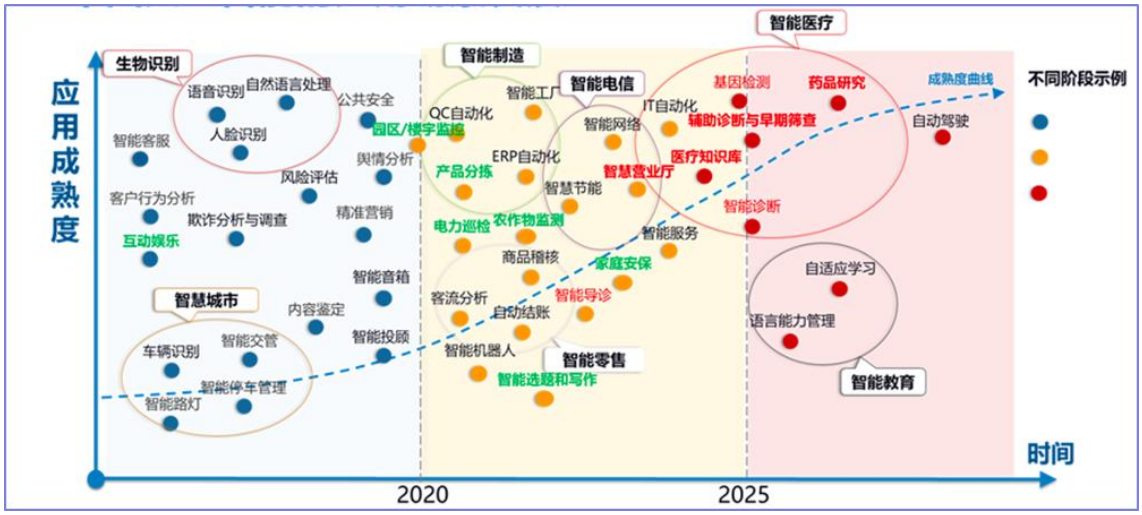
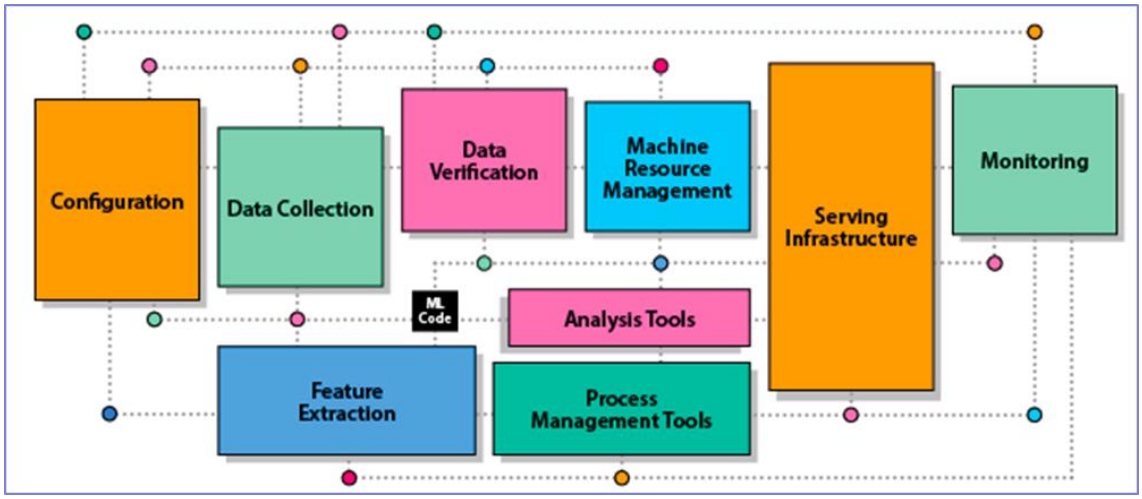
DevOps for AI - 人工智能应用开发面临的挑战

1. 概述

AI 是近十年来发展迅速、最有前途的技术，几乎应用于所有商业领域和研究领域。在许多领域（如医疗诊断），与传统应用程序相比，ML 在提供结果方面表现出优势，并在某些活动中表现出优于人类的智慧。这些趋势引发了对数据、AI 科学家以及软件开发人员的巨大需求。



早期研究表明，构建算法(即开发 ML 模型)只是成功开发并运行基于 AI 的软件系统所需全部工作的一小部分。



ML 软件实现与传统软件的区别在于，它们的逻辑不是显式编程的，而是通过从数据中

学习自动创建的。因此，基于 ML 的软件系统的开发过程涉及到不同的流程，包括数据收集、数据准备、定义 ML 模型(如深度网络模型)、通过模型训练的过程以量化模型参数并获得预期的结果。这个过程被称为 ML 工作流。ML 工作流需要一套复杂的工具支持，而获得一个高效的流程本身就是一个挑战。然而，ML 工作流并不涵盖整个软件开发过程。它并没有解决如何高效地进行软件开发的问题。

