分类号：TP311.5　　单位代码：10335

密　级：无　　学　　号：

浙江大学

博士学位论文

中文论文题目：针对 AI 系统的供应链安全分析与防护

英文论文题目：Security Analysis & Protection for AI System Supply Chains

申请人姓名：

指导教师：

合作导师：

学科 (专业)：网络与信息安全

研究方向：AI 软件与系统安全

所在学院：计算机科学与技术学院

论文递交日期　2026 年 3 月

针对 AI 系统的

供应链安全分析与防护

论文作者签名： _____

指导教师签名： _____

论文评阅人 1：_____

评阅人 2：_____

评阅人 3：_____

评阅人 4：_____

评阅人 5：_____

答辩委员会主席：_____

委员 1：_____

委员 2：_____

委员 3：_____

委员 4：_____

委员 5：_____

答辩日期 _____

Security Analysis & Protection

for AI System Supply Chains

Author's signature： _____

Supervisor's signature： _____

External reviewers: _____

_____

_____

_____

_____

Examining Committee Chairperson:

_____

Examining Committee Members:

_____

_____

_____

_____

_____

Date of oral defence: _____

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 **浙江大学** 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名: 签字日期： 年 月 日

学位论文版权使用授权书

本学位论文作者完全了解 **浙江大学** 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 **浙江大学** 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名: 导师签名:

签字日期： 年 月 日 签字日期： 年 月 日

勘误表

序言

摘要

Abstract

缩略词表

英文缩写	英文全称	中文全称
ZJU	Zhejiang University	浙江大学
ZJU	Zhejiang University	浙江大学
ZJU	Zhejiang University	浙江大学
ZJU	Zhejiang University	浙江大学
ZJU	Zhejiang University	浙江大学
ZJU	Zhejiang University	浙江大学
ZJU	Zhejiang University	浙江大学
ZJU	Zhejiang University	浙江大学

目录

图目录

表目录

1 绪论

1.1 研究背景及意义

随着人工智能 (Artificial Intelligence, AI) 技术的迅猛发展，AI 正在加速融入人类社会的各个领域，并逐渐成为推动社会进步与产业升级的重要引擎。在日常生活中，AI 技术已广泛应用于自动驾驶、智能助手、自然语言处理等关键场景。例如在自动驾驶领域，比亚迪推出的“天神之眼”高阶智能驾驶辅助系统，能够实现全场景的感知与控制辅助功能，为用户提供更加安全、高效的出行体验[1]；在智能助手方面，苹果公司的“Siri 助手”与华为的“小艺助手”能够执行语音指令，完成文件操作、应用启动等任务，显著提升了人机交互的便捷性[2-3]；在自然语言生成领域，OpenAI 于 2022 年发布的 ChatGPT 引发广泛关注，标志着以大参数语言模型 (Large Language Model, LLM) 为代表的生成式 AI 技术进入高速发展阶段[4]。AI 的广泛应用不仅加速了社会向数字化、信息化与智能化的转型，也成为衡量国家科技竞争力的重要标志。

AI 系统的分层架构。随着 AI 技术成体系地持续演化，目前业界研究重点已逐步从单一模型的性能和结构优化，扩展至模型在真实系统中的集成、部署与运行效率等更为系统性的问题。事实上，在复杂应用环境中，AI 模型往往被嵌入到一个多层次、异构化的 AI 系统中，形成从前端应用到后端算力硬件支持的一体化处理链。所谓 AI 系统，是指由 AI 模型、模型管理软件、运行时环境支持的 AI 框架以及底层硬件资源协同构建而成的综合性技术体系，其核心任务是对图像、语音、文本等输入数据进行智能化分析，并输出相应的决策结果或交互反馈。如图 1.1所示，现代 AI 系统通常由三层组成：软件应用层、模型框架层和硬件加速层，三者之间层层依赖、密切协同，共同构成支撑 AI 服务运行的完整技术栈。

软件应用层处于 AI 系统的最上层，直接面向终端用户，负责构建各类 AI 模型驱动的应用程序。在该层中，开发者主要使用 Python 语言调用预训练的 AI 模型，同时结合 Java、C++ 等高级编程语言实现定制化的业务逻辑和系统功能，例如自动驾驶、人脸识别、智能助手、文字生成等智能服务。这些应用可以通过嵌入式部署或远程服务调用的方式对接模型推理模块，从而灵活适配本地部署或云端服务等不同运行环境。

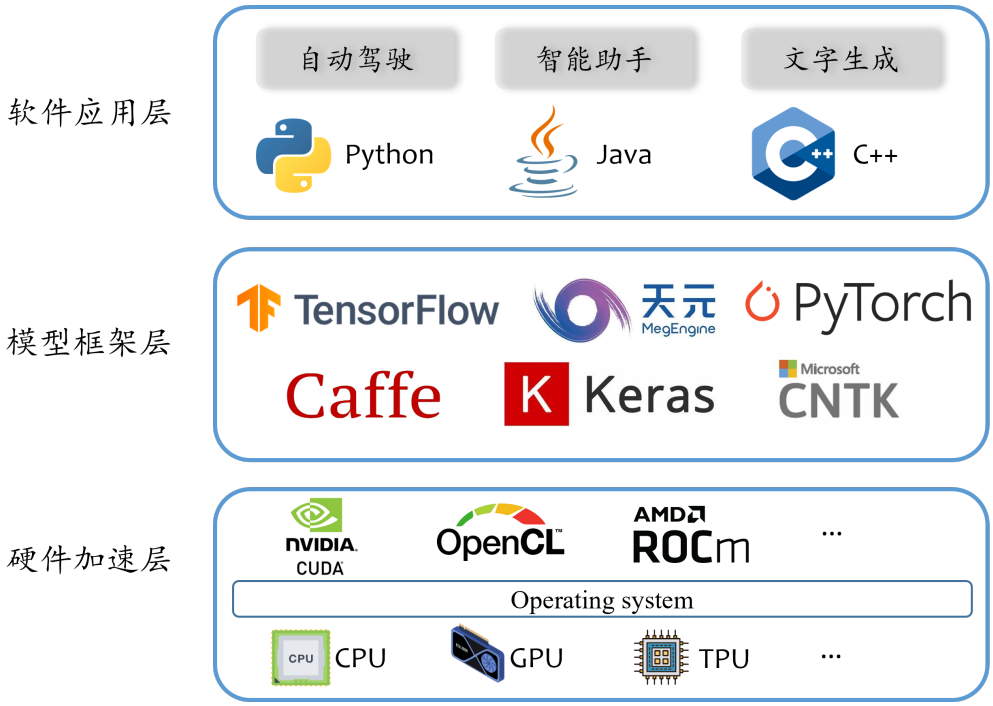

图 1.1 AI 系统架构图

模型框架层位于 AI 系统的中间层，是连接上层应用与下层硬件的核心支撑组件，承担模型训练、推理与部署的功能。在这一层，开发者通常依赖 TensorFlow、PyTorch 等主流深度学习框架[5-6]，通过其提供的高层 API（多以 Python 形式暴露）定义模型结构，并调用由 C++ 或通用并行计算语言实现的底层算子，高效完成模型计算与参数优化。此外，受限于训练过程对算力资源和高质量标注数据的高昂需求，开发者往往从开源模型平台引入预训练模型，并通过迁移学习或微调的方式实现定制化能力。这一实践在显著提升开发效率和迭代速度的同时，也使模型框架层成为 AI 系统中高度依赖外部资源的关键环节。

硬件加速层位于 AI 系统的最底层，为 AI 模型的算子运算提供实际的运行平台和算力保障。鉴于深度学习模型普遍具有高度并行的计算特性，单纯依赖 CPU 已难以满足性能需求，因此该层通常采用 NVIDIA GPU、Intel NPU、Google TPU 等专用加速硬件。同时，操作系统之上还运行着各类支持通用并行计算的平台，如 CUDA、OpenCL 等。这些平台通过底层驱动与编译器将 AI 框架中的算子编译为 GPU 或 TPU 等硬件指令，并由调度器分配至合适的计算单元，从而实现对模型计算过程的高效加速。

