使用 MongoDB 日志文件作为训练数据的 MongoDB 注入查询分类模型

Shaunak S. Pernia*，Minal Naresh Shiroadkar b，Ramdas Karmalli c

[a]果阿大学果阿商学院，*Taleigao*，*Durgavado*，果阿，*403206*，印度[b]果阿大学果阿商学院，*Taleigao*，*Durgavado*，果阿，*403206*，印度[c]果阿大学果阿商学院，*Taleigao*，*Durgavado*，果阿，*403206*，印度

摘要

NoSQL 注入攻击是一类网络安全攻击，攻击者会向 NoSQL 数据库发送经过特意构造的查询，从而使数据库执行未经授权的操作。为防御此类攻击，最初开发了基于规则的系统，但后来发现这类系统无法有效应对新型注入攻击，因此开发了基于模型的方法。

大多数基于模型的检测系统在测试期间都取得了指数级的正面结果，但它们仅使用发送到服务器的查询语句进行训练。然而，由于数据稀缺和类别不平衡，这些基于模型的系统被发现无法有效应对现实世界中的所有攻击。本文探讨了如何根据日志数据及其他提取的特征（不包括原始查询语句），对发送到 MongoDB 服务器的 NoSQL 注入攻击进行分类

日志数据采集自针对一台空的 MongoDB 服务器发起的模拟攻击，随后经过处理和探索。研究进行了判别分析，以确定能够区分注入查询与良性查询的统计显著特征，并由此生成了一个包含显著特征的数据集。

研究使用自动化机器学习库“FLAML”构建了若干基于机器学习的分类模型，并手动编写了 6 个模型；这些模型均在该数据集上训练，随后使用 50 个随机数据样本进行训练、交叉验证和评估。研究发现，表现最佳的模型是“FLAML”库中的“有限深度 XGBoost”模型，准确率为 71%。

所有数据集和 Python 笔记本均保存在以下 Git 仓库中：
https://github.com/ShaunakPerniUniGoa/NoSQLInjectionDetection

* 通讯作者电子邮箱：msci.2105@unigoa.ac.in
(Shaunak Perni) ~~电话号码：(+91) 8625946258~~

生成式人工智能内容免责声明

本文使用了生成式人工智能工具，包括 ChatGPT-4、Grammarly 和 Codium，用于代码生成、调试和语言润色等特定用途（例如语法和文风修正）。然而，作者声明，本文的研究、研究结果和核心内容均为其原创成果。

1 背景

2013年，（Son 等人，2013年）介绍了 DIGLOSSIA，这是一种运行时分析工具，用于检测 PHP 应用程序中的 SQL 和 MongoDB NoSQL 注入攻击。该工具通过将数据库查询划分为“代码”和“非代码”分量，然后分析“代码”部分中的用户提供的输入来运行。DIGLOSSIA 成功检测出了针对其基准 PHP 应用程序发起的所有攻击。然而，该工具的效能取决于查询是否严格遵循与数据库解析器相匹配的预定义语法；如果语法存在差异，就可能出现假阳性或假阴性。此外，其值影子机制可能并不完善；如果受输入污染的字符串经由未受监控的第三方扩展或某些 PHP 内置函数处理，就可能导致假阴性。

2015年出现了两项重要成果。（Joseph 和 Jevitha，2015年）开发了一种基于自动机的 NoSQL 注入检测系统，并专门在采用 MongoDB 的基于 Java 的后端上进行了测试。该系统在其测试数据中针对“基于时间”和“盲布尔值”注入攻击达到了 100% 的检测率，作者声称在这些场景下没有假阴性。不过，该系统仅适用于基于 Java 的系统和这些特定攻击类型；作者提出了未来的优化方向，例如使用令牌级自动机来降低复杂度。同年，（Ron 等人，2015年）对 NoSQL 数据库的安全措施进行了更全面的考察，证实尽管查询和访问机制有所更新，这些数据库仍无法免受注入攻击。研究详细介绍了多种攻击技术，包括 PHP数组注入、NoSQL OR JSON注入和 NoSQL JavaScript注入，并指出 JSON表示虽然总体定义明确，但仍会带来新的注入类型。研究得出结论，NoSQL技术领域在全面的安全措施和安全意识方面落后于传统 RDBMS SQL系统。建议的缓解策略包括使用适用于 JSON查询的原生编码工具、DAST/IAST、控制 REST API请求，以及认真落实身份验证和基于角色的访问控制。

进入2016年，Hou等人（2016年）进一步探讨了MongoDB安全措施的成熟度，从代码级角度分析了攻击与防御。该研究再次指出，NoSQL数据库仍然容易受到注入攻击，其JavaScript注入或JSON注入原理与传统SQL注入相似。实验表明，服务器端JavaScript注入可以暴露私人数据。讨论的防御机制包括输入验证（将输入限制为预期格式）和参数化语句（通过参数筛选用户输入）。研究还引入了恶意特征检测的概念，用于评估系统安全性。

其总体目标是提高开发人员和信息安全专业人员对NoSQL安全的认识，并倡导实施安全层。

2017年，(M. 等人，2017) 引入了“NoSQL Racket”，这是一款测试工具，旨在通过比较应用程序代码中 NoSQL 查询的静态结构与其动态运行时语句，检测 NoSQL 注入攻击。该工具利用预定义的“Driverstbl”生成可比较的查询模式，若二者不匹配，则表明存在注入。“NoSQL Racket”成功检测了使用 MongoDB、Cassandra、CouchDB 和 Amazon DynamoDB 的 Web 应用程序中的攻击，在各种注入场景中表现优于商业扫描器。性能测试显示，该工具能够以可接受的水平处理负载。然而，该工具依赖于以 PHP 函数的形式集成到应用程序代码中，因此作为外部扫描工具的通用性有限。其所针对的漏洞范围以及假阳性率等关键指标，均未得到详尽说明；此外，在超高吞吐量环境下的可扩展性也尚未量化。

2018年出现了两项值得关注的研究成果。(Eassa 等人，2018) 提出了 DNIARS（使用 RESTful 服务检测 NoSQL 注入攻击），这是一种用于检测 NoSQL 注入攻击的独立 RESTful Web 服务。DNIARS 对比 NoSQL 语句结构在静态和动态状态下生成的模式，以识别注入。该工具使用 PHP 实现，其主要优势在于检测过程与 Web 应用程序完全分离，从而增强了可移植性和可扩展性。评估结果表明，该工具运行有效，错误率较低（不超过 1%），并声称能够检测所测试的“所有类型”的 NoSQL 注入攻击。然而，其模式比较方法可能容易受到内部表未涵盖的高度多态或经过混淆的攻击变体影响。“所有类型”这一宽泛说法缺乏全面的对抗性测试作为依据，而报告中的低错误率也意味着可能存在尚未量化的假阳性或假阴性。(Sachdeva 和 Gupta，2018) 研究了针对 MongoDB 的基本 NoSQL 注入攻击，分析了尽管数据库采用不同的查询语言，攻击者仍可通过修改注入内容的语法形式使其易受攻击。他们展示了使用 PHP 和 JavaScript 对 MongoDB 发起的基本攻击，并提出将输入验证和参数化语句作为防御方法。此外，他们还提出了基于流程图的恶意特征检测概念。研究者承认，该研究仅关注基本攻击，因此未来还可考察更高级的攻击可能性。

