，没有安全问题的代码仓库规模较小，平均约为 4.5k 源代码行数。

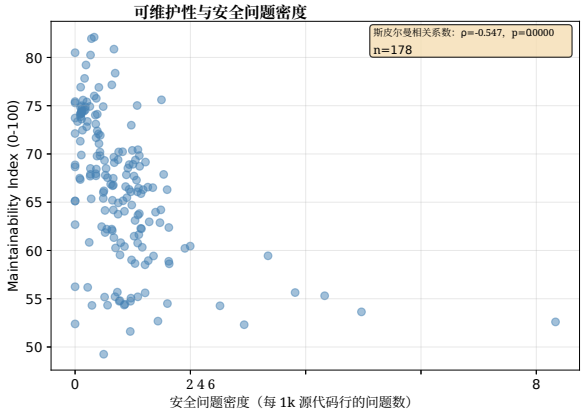

图 3. 可维护性与安全问题密度之间的相关性。

如图 3 所示，MI 与安全问题密度呈负相关。错误密度也存在这种关系（$\rho = -0.397$, $p < 0.001$），这表明可维护性较高的代码仓库，每千行源代码中的错误和安全漏洞往往更少。

错误和安全问题都会随着代码库规模的扩大而增加，这表明需要采取有针对性的干预措施来防止这些问题，并使自动驾驶汽车感知代码仓库达到生产就绪状态。具体而言，在错误方面，Pylint 共识别出 1，612 个错误，其中 1，424 个属于严重错误，且 90.4% 的代码仓库至少存在一个严重错误。防止这些错误相对直接：Pylint 等代码检查工具可以直接集成到常用的集成开发环境中，还可以配置 CI/CD 流水线，阻止包含严重错误的代码提交。

B. 安全漏洞预防

安全漏洞带来的挑战则不那么容易应对。与静态错误不同，如果开发者缺乏安全专业知识，就未必总能识别出这些漏洞；而要解决它们，则需要理解每种模式所引入的具体风险。这些知识在安全实践中早已确立，但正如我们的数据所示，在自动驾驶感知研究中却很少得到应用。然而，我们的分析揭示了一个高度可采取行动的集中趋势

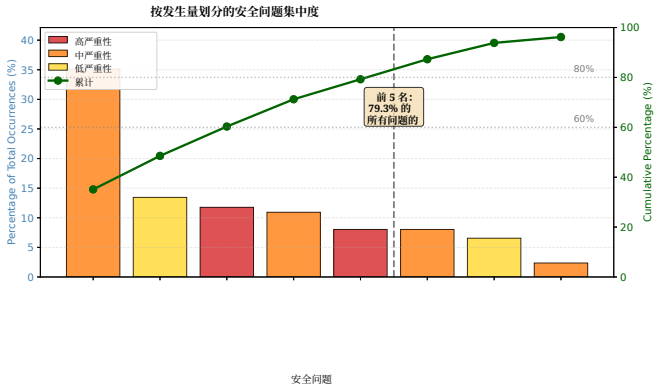

图4. 数据集中安全问题的集中度。并列时按严重性排序。

模式：如图4所示，少数安全问题占了绝大多数出现次数。

表 I 列出了 Bandit 识别出的所有安全问题。最常见的类别包括序列化（影响 83.1% 的代码仓库）、输入验证（62.4%）、Shell 注入（51.7%）和错误处理（39.3%）。在所有安全问题中，448 个（23.1%）严重性较低，1,180 个（58.1%）严重性中等，403 个（19.8%）严重性较高。它们分别影响 55.6%、87.1% 和 51.7% 的代码仓库。

因此，我们调查了这些问题的样本，并为五种最常见的模式制定了预防指南。

1) B614：不安全的 *PyTorch* 模型加载：在自动驾驶汽车感知流水线中，研究团队之间经常共享模型检查点，并在推理过程中加载。这些检查点通过 PyTorch 的 torch.load() 加载，该函数使用 Python 的 pickle 模块进行反序列化。该模块不仅能够存储数据，还能存储用于重建 Python 对象的可执行指令，这意味着在任何用户代码运行之前，加载恶意检查点文件就可能触发任意代码执行。因此，遭到破坏的检查点可能会悄无声息地更改检测阈值或分类边界，导致感知系统有规律地漏检障碍物或误分类道路使用者，且不会出现任何可观察到的错误。此漏洞影响了我们数据集中的大多数代码仓库，只有 34 个未受影响。

