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

勘误表 I
序言 III
摘要 V
Abstract VII
缩略词表 IX
目录 XI
图目录 XIII
表目录 XV
1 绪论 1
1.1 研究背景及意义 1
1.2 研究现状与目标 3
1.3 本文研究内容与贡献 5
1.4 本文组织与章节安排 5
参考文献 7
附录 9
A 一个附录 9
B 另一个附录 9

图目录

图 1.1　AI 系统架构图..... 1
图 A.1　附录中的图片..... 9

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。

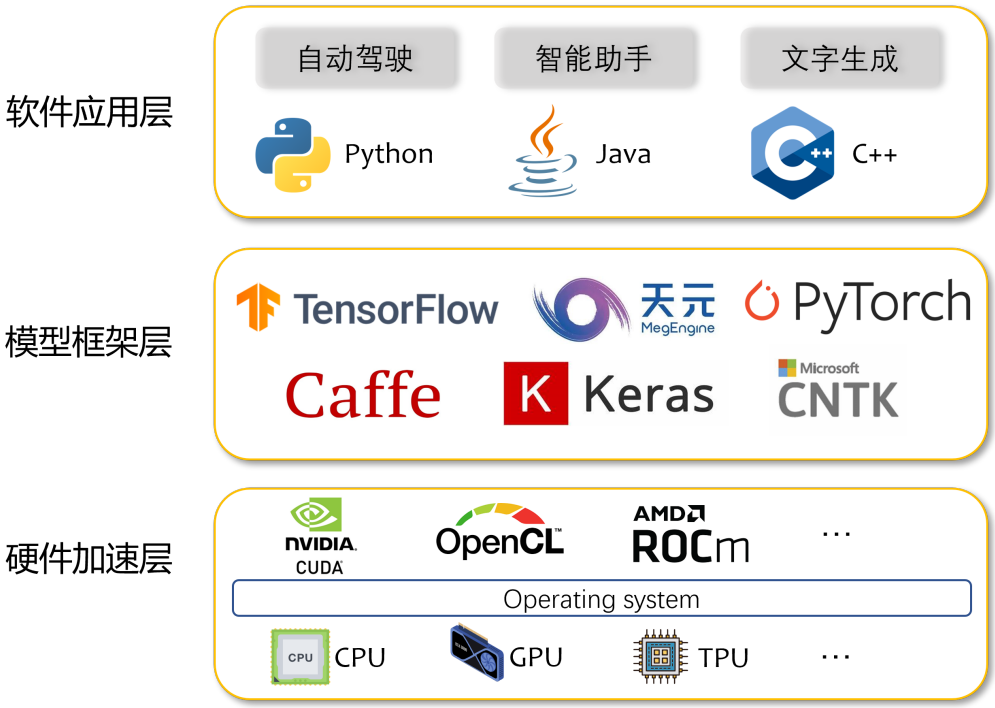

图 1.1 AI 系统架构图

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，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，连接模型推理模块，灵活适配本地或云端需求。

模型框架层位于 AI 系统的中间层，是 AI 系统的核心支撑部分，承担模型训练、推理与部署的功能。在这一层，开发者主要依赖深度学习框架如 TensorFlow 与 PyTorch 等[5-6]，使用框架提供的高层应用程序接口（Application Programming Interface，API）定义模型网络结构（通常是 Python 语言 API），调用封装好的底层算子（通常是由 C++ 或者通用计算语言实现）高效地完成模型构建与参数优化。此外，为了节省模型训练时的计算资源消耗和数据集成本，开发者往往会从开源模型网站引入预训练模型，并对其进行迁移学习或微调来实现 AI 系统的定制化能力和快速迭代。

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必要的运行平台和算力保障。由于 AI 模型普遍存在高度并行的计算特性，仅依赖传统 CPU 进行推理和训练将面临严重性能瓶颈。因此，该层通常依赖高性能计算设备，如 NVIDIA GPU、Intel NPU、Google TPU 等。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。

这种多层异构架构的 AI 系统显然极大地帮助开发者提升了开发效率，然而系统的复杂性也引入了更为复杂的 AI 软件与系统的供应链关系，每一层的组件与工具链均依

赖大量第三方库、开源框架和底层驱动，这使得整个 AI 系统会由于暴露在供应链侧存在的潜在风险之中而导致的自身安全性和稳定性下降，最终造成用户的信息泄露，资产损失等问题。在软件应用层，开发者在开发 AI 软件时，为了减少造轮子的代码，增加开发效率，往往会引入大量开源的软件包。例如 Python 软件开发中，图像处理模型往往依赖于“opencv-python”库进行开发[7]。该库提供了大量图形相关 API，能够在处理图像和计算机视觉方面，采用高效的算法进行增强、还原、除噪等。然而正是由于依赖第三方软件包的这一供应链关系，使得开发者不得不完全信任这些依赖包，如果这些依赖包被恶意攻击者投毒或篡改，可能在模型运行时引入恶意代码，导致整个 AI 系统被操控或监控。

在模型框架层中，由于训练模型的需要大量显卡支持的算力，以及人工标注的数据集的昂贵成本，开发者往往选择从已经训练好的模型进行优化，修改其结构或者对其参数进行微调，这些开源模型往往来自于外部开源社区，例如 HuggingFace、Model Zoo、TensorFlow Hub 等。这些模型来源复杂，且往往都是以二进制格式进行分发，一旦其包含恶意的后门或者隐蔽的可执行代码，便可能在模型推理运行时造成参数篡改，信息泄露，甚至任意代码执行，从而对整个 AI 系统造成严重的威胁。