AI 系统的供应链。在 AI 系统中这种多层异构架构显然极大地帮助开发者提升了开发效率和模型运行效率，然而系统的多层级复杂性也引入了高度复杂的供应链关系，使

系统整体暴露于跨层级、跨组件的安全风险之中。在这种分层结构下，每一层均依赖大量第三方库、开源框架或底层驱动组件。当某一层的组件受到攻击或被植入恶意行为时，由于下层为上层提供运行支撑、上层对下层进行功能抽象，这种威胁极易沿着依赖链条向上传播，最终影响整个 AI 系统的安全性与稳定性，造成信息泄露、资产损失甚至服务中断等严重后果。

在软件应用层，开发者为了提升开发效率、减少重复实现，通常会引入大量开源第三方软件包。例如在 Python 生态中，图像处理相关的 AI 应用往往依赖 `opencv-python` 库[7]，该库提供了丰富且高效的图像处理 API，能够在处理图像时采用高效的算法进行增强、还原、除噪。然而这种对第三方依赖的高度信任也构成了显著的供应链风险，一旦依赖包本身或其间接依赖被恶意投毒，或依赖包包含尚未修复的安全漏洞，恶意代码便可能在模型部署或运行阶段被触发，从而破坏整个 AI 系统的安全边界。

在模型框架层，从头开始训练模型的需要大量显卡算力的硬件支持，以及人工标注的数据集的昂贵成本，因此开发者往往选择基于现有预训练模型进行二次开发，修改模型结构或者对其参数进行微调。这些预训练开源模型广泛来源于 HuggingFace、Model Zoo、TensorFlow Hub 等开源模型平台[8-10]。然而，此类模型来源复杂，且多以二进制格式分发，其内部结构与执行行为对使用者而言往往不可完全验证。一旦模型中被植入恶意后门或隐蔽的可执行逻辑，便可能在推理过程中触发参数篡改、敏感信息泄露，甚至实现任意代码执行，对 AI 系统构成严重威胁。

在硬件加速层中，AI 系统的运行往往使用于不同的加速平台，这些加速平台都依赖底层驱动程序、编译器和固件将 AI 算子映射至具体硬件执行逻辑。然而，这些底层组件通常由硬件厂商封闭实现，缺乏透明性，其内部的内存管理机制、计算单元调度方式等细节对用户不可见。一旦这些驱动或固件中存在安全漏洞，或者没有实现特定的安全防护机制，攻击者便可能通过精心构造的模型输入或算子参数触发底层缓冲区溢出，进一步导致权限提升或敏感信息泄露。

综上所述，AI 系统的安全问题已不再局限于单一模型或单一组件，而是深度嵌入于其跨层级、跨组件的复杂供应链之中。因此，构建可信且安全的 AI 系统，必须从供应链全生命周期的角度出发，对各层依赖关系、潜在威胁与防护机制进行系统性分析与设计。

1.2 研究现状与目标

在 AI 系统日益复杂化的背景下，AI 供应链安全问题已逐步受到研究界与工业界的高度关注。随着 AI 应用从单一模型扩展为由多层组件协同构成的复杂系统，其安全性也愈发依赖于不同层级组件之间的依赖关系及每一层独有的供应链机制。

AI 系统软件应用层供应链研究现状。 Python 作为 AI 软件开发中最为主流的编程语言之一，围绕其软件应用层的供应链安全问题，也层出不穷，根据 Sonatype 自 2019 年以来的多年年度报告，不仅开源软件包的数量在逐年激增，恶意软件包的数量也随之层出不穷，截至 2024 年，Sonatype 组织已经发现超过 704 102 个恶意的开源软件包[11]。同时报告还指出 CVE 数量也持续呈指数级增长，开发者却无法跟上这样爆炸级的漏洞增长数，无法保证漏洞能够被及时修复。有相关研究表明，部分漏洞在开源软件包中存在的时间甚至长达 3 年以上未修复[12]。高风险的开源软件包不仅会对 AI 软件的开发造成影响，甚至能对整个 AI 系统造成威胁。

目前已有大量研究从开源依赖管理、软件包漏洞以及运行时环境风险等方面展开深入分析。Cheng 等人提出了 PyCRE 框架，采用静态分析方法修复 Python 供应链中存在的错误依赖问题。其核心思路是通过源码分析与抽象语法树技术 (Abstract Resource Tree，AST) 提取模块之间的依赖关系，并结合软件包配置文件构建依赖图，从而判断依赖图中的依赖项是否存在缺失和冲突，进而修复依赖冲突和版本不一致等问题，以避免因依赖错误导致的 AI 软件部署失败[13]。Mukherjee 等人提出了 PyDFix 框架，该框架通过在部署阶段收集运行时的控制台信息，判断安装过程中具体是哪些软件包出现错误，以及错误类型是依赖缺失还是版本不一致，并基于这些错误信息实现对依赖冲突的动态检测与修复[14]。此外，Pipreq 作为一种静态依赖生成工具，它可以通过自动化地分析 Python 项目中 `import` 语句引入了哪些模块，再通过一个一对一的模块与软件包名的映射，来判断该项目需要哪些软件包，从而能够自动从项目源码中推导出所需的依赖列表，用于生成标准化的 `requirements.txt` 配置文件[15]。在进一步扩展依赖修复范围方面，Ye 等人提出了 PyEGo 框架，该框架不仅关注软件包层面的依赖问题，还同时考虑系统环境依赖以及 Python 解释器版本兼容性，从而提升整体部署过程的可复现性与鲁棒性[16]。此外，Cao 等人提出了 PyDC 框架，针对由于 Python 软件依赖配置错误引发的 Dependency Smell 问题展开研究，系统分析了此类问题的普遍性、成因及其演化过程。

除依赖关系修复外，针对 Python 生态中软件漏洞的分析同样是软件应用层供应链研究的重要方向之一。由于 AI 软件通常依赖大量的核心 AI 组件包和其他开源软件包，这些关键依赖项中潜在的漏洞也是影响 AI 系统安全性的重要因素之一。Mahon 等人提出了 PyPitfall 工具，从整体视角系统分析了 PyPI 生态中的依赖结构及漏洞传播关系，揭示了直接依赖与传递依赖在系统漏洞暴露风险中的显著影响[17]。Alfadel 等人通过对 698 个 Python 包的 1396 条漏洞报告进行实证分析，发现 Python 软件包的漏洞数量呈上升趋势，且部分漏洞在被发现前的生命周期超过三年[18]。在更宏观的层面，Ladisa 等人对开源软件供应链的攻击实现了一个系统的分类，该分类独立于特定的编程语言或生态系统，并覆盖了从代码贡献到软件包分发的所有供应链阶段。其以攻击树的形式刻画了 107 种不同的攻击向量，并将其与 94 起真实世界事件及 33 类缓解措施进行映射[19]。类似地，Bogaerts 等人则更专注于 Python 语言，构建了包含 1026 个已公开 Python 漏洞的数据库，并提取了对应的补丁与易受攻击代码，为后续漏洞检测与修复研究提供数据基础[20]。

综上所述，现有研究在 AI 软件应用层已提出诸多有效工具和框架，可以用于自动修复 AI 项目中常见的依赖配置错误、漏洞风险检测、软件包部署的错误等问题，从而提升 AI 软件包的稳定性和安全性。然而，现有工作大多聚焦于已知漏洞或显式依赖关系分析，并且通常都是以软件包为分析粒度，对更细粒度的模块级行为关注度较少，同时也尚未深入探讨供应链机制本身如何被恶意利用的问题。

