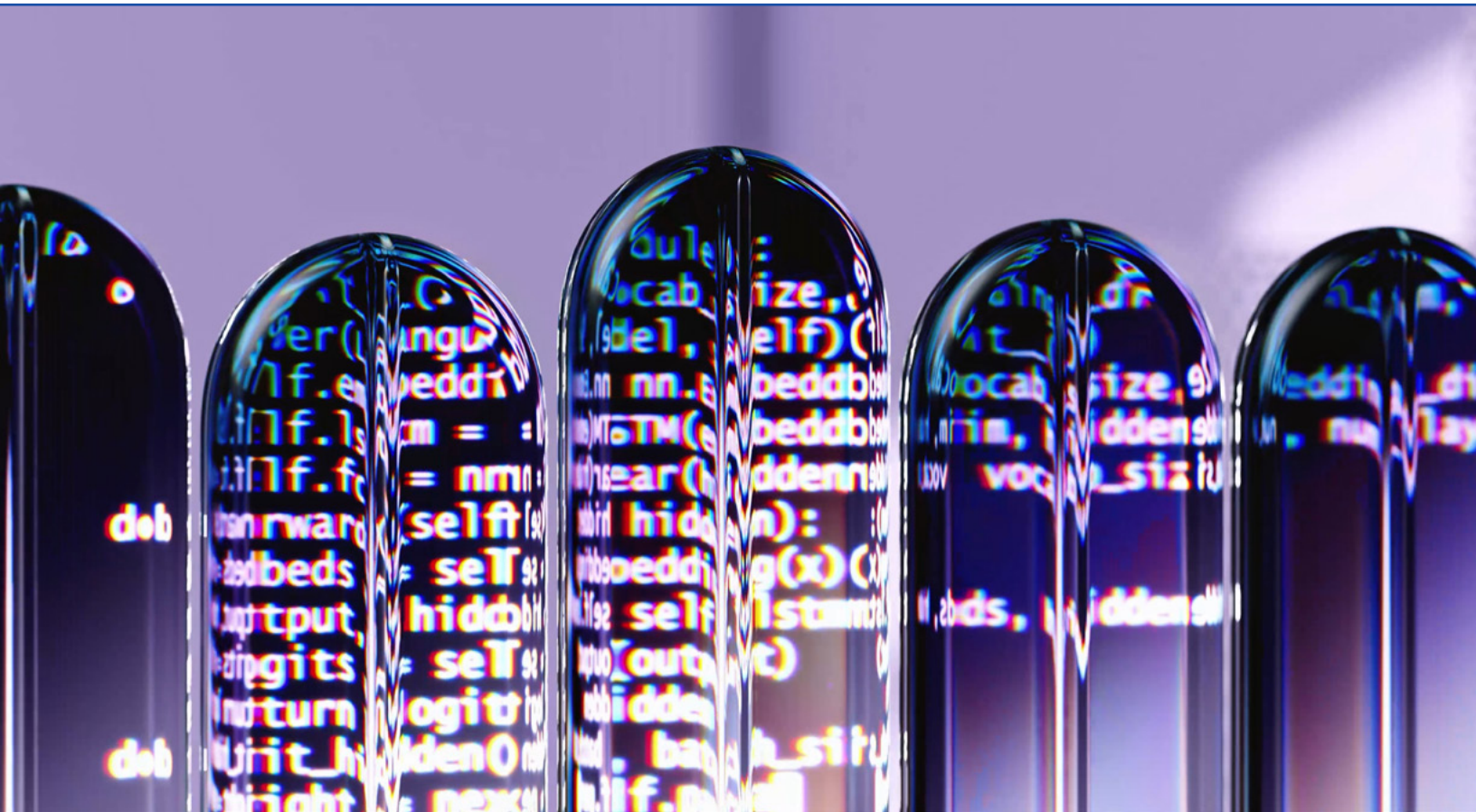
Google

2024年4月

安全的AI框架

保护人工智能软件供应链

作者：Shamik Chaudhuri，Kingshuk Dasgupta，Isaac Hepworth，Michael Le，Mark Lodato，Mihai Maruseac，Sarah Meiklejohn，Tehila Minkus，Kara Olive



摘要

随着AI功能在软件应用中的普及，我们看到许多与传统软件相同的问题，但速度加快。 随着AI进一步融入日常产品，威胁形势不断扩大，因此我们可以预期更多的攻击。 考虑到构建模型的费用，供应链解决方案显然是必要的。

本文解释了我们保护AI供应链的方法，使用溯源信息，并为其他组织提供指导。 尽管传统软件开发过程和风险与AI开发有所不同，但我们可以借鉴过去十年的工作，使用Bortg的二进制授权（BAB）、软件工件的供应链级别（SLSA）以及通过Sigsttorte的下一代加密签名解决方案，并将这些方案调整到AI供应链中，而不是重新发明轮子。

根据内部流程和平台的不同，每个组织对AI供应链安全的方法都会有所不同，但重点应放在可以在相对短时间内改进的领域。

读者应注意，本文的第一部分提供了关于“传统软件和AI软件的开发生命周期”的广泛概述。然后，我们具体探讨了AI供应链风险，并解释了我们利用来源信息保护AI供应链的方法。

更高级的从业者可能更喜欢直接转到“AI供应链风险”、“AI供应链安全控制”甚至本文末尾的“从业者指南”部分，这些内容可以根据任何组织的需求进行调整。

目录

目录

介绍

在2023年和2024年初，发现了几个恶意的AI模型——危险代码伪装成安全的、自由共享的模型。 下载这些模型来构建AI能力的毫无戒心的用户，实际上收到的是携带有害功能的程序，包括窃取数据或安装后门，使攻击者能够在用户的机器上执行代码。

开源和开放科学平台Hugging Face正在通过与安全研究人员合作来解决这些攻击问题。 该平台还提供了一种解决方案，即为将模型上传到平台的开发人员提供使用GPG密钥签署其模型的能力，这是一种公钥加密形式，允许用户在下载时验证模型，以确保它们未经修改来自可信的创建者。

不幸的是，这种解决方案并不经常被使用，可能是因为GPG签名引入了持续的密钥管理工作，这可能会增加工作量，容易出错，同时也会减慢上传过程。

这种类型的攻击对于涉足软件供应链领域的任何人来说并不新鲜。 随着人工智能扩展成为更主导的开发形式，我们发现许多过去在传统软件中出现的问题现在也在人工智能领域发生，但发生的速度加快了。

我们可以预期未来会看到更多类似这种攻击的情况，因为人工智能正在进一步融入日常产品中。
不过，也有好消息。 首先，已经存在的软件安全措施可以并且应该应用到人工智能生态系统中。 其次，我们已经学到了许多有关如何最有效地扩展这些解决方案的方法。 正如GPG密钥示例所示，如果开发人员不愿意或无法使用安全措施，那么这些安全措施就不起作用。

在过去几年里，软件行业与各国政府合作，修复传统软件供应链中的安全漏洞。

介绍

通常，这意味着改变全球开发人员使用的常见做法，并对现有基础设施进行改造，以使其能够抵御仅在被利用后才被发现的漏洞。 其中一些教训是艰难获得的，但幸运的是它们也是可转移的。

我们现在有独特的机会，随着人工智能开发变得更加普遍，从一开始就将这些解决方案构建到人工智能的新兴基础设施中，而不是在问题变得更难解决时再来解决。

本白皮书是描述我们实施谷歌安全人工智能框架（SAIF）方法的系列之一。 本文旨在面向广泛的技术受众，旨在帮助既想了解安全性的人工智能从业者，也想了解人工智能特定需求的安全从业者。

我们已经包含了两个实践领域的入门材料，因此专家可以选择跳过涵盖其领域的背景部分。

我们解释了我们保护AI供应链的方法，并为其他组织提供指导。[1]特别是，我们认为AI生态系统可以利用一种传统的供应链治理技术，即溯源，这是一种元数据文档，用于捕获和保护有关工件的信息以及它是如何创建的。

注重安全的用户可以通过验证模型生产者的身份来保护自己免受本文开头描述的攻击，以确认模型是否未经更改地来自他们期望和信任的生产者。 我们认为防篡改的溯源对于保护AI工件和数据以确保AI供应链的安全至关重要。 溯源还可以提供审计基础，以解决紧迫问题，例如允许训练管道推理版权以避免潜在的侵权问题。 更广泛地说，来源可以支持基本的水平，如治理和保证、合规性和事件响应。

1. 本文不涵盖机密性、隐私或模型行为质量等重要主题，这些将在专门的未来白皮书中讨论。

01

传统软件和AI软件的开发生命周期

以下各节介绍了我们在传统软件（指非AI软件）的开发生命周期和AI特定开发生命周期背景下所指的软件供应链的含义。

传统软件供应链

我们生活和工作的许多方面都由软件应用程序提供支持或协助。 但这些软件是从哪里来的呢？ 我们怎么知道我们是否可以相信它会按我们的期望行事？ 它的所有组件是否都已经正确获取？