易受攻击模式：

× 易受攻击

```
# 允许任意代码执行 检查点 = torch.load('model.pth')
```

安全实现：

✓ 安全

```
# 仅加载张量数据，阻止任意代码 checkpoint = torch.load('
```

2) B110：静默错误抑制： 尽管 Bandit 将此问题归类为低严重性，但它对安全关键型自动驾驶感知尤其危险。静默抑制异常会掩盖严重错误，导致激光雷达点云读取失败、相机帧损坏或传感器同步错误等问题无法被发现。随后，感知系统可能会基于错误数据运行，却向下游模块报告正常状态。

易受攻击模式：

× 易受攻击

```
尝试: model.load_state_dict(checkpoint)除外:通过
```

安全实现：

✓ 安全

```
尝试: model.load_state_dict(checkpoint)除外Exception作为e:
```

3) B605/B602：*Shell* 注入：感知系统通常会调用外部预处理工具来进行点云变换、传感器校准或数据格式转换。在这些流程中使用基于 Shell 的子进程调用会产生注入途径，攻击者可能利用这些途径在传感器数据到达检测模型之前对其进行篡改，从而破坏整个感知链的完整性。

B605 和 B602 都出现在最常见的五种安全问题中，但它们通过不同机制体现了同一种根本漏洞：B605 由 os.system() 及其他隐式调用 Shell 的相关函数触发，而 B602 则由明确设置 shell=True 的子进程调用触发。在这两种情况下，未经清理的字符串输入都会被传递给 Shell 解释器，修复策略也完全相同。

易受攻击模式：

× 易受攻击

```
导入 子进程 cmd = f"rm {user_input}"# 用
```

安全实现：

✓ 安全

```
导入 子进程# 列表语法可防止注入 子进程.
```

4) B307：*eval()* 的不安全使用：在感知系统中，有时会使用 eval() 动态实例化模型架构，或解析指定网络层的配置字符串。在部署环境中，如果攻击者能够修改配置文件或模型定义，就可以在感知进程中执行任意代码，从而可能导致目标检测失效，或在运行时注入虚假检测结果。

表 I：Bandit 检测到的安全问题及其严重性分类和安全类别。

ID	严重性	类别	问题	出现次数	受影响代码仓库	百分比代码仓库
B614	中	序列化	不安全地使用 PyTorch 加载	713	144	80.9
B110	LOW	错误处理	Try、Except、Pass	273	69	38.8
B605	HIGH	Shell 注入	使用 Shell 启动进程，检测到可能存在注入风险	239	79	44.4
B307	中	输入验证	不安全地使用 eval()，可能导致代码注入	222	106	59.6
B602	HIGH	Shell 注入	使用 shell=True 调用子进程	163	28	15.7
B506	中	序列化	使用不安全的yaml加载	163	64	36.0
B311	LOW	密码学	不安全的随机数生成器	133	36	20.2
B102	中	代码执行	检测到使用 exec	48	37	20.8
B106	LOW	身份验证	可能存在硬编码密码：‘predicted’	19	7	3.9
B310	中	输入验证	审查 URL 打开是否使用允许的协议	12	8	4.5
B107	LOW	身份验证	可能存在硬编码密码：‘train’	9	4	2.2
B615	中	依赖项	未固定版本的 HuggingFace 下载	8	2	1.1
B108	中	文件系统	可能不安全地使用临时文件/目录	7	6	3.4
B112	LOW	错误处理	检测到 Try、Except、Continue	5	4	2.2
B105	LOW	身份验证	可能存在硬编码密码：“”	5	3	1.7
B405	LOW	输入验证	正在使用 xml.etree.ElementTree 解析不可信 XML 数据	2	2	1.1
B314	中	输入验证	使用 xml.etree.ElementTree.parse 解析不可信 XML 数据	2	2	1.1
B408	LOW	输入验证	使用 xml.dom.minidom 解析不可信 XML 数据	2	1	0.6
B318	中	输入验证	使用 xml.dom.minidom.parse 解析不可信 XML 数据	2	1	0.6
B104	中	网络	可能绑定到所有接口	2	2	1.1
B324	HIGH	密码学	使用强度较弱的 SHA1 哈希进行安全保护	1	1	0.6
B113	中	网络	调用 requests 时未设置超时	1	1	0.6

禁用目标检测，或在运行时注入虚假检测结果。

易受攻击模式：

× 易受攻击

```
# 从字符串中执行任意代码 eval(args.task)(log_
```

安全实现：

✓ 安全

```
# 使用显式函数映射，而不是 evalTASKS = {'train': train_
```

C. 开发实践

在针对最普遍的安全问题制定了可操作的指南后，我们接下来考察开发实践是否能为提升代码质量提供互补途径。尽管我们从 GitHub 收集了每个

代码仓库大多与错误、安全问题或可维护性没有相关性。最值得注意的是，代码仓库的受欢迎程度并不能反映代码质量，星标数与错误数量之间也没有显著相关性（$\rho = 0.04$, $p = 0.56$）。这意味着，尽管从业者往往更青睐热门代码仓库，但它们并不比不那么热门的代码仓库更可用于生产。唯一与质量存在显著相关性的开发实践是采用持续集成/持续交付。