AI 系统模型框架层供应链研究现状。在 AI 系统的模型框架层，研究者逐渐意识到预训练模型的本身及其其所依赖的运行框架和算子在 AI 供应链中的关键地位。近年来，开源模型库中的模型数量呈爆炸式增长。以 Hugging Face 为例，仅在 2022 年至 2025 年期间，该平台上累计发布的开源模型数量已超过 200 万个[21]。如此庞大的模型规模在显著降低模型获取与复用成本的同时，也为恶意模型的传播提供了现实土壤。已有公开报告表明，开源模型库正逐步成为攻击者投放恶意载荷的新型渠道。JFrog 于 2024 年 2 月发布的分析报告指出，其在 Hugging Face 平台上发现了超过 100 个恶意模型，涉及 TensorFlow、PyTorch 等多个主流深度学习框架。这些模型在加载或推理阶段可触发反向 shell、任意文件读写、启动特定程序以及代码执行等恶意行为[22]。相较于传统的软件包投毒攻击，模型与框架层面的攻击更贴近模型的实际执行路径，能够自然嵌入正常的模型加载与推理流程中，因而通常具备更强的隐蔽性和更高的潜在危害性。

围绕模型框架层的安全风险，现有研究已从多个角度展开系统性探索，相关工作大体可归纳为恶意模型行为分析、模型安全检测机制以及模型框架层漏洞挖掘等方向。从攻击目标与实现方式的角度看，模型层面的恶意逻辑注入主要可以划分为两类。

第一类是传统机器学习语境下的恶意模型，其核心目标在于操纵模型的预测或决策结果，而非直接执行系统级恶意行为。例如，攻击者可通过精心设计的训练过程，使智能驾驶模型在特定条件下将红灯错误识别为绿灯，从而间接诱发交通事故。这类攻击主要关注模型推理行为本身的安全性，对系统执行环境的影响通常是间接的。代表性研究包括后门攻击，即在训练阶段向模型中植入隐蔽触发器，使模型在正常输入下表现正常，而在触发条件出现时输出攻击者预期结果[23-25]；以及对抗样本攻击，通过对输入样本施加微小扰动诱导模型产生错误分类[26-28]。近年来，随着大参数模型高效微调技术的发展，研究者进一步发现，可利用 LoRA 等轻量化微调机制在不显著影响模型整体性能的前提下植入恶意触发逻辑，从而实现更加隐蔽的攻击[29-30]。

第二类则是将 AI 模型本身作为恶意逻辑载体的攻击方式。在这一语境下，模型不再仅用于产生错误预测结果，而是被直接用于承载、隐藏并触发恶意软件或恶意代码，从而对运行模型的系统环境造成实质性威胁。现有研究表明，此类攻击主要通过以下三种方式实现。其一，攻击者将恶意软件或恶意逻辑嵌入模型的二进制参数或特定层次结构中，并在模型运行阶段对恶意载荷进行重组与触发。Hua 等人提出的 Malmodel 技术，将恶意模型嵌入 TensorFlow Lite 模型的层数、覆盖率等参数中，并利用 Java 反射机制主动触发[31]。Hitaj 等人提出的 MaleficNet，利用扩频信道编码结合纠错技术，将恶意负载注入深度学习网络参数中[32]。类似地，其他工作如 Evilmodel 1.0、Evilmodel 2.0 以及 StegoNet，则采用最低有效位 (Least Significant Bit，LSB) 隐写术将恶意软件隐藏于模型权重中[33-35]。其二，攻击者将恶意逻辑直接嵌入模型的 lambda 层中。这类攻击主要适用于支持 lambda 层的模型框架（如 TensorFlow），通过在模型执行过程中触发任意代码执行实现攻击。然而，该方式通常较易被检测，因为仅需检查模型中是否存在 lambda 层并分析其逻辑即可识别异常行为[36-37]。其三，也是目前最为普遍的一类方式，是利用 pickle 等不安全的模型序列化格式，将恶意逻辑嵌入模型文件中，并在模型反序列化过程中触发代码执行[38-40]。针对这一威胁，工业界已提出多种检测与分析工具。例如，Pickletools 可对 pickle 格式的模型文件进行反序列化分析，从而识别潜在的恶意函数调用[41]；Fickling 提供了对 Python pickle 对象的反编译、静态分析和字节码重写能力，既

可用于检测嵌入 PyTorch 模型的恶意行为，也可被用于构造攻击载荷[42]；Picklescan 同样支持对基于 pickle 的恶意 PyTorch 模型进行检测[43]。目前，业界较为先进的模型检测工具包括 Protect AI 公司推出的 ModelScan，该工具能够识别包括基于 pickle 的恶意模型和 TensorFlow lambda 层攻击在内的多种模型级恶意行为[44]。

综上所述，现有研究已从多个角度揭示了模型框架层在 AI 系统供应链中面临的安全风险，充分地证明了模型本身可以被用作攻击载体。然而，这些工作大多将风险归因于恶意模型本身或不安全的序列化机制，从而将模型框架层的安全边界界定在模型层面，这是不完备的，事实上模型框架层自身和为模型框架提供的算子层面的攻击仍未被充分研究。

AI 系统硬件加速层供应链研究现状。在硬件加速层，AI 系统高度依赖 GPU、NPU 等专用计算设备以满足大规模并行计算与高性能推理需求，其底层供应链通常由计算加速硬件、设备驱动、运行时库以及 CUDA、OpenCL 等编程框架共同构成。随着 GPU 架构与配套软件栈复杂度的持续提升，相关供应链组件逐渐暴露出新的安全风险，使得硬件加速层在 AI 系统中不再仅是被动的计算执行单元，而演变为潜在的重要攻击入口。

随着 GPU 架构与配套软件栈复杂度的持续提升，相关供应链组件逐渐暴露出新的安全风险，使得硬件加速层在 AI 系统中不再仅是被动的计算执行单元，而演变为潜在的重要攻击入口。Saileshwar 等人首次将 Rowhammer 类硬件攻击扩展至 GPU 平台，提出了 GPUHammer 攻击方法，利用 GPU 的高并行特性在显存中诱发比特翻转，从而显著破坏深度学习模型参数的完整性，甚至仅通过翻转单个模型权重比特即可导致模型准确率出现灾难性下降[45]。该工作表明，即便不直接攻击模型代码或框架逻辑，底层硬件的不可靠性本身亦可能成为影响 AI 系统可信性的关键因素。

除硬件本体外，围绕 GPU 构建的配套软件同样构成硬件加速层供应链中的重要组成部分，并已被多次证实存在安全隐患。已有公开漏洞表明，NVIDIA GPU 驱动中存在可被利用的高危漏洞，攻击者可借此实现权限提升或非法内存访问[46-48]。与此同时，面向 AI 场景广泛部署的 NVIDIA GPU 容器生态亦被发现存在配置缺陷与隔离失效问题，可能引发跨容器攻击或敏感数据泄漏[49-52]。此外，GPU 编译器及相关开发工具链同样曾被披露存在多项安全漏洞，这进一步扩大了硬件加速层在 AI 供应链中的攻击面[53-55]。更为严峻的是，上述驱动、容器与编译器等关键供应链组件多处于闭源或半开源状态，用户与研究者难以对其内部实现进行独立审计，使漏洞发现与修复高度依赖厂商响应，

一旦攻击者率先掌握可被稳定利用的漏洞，便可能借助硬件加速层对上层 AI 框架与应用产生连锁影响。

从技术研究角度看，现有国内外学术工作主要从 GPU 架构分析与漏洞利用两个方面对硬件加速层展开系统性研究。在 GPU 架构分析方面，研究者通过微架构测试与逆向工程方法，对不透明或半透明的 GPU 内部实现进行了深入探索。Jia 等人率先对 NVIDIA Volta 架构 GPU 的缓存层次结构与访存机制进行了系统分析[56]，随后又扩展至 Turing 架构[57]。此后，多项工作采用类似方法对 NVIDIA 不同代 GPU 架构进行逆向分析，旨在理解其内部设计与安全边界[58-60]。

在漏洞利用方面，针对 GPU 的攻击研究主要集中于侧信道 (Side-channel Attacks) 与隐蔽信道攻击 (Covert Channel Attack)。Naghibijouybari 等人首次证明，基于 OpenGL 或 CUDA 的间谍程序可以通过 GPU 侧信道提取网页指纹、用户交互行为，甚至恢复其他 CUDA 应用中神经网络模型的内部参数[61]。Zhang 等人进一步逆向了 Ampere 架构 GPU 的页表实现细节和多级缓存 (Cache) 的实现细节，并指出在多实例 GPU 特性 (Multi-Instance GPU, MIG) 场景下，由于 L3 Cache 共享机制仍然存在跨实例侧信道风险[62]。Nayak 等人利用统一虚拟内存 (Unified Virtual Memory, UVM) 和快表 (Translation Lookaside Buffer, TLB) 机制，在 GPU 上构建隐蔽信道，实现了对 GPU 加速数据库应用数据的泄漏[63]。此外，Dutta 等人利用 GPU 与 CPU 之间共享缓存与总线的特性，在 Intel 平台上构建了高带宽隐蔽信道，进一步拓展了跨硬件组件攻击的可能性[64]。