这就引申出一个问题，即 ML 工作流和软件开发过程应该如何关联起来，或者作为一个更普遍的问题：包含 AI 组件的软件系统的开发过程是什么？

本次课题将和大家一起探讨包含 ML 组件在内的软件开发过程带来的一些新的挑战，并讲述了可用于开发、运行和演进基于 ML 的软件系统的软件开发模型。

现代软件开发过程应用了敏捷原则，最常用的敏捷过程之一是 DevOps，具体来说，本文讲述如何将 ML 工作流与 DevOps 的集成。

2. ML 工作流

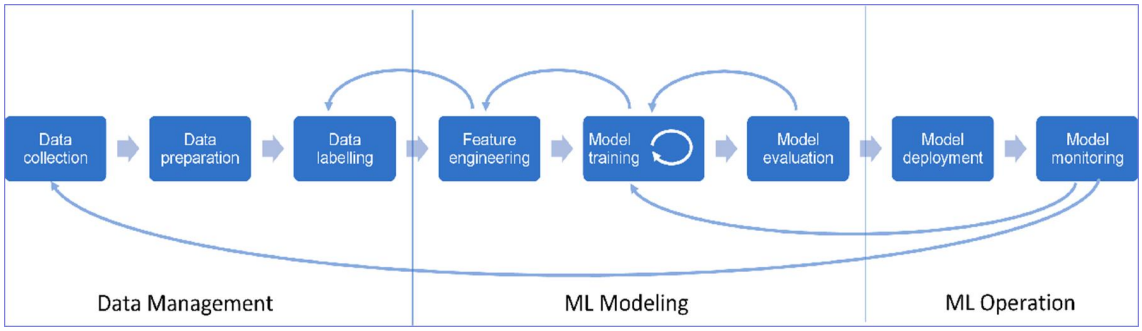
机器学习工作流描述了开发基于 ML 的软件系统时通常执行的开发阶段。ML 工作流包含的阶段：模型需求、数据收集、数据清洗、数据标签、特征工程、模型训练、模型评估、模型部署和模型监测。

对上述阶段进一步分组：数据管理、ML 建模和模型运营三组。

数据管理过程：包括数据采集及数据预处理、数据标注。数据通常存储在企业数据仓库或开放式存储，提供给不同的 ML 应用程序使用。

ML 建模过程：包括特征工程、模型训练、模型评估。此过程是高度迭代的，并由结果驱动（如模型精确率、准确率或 ML 模型训练期间使用的其它指标）。

模型运营过程：包括模型部署和模型监测。监测 ML 组件其有助于向 ML 建模提供反馈，可能会导致使用新数据集重新训练模型或者使用新的特征和新的模型架构重新设计模型。