在硬件加速层中，AI 系统的运行往往依赖于不同的加速平台。这些加速平台都依赖底层驱动程序、编译器和固件（firmware）将 AI 算子映射至具体硬件执行逻辑。而这些底层组件往往封闭、不透明，且极度依赖硬件厂商的实现细节。由此模型的算子在这些加速平台上运行时的具体运作机理，包括内存管理方式，计算核调度方式等都是对用户不可见的，因此一旦这些不透明的固件或者驱动存在漏洞，攻击者便可通过精心构造的输入数据触发底层缓冲区溢出，并进一步造成权限提升或信息泄露等问题。

因此，构建一个真正可信、安全的 AI 系统，必须从供应链的全生命周期角度出发，系统性地分析每一层的依赖结构、潜在威胁及防护机制。

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。

Python 作为 AI 软件开发的主要语言，目前已有大量工作在 Python 软件应用层供

应链方面进行了深入研究，它们围绕开源依赖、软件包漏洞与运行时环境的安全隐患。Cheng 等人提出了 PyCRE 框架，采用静态分析方式修复 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提取模块之间的依赖关系，并据此判断依赖项的准确性，从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部署失败[8]。Mukherjee 等人则提出了 PyDFix 框架，该方法通过在部署阶段收集运行时信息，实现对依赖冲突的动态检测与修复[9]。此外，Pipreq 作为一个静态依赖生成工具，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[10]。在进一步扩展依赖修复范围方面，Ye 等人提出了 PyEGo 框架，其不仅关注软件包本身的依赖问题，还考虑系统环境依赖及 Python 解释器版本兼容性，从而提升整体部署的可复现性与鲁棒性[11]。Cao 等人提出了 PyDC 框架在解决由于 Python 软件中依赖配置错误引起的 Dependency Smell 问题，调查其普遍性、成因以及演化过程。除依赖关系修复外，针对 Python 生态中软件漏洞的分析也是研究重点之一。Mahon 等人提出了 PyPitfall 工具，从整体视角系统分析了 PyPI 生态的依赖结构与漏洞传播关系，揭示了直接依赖与传递依赖对系统漏洞暴露风险的显著影响[12]。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[13]。Bogaerts 等人则构建了包含 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[14]。综上所述，现有研究在软件应用层已提出诸多有效工具和框架，用于修复依赖配置错误、缓解漏洞风险、提升软件包部署的稳定性。然而，这些工作大多聚焦于“已知漏洞”或“显式依赖关系”的分析，并且都是以包为分析粒度，对更细粒度的模块粒度少有分析，且尚未深入探讨供应链机制本身如何被恶意利用的问题。

在模型框架层，研究者逐渐关注到模型本体及其运行框架在 AI 供应链中的关键地位。相较于传统软件包，这一层的攻击面更贴近模型执行逻辑，具备更强的隐蔽性与破坏力。已有多项工作围绕模型文件的代码注入、恶意模型检测、模型转换过程的安全隐患等方面展开系统研究。

在硬件加速层，AI 系统高度依赖 GPU 等专用硬件资源以实现高性能计算，而 GPU 驱动、固件与用于计算的 CUDA、OpenCL 程序成为新的攻击入口。近年来，随着 CUDA 等平台复杂度的提升，越来越多研究开始探讨底层硬件与驱动对 AI 系统的供应链风险，包括驱动漏洞、微架构攻击、CUDA 程序漏洞检测等。

1.3 本文研究内容与贡献

1.4 本文组织与章节安排

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] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

附录

A 一个附录

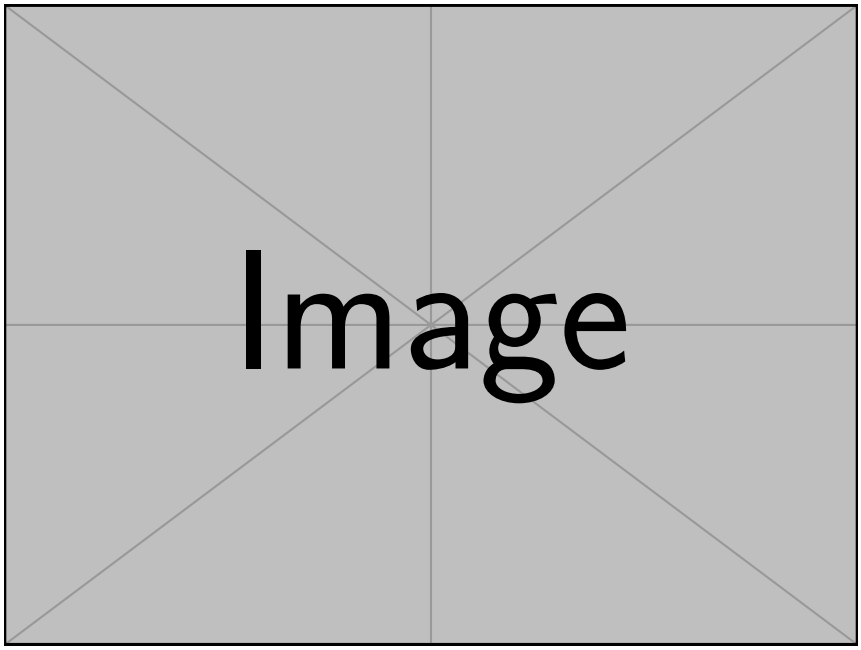

图 A.1 附录中的图片

B 另一个附录