在内存漏洞分析方面，已有研究揭示了 GPU 内存管理机制中存在的多种安全隐患。Guo 等人对 NVIDIA GPU 上的越界访问 (Out Of Bound, OOB) 漏洞进行了系统性分析，证实了 GPU 上 OOB BUG 利用的可能性，他们还对 GPU 栈内存布局进行了逆向工程[65]。Mittal 等人对 GPU 漏洞进行了全面综述，从数据泄露、侧信道与隐蔽信道等角度对攻击模式进行了系统分类[66]。Miele 等人利用 GPU 上的栈溢出漏洞劫持函数指针，并分析了在 GPU 环境中实施返回导向编程攻击 (Return-Oriented Programming, ROP) 的可行性[67]。此外，Park 等人提出的 `Mind Control` 攻击通过操纵 GPU 设备内存并利用固定 CUDA 库地址干扰深度学习系统推理过程[68]；Sorensen 等人提出的 `LeftoverLocals` 攻击，利用未初始化的 GPU 局部内存实现跨进程或跨容器的数据恢复，他们的研究恢复了 Apple、Qualcomm 和 AMD 等厂商的 GPU 上的局部内存数据，对其他用户的交互式大语言模型会话进行窃听[69]。Roels 等人进一步研究了 GPU 内存中的 ROP 小组件，并提出了绕过

NVIDIA 将返回地址存储在寄存器中的防御机制的组合式攻击方法[70]。

综上所述，现有研究从硬件架构、配套驱动和工具链软件，以及漏洞利用等多个维度系统揭示了硬件加速层的供应链在安全性方面的潜在风险。然而这些工作大多聚焦于单点漏洞、特定攻击技术或底层实现缺陷，并且仅仅将安全边界界定于 GPU 或者 NPU 等硬件加速设备，将其设为孤立的攻击目标，并未深入分析跨设备的安全性，例如是否可以从 GPU 侧威胁到 CPU 侧的安全性，再由此通过框架与运行时接口向上层 AI 系统传导安全影响。

1.3 本文研究内容与贡献

针对当前 AI 系统在多层次架构中逐渐显现的供应链安全风险，本文从系统整体视角出发，对 AI 系统的软件应用层、模型框架层以及硬件加速层三个关键层级开展了系统性的安全分析。针对当前 AI 系统在多层次架构中逐渐显现的供应链安全风险，本文从系统整体视角出发，对 AI 系统的软件应用层、模型框架层以及硬件加速层三个关键层级开展了系统性的安全分析。通过对上述三个层级的深入研究，本文不仅弥补了现有工作在跨层级系统性分析方面的不足，还在每一层级中发现了此前尚未被充分认识的安全问题，并提出了具有实践意义的分析方法与防护思路，从而构建了一套覆盖 AI 系统全栈的安全研究体系，为理解和保障 AI 系统在真实部署环境下的安全性提供了重要的理论与实践参考。

具体而言，本文的研究内容和主要贡献可概括为以下三个层级。

软件应用层：细粒度模块级依赖冲突安全分析。在软件应用层，现有研究主要关注依赖配置错误、已知漏洞传播以及漏洞检测等安全问题，且多以软件包为分析粒度。然而，在以 Python 为代表的 AI 软件生态中，大量项目由复杂的模块级依赖关系构成，不同软件包在安装与运行阶段可能产生细粒度的模块冲突行为，其安全影响尚未得到系统性研究。

为填补这一研究空白，本文围绕 Python 生态中的模块级依赖冲突问题开展了系统研究。首先，本文从 GitHub、Stack Overflow 等开发者社区出发，采用滚雪球式的数据收集方法，系统整理并分析了大量与模块冲突相关的真实事件，对模块冲突的触发场景、表现形式及潜在影响进行了实证研究总结[71-72]。在此基础上，本文进一步对整个 Python

开源生态 (The Python Package Index, PyPI) 生态中的全部软件包进行了大规模模块收集与分析，系统识别可能存在模块冲突风险的软件包及其潜在影响范围。最后，本文以真实世界的 AI 项目为对象，对 GitHub 上广泛使用的热门 AI 项目进行了深入分析，评估模块冲突问题在实际 AI 软件构建与运行过程中的安全影响。

围绕软件应用层的模块级依赖冲突问题，本文的主要贡献包括：

- **新攻击面：**本文揭示了一种软件应用层的新型攻击面——模块替换攻击。该攻击利用 Python 软件包在安装阶段将不同包的模块部署至同一默认路径的机制，通过构造同名模块引发冲突，从而干扰 AI 软件的正常执行，甚至实现任意代码执行等安全威胁。

- **新技术：**为支撑大规模生态分析，本文提出了两项关键技术：`InstSimulator` 与 `EnvResolution`。`InstSimulator` 通过 AST 解析与动态安装模拟相结合的方式，解决了软件包源代码结构与实际安装后模块布局不一致的问题；`EnvResolution` 则通过环境语义建模与免下载依赖解析机制，有效提升了依赖图构建的准确性与分析效率。

- **大规模实证研究：**基于上述技术，本文实现了 `ModuleGuard` 框架，对 PyPI 生态中 43 万余个软件包、420 万余个版本进行了系统性的模块依赖与冲突分析，系统总结了模块冲突问题的成因、特征及其在生态中的分布情况。

- **真实世界影响分析：**利用 `ModuleGuard`，本文进一步分析了 GitHub 上 3,711 个真实热门 AI 项目（涵盖 93,487 个版本标签），发现其中 108 个高星项目存在实际的模块冲突风险，并已向相关开发者报告问题并提供修复建议。

本部分研究的详细内容见本文第三章，相关代码开源于网站 https://sites.google.com/view/moduleguard，研究成果发表于 ICSE (CCF-A 类) 国际软件工程顶级学术会议。

模型框架层：基于合法 API 能力滥用的函数级攻击范式。在模型框架层，现有研究通常将恶意模型视为参数投毒或模型逻辑篡改问题，关注模型输出异常或性能退化等结果性表现。尽管已有少量工作尝试将模型本身作为恶意载体，但其攻击方式往往依赖不安全序列化或显式代码注入，不仅容易影响模型精度，也较易被现有检测工具发现。

随着深度学习框架功能的不断扩展，其所提供的大量高权限、通用型 API 已逐渐具备超出模型计算本身的系统交互能力。然而，这类“合法 API 所隐含的安全风险”在现有研究中尚未得到系统性分析。针对这一问题，本文以 TensorFlow 框架为研究对象，系统分析了其框架 API 的持久化机制及能力边界，提出了 TensorAbuse 攻击模型，揭示攻击者如何在不依赖传统漏洞的情况下，仅通过滥用框架提供的正常 API，将恶意行为嵌入 AI 模型之中，从而在模型运行阶段触发系统级攻击。

围绕模型框架层的 API 能力滥用问题，本文的主要贡献包括：

- **新攻击面：**本文提出了模型框架层的新型攻击范式——TensorAbuse。该攻击利用深度学习框架 API 自身具备的文件访问与网络通信能力，在不影响模型精度的前提下，实现任意代码执行、文件窃取等系统级攻击行为。

- **新技术：**本文提出了 `PersistExt` 与 `CapAnalysis` 两项关键技术，用于自动化提取并分析框架 API 的潜在高风险能力。`PersistExt` 结合静态分析与启发式规则，系统提取可被持久化至模型中的 API 及其跨语言调用链；`CapAnalysis` 则借助大语言模型能力，对 API 的潜在系统交互能力进行自动化判定。