这些是软件供应链安全领域致力于解决的一些问题。

当我们谈论软件供应链时，我们指的是导致软件制品创建的一系列步骤。

在传统软件开发生命周期中，开发人员向代码库贡献代码。 然后，使用外部依赖项，开发人员构建一个可执行文件，然后部署到软件包存储库。

稍后，其他开发人员将下载软件包以部署到生产服务中。

传统软件开发生命周期

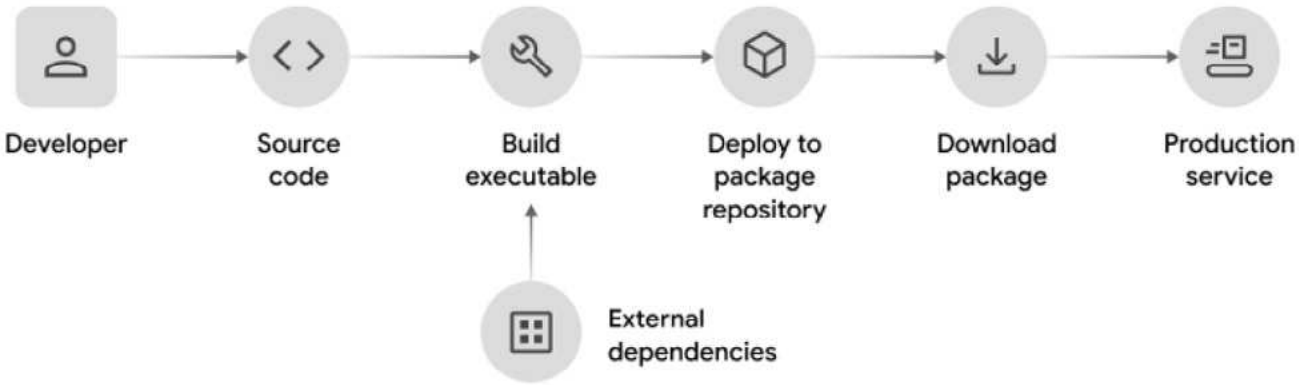


图1：传统软件开发生命周期：开发人员向代码库贡献代码。然后，使用外部依赖项，开发人员构建一个可执行文件，然后部署到软件包存储库。 稍后，其他开发人员将下载软件包以部署到生产服务中。

01 — 传统软件和AI软件的开发生命周期

图1展示了将开发人员贡献的源代码通过构建过程转换为制品的过程。 当我们谈论制品时，我们指的是一组序列化位的集合，可用作软件的输入或输出——数据集或代码库、软件包或OCI容器镜像、移动应用程序或AI模型。 当我们将输入工件转换为一些参数（如命令行参数或配置文件）并生成一些输出工件时，这构成了一个构建。2

如果没有保护，这个过程中的每个阶段几乎都可能成为供应链安全问题的源头。 Russ Cox，一位谷歌杰出软件工程师和Go编程语言的作者，将软件供应链安全框架描述为针对两类问题加固供应链：攻击和漏洞。

攻击和漏洞有何不同？ 在供应链的背景下，术语攻击表示软件在交付之前的恶意篡改。

例如，如果一家软件公司的工程师秘密用带后门的镜像替换新的操作系统版本，然后这个恶意镜像被发送给客户，我们会认为这是一次攻击。

2020年SolartWinds黑客事件是软件供应链攻击的一个显著例子。 入侵者在关键网络软件的软件更新过程中安装了木马，导致成千上万家公司和美国政府出现网络后门。

相比之下，漏洞通常是软件设计或实现中无意的缺陷或其依赖关系，这些缺陷可能只在软件发货后才对人类可见。

2. NIST SP 800-204D，“在DevSecOps CI/CD流水线中集成软件供应链安全策略的策略”提供了一个很好的概括性模型，展示了软件供应链包括个别转换“步骤”。请参阅htttps://csrtc.nistt.gov/pubs/sp/800/204/d/ftinal。

我们认为这些是软件中由于软件依赖关系而积累的可利用弱点——这是一个复合问题，因为许多项目具有大量依赖关系，这些依赖关系本身又有自己的依赖关系，依此类推。例如，Log4Shell是一个影响数百万基于Java的应用程序和设备的供应链漏洞。 这个漏洞让软件行业匆忙地修补和更新受影响的Log4j软件包，当发现一个为无辜目的创建的功能集可以被利用，在一个网页表单上输入一些微不足道的内容就可以在远程主机上执行远程代码。 这个漏洞在某种程度上如此具有影响力，因为Log4j是一个经常出现在软件依赖图中的软件包，有时深达十二层的依赖关系。

供应链安全存在两个主要关注领域：依赖跟踪以便在受到威胁时能够快速反应，以及防篡改保护以防止通过对软件构件进行恶意修改而受到威胁。

依赖跟踪

依赖跟踪对于管理漏洞和软件许可证都是有用的。 当一个软件项目导入一个现有的软件库来启用其功能的部分时，它可能会从这个依赖项中继承安全漏洞以及许可证限制。

如果一个广泛使用的软件库或组件包含可以被利用的意外行为，那么依赖于它的任何其他软件包可能会有漏洞。 然而，这并不是一定的—例如，包含它的许多软件包中可能不会执行易受攻击的代码路径。

确定易受攻击的代码是否被执行是困难的，因为这通常是一个不可判定的问题。 近似值依赖于良好的仪器设备和日志记录实践，以便启用事后分析。

此外，即使确定了软件包明确存在漏洞，修补也是一个困难的过程：首先需要修复主要错误，然后逐渐迭代地将补丁推出到所有受影响的下游依赖项中。 正如Log4Shell事件所展示的那样，在软件和受影响包之间存在许多层依赖关系时，这是一个困难的过程，因为一些语言生态系统需要多次中间更新才能完全修补下游包。

软件材料清单框架（SBOM）通过提供软件中的“成分”列表来帮助我们编码依赖关系。

SBOM还不是一个完整的解决方案，但它促进了行业范围内依赖关系发现的进一步进展，以便我们可以跟踪、衡量并最终纠正软件生态系统中漏洞的传播。

SBOM还可以用于跟踪与软件工件相关的软件许可证。 软件许可证用于指定包含它们的应用程序中开源库的预期使用方式。 当一个组织跟踪其软件工件中使用的所有软件许可证，并注意使用软件的方式时，它可以确保尊重开源库作者的意图。

篡改

当用户下载软件应用程序时，他们如何确定它是否按照其预期的源代码规范运行，而不包含恶意更改？

图2展示了攻击者或恶意内部人员可能在整个软件开发生命周期中向应用程序注入意外代码的一些关键点。

传统软件中的供应链风险

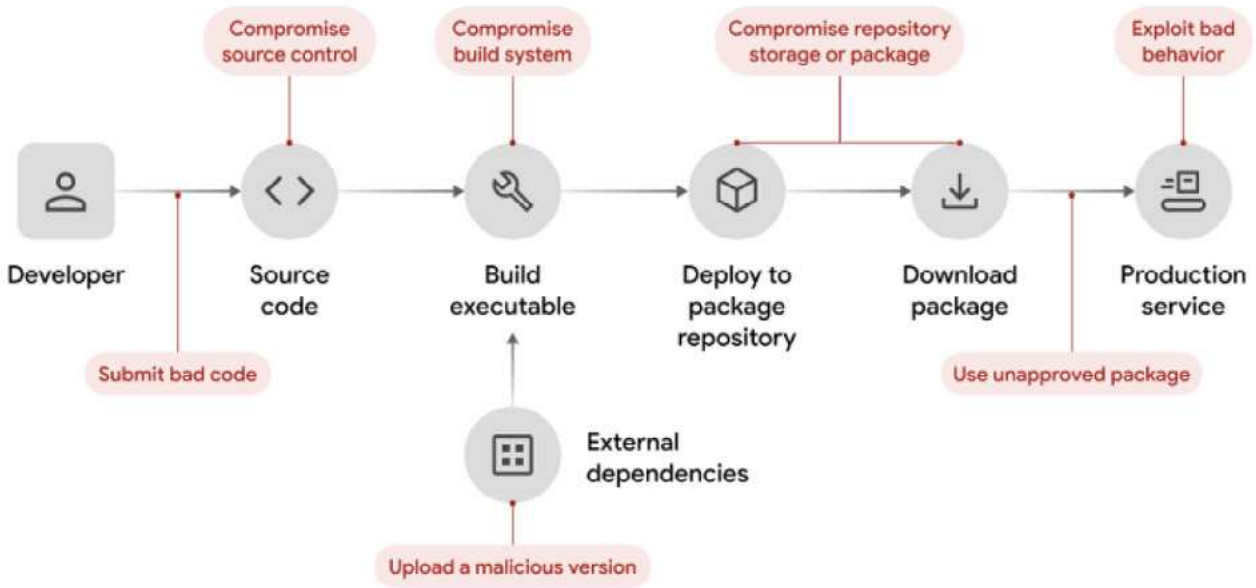


图2：与创建软件工件相关的供应链风险。

特别是，攻击者可以通过对代码、构建系统、任何相关的二进制依赖项或最终发布软件工件的软件包存储库进行未经批准的更改，从而修改软件工件的行为。 像软件工件供应链级别（SLSA）这样的解决方案帮助软件提供商采用维护将源代码与最终软件应用程序链接的加密监护链的最佳实践。

这个链条由一个**签名的起源**文档代表：一个关于软件工件如何生产的防篡改证明。

签名来源设计旨在通过在链条上施加额外的防篡改机制来补充SBOMs。[3]来源还通过进一步提供有关构建的信息（用于创建工件的工具和流程）和构建依赖关系，补充了有关工件来源和所需运行时依赖关系的信息。 简而言之，SBOM提供有关工件中包含什么的信息，而签名来源提供有关工件的创建方式的防篡改信息。有关工件创建细节的信息。