2019年，Ul Islam 等（2019）推出了一种利用监督学习自动检测 NoSQL注入的工具，并声称为此开发了首个带标签数据集。该数据集包含 1004 条 MongoDB 查询（其中 203 条为注入）和 350 条 CouchDB 查询（其中 50 条为注入），通过人工设计特征、利用交叉操作和变异操作扩充查询，并在易受攻击的网站上进行验证而生成。最初设计了 19 个特征，并选出效果最佳的 10 个。研究将问题建模为二元分类，并评估了多种监督学习算法。对于 MongoDB，神经网络分类器取得了最高的平均召回率（92.94%）和 0.9343 的 F2 分数，且该方法展现出与数据库无关的性能。比较研究显示，他们的工具比 Sqreen 高出 36.25%。其局限性包括：由于缺乏公开可用的基准数据集，必须生成合成数据集；关于数据集扩充的文献有限，影响了数据集的初始规模；此外，自动特征设计也面临挑战。

研究建议使用更大的数据集以进一步提升性能，并指出“OR 注入”示例的种类较少。

2020年，Scherzinger 和 Sidortschuck（2020）针对 NoSQL 数据库模式的设计与演化开展了一项实证研究，分析了来自 GitHub 的十个真实 Java 项目。该研究被称为同类研究中规模最大的一项，证实反规范化是一种常见实践，并观察到 NoSQL 模式通常会随着时间推移变得更加复杂，但其变更频率高于关系数据库研究（包含与模式相关变更的提交占比从 2.8% 到超过 30%）。实体类或与模式相关属性的新增是模式变更的主要形式。研究局限包括：通过静态分析识别第三方库中的所有与模式相关属性存在困难，可能低估模式复杂性，以及无法通过程序识别重命名或拆分实体等模式变更。由于研究重点是使用特定对象映射器的 Java 项目，其外部效度也受到质疑，这表明有必要在不同编程语言和 NoSQL 数据存储中开展更广泛的分析。

2021 年是 NoSQL 安全研究成果丰硕的一年。Shachi 等人（2021）对 SQL 和 NoSQL 注入攻击的检测与防御开展了全面综述，详细介绍了多种 NoSQL 注入技术（PHP 恒真式、联合查询、JavaScript 注入、搭载式查询、源违规）以及 MongoDB 特有的问题（哈希注入、密码哈希的弱点，以及旧版本缺乏身份验证和加密机制）。该综述同时考察了机器学习（ML）和非机器学习方法，并指出由于缺乏公开的带标签数据，机器学习模型主要依赖基于特征的监督学习。作为一篇综述，该研究综合了现有知识。与此同时，Mejia-（Mejia-Cabrera 等人，2021）提出了一种基于 JSON 结构构建 NoSQL 查询数据库的新方法，用于实验目的，因为此类数据库通常难以获得。他们评估了六种分类算法，其中神经网络和多层感知机达到了 97.6% 的准确率。Alizadehsani（2021）进一步倡导将人工神经网络应用于 NoSQL 攻击检测，并指出安全问题，尤其是注入攻击，在 NoSQL 领域的发展相对滞后。Babucea（2021）开展了一项综述，重点阐述 SQL 和 NoSQL 数据库之间的关键差异与局限，讨论了它们的演化、模式灵活性、可扩展性以及多样化的数据模型。论文得出结论：尽管 NoSQL 不断发展，但不存在单一的最优选择，两种模型通常会共存。最后，Weeratunga（2021）重点研究了 MongoDB Web 应用程序中的 NoSQL 注入漏洞识别，旨在自动化身份验证绕过、未授权数据访问和 JavaScript 注入的检测过程。该研究试图扩展 OWASP ZAP 动态应用程序安全测试工具，使其支持 NoSQL 注入检测，同时承认现有动态应用程序安全测试工具存在不足。其主要局限在于仅关注 MongoDB 及其特定漏洞。

2019年，(Sajid Ahmed 等，2022) 提出了一种用于服务器端应用程序的双层混合安全防火墙，可实时检测 SQL 和 NoSQL 注入攻击。第一层非机器学习层将传入的请求负载与恶意模式进行匹配，以阻止即时威胁并减轻第二层基于机器学习的层的负载；第二层则分析构造的查询，检测恶意

代码。NoSQL 数据集来源于早期研究（Ul Islam 等，2019），并使用 SMOTEENN 等重采样策略处理类别不平衡问题。针对 NoSQLIA 的特征工程涉及提取 14 个特征，并将其精简为 12 个最优特征。使用 SMOTEENN 后，通过支持向量机、AdaBoost 和随机森林等算法进行性能评估，结果表明准确率较高（部分模型的准确率、精确率、召回率和 F1 分数均达到 100%）。该系统使用 Python/Flask 和 NodeJS/ExpressJS 实现。其主要局限在于目前只能从文本数据中检测恶意代码，未来计划借助深度学习处理文件类型的数据。报告的 100% 准确率可能需要通过在高度多样化或对抗性的真实世界数据集上进一步验证，以确认其泛化能力并防止潜在的过拟合。Praveen 等提出了一种基于监督式文本分类的 NoSQL 注入检测模型，将监督学习与自然语言处理技术相结合。该模型面向多种 NoSQL 数据库（MongoDB、CouchDB、CassandraDB、Couchbase），通过 10 折交叉验证取得了 0.95 的 F1 分数。然而，摘要没有详细说明具体的注入类型或假阳性率；该模型抵御新型或高度混淆攻击模式的能力，仍有待进一步深入考察。

最近，2024年，D l 等，2024）.....推出了“MongoDB 注入数据集”，这是一个全面收集 MongoDB NoSQL 注入尝试和漏洞的数据集。鉴于非结构化数据库面临的漏洞日益增多，他们通过整合手动编写的命令，以及从 Nosqlmap 和 OWASP 等来源通过网页抓取得到的命令，整理出 400 条 NoSQL 注入命令（221 条恶意命令、179 条良性命令）。通过数据增强和优化（使用杰卡德相似度 > 80% 筛选命令），确保了数据的多样性。该数据集采用 JSON 格式，包含“文本”（载荷）和“标签”（恶意/良性）两个分量，涵盖复杂和简单的命令，因此适用于机器学习和数据分析，尤其适合用于训练或微调人工智能模型和大语言模型，以提高攻击检测的准确率。其公认的局限性包括专门针对 MongoDB 数据库进行定制，这意味着它在直接应用于其他类型的 NoSQL 数据库时可能存在局限，或无法全面覆盖相关内容。

2 引言

本文探讨了如何创建模型，将 MongoDB 会话的日志行分类为良性或注入攻击。由于缺乏专用于此目的的现有日志数据带标签数据集，因此生成了一个合成数据集。具体而言，研究人员将一组注入查询和良性查询在空的 MongoDB 服务器上执行，随后提取相应的日志数据。提取出的日志数据经过探索阶段分析，以识别并仅保留显著变量，最终得到的数据集仅由标签和这些判别特征构成。

随后在这些经过处理的数据上训练分类模型。建模方法采用了两组不同的模型：六个手动编程的机器学习模型，以及由 FLAML 自动机器学习库确定的一个模型。为评估可泛化性能，六个手动编程的模型使用同一数据集进行训练、验证和测试，但采用了不同的随机种子。该方法产生了不同的训练、验证和测试数据组合，并据此