- **真实世界影响验证：**基于上述技术，本文从 TensorFlow 中识别出 1,083 个可嵌入模型的 API，并进一步筛选出 31 个可用于构造恶意行为的高风险 API，构造了五类攻击原语与四类完整攻击模型，成功绕过 Hugging Face、ModelScan 等主流检测工具，并获得厂商认可与安全奖励。本研究向这些厂商提供相关利用的详细代码，帮助他们构建能力更强的模型扫描工具，同时收获了 1,650 美元的奖励。

- **恶意模型检测工具：**在攻击分析的基础上，本文实现并开源了一套恶意模型检测工具，可对 TensorAbuse 及相关隐蔽嵌入方式进行自动化检测并生成分析报告。

本部分研究的详细内容见本文第四章，相关代码开源于 https://github.com/ZJU-SEC/TensorAbuse，研究成果发表于 IEEE S&P (CCF-A 类) 网络安全领域国际顶级学术会议。

硬件加速层：GPU 内存地址随机化安全机制分析与跨设备攻击。在硬件加速层，GPU 等专用计算设备已成为 AI 系统不可或缺的基础设施。尽管已有大量研究对 GPU 架构、侧信道攻击以及驱动与固件漏洞进行了深入分析，但现有工作大多将 GPU 视为相对独立的计算单元，主要关注 GPU 内部的安全问题，而缺乏对 GPU 与 CPU 等异构

计算设备之间安全关联性的系统研究，尤其尚未从供应链视角分析 GPU 侧安全机制失效对整个 AI 系统的潜在影响。

针对这一不足，本文从 GPU 侧防护机制的安全性出发，重点分析了 NVIDIA GPU 中地址随机化 (Address Space Layout Randomization, ASLR) 机制的设计假设与实际实现之间的差异。本文发现，GPU ASLR 在实现层面存在非随机区域、粗粒度随机化以及地址相关性等问题，使得攻击者仅需利用 GPU 侧的简单内存安全漏洞（如越界读），即可泄露 GPU 内部的敏感地址信息。进一步地，本文系统分析了 GPU 与 CPU 在统一虚拟内存和物理内存映射机制下的交互关系，揭示了 GPU 地址泄露如何被用于逐步推断 CPU 侧地址空间布局，最终首次实现了从 GPU 出发影响 CPU 地址随机化安全性的跨设备攻击路径。

围绕硬件加速层的 GPU 地址随机化机制，本文的主要贡献包括：

- **新攻击面：**本文揭示了一种硬件加速层的新型攻击面。攻击者可利用 NVIDIA GPU 中地址随机化机制的弱点，通过 GPU 侧的 OOB 漏洞泄露关键地址信息，并进一步借助 GPU 与 CPU 地址空间之间的相关性，推断 CPU 侧 glibc 等关键段的地址布局，从而削弱 CPU 侧 ASLR 的安全性。

- **新技术：**本文提出了 `FlagProbe` 与 `AnchorTrace` 两项关键逆向分析技术，用于在缺乏文档支持的黑盒环境下恢复 NVIDIA GPU 虚拟地址空间的语义结构。`FlagProbe` 通过启发式标志插入与访问行为分析，恢复不同 GPU 内存区域的语义；`AnchorTrace` 则利用系统常量内存作为锚点，递归追踪稳定指针链，从而高效收集 GPU 侧随机化地址信息。这两项技术共同构成了一套可复用的 GPU 内存布局分析框架。

- **新发现：**本文系统揭示了多个此前未被公开披露的 GPU 内存布局特性：多个 CUDA 进程共享关键 GPU 内存区域；GPU 与 CPU 在不同权限模型下映射相同的物理内存页；GPU 堆内存完全缺乏随机化，其余内存区域采用统一的粗粒度随机偏移；此外，部分 CPU 地址空间（如 glibc）与 GPU 地址空间之间仅存在极低熵差异。这些特性共同削弱了 GPU ASLR 的防护效果，并为跨处理器攻击提供了现实基础。

- **跨设备攻击验证：**基于上述发现与技术，本文发现了一个 GPU 缺陷。基于该缺陷，本文构建并验证了一条完整的跨设备攻击路径，展示了攻击者如何从 GPU 侧漏洞

出发，逐步破坏 GPU ASLR，并进一步推断 CPU 地址随机化信息，从而对整个 AI 系统的安全性产生连锁影响。相关缺陷已经汇报给 NVIDIA 官方，并已经在 570 版本驱动中修复。

本部分研究的详细内容见本文第五章，相关代码开源于 https://github.com/ZJU-SEC/NvidiaASLR，研究成果发表于 IEEE S&P (CCF-A 类) 网络安全领域国际顶级学术会议。

1.4 本文组织与章节安排

2 AI 系统供应链背景知识

随着 AI 技术的大力发展，其被应用在现代社会的多个关键领域，并成为当前时代下的核心生产力。在这一过程中，AI 系统也由原本的单一推理模型，轻量级数据和简单应用交互的特点演进为由软件应用、模型框架、硬件加速平台等多层组件共同构成的复杂系统。在这一结构下，AI 系统的开发、训练、部署与运行高度依赖第三方代码库、预训练模型、运行时框架以及异构硬件与驱动程序，逐步形成了一条跨越多个技术层级与信任边界的 AI 系统供应链。相比于传统软件供应链，AI 系统的供应链结构更加复杂，也引入了大量的新型攻击面。本章首先对 AI 系统供应链的整体架构进行梳理，从软件应用层、模型框架层以及硬件加速层三个关键层级出发，介绍各个层级的供应链基本组成，功能职责以及所面临的威胁，从而为后续章节针对不同层级展开的安全分析与攻击奠定背景基础。

2.1 软件应用层供应链

在 AI 软件开发过程中，为了显著降低研发成本并加速功能迭代，开发者通常遵循“避免重复造轮子”的工程实践，广泛复用已有的开源代码、第三方组件以及成熟的软件开发工具链。与传统软件相比，AI 软件在数据处理、数值计算与模型推理等方面对功能组件的依赖更为密集，这使得其软件应用层天然构建在一条高度依赖外部资源的供应链之上。

总体而言，AI 系统的软件应用层供应链主要由两条相互交织的子链路构成：一方面是开源组件自身的开发、演化与分发过程；另一方面是 AI 软件在开发、构建与维护过程中对外部代码或组件的引入和复用过程。如图 2.1 所示，这两条链路共同构成了 AI 软件应用层的基础生态，并在实际工程实践中紧密联系。

开源组件开发过程链路。在组件开发阶段，组件开发者通常围绕通用功能或特定需求实现可复用的软件模块，并将其发布至公共的开源组件库中，供其他开发者下载与使用。与此同时，组件开发者自身也往往依赖已有的第三方组件，通过直接引入、二次封装或定制化修改的方式完成新组件的开发。因此，组件开发过程本身同样嵌套在更上

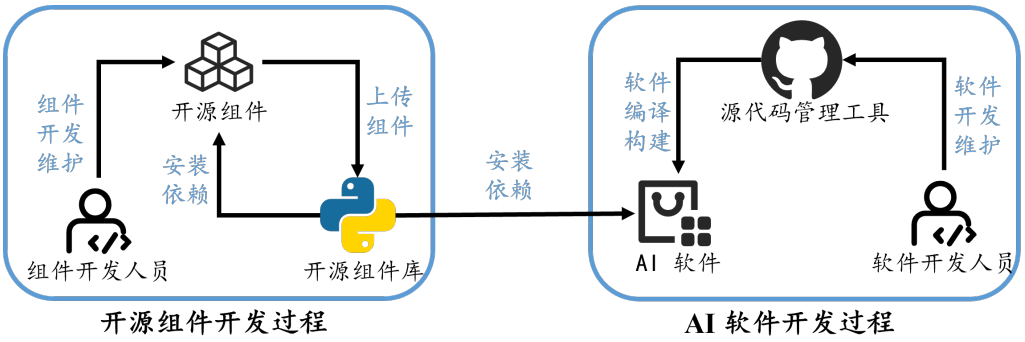

图 2.1 软件应用层供应链架构图

游的组件供应链之中。不同编程语言通常对应不同的组件分发生态。例如，Python 语言主要依赖 PyPI，Java 语言对应 Maven Central，Rust 语言则以 Crates.io 作为官方组件仓库[73-75]。这些集中式组件库极大地降低了依赖获取与分发成本，使得复杂 AI 功能可以通过少量依赖声明快速集成。然而，与之相伴的是依赖规模和复杂度的持续增长。对于 C/C++ 等底层语言而言，由于其发展历史较早、应用场景高度多样，整体生态呈现去中心化特征，缺乏统一的官方组件仓库。这类语言通常依赖系统原生库（如 GNU C Library）或厂商提供的专有库（如 NVIDIA CUDA-X 库）[76-77]。近年来，尽管出现了诸如 Conan 等第三方包管理工具，但其应用范围与生态成熟度仍相对有限[78]。这种多样化的分发模式进一步增加了 AI 软件应用层供应链的异构性。