3. htttps://slsa.dev/blog/2022/05/slsa-sbom

AI软件供应链

以下部分将将传统软件供应链的分析应用于AI软件供应链，以说明相似之处和差异之处。

前面讨论的这两个关注领域——依赖跟踪和篡改——也适用于AI软件供应链，但对于AI特定方面的框架略有不同：

依赖跟踪：与传统供应链一样，查找并修复引入AI工件和基础设施中的错误很重要。 然而，对于AI，出现了一类新的依赖关系：用于训练模型的数据集。 跟踪能力不仅与安全和隐私问题相关，还与基于版权的使用限制管理相关，类似于与软件许可相关的问题。

工件完整性：签名和验证**AI**模型的能力，并确保它没有被篡改，类似于签署传统软件工件。

01 — 传统软件和AI软件的开发生命周期

要了解这些关注点在应用于AI时的差异，让我们看看AI应用通常是如何开发的，而不涉及特定模型的细节。

AI应用的核心概念是模型[4]。 从高层次来看，模型可以看作是代码和权重的配对，作为训练过程的一部分创建，这只有在

当部署在基于人工智能的应用程序中时。 模型的目的是从数据中提取统计模式，并利用这些模式对使用人工智能的应用程序中的新数据进行预测（也称为推理）。在这个高层次上，该过程看起来像下面的图表：

ML模型开发生命周期

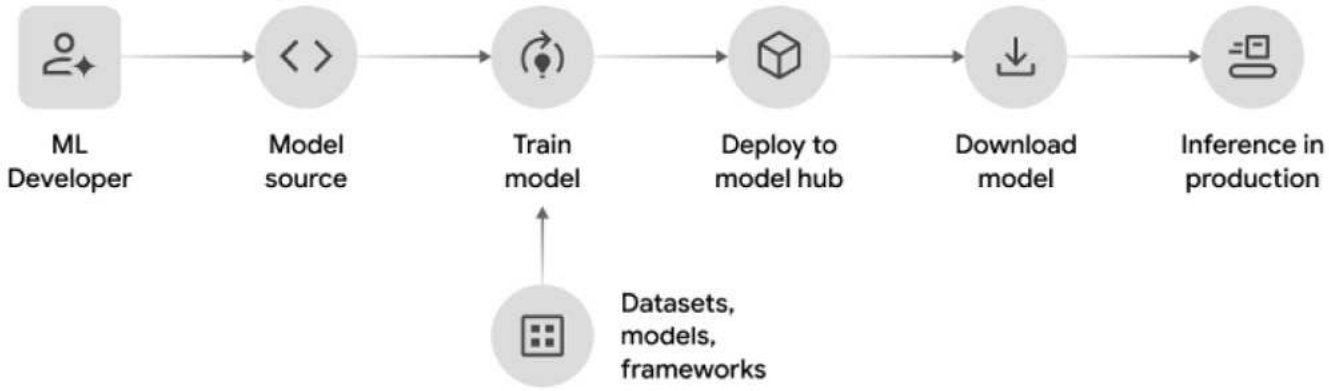


图3：AI模型的生命周期：AI开发人员选择模型架构，并使用外部数据集、预训练模型和模型训练框架来训练模型。 模型部署到模型中心，稍后将从中下载以在生产环境中用于推理。

4. 值得注意的是，仅保护模型本身并不足以保护AI系统。 您还需要保护其训练数据、用于训练和提供模型的基础设施，以及使用模型的应用程序。

以下对训练AI模型的简要介绍侧重于与供应链风险相关的开发方面。 有多种类型的模型，各种不同格式、大小和不同目的；然而，主要的供应链关注点对所有情况都适用。 正如图表所示，高层次的故事与传统软件开发非常相似。

数据集

要开始训练一个模型，我们需要从中提取模式的数据。
数据集可能比模型架构源代码更重要：无论模型有多复杂，只有在合适的数据上训练时才能表现良好。 因此，数据采集要求从业者在早期提出一些问题：

- 系统的预期用例是什么—它有助于完成什么任务？
- 为了执行模型需要执行的任务，需要回答哪些问题？
- 什么数据可以训练模型来回答这些问题？
- 哪些数据来源可能符合从业者和最终用户的需求？
- 数据是否高质量、完整、准确和相关？
- 在训练中使用的数据集是否存在道德和法律问题？

一旦确定并获取了足够的数据来源，从业者将其导入本地存储以加快训练速度。 在这里，我们看到了第一个供应链风险：(a)[5]数据可能在导入过程中或之前被恶意篡改。

一旦数据被导入，通常需要进行清洗和转换，这些过程通称为数据增强。 数据可能需要新的标签来帮助训练模型；它可能包含低质量、重复或不一致的记录；或者它可能与任务要求的格式不同。 如果从业者使用多个数据集，他们可能还需要解决格式和内容上的不一致性。

5. 本节讨论的风险在图5中用相同的字母标记。

人类和算法标记者可以用来标记、过滤或转换数据。 从供应链的角度来看，人类可能会恶意地错误标记数据。 或者，算法标记者可能存在错误，导致数据转换不当。 所有这些都会影响训练模型的性能。 这是另一个供应链风险：(b) 意外的（恶意或不正确的）训练数据可能被用来训练模型。 从训练过程的角度来看，这种风险包括数据污染（上述风险(a)），但这种风险更普遍，因为它影响了最终用户所看到的工件。

一些转换的例子包括：

- 删除重复记录
- 从另一个数据集中补充缺失字段
- 更改特定字段的格式或比例
- 通过转换现有数据点来添加新示例（例如，旋转图像）
- 通过使用不同模型生成合成数据来增加用于训练的数据量

由于所有这些转换导致的数据集与输入版本不同，从供应链完整性的角度来看，我们需要跟踪这些操作。 这引发了血统的概念：元数据捕获所有在数据集和其生成的模型上执行的预训练转换。 血统类似于传统软件供应链中的溯源，但溯源更广泛，因为它还涵盖基础设施元数据和输入输出的加密签名。 由于训练期间使用的数据对模型的后续性能至关重要，因此我们必须捕获与数据集操作相关的所有供应链血统和溯源信息。 血统和溯源将构成AI模型治理的基础，包括为训练模型所接受的受版权材料建立政策和控制。

模型

模型是开发AI应用程序的核心概念。
在非常高的层面上，模型是一组权重 - 决定每个特征（数据属性，数据列等）如何影响输出的参数。
训练模型意味着调整权重，直到预测输出接近示例目标标签来自训练数据集。 示例标签和实际标签之间的距离由损失值来衡量，训练过程旨在最小化该值。

(有关该过程的入门解释，请参阅这个机器学习速成课程。) 举个轻松的例子，想象一下为

挑食者选择每日午餐的任务。 你可能会考虑许多因素：他们上次吃了什么；建议食物的温度；它的颜色；它包含的口味；营养价值；口感；一年中的时间；挑食者是独自用餐还是与朋友一起。 有些特征将起更大的作用，而其他特征的权重有时可能取决于其他特征值。

通过对所有这些特征进行分类并跟踪挑食者偏爱的菜单，您可以随着时间推移学习特征的权重，从而逐渐推断出一棵决策树，帮助您未来更好地规划菜单。

在开发更复杂的模型时，包括大型语言模型（LLMs）和多模态基础模型（在大量数据上训练以执行各种任务的模型），也使用相同的特征、目标标签和损失概念。 在这里，模型执行多个计算，合并数据和权重以创建中间计算。 这些计算的结果然后与其他权重和计算混合在迭代过程中，直到最终预测可以产生。

生成的模型架构被称为**计算图**：它表示数据从输入到预测的正向流动。 这个图记录了推断过程中发生的所有计算。

从头开始训练大型模型是昂贵的，需要大量的时间和资源。 通常，开发人员会从预训练模型开始，以减轻这种负担，然后在其上构建一个新模型。例如，开发人员可以采用预训练模型并对其进行迁移学习。

迁移学习教一个为特定任务训练过的模型执行不同任务。微调是一种迁移学习类型，它冻结大部分模型权重，只更新模型架构中的最后几个计算。

迁移学习过程

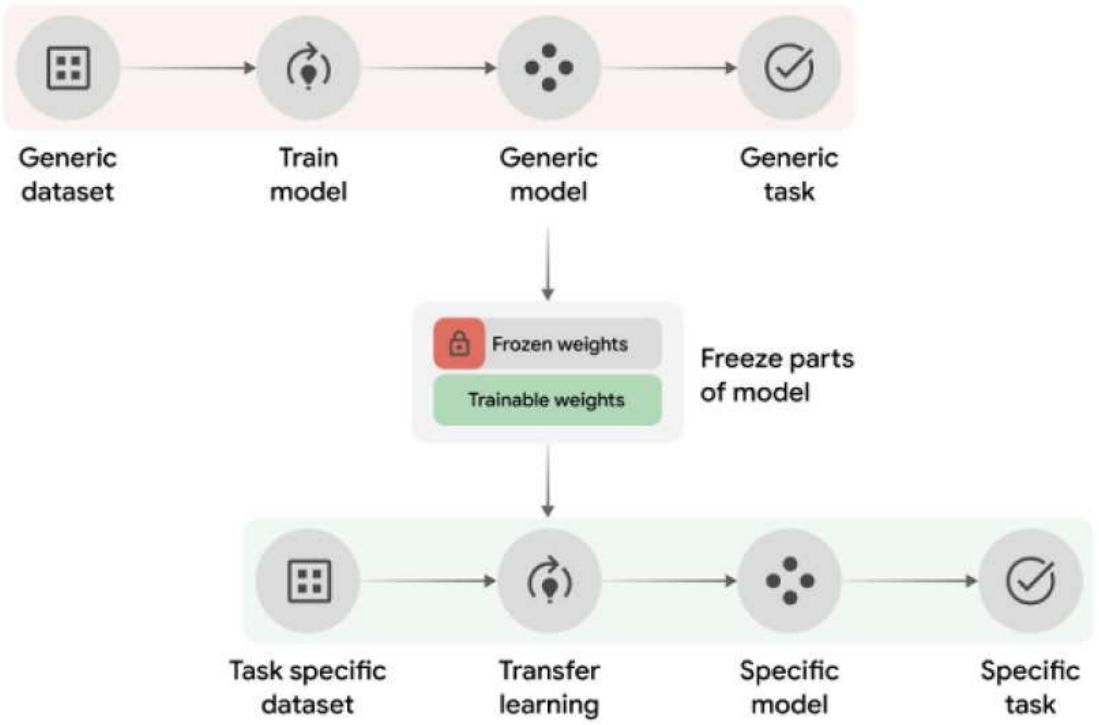


图4：迁移学习过程概述。 一个通用模型在通用数据集上训练，以在通用任务上高效执行。 对于微调，模型权重的前几个阶段被冻结，然后模型再次在特定任务数据集上进行训练。