3. 开发 ML 系统面临的一些挑战

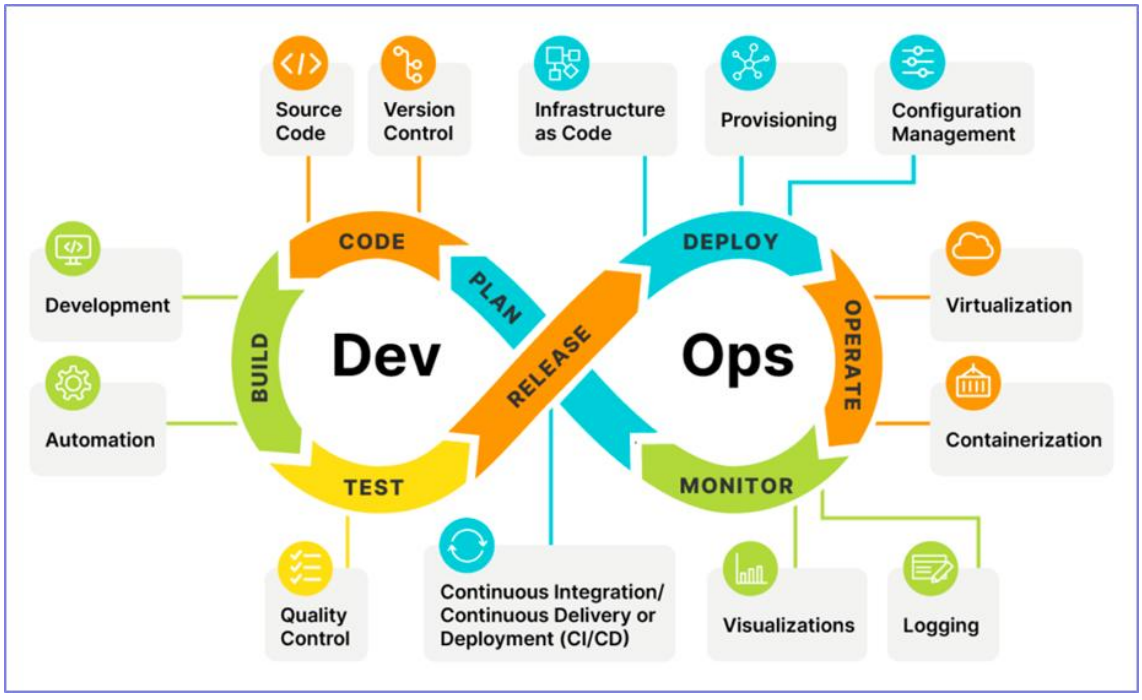
在实际环境中，由于各种原因，使得开发 ML 系统具有挑战性。例如：

- 用于构建模型的数据存在大量的数据质量问题，并且在训练模型之前需要花费大量的精力来准备数据，因为通常这些数据不能轻易地作为模型的输入。
- 此外，在 ML 建模阶段，手动和临时程序的使用也会使模型试验结果难以重现，降低了试验的效率。
- 尽管模型的开发存在挑战，但其仍然是构成整个 ML 系统的一小部分。一旦数据科学家选择了最终的模型，就需要将其部署并集成到终端用户使用的应用程序中。**对于复杂的系统，模型的集成和部署需要考虑与软件系统相关的大量需求。**基于 ML 的应用程序还需要持续监测，以便检测预测结果和数据中的错误（例如，**训练服务偏差**），并重新训练模型。

4. DevOps 流程

DevOps 是一种软件开发方法，强调软件开发和运营之间的协作，以便运营软件系统并加快软件变更的交付。

在持续部署（CD）的背景下，这有助于创建一个可重复、可靠的流程，以便在生产环境中频繁发布变更软件版本。



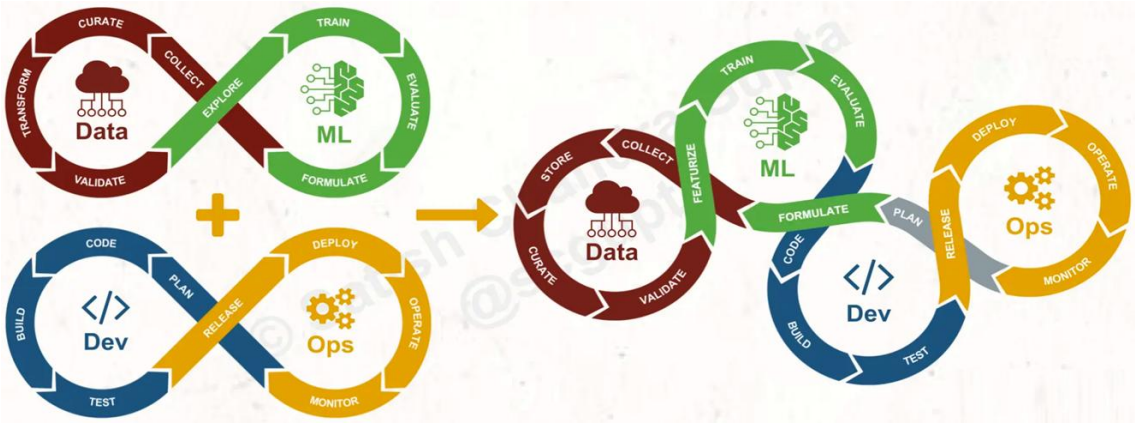
基于**敏捷软件开发**方法，DevOps 的一个重要原则是**构建、测试、部署和运营流程的**

自动化，同时确保所有的软件制品都有版本控制。

5. 集成 ML 工作流和软件系统生命周期的挑战

ML 工作流定义了构建 ML 模型的端到端过程，但它忽略了软件开发过程。ML 模型始终是应用程序或软件系统的一部分。

软件系统的开发有另一种逻辑和过程。开发和运营过程是相互反馈循环紧密相连，但并没有很好的定义如何与 ML 工作流连接。



这就导致执行 ML 工作流和 DevOps 时会面临很多挑战。 下面来介绍下推动 ML 工作流和 DevOps 集成的一些挑战。

多环境

假设 ML 模型训练有一个用于构建 ML 模型的数据存储。在 ML 工作流中，大多数与数据处理相关的活动都可以和其他活动解耦。ML 模型训练对算力要求很高，大型模型通常需要分布式方式增加高性能的算力资源来训练，以加快训练速度。因此，**开发环境需要交互式软件开发工具来调度本地或者分布计算单元的训练**，通常是 CPU 和 GPU 的组合。