评估每个模型的平均性能。FLAML 自动机器学习模型拥有用于模型选择和性能确定的内部系统，可直接提供性能指标；随后将这些指标与六个手动编程模型的平均性能进行比较。

3 数据收集

（有关代码及输出，请参阅代码仓库中的 *./notebooks/dataProcessing.ipynb*）

在一台运行 Fedora 39 Linux、内核版本为 6.5 的机器上搭建了一个空的 MongoDB 6.0.15 服务器。该机器配备 AMD Ryzen 7 57000U 处理器、8.0 GiB 内存和 1TB HDD 存储，并使用 Mongosh 2.25（一款用于与 MongoDB 数据库交互的程序）向数据库发送指令。服务器的性能分析级别设为 2，这是允许在日志中记录最多数据的最高级别。

本文使用数据集（D l 等，2024）（以下简称“查询数据集”），将数据集中的每条查询发送至 MongoDB 数据库并执行。执行完所有查询后，提取日志文件。保留日志文件中与所发送查询对应的行，并删除其余无关行，因为这些行与本研究所需的数据无关。随后，将剩余数据从非结构化 JSON 格式转换为表格形式，本文将其称为“日志数据”。

由于 JSON 数据的特性，每个嵌套键都被进一步展开，直到每个键成为一个独立的特征。使用查询数据集中的“Text”变量和日志数据中的“filter”列，将查询数据集中的目标属性“label”与日志数据关联起来。

根据日志数据中的“Filter”变量（其中包含原始查询字符串），进一步提取了更多特征，具体依据如下：

- 根据 MongoDB 规范，筛选器中存在的运算符类别 • 根据 MongoDB 规范，筛选器中存在的选择器类型 • 是否存在空操作数 • 查询的长度 • 仅包含 MongoDB 关键词、移除数据库变量名称后的查询，以及该查询的长度

添加这些工程化变量后，所有常量变量均被移除。

T收集数据的最终结构如下表所示（表 1).

字段	显示名称	数据类型	描述
t	时间戳	时间戳	日志的时间戳 ISO-8601 格式的消息。
计划摘要	计划摘要	字符串	用于执行 查询的计划。
规划时间（微秒）	规划时间（位于	浮点数	开发所花费的时间

	微秒		以微秒为单位的查询计划。
CPU纳秒数	CPU 纳秒	整数	CPU 处理时间，以 纳秒。
过滤 器	筛选器	字符串	查询中使用的筛选器。
$eq	$EQ	布尔值	是否存在 $eq 运算符 于查询筛选器中。
$gt	$GT	布尔值	是否存在 $gt 运算符 于查询筛选器中。
$in	$IN	布尔值	$in 运算符是否 存在于查询筛选器中。
$ne	$NE	布尔值	$ne 运算符是否 存在于查询筛选器中。
$nin	$NIN	布尔值	查询筛选器中是否存在 $nin 运算符 存在于查询筛选器中。
$type	$TYPE	布尔值	是否存在 $type 运算符 存在于查询筛选器中。
$mod	$MOD	布尔值	$mod 运算符是否 存在于查询筛选器中。
$regex	$REGEX	布尔值	是否存在 $regex 运算符 于查询筛选器中。
$ where	$WHERE	布尔值	是否存在 $where 运算符 于查询筛选器中。
$elemMatch	$ELEM _MATCH	布尔值	是否存在 $elemMatch 运算符 查询筛选器中。
$size	$SIZE	布尔值	查询筛选器中是否存在 $size 运算符 存在于查询筛选器中。
$	位置运算符	布尔值	是否使用了位置 $ 运算符。
>=	大于或等于布尔值		查询是否使用了 >= 比较。
<=	小于或等于	布尔值	查询是否使用了 <= 比较。
<	小于	布尔值	查询是否使用 < 比较。
>	大于	布尔值	查询是否使用 > 比较。
选择器_比较	选择器比较	布尔值	是否为比较选择器

			在查询中使用。
选择器_逻辑	逻辑选择器	布尔值	是否使用了逻辑选择器 在查询中使用。
选择器元素_	元素选择器	布尔值	是否在查询中使用了 元素选择器。
选择器评估_	选择器评估	布尔值	是否使用求值选择器 于查询中。
选择器_数组	选择器数组	布尔值	是否使用了数组选择器 进行查询。
选择器 按位运算_	选择器 按位运算	布尔值	是否使用了按位选择器 进行查询。
投影	投影	布尔值	是否使用了投影选择器 在查询中使用。
misc	杂项	布尔值	是否使用杂项选择器 在查询中使用。
选择器	选择器	布尔值	是否有任何选择器 运算符用于 查询中。
标准_逻辑	标准逻辑	布尔值	是否使用标准逻辑 运算符 查询。
所有_运算符	所有运算符	布尔值	是否所有运算符都 包含在查询分析中。
空_操作数	空操作数	布尔值	查询中是否使用了空操作数 在查询中使用的。
正则表达式_空_操作数	正则表达式空操作数	布尔值	是否使用基于正则表达式的空 操作数。
text	Text	字符串	筛选查询的原始形式
查询_长度_原始	查询长度（原始）	整数	查询字符串的长度，如 上所述。
关键词_仅	仅关键字	字符串	查询字符串，但其中包含 仅有 MongoDB 查询 关键词
查询_长度_关键词 s_仅	查询长度（关键词 仅）	整数	提取关键词后的查询长度 仅提取关键词。
标签	标签	布尔值	分配给查询的标签 用于识别或 分类。

表1：收集数据的最终结构

请注意，由于技术限制，自动化处理后仍有一些恒定列，即*“cursorExhausted”*、*“queryFramework”*和*“reslen”*，这些列在人工评估时被移除

4 数据探索

数据收集和处理完成后，对数据集进行了分析。所有文本类变量都被移除，只保留了数值变量和布尔/隐藏变量，因为这正是本项目的研究范围。随后进行了判别分析，以确定能够区分注入查询和良性查询的显著变量。

4.1 数值特征判别分析

数值特征“planningTimeMicros”、“cpuNanos”、query_length_raw”和“query_length_keywords_only”根据标签变量划分为两个样本，即每个变量分别对应注入查询和良性查询的两个分布。为了确定这些特征的分布是否存在显著差异，也就是注入查询与良性查询在这些特征上是否存在差异，使用直方图（图1）和核密度估计图（图2）对各特征的分布进行了可视化。为确认这些样本彼此独立，采用了曼-惠特尼 U 检验，其原假设为该特征的样本均值彼此不独立，

选择曼-惠特尼检验的原因如下：

1. 没有信息表明这些样本来自同一总体 2. 没有信息表明这些样本彼此相关 3. 没有信息表明这些样本服从正态分布，也没有信息表明它们所代表的总体服从正态分布

由于曼-惠特尼 U 检验不要求满足上述任何条件，因此决定使用该检验来判断这些样本是否来自同一总体，即注入查询与良性查询的特定特征值是否存在差异；或者这些样本是否相互独立，因而来自不同的总体，即注入查询与良性查询的特定特征是否存在差异。