这些技术使模型能够以比从头开始训练更低的成本学习新任务。 例如，开发人员可以拿一个针对大型通用图像识别数据集训练过的模型，然后对医学诊断扫描的一个更小的数据集进行微调，从而得到一个能在X光成像中检测异常的模型。

类似地，在LLM中用于语言和语法表示的组件可以在一个新模型中重复使用，该模型在特定文本生成类别上表现良好。

当我们为特定应用程序调整预训练模型时，我们需要确保供应链元数据被正确记录。 这意味着为每个训练数据集捕获来源，从任何训练过程中提取元数据，以及（因为通用模型可能由不同实体训练）预训练模型的来源。 一个未受良好保护或来源的通用模型可能对从中派生的任何模型构成供应链威胁—(c) 它可能已被恶意训练（包含后门或在特定任务上表现不佳），或者在训练和微调之间可能已被篡改。

其他模型开发场景涉及将多个模型的输出合并为一个更大的模型。

举个例子，考虑一种称为专家混合（MoE）的技术。
通过这种技术，每个单独的模型可以解决问题空间的一部分，但在其他情况下表现不佳（例如当特定类型的特征存在时）。 通过综合每个单独模型的预测，我们可以实现一个整体性能更好的模型。 在使用MoE时，我们再次需要所有涉及模型的来源元数据，以便完全了解生产中使用的最终模型的供应链。

模型序列化

一旦模型训练完成，我们将其投入生产管道中使用。 例如，我们可以创建一个基于一些关键输入生成房地产列表的Web应用程序。 为了创建这样的应用程序，我们需要序列化我们的模型，然后可以选择将其存储在一个中心，从中可以为每个新应用程序下载。

序列化允许我们将模型转移到新环境中，这很有用，因为用于训练的硬件和基础设施通常与生产推理基础设施不同。 从供应链完整性的角度来看，我们需要确保模型在存储过程中不能被篡改（图5中的风险（d））。

一种模型存储的方法是将权重记录在一个文件中，每个应用程序在使用模型之前都会解析该文件。 这被称为检查点，广泛用于训练过程中，AI从业者会定期保存长时间训练循环中的权重。

如果进程中止或行为异常，训练可以从最后一组权重或**检查点**重新开始。

有几种将模型序列化为检查点文件的方法：

- 使用语言提供的预定义序列化功能，例如在Pytthon中使用pickle。根据实现方式，这可能存在安全风险，因此我们不建议使用—SafteTensorts库是一个很好的替代方案。

- 使用库将权重存储在库可理解的格式中，例如，对于使用 numpy训练的模型，我们可以按照NPY格式将检查点保存在文件中—在某些情况下可能存在安全风险。然而，这意味着每个推断应用程序必须重用训练过程中使用的相同代码。 因为这会在训练代码和推理代码之间创建一个硬依赖关系，我们不建议将其用于更高级的模型。

- 将权重和模型架构打包成一个单独的实体。 这可以是一个单个文件（例如，用于TFLitte的ftlattbufterts文件或用于Pyttortch的.ptth模型的压缩包）或一个结构化的文件和目录集合（例如，TensortFlow SavedModel）。有时，本应存储在单个文件中的大型模型也会分割成多个文件以加快加载速度。 请注意，即使这些格式也可能存在安全风险，例如。pth文件可以使用 pickle，而SavedModels可以使用 Lambda 层来运行任意代码。

将权重和架构捆绑到单个软件包格式中，使开发人员能够更轻松地在应用程序之间传输模型，只要他们在其基于人工智能的应用程序中集成了模型格式的解释器。

解释器解析模型结构和权重，构建适当的内存布局以执行推断。

一个单一的软件包还允许解释器和用于训练模型的框架分别演变。

只要保持兼容性，使用一个版本的框架训练的模型可以与匹配另一个版本的解释器一起使用。

对于一些现有的序列化格式，可以实现前向和后向兼容性，实现训练过程和在生产中使用模型的应用程序之间的完全解耦。

由于模型序列化代表在我们供应链中创建一个新的工件——即检查点——这是另一个我们应该记录起源的地方，纪念执行的操作（图5中的风险（c））。

为每个新工件或检查点记录完整的起源有助于开发人员跟踪存储或模型序列化过程中引入的风险。

在记录模型之后，它也可能经历模型量化。这个过程将一个完全训练好的模型缩小，通过将其权重转换为低精度整数，并在某些情况下，还用可以操作量化权重的操作替换模型的计算图中的操作。这增加了模型的效率，特别是当它部署到嵌入式/移动应用程序时，通过允许整数操作消耗更少的计算资源和时间。 这种转换导致的精度损失不足以在推断过程中导致模型错误预测。

量化模型的过程也是在构建一个新的工件，因此我们需要记录相关的供应链元数据，以维护更新模型的来源（这也在图5中以风险（c）表示）。

总的来说，我们需要记住模型不容易检查：模型的行为受其权重的影响很大，但考虑到大量的权重和二进制格式，人类不可能分析权重以预测模型可能做什么。在某些存储格式中，分析计算图也很困难。 相反，我们采用这样的观点，即模型是程序；它们类似于在运行时解释的字节码，以产生一些有价值的行为集。

然而，与传统软件不同，通过逆向工程理解二进制是可行的，模型是不透明的，意味着只能通过观察所有可能输入的一小部分来部分理解它们的行为。 考虑到构建模型的费用，完整的供应链来源信息是必不可少的。 例如，在训练平台遭受攻击的情况下，我们可以使用这些信息快速识别需要分析和重新训练以消除潜在篡改的模型（图5中的风险（e））。

训练框架

AI从业者通常不会从头开始编写代码来训练模型。 相反，他们使用针对可以利用可用硬件的操作进行优化的库。

强大的库——如JAX、TensortFlow和PyTortch——能够利用训练主机上的硬件加速器——GPU和TPU——显著提高训练速度。

对于大型模型，框架可以跨多个主机分布计算，有效地管理调度和网络通信。

这些特性使训练框架变得复杂，为引入漏洞提供了机会（风险（ft）见图5）。

因此，训练库是供应链的关键部分，因为它们可以生成受漏洞影响的模型。

记录有关训练框架的来源可以帮助我们识别使用后来发现存在错误的数学运算实现或编译模型特定代码以在硬件加速器上运行的框架训练的模型。

- 训练期间的人工评估：LLMs和基础模型还通过人类使用强化学习与人类反馈（RLHF）进行评估。 定期，一个检查点用于回答各种提示，人类对答案进行评分。 这为模型提供了关于哪些答案适合提示的信号，训练过程将相应地更新模型权重。

评估

在训练模型时，我们希望确保它们在其任务的示例上有效执行，而不会过度拟合到它们训练过的特定示例上。 在过程中有多个关键时刻我们评估性能：

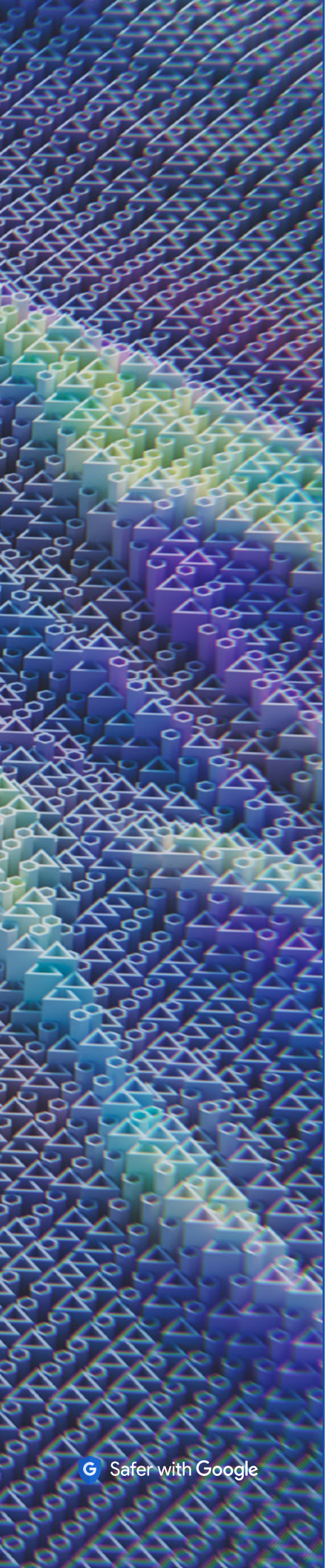
- **训练期间的自动化测试：**这使用训练数据的专用部分。 在训练循环期间，不是将数据提供给模型，而是在每个检查点评估模型，作为模型在训练期间未见数据上的表现的代理。