AI 软件的开发过程链路。在 AI 软件的实际开发过程中，软件开发者通常借助源代码管理工具对项目进行协作开发与持续维护。除通过包管理器引入第三方开源组件外，开发者还可能直接采用源码克隆、代码片段复用等方式，将外部项目中的实现集成至自身代码库中，从而进一步缩短开发周期。这一过程中，版本控制系统与代码托管平台成为软件应用层供应链的关键基础设施。当前，Git 已成为事实上的标准版本控制工具，而 GitHub、GitLab、Gitee 以及企业内部代码托管平台则承担了代码协作、审计与发布的重要角色。在软件开发完成后，构建与打包工具被用于生成可部署的软件形态，例如可执行二进制文件、安装包或容器镜像，并最终交付给终端用户。

依赖解析的复杂性和动态性。在上述两条供应链链路的共同作用下，AI 软件应用层逐步形成了结构复杂、层级深度较大的依赖关系网络，通常以依赖图的形式进行刻画。如图 2.2 所示，albumentationsx 软件在 2.0.13 版本中依赖多个第三方组件，并进一步引入多层间接依赖[79]。在一个依赖图中，单个软件包通常对应一个特定版本，但在某些语

言生态（如 JavaScript）中，同一依赖图中可能同时存在同一组件的多个版本。每一个软件包既可能作为直接依赖被上层软件显式引入，也可能作为间接依赖隐藏在更深层的依赖链路中。例如，图中 numpy 2.2.6 版本既是 albumentationsx 的直接依赖，同时也是其经由 scipy 1.16.3 引入的间接依赖。此外，AI 软件的依赖图并非静态不变。依赖版本往往通过范围约束进行声明，例如在 albumentationsx 的配置文件 setup.py 中指定的一条依赖为 numpy>=1.24.4[80]。在此情况下，依赖解析工具通常会选择满足约束条件的最新版本。一旦上游组件发布新版本，软件在重新构建或安装时，其依赖图结构便可能随之发生变化，从而引入新的行为差异。

依赖图的动态性也使得依赖解析算法变得极其复杂。以 Python 生态为例，用户通常通过官方包管理器 pip 安装第三方依赖组件[81]。在安装过程中，pip 会解析配置文件中的依赖声明，并从 PyPI 获取候选版本列表，依据版本约束与兼容规则逐步选择合适的版本进行下载与安装。该解析过程会递归地处理新引入的依赖，并在出现冲突时进行回溯与重新选择。由于该过程采用边解析边安装的策略，依赖规模较大时往往面临解析效率低下、冲突频繁等问题，甚至可能因为依赖地狱或者依赖冲突问题导致整个依赖解析过程失败[82-83]。这种高度动态且依赖密集的安装机制，使得 AI 软件应用层在功能灵活性提升的同时，也不可避免地引入了更复杂的且难以全面发现的供应链安全风险。

软件应用层供应链对 AI 系统的威胁。

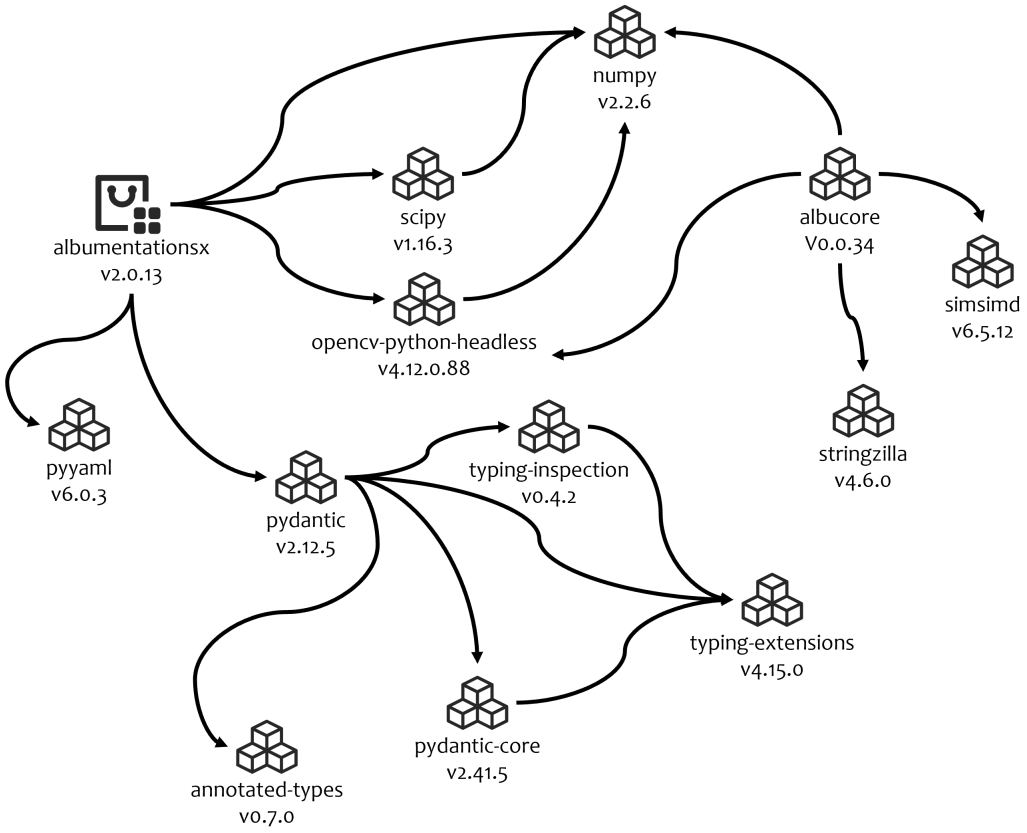

图 2.2 albumentationsx 软件 2.0.13 版本依赖图

参考文献

[1] 比亚迪. 比亚迪获全国首张 L3 自动驾驶高快速路测试牌照，全面加速智能化布局[EB/OL]. 2023. https://www.byd.com/cn/news/2023/detail496.

[2] Apple Inc. Siri[EB/OL]. 2025. https://www.apple.com/siri/.

[3] Huawei. 小艺助手[EB/OL]. 2025. https://xiaoyi.huawei.com/chat/.

[4] OpenAI. ChatGPT[EB/OL]. 2022. https://openai.com/zh-Hans-CN/index/chatgpt/.

[5] ABADI M, BARHAM P, CHEN J, et al. {TensorFlow}: a system for {Large-Scale} machine learning [C]//12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016: 265-283.

[6] PASZKE A, GROSS S, MASSA F, et al. Pytorch: An imperative style, high-performance deep learning library[J]. Advances in neural information processing systems, 2019, 32.

[7] BRADSKI G, the OpenCV team. opencv-python: OpenCV library for Python[EB/OL]. 2025. https://pypi.org/project/opencv-python/.

[8] INC. H F. Hugging Face Hub: A Platform for Sharing Machine Learning Models, Datasets and Demos [EB]. 2026.

[9] Various Contributors. Model Zoo: A Collection of Pre-trained Deep Learning Models[EB]. 2026.

[10] Google Research. TensorFlow Hub: A Repository of Trained Machine Learning Models[EB]. 2026.