另一方面，**由于计算和存储资源有限，可能会上云。**因此。**运行环境和传统的软件可能类似**(例如：基于 web 的应用程序)，**也可能完全不同**(例如：自动驾驶系统)。
与传统软件开发相比，ML 软件开发的主要挑战是如何处理数据和 ML 模型，而不仅仅是代码。**用于跟踪和管理代码的工具，如：Git，不足以确保有效跟踪和管理 ML 组件（代码、数据、模型）的版本依赖关系。**

类似地，**持续集成（CI）工具不仅是测试代码，还用于测试和验证用于训练模型的数据**。虽然一般使用 CI 工具来协调 ML 模型开发，但在分布式训练的背景下，他们不足以确保在专用硬件（如：GPU）上有效训练模型。

除了不同的工具，**ML 软件开发过程还需要拥有不同的专业知识的开发者——数据科学家、AI 专家和软件工程师。**

以上这些在环境、工具、流程和专业知识方面差异，对 ML 软件开发提出了新的挑战：

如何确保成功管理多个开发环境？

如何确保组件之间无缝的相互通信和交互？

如何确保多学科专业知识，并实现领域专家之间的高效协作？

互补需求

现代软件开发流程将**传统的需求管理**与**利益相关者的结果/数据反馈**结合起来，最近还**使用 AI 来增强系统运营的反馈**。

与用户验收相关的**软件系统**特性**通常通过 A/B 测试进行评估**。

在许多领域，与质量属性相关的**非功能性需求**是开发过程的主要关注点。例如，与可靠性（safety、security、可靠性、健壮性等）、资源（实时性、性能、能耗、计算和存储约束）、用户接口（可用性）相关的属性。

ML 工作流的要求则不同，其主要关注：数据质量、数据管理的有效性（数据准备、标注）、**训练过程的有效性**（CPU 训练时间）以及**训练过程中所需的资源**。

关于 ML 模型的质量，有一组指标可以表示预测的质量：准确率、精确率和召回率等。

在特征工程中，**特征选择及其处理**(归一化、转换等)**的质量**、**有效性**与**预测质量**有关。

在**集成过程**中，主要的挑战在于**理解软件系统需求和 ML 需求之间的关系**。例如，**系统特征与数据集特征之间的关系**。显然，这些并不相同，但它们可能是相关的。**当将特性指定为系统需求时，哪些特性应该包括在 ML 模型中**？

另一个例子：为了**实现系统的可靠性**，我们是否需要**确保 ML 模型的精度达到一定的水平**？性能要求也存在类似的问题：**ML 模型性能对系统性能的影响是什么**，例如：实时性要求？

上述例子可以总结为一个共同的挑战：

如何将 ML 需求与软件系统需求联系起来？

如何从系统需求中得出 ML 需求？

系统验证 VS ML 评估

ML 评估的目标是提高预测质量。首先找到指定代价函数的最小值，然后以准确性等 ML 指标评估结果。

这些指标的度量在开始时是未知的，需要通过多次迭代、调整 ML 模型和输入数据集来寻求最优值。**该过程没有预定的目标，也没有过程的预判**（比如时间和资源）。

但软件系统有规范去度量其正确性。**正确性**可以是**二进制数据类型**(正确/失败)，也可以是和**非功能属性(如可靠性)相关**的一种方式。

在验证过程中，由于系统不应该被更改/调整，因此，验证过程通常不是一个迭代过程。这种验证通常通过测试或者分析（静态/动态、正式/经验）。

这里又有一个问题，即**这些过程是如何关联的？我们能使用 ML 评估指标来分析系统的正确性吗**？如果系统中集成了不同的 AI 模型，那么这个问题就尤为重要。

或者反过来说，**我们能从系统验证中得出 ML 模型不够准确或精确的结论吗？软件系统验证与 ML 评估如何相关？**

系统演进 VS ML 模型演进

软件的持续变化促成了像 DevOps 等开发流程的出现，这些流程明确地支持**持续集成、持续部署、自动验证**和**生产中特定类型的测试**，如 A/B 测试。

软件评估包括**新功能和改进功能的变更以及内部基础结构的变更。这些更改会影响 ML 模块的更改请求，包含功能和非功能属性**。

ML 工作流侧重于另一种类型的**演进**：

由于**数据集变化引起的重新训练需求**而演进，来自新环境的新数据可能需要通过再训练来重新建模。

模型本身的变化/优化产生的演进，但使用相同的数据集训练。

虽然**连续的软件演进会以一种可控的方式映射到 ML 模型的演进**(尽管这需要建立程序)，但**模型的数据输入的变化，可能会以不受控和意外的方式发生，这可能会直接影响 ML 模型的准确性和性能**。