由于RLHF和类似技术会影响最终模型，我们应该捕获来源以在AI供应链中包含有关这些过程的信息。

如果测试数据被篡改，或者人类评分者恶意鼓励错误答案，这可能会影响模型的行为（图5中的风险(a)、(b)、(c)）；记录来源可以让我们在发现时检测到这些篡改的影响。

6. 过拟合是指模型在训练数据上表现出色，但在新数据上失败的情况。

较大的模型，如LLMs和基础模型，在发布后也经常由第三方进行评估。 这些评估类似于传统软件中的集成测试或验收测试：它们不会改变软件，但它们允许组织在承认或部署之前决定它是否按预期运行。注重生产卫生和可观察性的组织可能选择在受信任的执行环境中执行此类测试，并将结果记录在签名证明中，以便他们可以确保模型在使用之前已经得到充分评估（图5中的风险(g)和(h)，其中(h)发生在模型在生产中被不当使用后）。



02

人工智能供应链风险

本节更详细地讨论了先前描述的AI开发生命周期如何导致AI特定的供应链风险。

在许多方面，训练、发布和提供模型的过程与传统的软件开发生命周期（SDLC）非常相似。 我们可以认为训练过程

（或在进行数据集增强时的数据转换）代表了一个“构建”。构建的“源”和“依赖关系”由训练框架、用于定义模型架构的代码以及数据集表示。最终，产生的“包”就是模型（或在数据集增强的情况下是训练数据）。

7. 同样，NIST SP 800-204D第2.4节中的“SSC模型”，即“软件供应链安全在DevSecOps CI/CD流水线中集成策略”，是这些概念的有用抽象 — 对于人工智能和传统软件同样适用。

与传统供应链风险的相似之处

在前一节中，我们已经展示了训练基础模型涉及多个组件和过程：

1. 从数据集开始，我们执行多个数据清洗和数据增强步骤。

2. 我们选择一个框架来训练一个新模型，将清洗和增强后的数据与先前训练过的模型结合起来。

3. 模型被记录为一个检查点，并可能被量化为更小的占用空间。

4. 检查点存储在模型中心。

5. 然后，检查点可以用于未来的训练步骤或部署在AI应用程序中。

每个步骤都可能受到无意中的缺陷或设计选择的影响，这可能导致供应链的妥协。 总结我们之前在模型生命周期中确定的风险：

- 数据可能在摄入过程中或在整理和清洗过程中被恶意或无意地污染。
- 训练平台可能容易受到攻击。
- 训练框架和库可能包含影响其计算的漏洞或后门。 例如，模型检查点或量化代码可能以可能涉及安全敏感方式更改模型架构或权重。
- 人工评分员，或在训练过程中的自动化测试步骤，可能会引入错误或恶意推断。
- 模型中心可能会受到损害，允许恶意开发人员操纵模型权重或数据集，从而影响生产用途以及使用预训练模型进行未来训练步骤。
- 在部署模型以创建AI应用程序时，开发人员可能会无意中使用训练不足或未经评估的模型，如果他们收到的模型与预期不同。

这些风险中的任何一个都可能使生产中的模型容易受到利用。

基础模型训练中的供应链风险

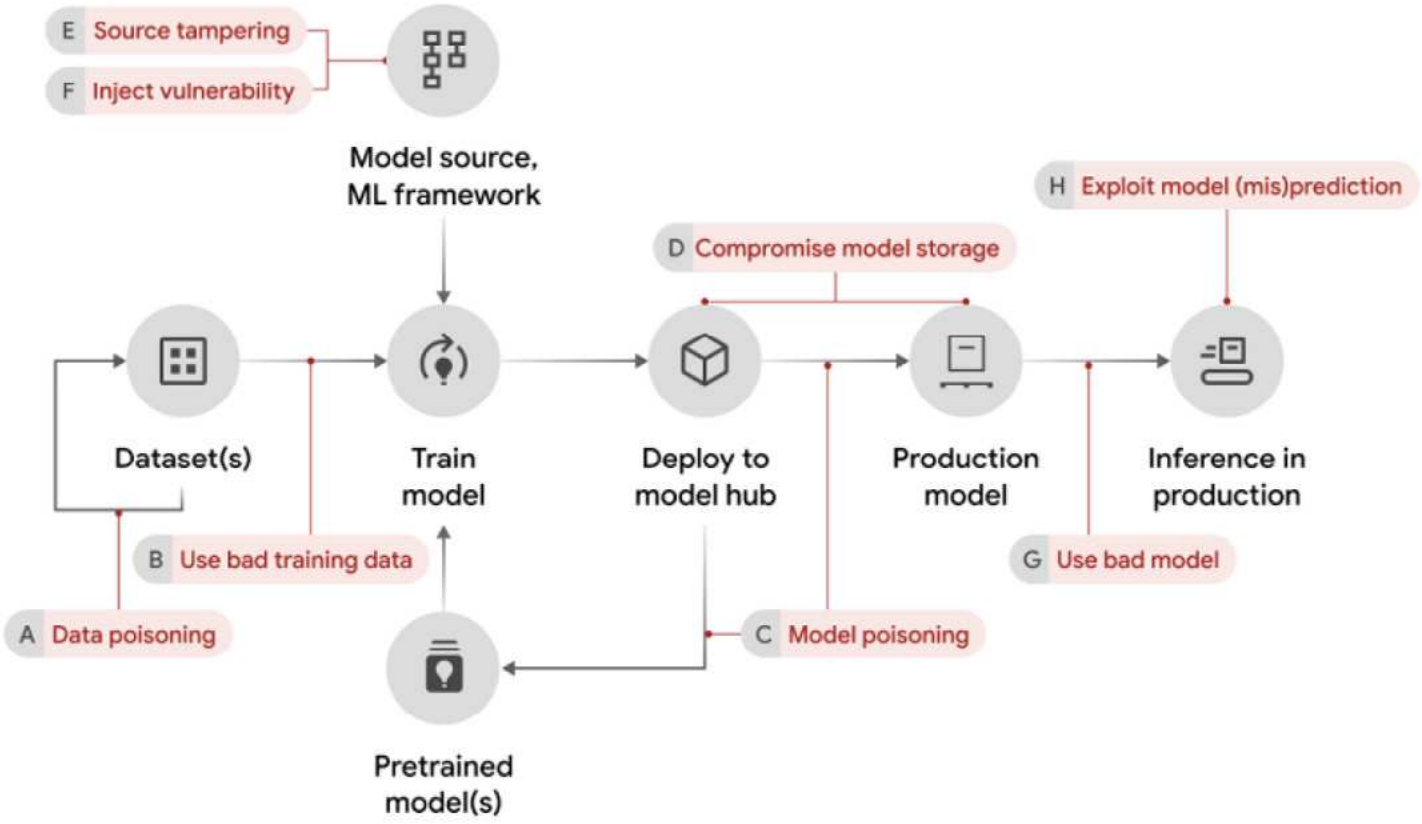


图5：基础模型的整体训练和相关供应链风险

为了保护这个复杂系统并确保模型供应链的安全，我们应该分析这个过程的每一步如何作为一个整体（图5）以及单独地运作。 由于训练是任何AI供应链中的共同因素，让我们来看看训练