尽管我们的数据集中只有 7.3% 的代码仓库采用了持续集成/持续交付实践，但如图 5 所示，采用持续集成/持续交付的代码仓库的平均可维护性指数高于未采用的代码仓库。鉴于可维护性指数与较低的错误和安全问题密度相关（第 V-A 节），这一点值得关注。不过，代码仓库

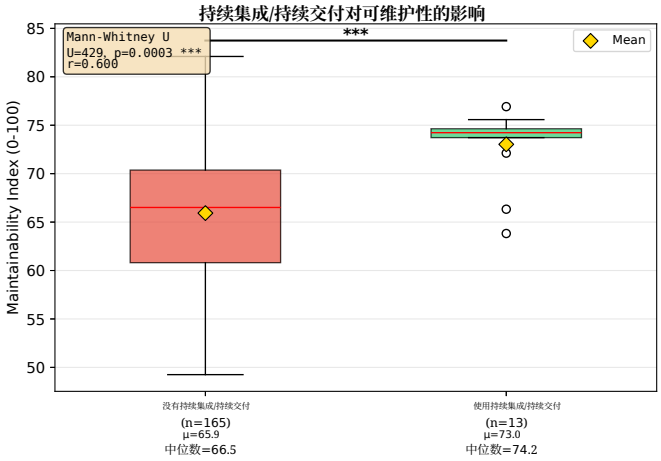

图 5. 采用持续集成/持续交付的代码仓库，其平均可维护性指数显著更高。

采用持续集成/持续交付的代码仓库往往也有更多贡献者（平均为 34.4 人，而非 2.3 人），因此尚不清楚持续集成/持续交付是否会直接影响可维护性指数，还是仅仅因为贡献者数量较多。

我们进一步调查了这一问题，发现无论团队规模如何，采用持续集成/持续交付的代码仓库，其平均可维护性指数都高于未采用者。在所有团队规模中，采用持续集成/持续交付的代码仓库的平均可维护性指数始终较高（69.2 至 74.2），而未采用者为（65.4 至 69.9）；其中，团队规模为中等的团队（$3-5$ 贡献者）差异最大。根据既定的可维护性指数阈值 [14]，这些数值处于中等可维护性范围（$65 \leq MI < 85$），而未采用持续集成/持续交付的代码仓库正接近低可维护性阈值

（$MI < 65$）。这表明，无论团队规模如何，持续集成/持续交付的采用都与较高的可维护性指数相关。然而，由于样本量较小，这一比较不具有统计显著性，仍需开展进一步研究以验证这种关系。

由于我们的数据集中只有 7.3% 的代码仓库采用了持续集成/持续交付，这些发现表明，推广其应用是弥合自动驾驶感知代码研究与生产之间差距的最切实可行途径之一。

VI. 局限性

尽管我们的数据集取自规模最大的两个自动驾驶感知排行榜，但仅以这两个排行榜作为数据来源，对外部效度构成了威胁。我们通过分析大量代码仓库（178 个），力求减轻这一威胁。选择偏差也可能影响了我们的发现，因为我们只分析了排行榜上发布的代码仓库。已发布的自动驾驶感知代码库在成熟度和质量方面可能不同于未发布的代码库。

内部效度方面的威胁包括使用静态分析工具评估代码仓库质量，尤其是 Pylint 的错误检测可能产生假阳性和假阴性。此外还涉及构念效度问题，即我们选择的指标是否确实能够衡量代码质量。尽管错误、安全问题和可维护性指数（MI）被广泛认为是代码质量指标，但它们可能无法涵盖与生产部署相关的软件质量的所有方面。

VII. 结论

我们首次对自动驾驶汽车感知代码仓库进行了大规模软件质量分析，评估了来自最大的自动驾驶三维目标检测排行榜 KITTI 和 NuScenes 的 178 个代码仓库。我们的主要发现如下：

- 代码质量尤其低，只有 2.8% 的代码仓库没有错误，只有 6.7% 的代码仓库不存在安全漏洞。
- 考虑到严重错误和高严重性安全问题的缺失情况，只有 7.3% 的代码仓库被认为可用于生产。
- 我们发现安全问题高度集中，其中五个问题占出现次数的近 80%。
- 虽然分析的代码仓库中只有 7.3% 采用了持续集成/持续交付实践，但我们发现，采用持续集成/持续交付与代码更易维护相关。