上下文的变更可能需要**使用新的数据集重新训练 ML 模型**。然而，重新训练的模型可

能会在新的上下文中正常工作，但它的性能在旧的环境中依然会下降。

因此，衍生出新的挑战：

软件的演进如何影响 ML 模型的演进？

如何控制环境变化对 ML 模型性能和模型演进的需求？

如何控制 ML 模型的演进以实现对软件系统演进同样的控制水准？

运营和反馈循环

ML 工作流和 DevOps 本身被设计为独立的迭代过程，但它们也相互依赖。

数据管理包括**数据收集**，收集的数据**用于在开发的初始阶段为训练提供数据**。然而，**在许多情况下，新数据是在系统运营期间收集的，尤其是新数据覆盖现有环境数据时，ML 模型必须重新训练**。ML 模型必须部署在开发环境中，才能在开发过程中使用，并需要与可执行系统一起部署。

在运营中，系统必须提供**有关系统性能和用户对开发环境的接受程度信息**，以便为系统的进一步演进提供反馈。系统还需要**监控已部署的 ML 模型，以便检查其在当前环境中的性能**。

要获得成功和高效的端到端流程，必须不断交换不同类型的信息。这里面临的挑战有：

流程之间应交换哪些信息？

信息交换的触发机制是什么，信息交换的频率是什么？

6. 集成 ML 工作流和 DevOps 流程

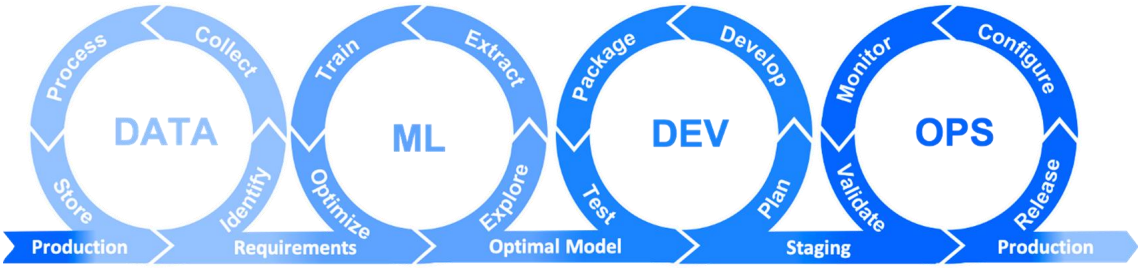
我们最终的目的是通过将 ML 工作流和 DevOps 流程的集成来**实现在生产中快速迭代和持续地开发、部署和运营基于 AI 的软件系统**。