供应链图和相关风险代表了整个模型开发过程中的风险。

这些风险的图表扩展了图3中所示的AI模型开发生命周期：

机器学习中的供应链风险

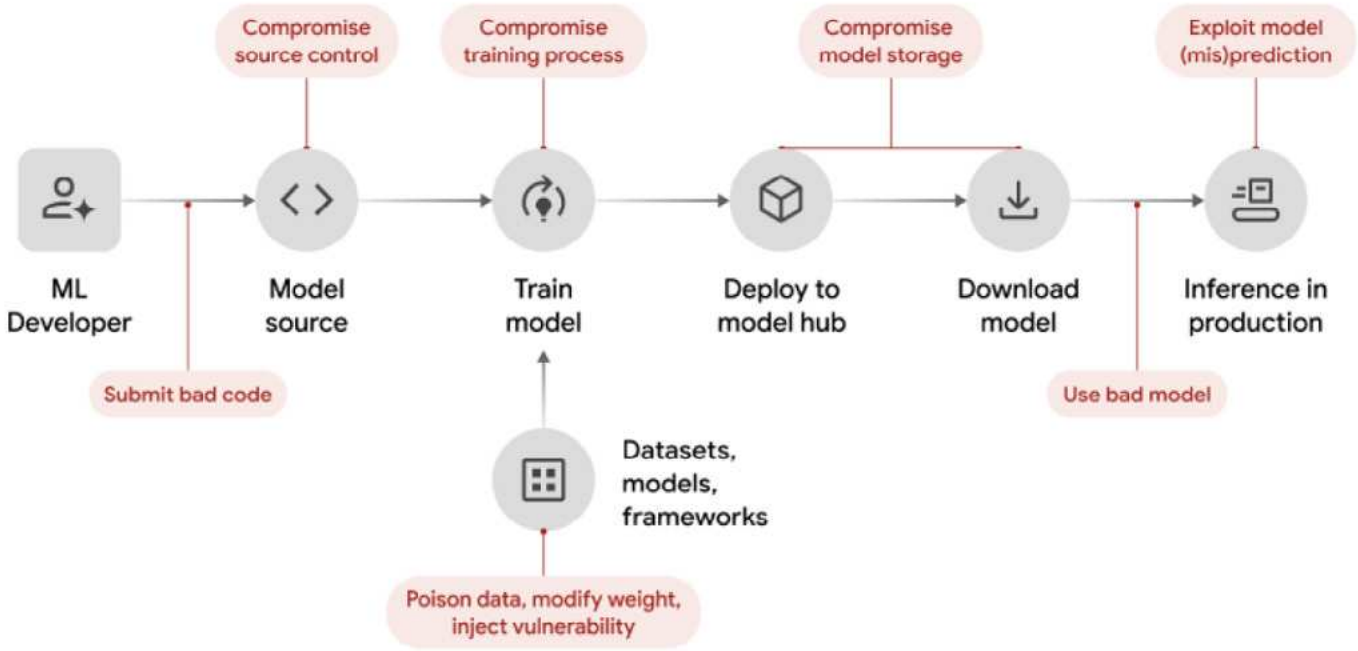


图6：训练模型相关的供应链风险

当我们将图6中的AI供应链与图2中传统软件供应链的风险进行比较时，我们会注意到明显的相似之处：即它们都包含了代码、版本控制、部署到存储库或中心，以及最终被包消费者下载的概念。 这些相似之处为本文后面讨论的风险和控制提供了信息。

AI开发中的特殊差异

正如我们刚刚解释的，AI开发与传统软件开发共享一个共同的形状
。
然而，在软件开发生命周期中
也有一些做法与当前AI开发的状态不同
。

让我们来看看一些传统软
件开发的特点，这些特点可能不会直接转移到人工智能领域：

- **源代码控制:**代码通常存储在版本控制系统中。
  这使开发人员能够共同在一个代码库上合作，跟踪代码库随时间的状态，并根据需要回滚或前进更改
  。

- **代码审查:**许多组织使用代码审查作为一种工具，以确保新的代码更改符合其可读性或正确性标准。

- **托管的脚本构建:**为了将代码转换为一个软件包，许多组织使用一个遵循一组预定义配置步骤的托管构建系统。

  这种自动化可以实现开发人员和发布周期之间更高的一致性，同时也减少了每个开发人员的维护负担。

- **短、廉价的构建周期:**如果开发人员发现最近构建的软件包中存在错误，或者决定他们想要更改一些功能，他们可以开始一个新的构建而不会产生太多额外的开销（尽管人们经常抱怨构建延迟问题）。

当我们看AI开发时，这些属性可能在很大程度上有所不同：

- **数据集的不成熟版本控制：**用于存储、更改和检索数据集的生态系统通常比用于代码管理的对应生态系统更少偏见和健壮。 大多数数据集的巨大大小也给版本控制解决方案的健壮性增加了额外的约束。 这意味着给定数据集的内容可能会通过清理、复制和扩展而发生变化，通常不会引起明确的语义版本号的增加。

- **数据的人工审查具有挑战性**：训练数据集很大，可能经常更新。
  在每次数据转换后进行高信心的人工审查可能是昂贵或不可行的。

- **ML训练通常不是完全脚本化的：**ML训练通常包括一系列不记录在任何中央配置中的临时增量步骤。 它也可能涉及一系列系统和框架，而不是完全在单个托管构建系统的范围内运行。 训练也可以依赖于专门的硬件，这使得在托管构建系统中常用的沙盒技术更难应用。 遵循隔离、可重现和确定性构建的黄金标准在训练ML模型时更难实现。

- 长时间、昂贵的训练周期：由于训练成本高昂，许多训练库支持额外的方式将新数据或代码“侧载”到已运行的进程中。

  这意味着注入输入到ML训练的控制平面比传统构建系统更多样化和复杂。

由于数据版本控制不如代码源控制成熟，跟踪数据集的来源并为训练准备和提供数据集变得更加困难。同样，由于训练过程可能比构建过程更复杂，因此随着输入和输出通过训练流动，跟踪模型输入和输出的来源变得更加困难。

Safer with Google

03

人工智能供应链安全控制

以下部分讨论可用的控制措施，以保护免受关键AI供应链风险的影响。

我们相信传统的供应链解决方案可以并且应该扩展到适用于*AI*开发。

尽管传统开发过程和风险与AI有所不同，但我们可以调整传统方法，例如将为代码设计的保护措施也扩展到数据。 例如，我们可以扩展来源概念以涵盖数据转换，扩展到ML从业者所称的血统。 没有必要重复造轮子。

AI供应链安全的端到端方法应该解决本文前面讨论过的两个需求：跟踪AI应用程序的所有依赖关系，从数据开始到生产模型结束，并确保所有工件的完整性。

为了满足这些需求，组织应努力能够自信地回答一系列关于生产中使用的AI模型的问题：

- 谁创建了这个模型？
- 用什么系统训练了这个模型？哪些系统处理了数据集？
- 自发布以来，模型有过任何更改吗？
- 这是哪个版本的模型？哪些版本正在生产中？
- 训练、测试和评估模型时使用了哪些数据源？
- 数据源在训练之前是如何处理或清洗的？
- 用于训练、测试和评估模型的代码框架是哪些？
- 对模型运行了哪些评估？
- 数据集是否适合在给定的训练环境中使用？
- 数据集是否具有任何特定属性（受版权保护的数据、许可限制、位置数据等），需要在训练之前特别注意和批准？

指导原则

为了回答关于供应链安全的关键问题，组织可以专注于能够在相对短时间内推动AI供应链安全的领域。

这包括：保护

处理、训练或提供AI模型的生产系统的完整性[8]。

组织需要确保所有用于预处理、训练、微调或提供的基础设施都能够抵御软件供应链威胁。

由于这是一个快速发展的领域，要确保系统在演变或添加新功能时仍然符合规定可能会具有挑战性。

为所有数据集和AI模型编目。除了保护AI训练和提供工作流程使用的基础设施外，我们还需要了解数据集（用于训练或基准测试）和模型本身的来源。 用于生成模型的所有输入是什么？

组织存储和使用哪些数据集，它们的相关属性是什么？

跟踪AI工件的来源意味着全面记录它们之间的使用关系。 除了实现供应链完整性，来源还有助于加快上线批准（确定AI应用是否可以作为用户面向产品发布）并调试在生产中观察到的行为（例如，确定在训练或提供模型过程中使用的编译器中的错误是否会影响模型推理的结果）。保护模型免受篡改，数据集免受污染。即使模型是在安全基础设施上训练的，它仍然可能容易受到进一步篡改的影响。

8. 第14章：部署代码构建安全可靠系统是如何保护传统软件生产系统完整性的良好参考。 那里提出的许多项目也适用于人工智能领域。

例如，攻击者可以复制模型并针对精心策划的数据集对新版本进行微调，导致其发出令人反感的推断，或者内部人员可以精确地覆盖一些模型权重以引起意外行为。

已经展示了针对野外模型的有针对性的供应链攻击，其中一个概念验证攻击通过精确修改一个流行模型来传播错误信息，并将其上传到看起来合法的存储库路径，以欺骗用户下载。对于模型以及一些数据集，通过使用签名和来源注释工件，可以减轻中毒的可能性。 这也凸显了健壮的数据清洗流程的重要性：如果数据清洗流程在溯源文件中得到妥善跟踪，组织就能够检测内部人员的恶意操纵。 组织还可以通过早期数据清洗实践来减轻从外部来源导入的受污染数据带来的影响。

通过早期数据清洗实践，组织可以在供应链的起始阶段发现并修复或替换有缺陷或易受攻击的构件。

发现并修补或替换有缺陷或易受攻击的构件。由于训练经历了多次迭代，在整个过程中不断添加数据和代码，模型可能以多种方式受到漏洞的影响。 为了防止利用并实现补救，我们必须跟踪用于生成模型的所有输入——包括代码和数据。 同样，我们必须维护组织使用或生成的所有模型的准确清单，以便在漏洞出现时做出响应。

此外，由于人工智能开发依赖于大规模数据处理，它也继承了与数据策划和共享相关的问题类别。

防止意外或恶意数据权利侵犯。在AI环境中，这个问题变得越来越重要。 特别强调需要确保训练过程只能使用经批准的数据集。

9. 数据集可以通过两种不同的方式收集。 第一种类型的数据集可以作为相对稳定的、版本化的工件导入：例如，导入、清理并在离散时间间隔内存储的公共领域数据集。 这种类型的数据集可以类似地对待其他类型的软件工件，并且非常适合签名和溯源工具。 然而，并非所有数据都符合这种形式：例如，如果考虑动态生成的数据库记录，比如基于真实世界互动不断写入的匿名化搜索日志，这可能需要更专门的方法。

谷歌的方法：BAB、SLSA和工件完整性

下一节讨论了谷歌内部对AI模型供应链安全的方法。

本文主要关注之前讨论的AI供应链关注点中的完整性和数据溯源方面。 漏洞管理是一个广泛的话题，将在另一份白皮书中更深入地讨论，但这里讨论的解决方案支持并促进了一个强大的漏洞管理计划。