[11] Sonatype. State of the 2021 Software Supply Chain[J/OL]. Sonatype Blog, 2021. https://www.sonatype.com/blog/software-supply-chain-2021.

[12] AKHOUNDALI J, NOURI S R, RIETVELD K, et al. MoreFixes: A large-scale dataset of CVE fix commits mined through enhanced repository discovery[C]//Proceedings of the 20th International Conference on Predictive Models and Data Analytics in Software Engineering. 2024: 42-51.

[13] CHENG W, ZHU X, HU W. Conflict-Aware Inference of Python Compatible Runtime Environments with Domain Knowledge Graph[C/OL]//ICSE '22: Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022: 451-461. https://doi.org/10.1145/3510003.3510078. DOI: 10.1145/3510003.3510078.

[14] MUKHERJEE S, ALMANZA A, RUBIO-GONZÁLEZ C. Fixing dependency errors for Python build reproducibility[C]//Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis. 2021: 439-451.

[15] SMITH J. pipreq[EB/OL]. 2023. https://github.com/bndr/pipreqs/.

[16] YE H, CHEN W, DOU W, et al. Knowledge-based environment dependency inference for Python programs[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 1245-1256.

[17] MAHON J, HOU C, YAO Z. PyPitfall: Dependency Chaos and Software Supply Chain Vulnerabilities in Python[J]. arXiv preprint arXiv:2507.18075, 2025.

[18] ALFADEL M, COSTA D E, SHIHAB E. Empirical analysis of security vulnerabilities in Python packages[J/OL]. Empirical Softw. Engg., 2023, 28(3). https://doi.org/10.1007/s10664-022-10278-4. DOI: 10.1007/s10664-022-10278-4.

[19] LADISA P, PLATE H, MARTINEZ M, et al. SoK: Taxonomy of Attacks on Open-Source Software Supply Chains[C]//2023 IEEE Symposium on Security and Privacy (SP). 2023: 1509-1526. DOI: 10.1109/SP46215.2023.10179304.

[20] BOGAERTS F C G, IVAKI N, FONSECA J. A Taxonomy for Python Vulnerabilities[J]. IEEE Open Journal of the Computer Society, 2024, 5: 368-379. DOI: 10.1109/OJCS.2024.3422686.

[21] RÍOS F. Hugging Face's two million models and counting[EB/OL]. AI World. 2025. https://aiworld.eu/stories/hugging-face-two-million-models.

[22] JFrog Security Research Team. Examining Malicious Hugging Face ML Models with Silent Backdoor [EB/OL]. JFrog Security Research. 2025. https://research.jfrog.com/examining-malicious-hugging-face-ml-models-with-silent-backdoor/.

[23] JI Y, ZHANG X, WANG T. Backdoor attacks against learning systems[C]//2017 IEEE Conference on Communications and Network Security (CNS). 2017: 1-9. DOI: 10.1109/CNS.2017.8228656.

[24] GU T, LIU K, DOLAN-GAVITT B, et al. Badnets: Evaluating backdooring attacks on deep neural networks[J]. Ieee Access, 2019, 7: 47230-47244.
[25] TURNER A, TSIPRAS D, MADRY A. Label-consistent backdoor attacks[J]. arXiv preprint arXiv:1912.02771, 2019.
[26] KURAKIN A, GOODFELLOW I, BENGIO S. Adversarial machine learning at scale[J]. arXiv preprint arXiv:1611.01236, 2016.
[27] HUANG S, PAPERNOT N, GOODFELLOW I, et al. Adversarial attacks on neural network policies [J]. arXiv preprint arXiv:1702.02284, 2017.
[28] MADRY A, MAKELOV A, SCHMIDT L, et al. Towards deep learning models resistant to adversarial attacks[J]. arXiv preprint arXiv:1706.06083, 2017.
[29] YIN M, ZHANG J, SUN J, et al. LoBAM: LoRA-Based Backdoor Attack on Model Merging[J]. arXiv preprint arXiv:2411.16746, 2024.
[30] LIU H, LIU Z, TANG R, et al. Lora-as-an-attack! piercing llm safety under the share-and-play scenario [J]. arXiv e-prints, 2024: arXiv-2403.
[31] HUA J, WANG K, WANG M, et al. MalModel: Hiding Malicious Payload in Mobile Deep Learning Models with Black-box Backdoor Attack[J]. arXiv preprint arXiv:2401.02659, 2024.
[32] HITAJ D, PAGNOTTA G, DE GASPARI F, et al. Do You Trust Your Model? Emerging Malware Threats in the Deep Learning Ecosystem[J]. arXiv preprint arXiv:2403.03593, 2024.
[33] WANG Z, LIU C, CUI X. Evilmodel: hiding malware inside of neural network models[C]//2021 IEEE Symposium on Computers and Communications (ISCC). 2021: 1-7.
[34] WANG Z, LIU C, CUI X, et al. Evilmodel 2.0: bringing neural network models into malware attacks [J]. Computers & Security, 2022, 120: 102807.
[35] LIU T, LIU Z, LIU Q, et al. StegoNet: Turn Deep Neural Network into a Stegomalware[C/OL]// ACSAC '20: Proceedings of the 36th Annual Computer Security Applications Conference. Austin, USA: Association for Computing Machinery, 2020: 928-938. https://doi.org/10.1145/3427228.3427268. DOI: 10.1145/3427228.3427268.
[36] Splinter0. Tensorflow Remote Code Execution with Malicious Model[EB/OL]. GitHub. 2024. https://github.com/Splinter0/tensorflow-rce.
[37] CERT Coordination Center. Vulnerability Note VU#253266[EB/OL]. CERT/CC. 2025. https://kb.cert.org/vuls/id/253266.
[38] The Hacker News. New Attack Technique 'Sleepy Pickle' Targets Machine Learning Models [EB/OL]. The Hacker News. 2024. https://thehackernews.com/2024/06/new-attack-technique-sleepy-pickle.html.
[39] Pjcampbe11. Pickle-File-Attacks[EB/OL]. GitHub. 2024. https://github.com/pjcampbe11/Pickle-File-Attacks.
[40] Trail of Bits. Exploiting ML Models with Pickle File Attacks (Part 1)[EB/OL]. Trail of Bits. 2024. https://blog.trailofbits.com/2024/06/11/exploiting-ml-models-with-pickle-file-attacks-part-1/.
[41] Python Software Foundation. pickletools —Tools for pickle developers[EB]. 2023.
[42] Of BITS T. Fickling @ DEFCON AI Village 2021[EB]. 2021.
[43] Mmaitre314. Python Pickle Malware Scanner[EB]. 2024.
[44] Protectai. ModelScan: Protection against Model Serialization Attacks[EB]. GitHub. 2024.
[45] University of Toronto. How three U of T researchers discovered a GPU vulnerability that threatened AI models[EB/OL]. University of Toronto. 2024. https://www.utoronto.ca/news/how-three-u-t-researchers-discovered-gpu-vulnerability-threatened-ai-models.
[46] NVIDIA Corporation. Product Security[EB/OL]. NVIDIA. 2025. https://nvidia.custhelp.com/app/answers/detail/a_id/5630.
[47] NVIDIA Corporation. Product Security[EB/OL]. NVIDIA. 2025. https://nvidia.custhelp.com/app/answers/detail/a_id/5703.
[48] NVIDIA Corporation. Product Security[EB/OL]. NVIDIA. 2025. https://nvidia.custhelp.com/app/answers/detail/a_id/5670.
[49] NVIDIA Corporation. Vulnerability Analysis for Container Security[EB/OL]. NVIDIA. 2024. https://build.nvidia.com/nvidia/vulnerability-analysis-for-container-security.
[50] AIMonks. The NVIDIAScape Vulnerability: A Wake-Up Call for Closed-Source AI Infrastructure