集成 ML 工作流和 DevOps 流程的期望有：

- 在基于 AI 的系统开发中实现所有流程步骤内和跨流程步骤间的自动化，包括构建、测试、部署和基础设施管理。

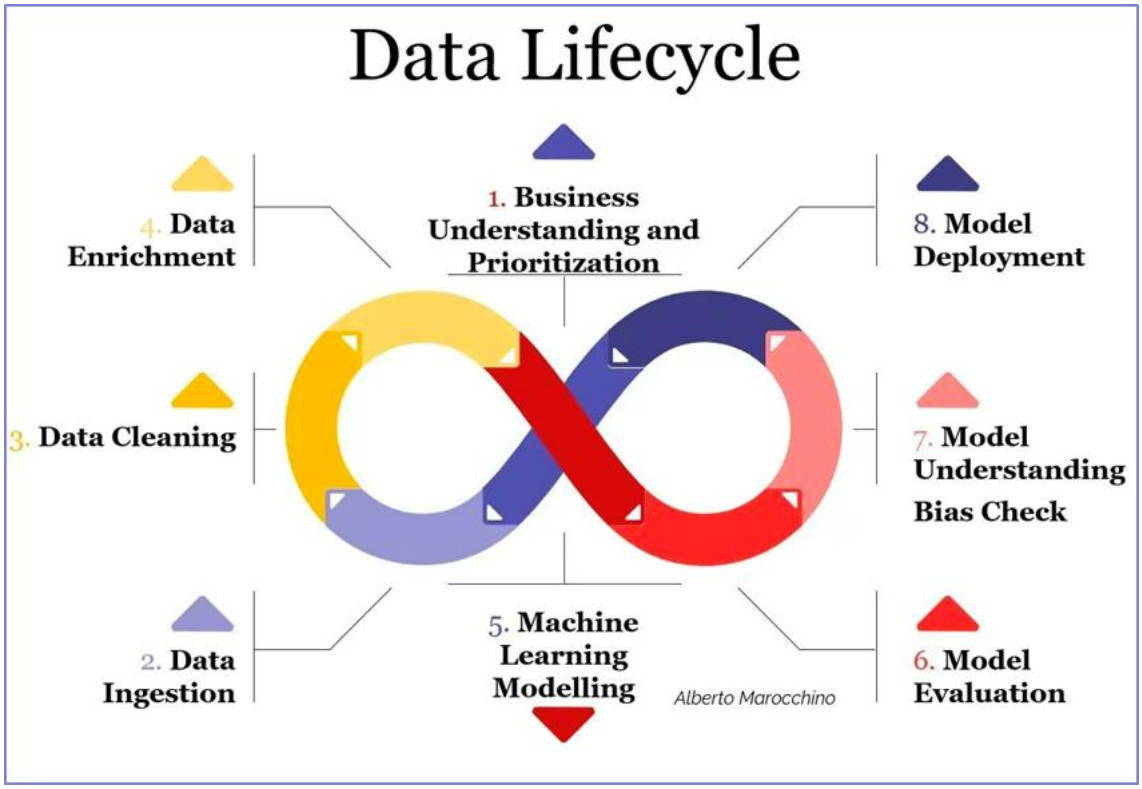
- 所有对构建基于 AI 的系统重要的制品（代码、数据和模型）都是可复现的，能够适应并以小增量可靠地发布。
- 集成过程要特别适用于基于新的流数据或 ML 模型参数优化（如更改超参数）重新训练部署 ML 模型的场景。

整个基于 AI 的系统的生命周期将 ML 工作流和 DevOps 集成到四个不同的流程中，即数据管理(DM)、ML 建模（Mod）、软件开发（Dev）和系统运营（Ops）。



6.1 数据管理(DM)

数据管理阶段通过建立流程来**收集数据、选择数据、标注和策划用于训练 ML 模型的特征**。



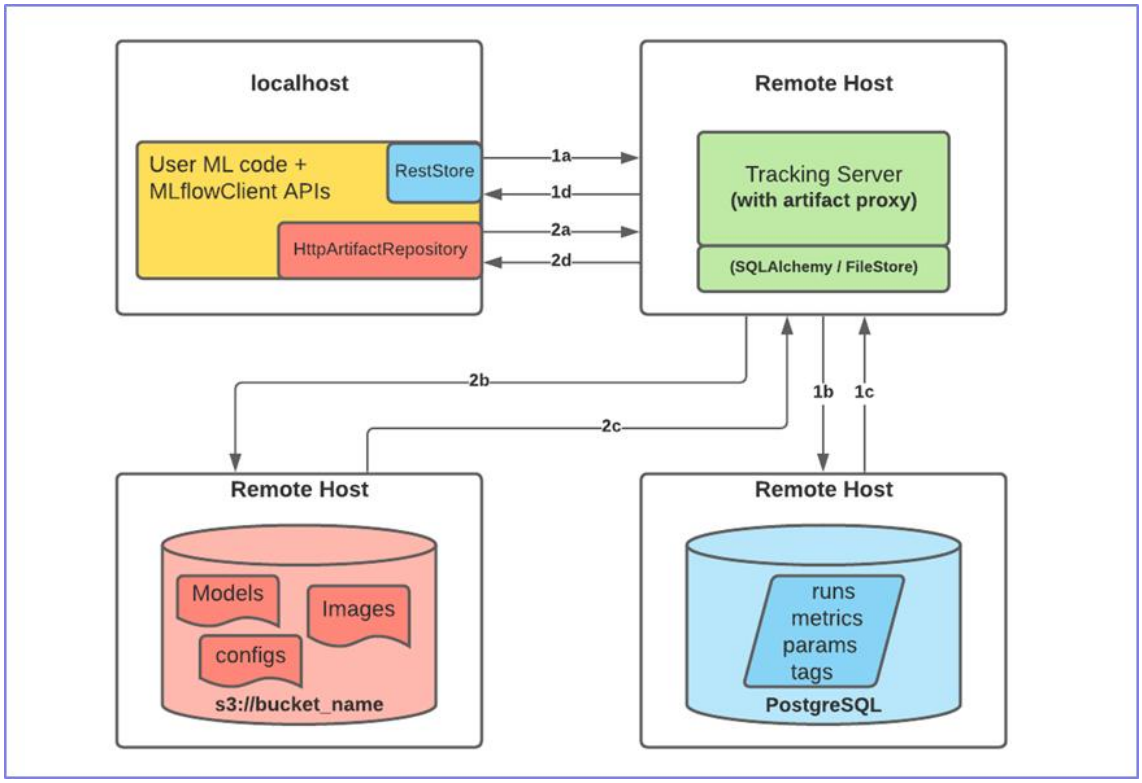
标准化流程有助于**以最优化的格式提取和存储数据**，便于在 ML 建模阶段使用，同时加强数据的安全访问控制。