各项特征的曼-惠特尼检验结果如（表 2）所示

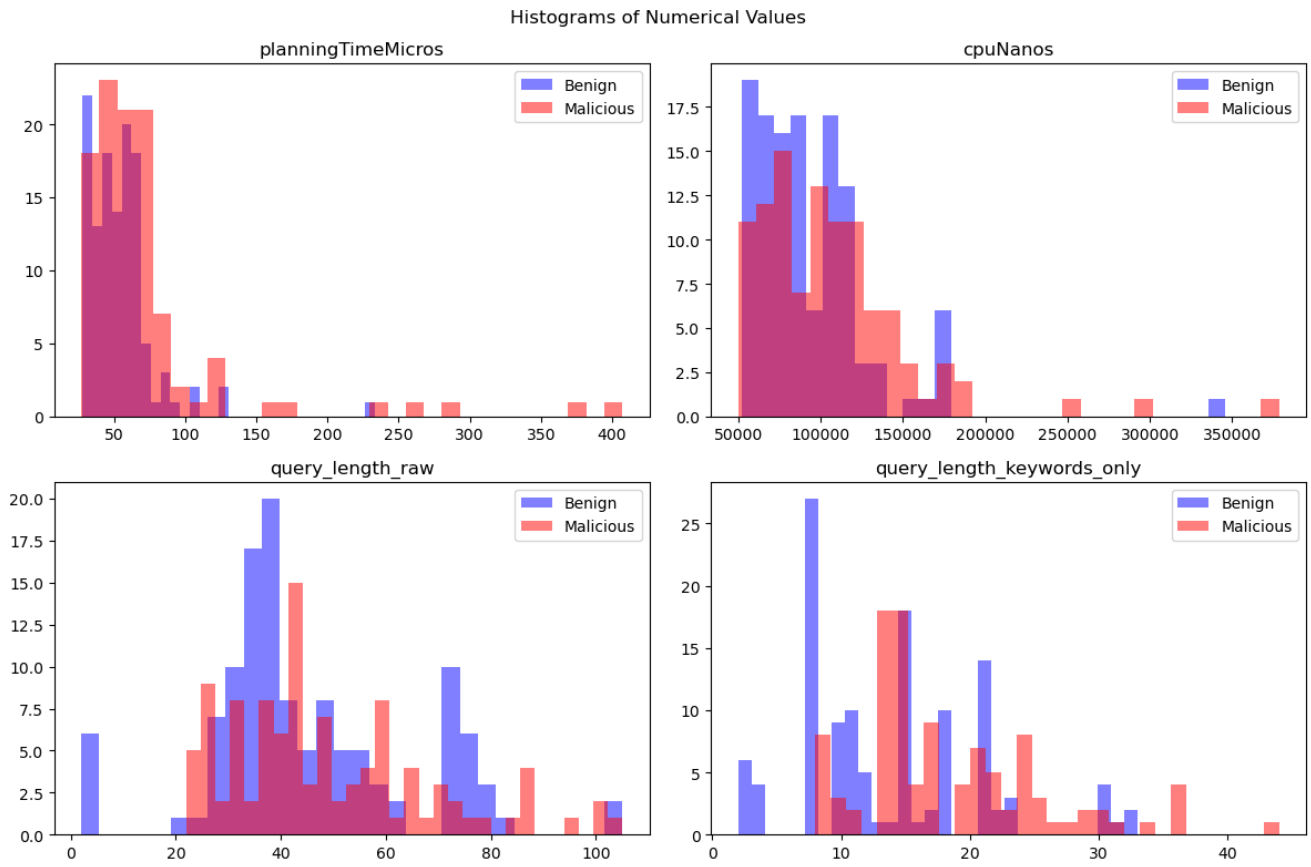


图*1*：数值变量的直方图，蓝色表示良性查询样本，红色表示注入查询样本，从左上角开始顺时针依次为：规划时间（微秒）、*CPU*纳秒数、查询_长度_原始、查询_长度_关键词_仅

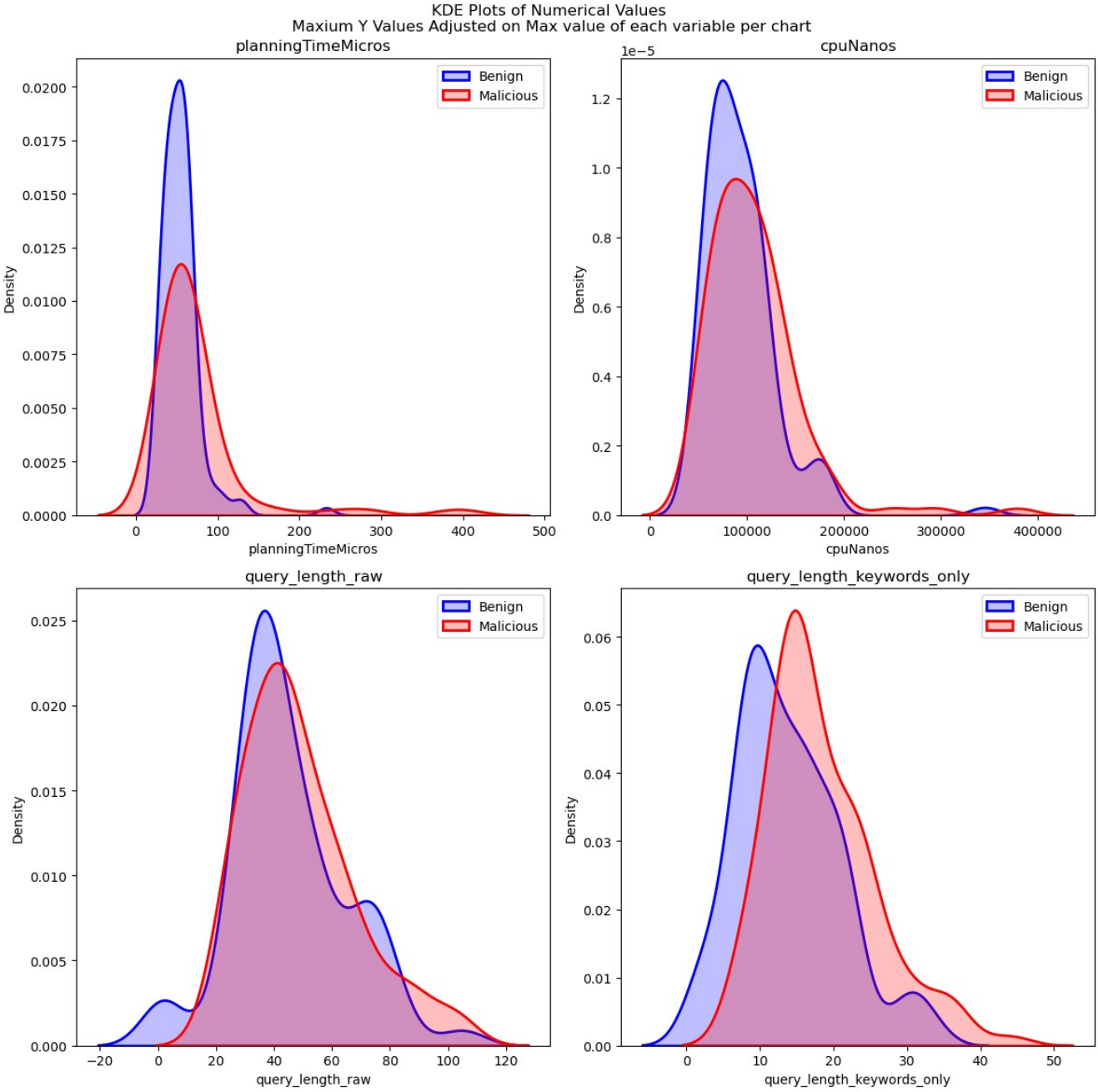


图*2*：数值变量的核密度估计图，蓝色表示良性查询样本，红色表示注入查询样本。从左上角开始顺时针依次为：规划时间（微秒）、*CPU*纳秒数、查询_长度_原始、查询_长度_关键词_仅