[EB/OL]. Medium. 2025. https://medium.com/aimonks/the-nvidiascape-vulnerability-a-wake-up-call-for-closed-source-ai-infrastructure-b07a745bdac2.
[51] NVIDIA Corporation. Security Bulletin: NVIDIA Container Toolkit - July 2025[EB/OL]. NVIDIA. 2025. https://nvidia.custhelp.com/app/answers/detail/a_id/5659/~/security-bulletin%3A-nvidia-container-toolkit---july-2025.
[52] Wiz Research. NVIDIA AI Vulnerability CVE-2025-23266 (NVIDIAScape)[EB/OL]. Wiz. 2025. https://www.wiz.io/blog/nvidia-ai-vulnerability-cve-2025-23266-nvidiascape.
[53] MITRE Corporation. CVE-2024-53870[EB/OL]. CVE.org. 2024. https://www.cve.org/CVERecord?id=CVE-2024-53870.
[54] MITRE Corporation. CVE-2024-53871[EB/OL]. CVE.org. 2024. https://www.cve.org/CVERecord?id=CVE-2024-53871.
[55] MITRE Corporation. CVE-2024-53872[EB/OL]. CVE.org. 2024. https://www.cve.org/CVERecord?id=CVE-2024-53872.
[56] JIA Z, MAGGIONI M, STAIGER B, et al. Dissecting the NVIDIA volta GPU architecture via microbenchmarking[J]. arXiv preprint arXiv:1804.06826, 2018.
[57] JIA Z, MAGGIONI M, SMITH J, et al. Dissecting the nvidia turing t4 gpu via microbenchmarking [J]. arXiv preprint arXiv:1903.07486, 2019.
[58] ABDELKHALIK H, ARAFA Y, SANTHI N, et al. Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis[C]//2022 IEEE High Performance Extreme Computing Conference (HPEC). 2022: 1-8.
[59] JARMUSCH A, GRADDON N, CHANDRASEKARAN S. Dissecting the NVIDIA Blackwell Architecture with Microbenchmarks[J]. arXiv preprint arXiv:2507.10789, 2025.
[60] LUO W, FAN R, LI Z, et al. Dissecting the NVIDIA Hopper Architecture through Microbenchmarking and Multiple Level Analysis[J]. arXiv preprint arXiv:2501.12084, 2025.
[61] NAGHIBIJOUYBARI H, NEUPANE A, QIAN Z, et al. Rendered insecure: GPU side channel attacks are practical[C]//Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. 2018: 2139-2153.
[62] ZHANG Z, ALLEN T, YAO F, et al. T unne L s for B ootlegging: Fully Reverse-Engineering GPU TLBs for Challenging Isolation Guarantees of NVIDIA MIG[C]//Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023: 960-974.
[63] NAYAK A, B P, GANAPATHY V, et al. (mis) managed: A novel tlb-based covert channel on gpus[C]//Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security. 2021: 872-885.
[64] DUTTA S B, NAGHIBIJOUYBARI H, ABU-GHAZALEH N, et al. Leaky buddies: Cross-component covert channels on integrated cpu-gpu systems[C]//2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). 2021: 972-984.
[65] GUO Y, ZHANG Z, YANG J. GPU Memory Exploitation for Fun and Profit[C/OL]//33rd USENIX Security Symposium (USENIX Security 24). Philadelphia, PA: USENIX Association, 2024: 4033-4050. https://www.usenix.org/conference/usenixsecurity24/presentation/guo-yanan.
[66] MITTAL S, ABHINAYA S B, REDDY M, et al. A Survey of Techniques for Improving Security of GPUs[J]. Journal of Hardware and Systems Security, 2018, 2(3): 266-285.
[67] MIELE A. Buffer Overflow Vulnerabilities in CUDA: A Preliminary Analysis[J]. Journal of Computer Virology and Hacking Techniques, 2016, 12(2): 113-120.
[68] PARK S O, KWON O, KIM Y, et al. Mind Control Attack: Undermining Deep Learning with GPU Memory Exploitation[J]. Computers & Security, 2020, 102: 102115.
[69] SORENSEN T, KHLAAF H. LeftoverLocals: Listening to LLM Responses Through Leaked GPU Local Memory[J]. arXiv preprint arXiv:2401.16603, 2024.
[70] ROELS J, JACOBS A, VOLCKAERT S. CUDA, Woulda, Shoulda: Returning Exploits in a SASSy World[C/OL]//EuroSec'25: Proceedings of the 18th European Workshop on Systems Security. Rotterdam, Netherlands: Association for Computing Machinery, 2025: 40-48. https://doi.org/10.1145/3722041.3723099. DOI: 10.1145/3722041.3723099.
[71] GitHub, Inc. GitHub[EB/OL]. GitHub. 2008. https://github.com.
[72] Stack Overflow. Stack Overflow[EB/OL]. Stack Overflow. 2008. https://stackoverflow.com.

[73] Python Software Foundation. Python Package Index (PyPI)[EB/OL]. Python Software Foundation. 2003. https://pypi.org.
[74] Apache Software Foundation. Maven Central Repository[EB/OL]. Sonatype. 2003. https://central.sonatype.org.
[75] Rust Foundation. Crates.io[EB/OL]. Rust Foundation. 2014. https://crates.io.
[76] GNU Project. The GNU C Library (glibc)[EB/OL]. Free Software Foundation. 1992. https://www.gnu.org/software/libc/.
[77] NVIDIA Corporation. CUDA-X Libraries[EB/OL]. NVIDIA. 2007. https://developer.nvidia.com/cuda/cuda-x-libraries.
[78] SARKAR V, KAMPKÖTTER A, HERMANN B. CoPhi-Mining C/C++ Packages for Conan Ecosystem Analysis[C]//2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR). 2025: 586-590.
[79] Albumentations Team. AlbumentationsX[EB/OL]. GitHub. 2024. https://github.com/albumentations-team/AlbumentationsX.
[80] Albumentations Team. AlbumentationsX setup.py[EB/OL]. GitHub. 2024. https://github.com/albumentations-team/AlbumentationsX/blob/main/setup.py.
[81] Python Packaging Authority. pip Documentation[EB/OL]. Python Packaging Authority. 2008. https://pip.pypa.io/en/stable/index.html.
[82] WANG C, WU R, SONG H, et al. smartpip: A smart approach to resolving python dependency conflict issues[C]//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. 2022: 1-12.
[83] FAN G, WANG C, WU R, et al. Escaping dependency hell: finding build dependency errors with the unified dependency graph[C]//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020: 463-474.

附录

A 一个附录

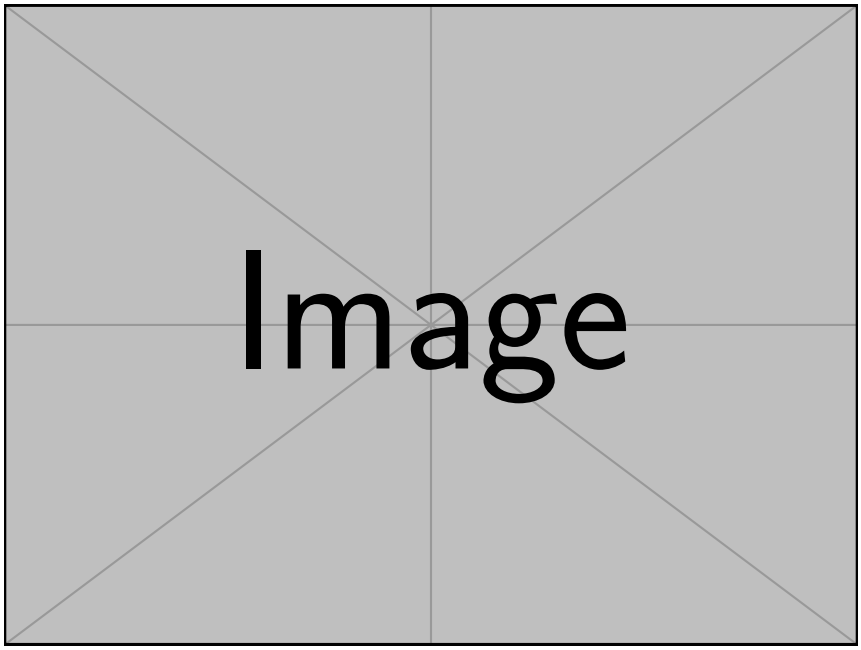

图 A.1 附录中的图片

B 另一个附录