6.2 ML 建模（Mod）

一旦在 **ML 建模阶段提供了新的数据集进行训练**，就需要**提供训练环境以**方便模型训练和评估。

ML 模型实验可以在 ML 平台上进行，该平台由内部团队开发或使用开源工具，提供**分析数据集和训练 ML 模型的能力**。

在每个 ML 模型实验运行的阶段，**数据集、训练模型**(包括模型配置)和**评估结果都被存储起来，以便与历史基线（例如当前正在生产的模型）进行比较**。



为了**方便**在 ML 模型实验顺利完成后跟踪制品依赖关系，开发人员需要显式地将制品及其依赖关系(数据集、模型和代码)指定并注册到依赖关系跟踪系统中，该系统将实验运行历史数据存储在数据库中。

对于每个制品，开发人员指定其元数据（名称、版本号、注册日期）和依赖关系信息，这些信息包含对制品依赖的其他元素的引用。例如，数据集依赖关系信息可以是

- 数据源，例如：日志 ID，
- 用于从原始数据中提取数据集的代码的 Git 哈希，
- 版本化的数据标签集
- 提取脚本的入口点等。

对于存储制品及其依赖关系信息来说，重要的是它应该具有不变性，即能够每次重新创建/再现完全相同的制品。

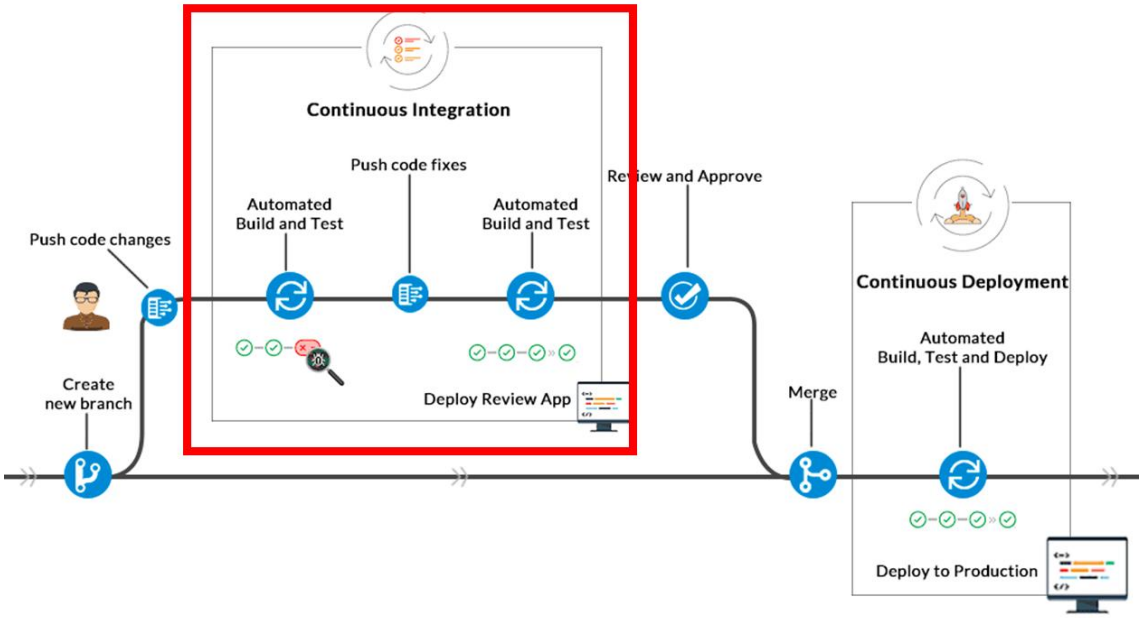
此外，**版本化的制品和它们的依赖信息可以被其他系统从外部访问**，例如，通过简单的 API 调用构建系统。