变量	U统计量	P值	备择假设 在0.01显著性水平下接受
规划时间（微秒）	4926.5	0.006628	True
CPU纳秒数	5013.0	0.011288	False
查询_长度_原始	5589.0	0.178336	False
查询_长度_关键词_仅	3397.0	0.000003	True

表 *2*：数值变量的曼-惠特尼 *U* 检验结果及 *0.01*显著性水平下的统计显著性

如表 2 所示，只有 planningTimeMicros 和 query_length_keywords_only 在 0.01显著性水平上被发现具有显著性，因此确认这 2 个特征的样本在注入查询与良性查询之间存在显著差异

4.2 布尔特征的判别分析

对于布尔特征，我们进行了卡方（χ2）独立性检验，以确定每个布尔变量与查询类型（即是否为注入或良性查询）之间是否存在统计学上显著的关联。之所以选择该检验，是因为它适用于分析两个分类变量之间的关系，且无需假设数据分布的正态性或总体的特定形态。每项检验的原假设均为该布尔特征与查询类型相互独立，这意味着注入查询与良性查询中该特征的分布没有显著差异。各布尔特征的卡方检验结果见表3。

变量	**φ 值**	**P 值**	**备用方案 假设已接受 在0.01显著性水平下**
$eq	0.08613523235361699	0.1973445139252078	假
$gt	0.055968356591591495	0.4022229995323431	假
$in	0.07056089519199879	0.2909413614822473	假
$ne	0.1727185665577534	0.009737481705364296	True
$nin	0.004795459889571529	0.9427834582253255	假
$type	0.09504298325597839	0.15488894901564945	False
$mod	0.004795459889571529	0.9427834582253255	False
$regex	0.08006523405673249	0.23079764009860607	False
$where	0.11104504640461153	0.09651877524572479	False
$elemMatch	0.11104504640461153	0.09651877524572479	False
$size	0.0369145423668659	0.5806152315243337	False

$	0.32600281665209946	1.0653678148033842e-06	True
>=	0.004795459889571529	0.9427834582253255	假
<=	0.05437649268574571	0.4157407293669543	假
<	0.05437649268574571	0.4157407293669543	假
>	0.05437649268574571	0.4157407293669543	假
选择器_比较	0.12672397767463786	0.05787667883941351	假
选择器_逻辑	0.12672397767463786	0.05787667883941351	假
选择器_元素	0.12672397767463786	0.05787667883941351	假
选择器_评估	0.12672397767463786	0.05787667883941351	假
选择器_数组	0.12672397767463786	0.05787667883941351	假
选择器_按位运算	0.12672397767463786	0.05787667883941351	假
投影	0.12672397767463786	0.05787667883941351	假
misc	0.12672397767463786	0.05787667883941351	假
选择器	0.12672397767463786	0.05787667883941351	假
标准_逻辑	0.12672397767463786	0.05787667883941351	假
所有_运算符	0.12672397767463786	0.05787667883941351	假
空_操作数	0.12672397767463786	0.05787667883941351	假
正则表达式_空值_操作数	0.12672397767463786	0.05787667883941351	假

表*3*：布尔变量相对于目标“标签”的 φ系数和统计显著性

如表3所示，只有变量 $ne 和‘$’在 0.01显著性水平下显著，因此保留了这两个变量，并删除了其余变量。

4.3 可分性分析

通过前述统计检验识别出显著变量后，将这些判别特征与标签变量合并，形成“最终数据集”。随后使用该数据集进行可分性分析，旨在判断“注入”类和“良性”类是否在特定维度上呈现明显的分离。

进行了以下可分性分析方法，并将结果可视化

1. 线性判别分析（LDA）（图3）此分析用于确定数据是否线性可分 2. 主成分分析（PCA）（图4）此分析对 LDA 结果应用变换，并确定数据是否线性可分

3. t分布随机邻域嵌入（t-SNE）（图5）该分析配置为包含两个分量（或维度），用于确定数据的非线性可分性。"