我们对AI完整性的方法是通过扩展过去十年我们所做的现有供应链安全工作，使用二进制授权（BAB）为Borg，软件工件供应链级别（SLSA）以及下一代加密签名解决方案，如Sigstore。我们还使用特定的工具来编目数据，并捕获有关可能用于安全控制、治理和其他AI特定问题（如版权侵权）的AI工件的信息。

BAB和SLSA几乎涵盖了本文讨论的每一个风险；其余风险通过对AI工件进行数据编目来解决。

Bortg的二进制授权

BAB，或二进制授权为Bortg，指的是确保只有经授权的二进制文件可以在我们的生产集群管理系统（“Bortg”）上运行的控制。10

BAB最初是为了防范内部威胁而设计的。 在开发团队内部，每个人都彼此了解并信任；然而，统计数据显示，随着组织规模的增长，恶意行为者可能会尝试篡改代码或数据的风险也在增加。 此外，还存在这样的风险，即开发团队成员的工作站可能会被外部攻击者入侵，然后利用团队成员的权限篡改代码或数据。

10. BAB也被称为BCID，代表二进制和配置标识。

为了应对这一问题，BAB被引入以防止单方面访问：没有单个人可以对生产中的代码进行未经批准的更改。 随着时间的推移，BAB的保证范围已经扩大。
现在，它确保所有发布的软件都没有被篡改，并且其内容被充分理解。

BAB政策确保：

- 代码更改已经得到多人审查和批准。
- 构件是在经过防篡改的批准构建系统上构建的。
- 构建是可验证的，意味着构建生成的二进制或软件包可以通过检查元数据来进行可验证的审计，这些元数据标识了构建过程中使用的所有源代码。
- 元数据（来源）本身以防篡改的方式捕获。[11]
- 托管软件服务的作业配置也受到单方面修改的保护。

类似于非常详细的SBOM，这提供了深入了解用于在生产中创建工件的源代码依赖关系。 此外，BAB对谷歌构建系统施加的隔离和加密签名要求使我们相信工件在创建过程中或之后都没有被篡改。

现在谷歌在全公司范围内使用BAB来强制执行围绕生产完整性的技术政策。 通过使用技术手段在规模上强制执行供应链完整性政策，谷歌减少了内部风险，促进了可靠性和一致性，并使成千上万的团队能够以最小的开销遵守与SDLC相关的标准。

11. 捕获来源和捕获安全、防篡改的来源之间存在差异。 我们认为只有后者对供应链保护有用。

安全来源是在构建物品时生成的，在受信任的构建者的隔离环境中进行，受到干扰的保护。 加密签名确保来源在生成后不被篡改。

为了在大规模上实施技术变革，我们依靠了两个关键原则：

- 我们将供应链完整性定义为一个逐步攀登的阶梯，而不是一个全有或全无的“登月”努力。
  我们逐渐定义、实施并推出了一系列较小的改进，这些改进相互衔接，随着时间的推移大幅减少了攻击面。

- 我们与谷歌的基础设施所有者合作，实施“默认安全”的系统。 例如，谷歌的托管构建服务现在默认使用广泛的来源元数据对其构建进行签名，用户无需更改任何行为。 我们还自动化了在Bortg上管理来源验证和准入控制的政策，减少了谷歌部署服务团队的安全负担。

软件构件的供应链级别

在2021年，我们将BAB外部化为SLSA，即软件构件的供应链级别，以便其他组织能够从我们十年的进步中受益。 不久之后，我们将SLSA捐赠给开放源安全基金会（OpenSSF）进行跨行业合作。 SLSA已经从其谷歌特定的根源扩展到现在对各种规模的组织和软件生产商有用，包括开源软件生产商和项目维护者。

像BAB一样，SLSA被组织成明确定义的级别，每个级别都制定了越来越严格的安全期望。 这是有意为之，以便实现系统性的组织变革：一个给定的团队、部门或运营领域可以从任何点开始攀登阶梯，每个梯级都可以从前一个梯级到达。

在使用BAB来保护谷歌的生产系统时，这些级别的逐步提升为转变安全控制的成熟度提供了重要的结构。

同样，SLSA提供了一个清晰的路线图，包括基准、评估和目标。 分级结构通过定义明确的标准和可实现的步骤来使复杂变化更易管理。 该框架还为变革倡议创造了一个共同的词汇和背景。 这促进了团队和部门之间关于期望和进展的清晰沟通和对齐。

后一点——共享语言——反过来又提到了SLSA框架在整个行业中带来的一个重要好处。 除了概述供应链最佳实践，并启用自动化策略来管理和减轻风险，SLSA还为整个问题空间提供了一个共享的词汇和概念模型，这反过来适用于AI供应链空间。

模型来源

在谷歌的第一方和开源人工智能开发生态系统中，我们正在采用SLSA框架和格式来签署模型来源。

这个元数据文档通过加密将模型与服务账户绑定在一起——这是一个代表应用程序而不是人类用户的标识账户，用于训练模型。这也将使谷歌能够验证所有模型是否符合预期的签名密钥，这样内部人员就无法在不被察觉的情况下覆盖或更改模型（包括决定其行为的权重）。

与其他软件类型相比，实现更高的SLSA构建级别对模型来说尤为具有挑战性，因为训练过程漫长、资源密集且难以隔离。

我们目前正在探索从谷歌现有的全面数据访问日志基础设施中收集模型来源的方法，而不仅仅依赖于人工智能平台提供完整的材料清单。

这将使我们能够将签名的SLSA来源证书扩展到包括有关用于训练模型的数据的附加元数据：数据来源。

数据溯源

为了完全理解模型中的内容，我们必须了解用于训练的数据。数据溯源涉及记录在训练和评估模型过程中使用的所有数据示例的来源。 如果数据在训练之前进行了转换（例如，用于数据清洗目的），我们建议建立一个带有自己关联的溯源文档的单独数据集，链接到原始数据。

目前，数据溯源并未传达数据的供应链完整性。 正如前面提到的，我们正在研究用其自己的签名注释数据集的方法，这将提供完整性，以便知道它们在签名后是否被篡改。 我们目前对记录保留的细粒度有一些开放性问题：我们可以仅通过URI记录数据集，或者我们可以将数据集中的每个记录加密编码成一个结构，以便检测特定示例是否在训练过程中使用过。

在两个极端之间也存在可能性，并且我们正在分析沿着这个范围的权衡。

在谷歌内部，我们提供数据和模型目录工具，以改善团队开发和使用的资产在AI模型开发和部署中的分发和发现。 数据和模型目录侧重于元数据的快照和版本，以及数据或模型的属性和特性。

目录中的条目与存储在一个或多个位置的基础资产密切相关，这有助于用户筛选符合其正在进行的AI项目标准的可用资产，请求访问和合规批准，并在此后立即开始使用该资产。

每个数据集版本和模型版本还会被分配一个全局唯一的条目ID，该ID可作为我们在谷歌支持的许多数据和模型加载库的一部分。使用这些ID，我们能够将下游作业和任务的谱系关联到准确的数据集或模型摄取的版本或快照。

在谷歌内部，我们记录模型和数据集的
以下资产元数据：

模型属性	数据属性
名称	名称
唯一全局版本标识符	唯一全局版本标识符
创建日期	创建日期
更新日期	更新日期
格式	格式
位置	位置
大小	分类结果 (数据包含内容)
所有者	所有者
架构	架构
文档	文档
SHA-256哈希	SHA-256哈希

03 — AI供应链安全控制

溯源收集通常是组织中的一个事后想法，但我们已经了解到将其纳入模型训练工作流程中是困难的。

作为第一步，对于这些工件本身进行严格的簿记非常重要。 每个模型和数据集都需要具有全局唯一标识符，并且需要以不可变版本作为溯源收集的基础。

可靠记录溯源的方法包括：

- 显式溯源日志记录：在数据摄入或模型检查点库等I/O库中记录血统关系。 挑战在于实现对数据集和模型读取或写入的所有方式进行全面覆盖。

- 基础设施日志收集：记录所有文件访问的低级基础设施日志可以用于挖掘AI工件的溯源，但往往难以准确地将低级日志与模型和数据集关联起来。

- 执行沙箱：理想情况下，像训练或数据丰富作业这样的AI工作流提供输入和输出清单，并且沙箱限制了在清单之外的任何访问，同时记录每个输入和输出。

  这需要组织要求所有工作流在这样的执行沙箱内运行。

一旦组织证明了AI供应链中所有步骤的来源，他们可以制定政策，这些政策可以自动评估，以检测模型是否可以用于生产或训练新模型。 这导致了一种向左转的模型开发方法，因为在开发生命周期的早期阶段就可以检测到糟糕的模型或数据集，而不是在启动批准阶段之后，经过昂贵的训练和测试。

第三方AI模型开发解决方案

谷歌在保护开源软件世界方面投入了大量资金，包括支持SLSA和Sigsttorte项目。 在2023年，我们开源了我们的工作，以应用现有解决方案，如SLSA和Sigsttorte到AI。

在本节中，我们专注于开源解决方案，如SLSA和Sigstore，用于由不同组织的作者开发的模型，这些模型与使用这些模型的软件系统的开发者不在同一组织中。 这些可以是开放模型（模型的权重对终端用户可用），或者是完全开源模型（权重和训练源代码都可用）。 无论哪种情况，在使用这些模型时，我们应采取更严格的安全姿态，将模型发布者视为不受信任的。 我们正在探讨可以对模型提出的不同类型声明，以及这些声明如何可以公开验证。