这些发现与自动驾驶汽车的部署直接相关，因为 ISO 26262 和预期功能安全（ISO 21448）等安全标准要求系统地使用静态分析和编码指南。研究类感知代码仓库中安全漏洞普遍存在、持续集成/持续交付的采用几乎为零，这表明在将其集成到通过安全认证的自动驾驶系统之前，仍需进行大量整改。

未来的工作可以将我们的分析范围扩展到 3D 目标检测之外，涵盖语义分割、跟踪和协同感知。此外，通过代码执行进行动态分析，有助于深入了解运行时行为和模型质量。

参考文献

[1]A. Geiger、P. Lenz、C. Stiller 和 R. Urtasun，“视觉与机器人学相遇：KITTI 数据集”，《国际机器人研究期刊》，第 32 卷，第 1231–1237 页，2013 年。 [在线]。网址：http://www.cvlibs.net/datasets/kitti。[2] H. Caesar、V. Bankiti、A. H. Lang、S. Vora、V. E. Liong、Q. Xu、A. Krishnan、Y. Pan、G. Baldan、O. Beijbom 和 A. Company，“nuscenes：用于自动驾驶的多模态数据集”，*2020* 年 *IEEE/CVF* 计算机视觉与模式识别会议（*CVPR*），第 11 618–11 628 页，2020 年 6 月。[3] MISRA，“MISRA C:2012 关键系统中 C 语言使用指南”，汽车工业软件可靠性协会，技术报告，2012 年，ISBN 978-1-906400-10-1。[4] ISO，“ISO 26262：道路车辆——功能安全”，国际标准化组织，瑞士日内瓦，技术报告，2018 年。[5] ——，“ISO 21448：道路车辆——预期功能安全（SOTIF）”，国际标准化组织，瑞士日内瓦，技术报告，2022 年。[6] Pylint 开发团队，“Pylint”，https://github.com/pylint-dev/pylint，2025 年，一款 Python 静态代码分析工具，可检查错误、强制执行编码规范并查找代码异味。 [在线]。网址：https://pylint.readthedocs.io/ [7] PyCQA，“Bandit：Python 安全检查工具”，https://github.com/PyCQA/bandit，2025 年，访问于：2025 年 11 月。[8] M. Lacchia，“Radon：Python 代码度量指标”，https://github.com/rubik/radon，2025 年，访问于：2025 年 11 月。[9] R. Salay、R. Queiroz 和 K. Czarnecki，“ISO 26262 分析：机器学习与汽车软件安全”，SAE 技术论文，技术报告，2018 年。[10] P. Koopman 和 M. Wagner，“自动驾驶汽车测试与验证中的挑战”，《*SAE* 国际交通安全期刊》，第 4 卷，第 1 期，第 15–24 页，2016 年。[11] H. Tabani、L. Kosmidis、J. Abella、F. J. Cazorla 和 G. Bernat，“评估工业自动驾驶框架对 ISO 26262 软件指南的遵循情况”，载于*2019* 年第 *56* 届年度设计自动化会议论文集，2019 年，第 1–6 页。[12] W. Cheng、Z. Li、P. Liang、R. Mo 和 H. Liu，“揭示自动驾驶系统中的安全弱点：一项深入的实证研究”，《信息与软件技术》，第 182 卷，文章编号 107709，2025 年。[13] S. Kochanthara、T. Singh、A. Forrai 和 L. Cleophas，“自动化驾驶感知系统的安全性：以 Apollo 为例的案例研究”，《*ACM* 软件工程与方法学汇刊》，第 33 卷，第 3 期，第 1–28 页，2024 年。[14] D. Coleman、D. Ash、B. Lowther 和 P. Oman，“使用度量指标评估软件系统的可维护性”，《*Computer*》，第 27 卷，第 8 期，第 44–49 页，1994 年。[15] E. Breck、S. Cai、E. Nielsen、M. Salib 和 D. Sculley，“ 机器学习测试评分：机器学习生产就绪度与技术债务削减的评估准则”，载于*2017* 年 *IEEE* 国际大数据会议（*Big Data*），2017 年，第 1123–1132 页。[16] J. Ruohonen、K. Hjerppe 和 K. Rindell，“对 PyPI 中 Python 软件包开展的大规模安全导向静态分析”，载于 *2021* 年第 *18* 届隐私、安全与信任国际会议（*PST*）。美国加利福尼亚州洛斯阿拉米托斯：IEEE 计算机学会，2021 年 12 月，第 1–10 页。 [在线]。网址：https://doi.ieeecomputersociety.org/10.1109/PST52912.2021.9647791