4.3.1 LDA

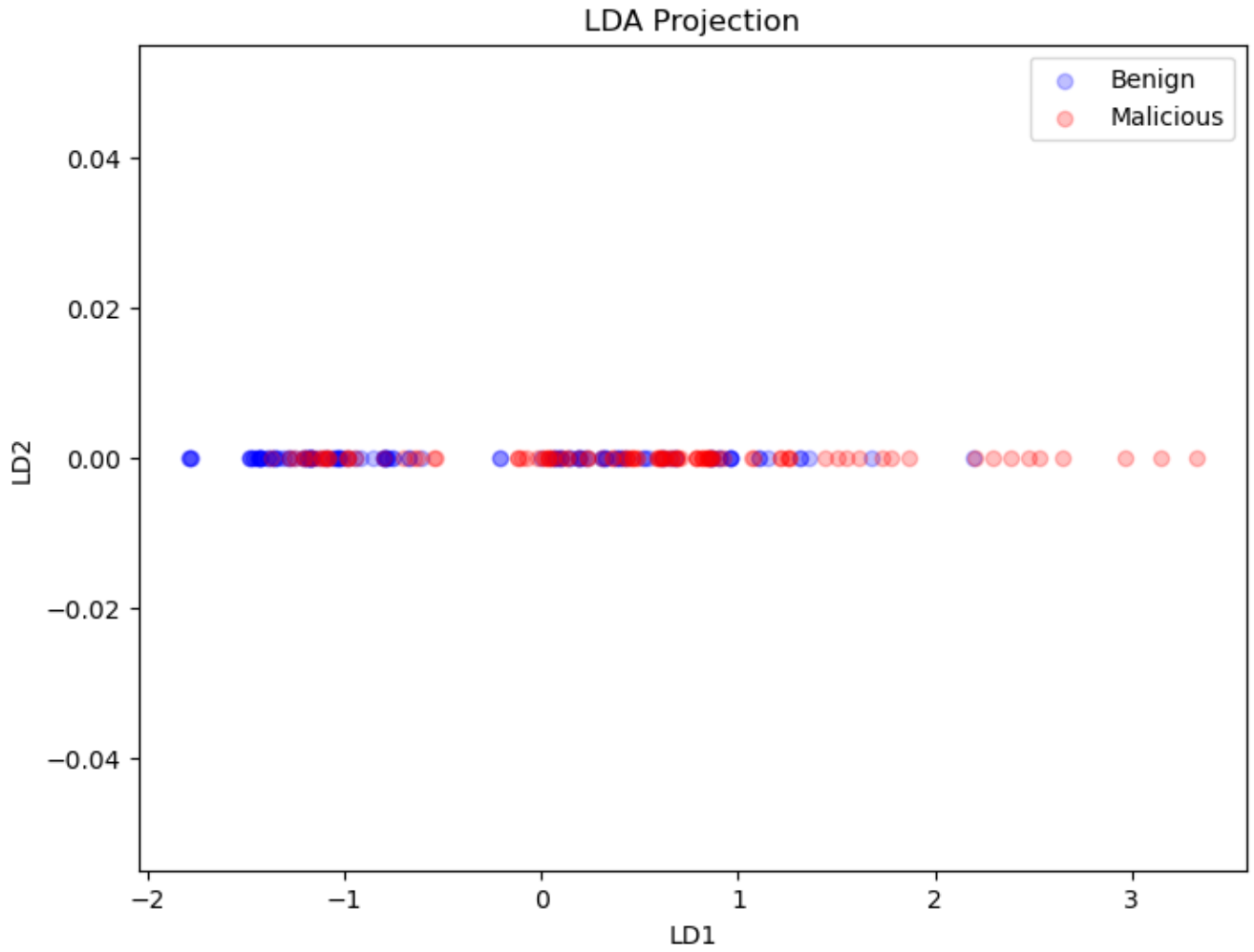


图*3*：线性判别分析投影图，蓝色圆圈表示良性查询数据点，红色圆圈表示注入查询数据点

下图展示了线性判别分析可视化

在线性判别分析投影中未观察到令人满意的可分性，这表明数据可能并非线性可分

4.3.2 PCA

下图展示了主成分分析可视化结果

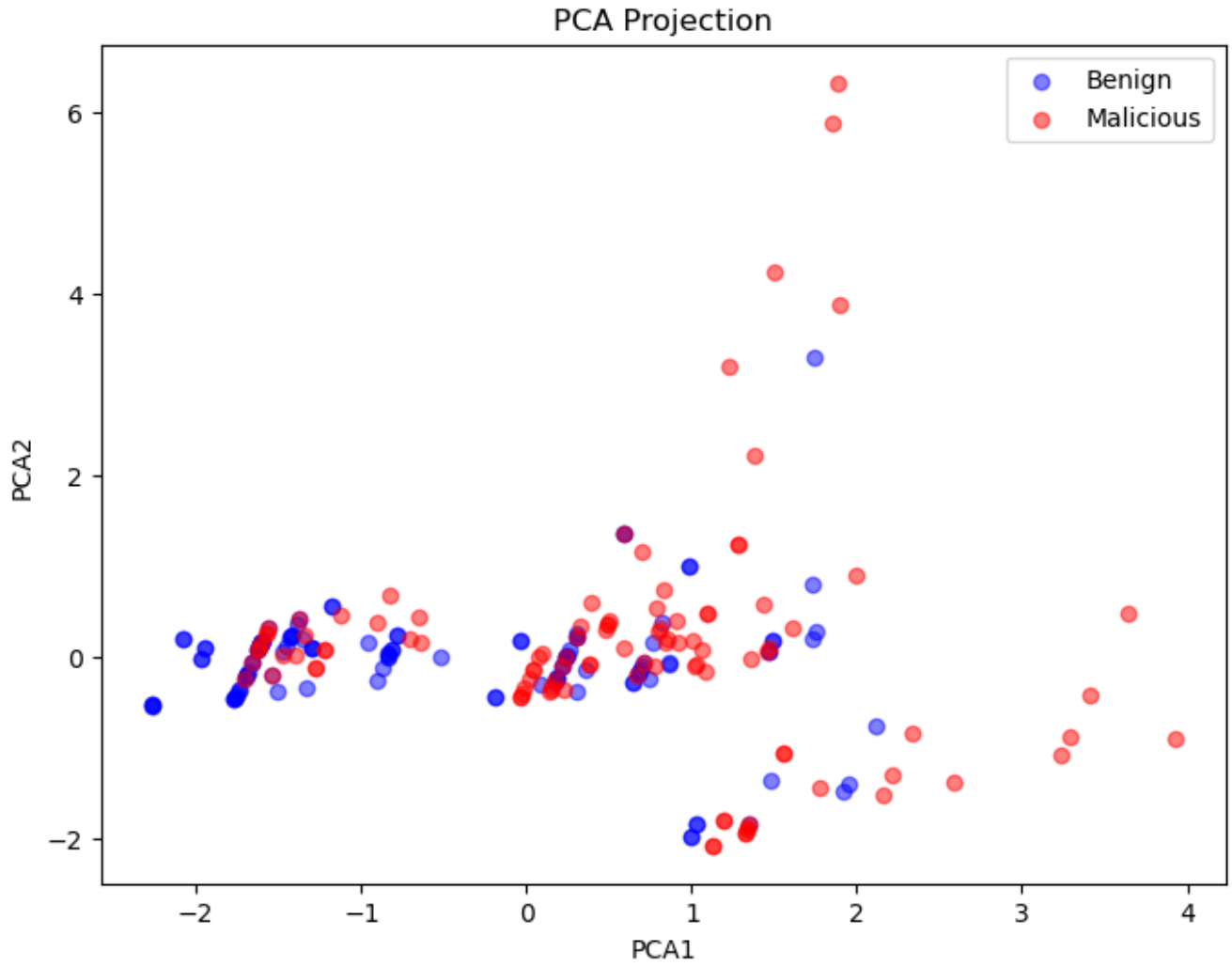


图 *4*：主成分分析投影图，蓝色圆圈表示良性查询数据点，红色圆圈表示注入查询数据点

虽然 PCA 投影的可分性略优于线性判别分析投影，但仍不理想，这表明数据可能不是线性可分的。

4.3.3 t-SNE

下图展示了 t-SNE 可视化结果

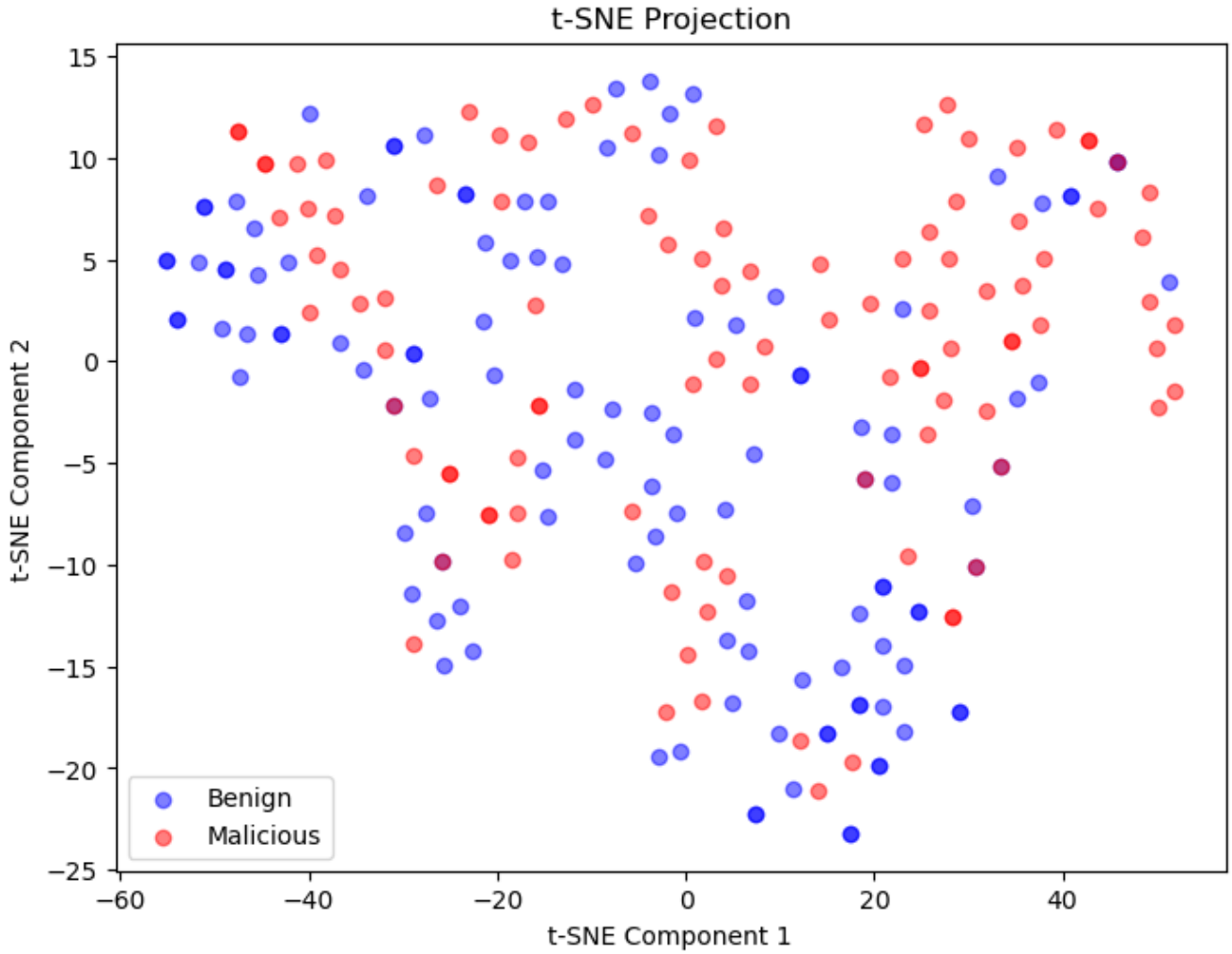


图*5*：*t-SNE* 投影图，蓝色圆圈表示良性查询数据点，红色圆圈表示注入查询数据点

t-SNE 投影显示出良好的可分性，表明数据至少在二维降维空间中呈现非线性可分。因此可以得出结论：在线性分类模型基于该数据集进行训练和测试时，可能无法取得良好效果，而非线性分类模型可能表现更佳

仅包含目标类别和重要特征的最终数据集结构如下

字段	显示名称	数据类型	描述
$ne	$NE	布尔值	如果查询筛选器中存在 $ne 运算符 在查询筛选器中
规划时间（微秒）	规划时间 微秒	浮点 数	开发所需时间： 查询计划的生成时间（以 微秒为单位）。
$	$	布尔值	如果查询筛选器中存在 $ 字符 在查询筛选器中
查询_长度_关键词 _only	仅含 关键词	int	查询的长度 仅含过滤字符串 变量名为 已删除
标签	标签	布尔值	如果查询是 注入查询还是非注入查询

表4：　仅包含具有统计显著性的变量和目标类别的数据结构　　*ss*

5 模型构建

构建了一个具有以下通用结构的分类模型：

标签~ $ne + 规划时间（微秒） + $ + 查询_长度_关键词_仅

该模型旨在根据指定的输入属性，将查询分类为注入或良性。

为评估模型，选取了六种不同的机器学习算法，使用准备好的数据集进行训练和测试：

选择的算法如下：