在谷歌之外签名

对于签署开放模型，我们推荐使用Sigsttorte，这是一个免费的、公共的服务，通过抽象化大部分密钥管理复杂性，简化了代码签名和验证。 Sigsttorte的公共签名账本构成了SLSA来源的信任基础（在本文中早已介绍）。 构建后，可以使用Sigsttorte对工件进行签名。

用户随后可以验证签名，证明该工件自构建以来未被篡改。 由于Sigsttorte的签名过程支持使用临时或“短暂”的密钥，在签名事件后不久就会过期，因此开发人员无需长期维护或轮换密钥。 这解决了后续密钥泄露可能导致签名无效的风险。 与SLSA类似，Sigsttorte的优势也适用于人工智能供应链领域。

我们目前正在考虑如何对模型进行签名。 这可以被视为模型发布者声称用户正在看到该模型的权威版本——即发布者打算发布的版本。 请注意，我们可以区分模型的发布者（将模型上传到外部模型中心的身份）和模型的训练者（触发生成模型的训练作业的身份）。 我们在这里采用的威胁模型可能假设训练者可能不信任发布者或内部存储系统，并且可能希望在这些步骤中保护模型的完整性。

为了这些目的，我们有一个用于使用Sigsttorte签名模型的库。根据威胁模型，该库支持仅签署最终模型或在训练期间的检查点。

对于在共享环境中运行的长时间训练作业，最好签署所有检查点，以防止攻击者在训练作业中断后强制从损坏的检查点加载。

然而，我们意识到检查点可能非常庞大，签名过程可能会增加延迟，因此检查点签名默认情况下不激活。 我们正在探索并行化方法来进行签名过程，以便可以部署完整性保护而不会产生大量的延迟/效率成本。

我们计划将签名库与模型训练框架集成，以实现对模型的透明签名，而无需要求模型训练者更新其代码。

我们还将该库与模型中心集成，以允许在上传模型时对模型进行签名。 这两种模式之间的区别在于，尽早签名可以确保供应链的更大部分完整性，但并非所有模型训练框架都支持签名。 因此，与其拒绝上传未签名的模型，模型中心有机会在上传前提供签名。

一旦模型上传，模型中心可以验证其签名，并在模型卡上显示特定标签—这是模型的公开信息源，类似于代码存储库中的README.md文件。 这些简短的文档附加在模型上，用于描述模型是什么，如何训练的，其用例以及其他有用信息。 在模型卡中显示已验证签名的标签向用户发出信号，表明该模型具有完整性保护，安全意识强的用户还可以在将模型加载到生产环境之前自行验证签名。

我们建议进行模型验证，以覆盖模型中心本身被 comprtomise 的风险。

此外，我们计划为数据集提供签名，以减少数据污染的机会窗口。

这将加强数据存储系统已实施的其他缓解措施。

谷歌之外的完整性

虽然工件完整性有助于在工件构建后检测工件篡改，但从长远来看

我们还在考虑关于模型训练流程可以提出的可验证声明。

我们正在探索将控制功能构建到现有训练环境中，使模型发布者能够输出关于模型的可验证来源（例如，其代码依赖关系、训练环境，甚至使用的训练数据或预训练模型）。 这就是模型的SLSA发挥作用的地方。

然而，为了实现高水平的SLSA保证，生成来源的人必须在不受触发构建的人控制的环境中运行。 这意味着我们需要为AI模型定制可重用的训练器。 这些必须支持硬件加速器，这是传统软件世界通常不具备的要求。

我们目前正在努力将SLSA标准扩展到包括一个单独的依赖跟踪（以减少使用外部依赖项带来的风险）和一个单独的源跟踪（以提供在构建之前防止篡改的保护）。

有了来源追溯，我们还在探索更多高级功能，比如围绕训练数据中包含或不包含特定工件的可验证声明。

首先，我们考虑使用Crtoissantt格式来描述数据集。 虽然这是为了简化外部数据集的发现和理解而开发的，但我们可以将其用作确保以确定性方式计算数据集摘要的规范化层。

正如我们之前提到的，LLM的生命周期中一个重要组成部分是训练后评估。 目前，这些操作都没有安全保障，完全依赖于对评估者的隐性信任。 我们考虑在受信任的隔离环境中进行这些评估，以一种防篡改的方式报告基准分数，使用 in-toto认证框架。

最后，当涉及到开放模型时，我们需要考虑使用审查。 最常见的情况是当一个新的AI应用程序等待启动批准时，但我们也在考虑如何安全地将使用AI模型的库导入公司的内部源代码控制。 在任何这些情况下，我们需要一种方式来查看所有相关的供应链元数据（基于本节中提出的解决方案生成），并将它们合并成既能人类阅读又能机器阅读的信息。 这将加快审查速度，同时自动化和向左移动批准使用开放模型的过程。

对于这部分问题空间，我们建议使用理解构件组成的图（GUAC），因为这是首选系统来理解开源和公司内部系统与开源软件之间的大型供应链。 通过最小的更改，GUAC可以作为AI模型开发生命周期的窗口，作为在AI供应链中聚合和查询元数据的工具。

行业共识的开放问题

尽管已经有很多解决方案可用于AI供应链安全，但仍然存在一些未解决的问题需要解决。 虽然我们已经提出了一些观点，但我们认为作为行业应该共同努力，特别是就以下开放性问题达成共识会更有成效：

- 模型和数据来源应包含哪些信息？
- 为了实现互操作性，应该标准化模型和数据来源的格式是什么？
- 模型和数据来源应该如何存储、共享和验证？
- 模型和数据溯源应该捕获到什么级别的细节？ 例如，在溯源中记录数据集信息时，我们是只承诺数据集的名称还是每个数据示例？ 后者将允许回答关于在训练过程中是否使用了特定信息的问题，但它会更大并且需要更多的努力来生成。
- 数据集应该签名以确保完整性吗？ 如果是这样，是否有额外的数据特定字段需要添加到SLSA标准中？
- 我们如何保护模型训练免受消耗不受信任的来源或在签名之前篡改溯源的影响？[12]完全沙盒化或资源隔离似乎困难且昂贵，那么是否有其他探索的替代机制？
- 模型中心应该支持溯源验证作为内置功能吗？
- 行业应该采用ML-BOM[13]还是溯源？

12. 在更高的SLSA级别上，完整的沙箱或资源隔离非常重要，以确保生成过程中的来源不受篡改。 将相同的沙箱应用于训练过程似乎很困难且昂贵。 我们正在考虑是否有其他探索方法，例如通过利用数据集存储系统日志中的可观察性来减轻训练过程中沙箱的负担，作为充实来源文档的补充机制（“日志收集”）。

13. 有关将SBOM概念扩展到新的ML-BOM的行业讨论。然而，我们认为SLSA来源可以捕获有关供应链的相同信息，而ML-BOM中的其余信息通常包含在模型卡中。 这种观点可能会在未来发生变化，我们正在关注相关行业发展，并将根据需要进行调整。

04

实践者的指导

每个组织在供应链安全方面的方法都会有所不同，这取决于内部流程和平台。

即使捕获和处理供应链洞察所需的元数据的方法各不相同，但最终目标可能看起来相似：以洞察和透明度构建您的供应链。 我们建议您查看本文中关于实现此目标的技术考虑的指导原则部分。

如果一个组织能够为所有软件和AI工件做到以下几点，
那么它将处于一个良好的位置：

捕获元数据

捕获足够的元数据以了解每个工件（模型或其他）的渊源。您希望能够回答基本问题：工件来自哪里；谁创作、更改或训练了它；训练时使用了哪些数据集；以及用于生成工件的源代码是什么。

增加完整性

随着时间的推移，努力提高工件和相关元数据的完整性。 最终，元数据应在工件创建过程中以不可修改、防篡改的方式捕获。工件本身应使用下一代签名技术进行加密签名，以减少密钥管理的负担，以显示工件在事后是否被篡改。

整理元数据

整理信息以支持查询和控制。 在发生事件时，您将能够了解受影响组件的爆炸半径。

在启动时，您可以实施适当的治理。在开发过程中，控制将确定工件是否符合使用指南。

与他人分享

最后，作为最佳实践，分享您在SBOM、溯源文档、模型卡或其他某种工具中捕获的元数据，以帮助其他开发人员。 这种对模型创建过程的透明度增加了信任，并有助于跟踪模型在复杂供应链中的意外行为，以发现问题的根源。

结论

我们在本白皮书中概述的方法旨在指导寻求保护其AI软件供应链的行业和组织。 我们相信扩展现有的软件供应链解决方案是应对AI软件供应链风险的有效方式。 与其创建新解决方案，我们可以将AI模型视为传统软件。 通过认真应用已建立的软件供应链安全实践并仔细跟踪数据集，组织可以加强对恶意攻击的防御，并更快地从意外漏洞中恢复。

赌注很高。 AI系统在从医疗保健到金融和基础设施等各个领域的应用越来越广泛。 在过去的十年里，我们在传统软件领域看到，你的安全性取决于最薄弱的环节，而最薄弱的环节通常是供应链中被忽视的部分。 通过应用本文中提出的解决方案，我们相信我们可以共同加强连接AI软件供应链的环节。

集体行动至关重要。与传统软件一样，没有一个AI模型是独立的。 无论一个组织有多么自给自足，总会涉及依赖关系、数据集和其他共享组件。 通过增加我们对这些组件的捕获和共享的信息，我们可以保护共享AI生态系统的基本构建模块，帮助确保每个人的AI软件供应链。