1. 逻辑回归 2. 随机森林 3. 核支持向量机 4. K近邻 5. 决策树 6. 朴素贝叶斯

此外，FLAML 自动机器学习模型配置为执行分类任务，时间预算为 60 秒，并采用 5 折交叉验证（模型的默认设置）。

5.1 模型训练与测试概述

最终数据集被系统地划分为训练集、测试集和验证集，分别占数据集的 60%、20% 和 20%。模型 1–6 使用这些数据集进行训练。通过循环遍历从 1 到 51 的随机种子，生成 50 种不同的数据集划分（X_train、Y_train、X_test、Y_test、X_val、Y_val）。每次迭代均记录以下性能指标：准确率、精确率、召回率和 F1 分数。每项指标均通过 5 个数据集划分进行交叉验证。所有迭代完成后，计算每个模型各项指标的平均值。FLAML 自动机器学习模型以训练数据集为输入，并配置为进行 5 折交叉验证（默认值）；结果则按照其内部评估方法直接记录。整体训练算法如（图6）所示。

5.2 模型训练过程

模型训练在 Python 3.13 环境中进行，使用了 pandas、scikit-learn（sk-learn）和 FLAML 库。以 CSV 文件加载的“最终数据集”被读入 pandas DataFrame 对象中。随后，将该 DataFrame 传递给 scikit-learn 的 `train_test_split` 方法，以在每次迭代中生成六个不同的变量：

- X_train：数据集中 60% 的唯一行，仅包含用于训练的显著变量。
- Y_train：数据集中 60% 的唯一行，仅包含用于训练的目标变量。
- X_test：数据集中 20% 的唯一行，仅包含用于测试的显著变量。
- Y_test：数据集中 20% 的唯一行，仅包含用于测试的目标变量。
- X_val：数据集中 20% 的唯一行，仅包含用于验证的显著变量。
- Y_val：数据集中 20% 的唯一行，仅包含用于验证的目标变量。

这些变量用于训练、验证和测试六个手动实现的模型。数据集划分直接提供给 FLAML 自动机器学习模型，未进行外部随机化，因为 FLAML 内置了自己的数据划分机制。针对每个随机种子进行的迭代过程确保了全面的训练与评估，每次运行的结果都会被记录下来，随后取平均值，以评估整体性能。

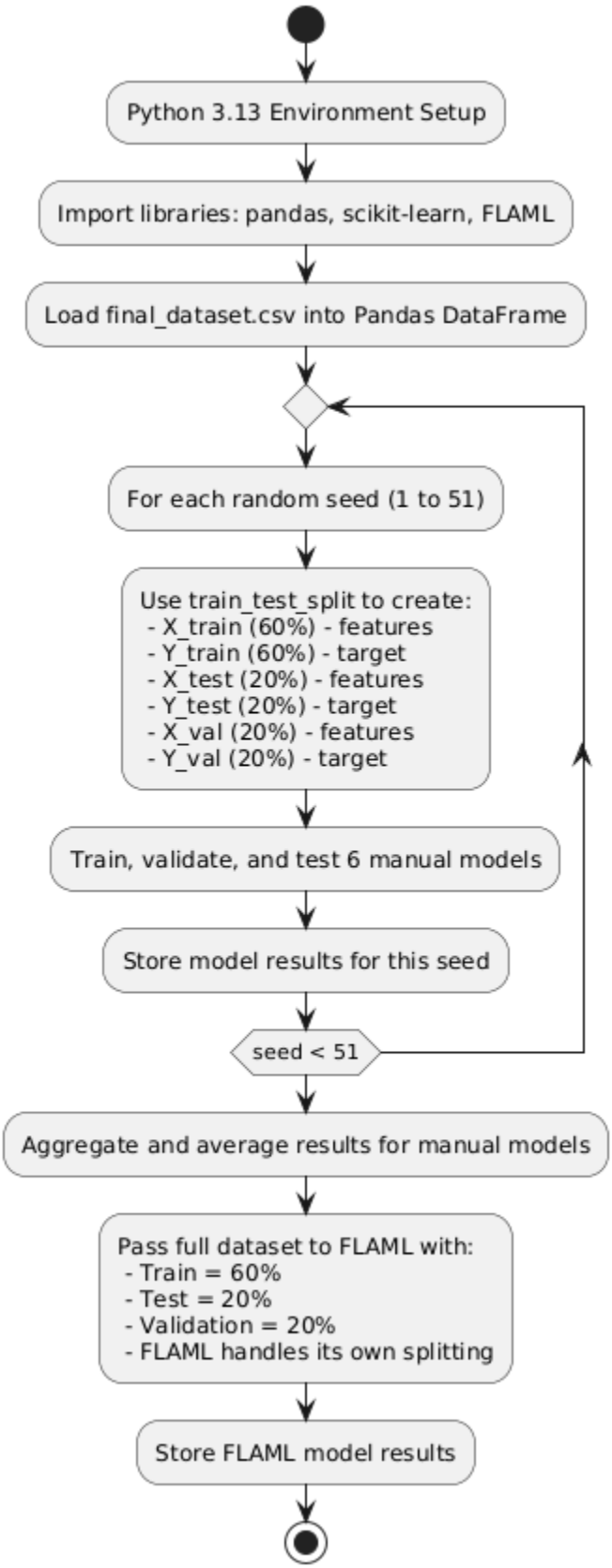


图*6*：模型训练算法流程图

6 评估

对所有模型进行训练、测试和交叉验证后，记录了以下结果（表5）

序号	分类器	准确率（%）	精确率（%）	召回率（%）	F1 分数（%）
1	逻辑回归	65.45%（平均值）	67.73%（平均值）	66.73%（平均值）	66.64%（平均值）
2	随机森林	67.23%（平均值）	68.38%(平均值)	70.36%(平均值)	67.97%（平均值）
3	支持向量机	55.97%(平均值)	59.55%(平均值)	46.36%(平均值)	50.00% (平均值)
4	K近邻	53.25%（平均值）	53.59%（平均值）	63.27%（平均值）	56.66%（平均值）
5	决策树	66.32%（平均值）	67.49%（平均值）	66.91%（平均值）	66.71%（平均值）
6	朴素贝叶斯	59.87%(平均值)	72.00%(平均值)	39.27%(平均值)	49.28% (平均值)
7	FLAML 最佳估计器（XBG 有限深度）	73.00%	75%	73%	74%

表*5*：模型（*1-6*）的交叉验证及平均评估指标，以及 *FLAML* 自动机器学习模型针对不同随机化样本测试数据的交叉验证及最佳评估指标

因此，得出的结论是，XBG 有限深度算法在所有其他模型中表现最佳，具有最高的准确率、精确率、召回率和 F1 分数。

7 结论

本项目展示了仅使用从日志数据中提取的特征，构建机器学习模型并将 MongoDB 日志条目分类为良性或表明存在注入攻击的可行性。由于目前没有适用于此任务的现有带标签数据集，我们通过在空的 MongoDB 实例上执行良性和恶意查询并捕获由此产生的日志输出，成功生成了一个完全合成的数据集。通过探索性分析，我们识别并保留了最具判别性的变量，从而生成了一个由标签和统计显著特征组成的精简数据集。

8 局限性

本研究的主要局限在于依赖从空的 MongoDB 环境中获取的合成生成数据。虽然这种方法能够实现受控的数据收集和精确的标注，但可能无法充分反映现实生产系统中的复杂性、噪声和行为多样性。因此，模型在实际运行环境中的泛化能力仍不确定。未来的工作应侧重于使用真实世界的日志数据验证这些研究结果，并将该方法扩展到包含真实模式、工作负载和后台活动的数据库。

参考文献

Alizadehsani, Z., 2021年。提出使用人工神经网络进行NoSQL攻击检测。ResearchGate。https://doi.org/10.1007/978-3-030-53829-3_29 Babucea, A.-G., 2021年。SQL还是NoSQL数据库？关键差异。D l，R.，H v，A.，Honnavalli, P.B.，S.，N.，2024年。MongoDB注入数据集：全面收集MongoDB NoSQL注入尝试与漏洞。Data in Brief 54, 110289 10 1016 2024 110289。https://doi.org/ . /j.dib. . Eassa, A.M.，Elhoseny, M.，El-Bakry, H.M.，Salama, A.S.，2018年。使用RESTful服务检测Web应用程序中的NoSQL注入攻击。Program Comput Soft 44, 435–444。https://doi.org/10.1134/S036176881901002X Hou, B.，Qian, K.，Li, L.，Shi, Y.，Tao, L.，Liu, J.，2016年。MongoDB NoSQL注入分析与检测，载于：2016年IEEE第三届网络安全与云计算国际会议（CSCloud）。发表于2016年IEEE第三届网络安全与云计算国际会议（CSCloud），第75–78页。https://doi.org/10.1109/CSCloud.2016.57 Joseph, S.，Jevitha, K.P.，2015年。用于防御NoSQL注入的基于自动机的方法，载于：Abawajy, J.H.、Mukherjea, S.、Thampi, S.M.、Ruiz-Martínez, A.（编），计算与通信安全。Springer国际出版，Cham，第538–546页。https://doi.org/10.1007/978-3-319-22915-7_49 M.，A.，H.，O.，M.，H.，S.，A.，2017年。NoSQL Racket：用于检测Web应用程序中NoSQL注入攻击的测试工具。ijacsa 8。https://doi.org/10.14569/IJACSA.2017.081178 Mejia-Cabrera, H.I.，Paico-Chileno, D.，Valdera-Contreras, J.H.，Tuesta-Monteza, V.A.，Forero, M.G.，2021年。通过机器学习自动检测NoSQL数据库中的注入攻击。计算机科学讲义（包括人工智能讲义和生物信息学讲义子系列）。https://doi.org/10.1007/978-3-030-77004-4_3Ron, A.，Shulman-Peleg, A.，Bronshtein, E.，2015年。没有SQL，就没有注入？审视NoSQL安全性。https://doi.org/10.48550/arXiv.1506.04082 Sachdeva, V.，Gupta, S.，2018年。MongoDB上的NOSQL注入基础分析与检测，载于：2018年先进计算与电信国际会议（ICACAT）。发表于2018年先进计算与电信国际会议（ICACAT），第1–5页。https://doi.org/10.1109/ICACAT.2018.8933707 Sajid Ahmed, A.S.，Shachi, M.，Shourav, N.，2022年。使用数据挖掘技术检测服务器端应用程序注入攻击的混合方法。https://doi.org/10.13140/RG.2.2.27421.18407 Scherzinger, S.，Sidortschuck, S.，2020年。关于NoSQL数据库模式设计与演进的实证研究。Shachi, M.，Shourav, N.，Sajid Ahmed, A.S.，Brishty, A.，Sakib, N.，2021年。服务器端应用程序SQL和NoSQL注入攻击检测与防御综述。国际计算机应用期刊183, 1–7。https://doi.org/10.5120/ijca2021921396 Son, S.，McKinley, K.S.，Shmatikov, V.，2013年。DIGLOSSIA：以高精度和高效率检测代码注入攻击。ACM计算机与通信安全会议论文集。https://doi.org/10.1145/2508859.2516696 Ul Islam, M.R.，Islam, Md.S.，Ahmed, Z.，Iqbal, A.，Shahriyar, R.，2019年。使用监督学习自动检测NoSQL注入，载于：2019年IEEE第43届年度计算机软件与应用会议（COMPSAC）。发表于2019年IEEE第43届年度计算机

软件与应用会议（COMPSAC），第 760–769 页。
https://doi.org/10.1109/COMPSAC.2019.00113 Weeratunga, A.M., 2021. MongoDB Web 应用程序中的 NoSQL 注入漏洞识别（学位论文）。