在模型实验阶段结束时，**选择最终的训练模型，并验证该模型在与其他 ML 组件隔离的情况下能够工作正常**。

6.3 软件开发（Dev）

当部署经过训练的模型时，首先**要验证它与系统的其他部分(即其他组件)是否正确运行**。

我们需要**构建系统**在大型测试集上执行系统范围的集成和验证，以给出系统所有部分在不同组件之间如何执行的整体视图。



由于构建系统可以访问制品依赖信息和功能，例如：使用代码对制品进行版本控制和存储。因此，**构建系统**可以以这样的方式实现，即它**具有定期编排和运行整个模型训练作业的功能。** 在这种情况下，重新训练模型将是自动化的，构建过程状态将被指定为失败、中止或成功。

如果使用**最佳模型成功构建**，生成的新模型性能优于生产模型，**将在模型注册表中注册。如果构建失败或中止，开发人员可以选择使用优化的参数集重新构建模型**。

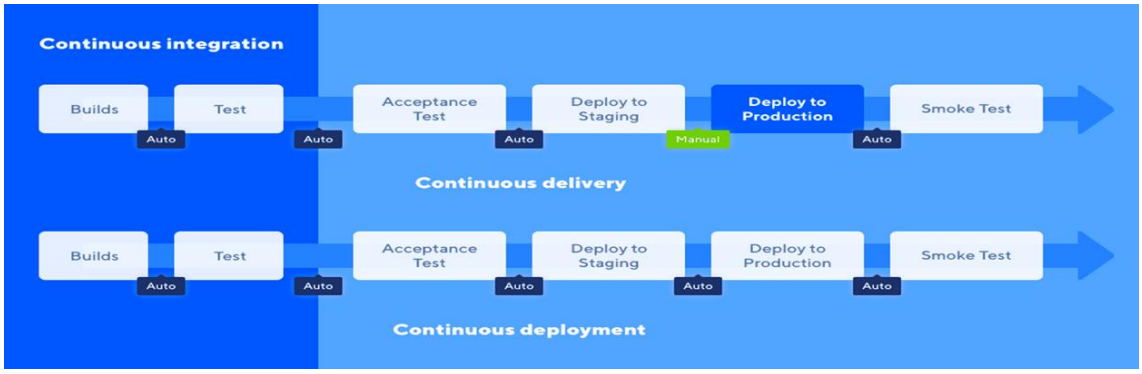
此外，CI 系统执行的历史构建还可以用来**识别和分析由于模型更新而在其他组件中出现的 bug**。

最后，只有在成功构建后，包括通过**定期集成和系统测试**，才会将经过验证的模型的最新注册版本发布/推广到后续的验收环境或生产环境从而保证质量。定期测试包括测试不同输入的 AI 组件，以确保适当的响应。

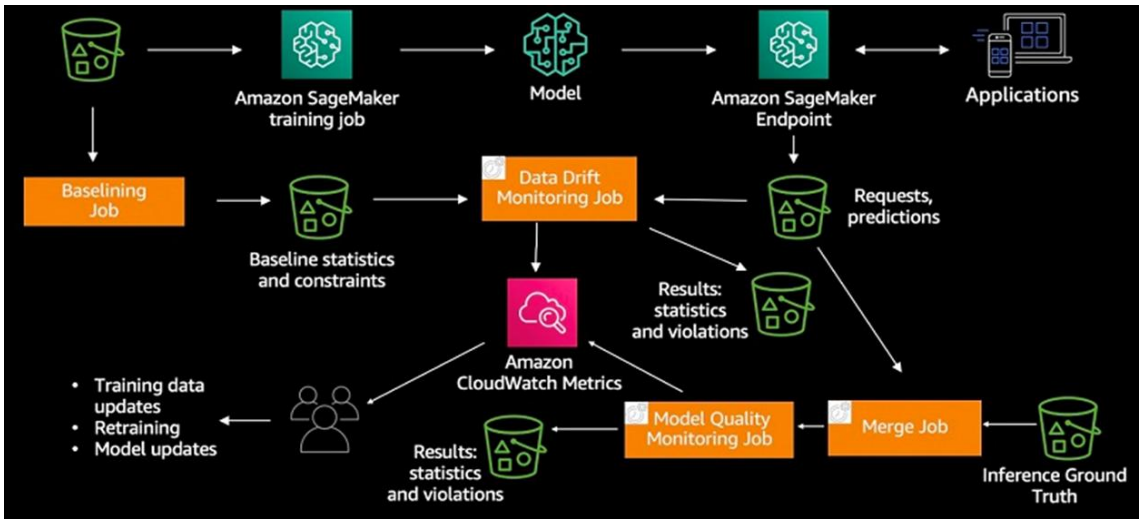
6.4 系统运营（Ops）

一旦 ML 系统组件在生产环境中运行(包括训练过的模型)，系统将被持续监视，以检测性能下降等问题。

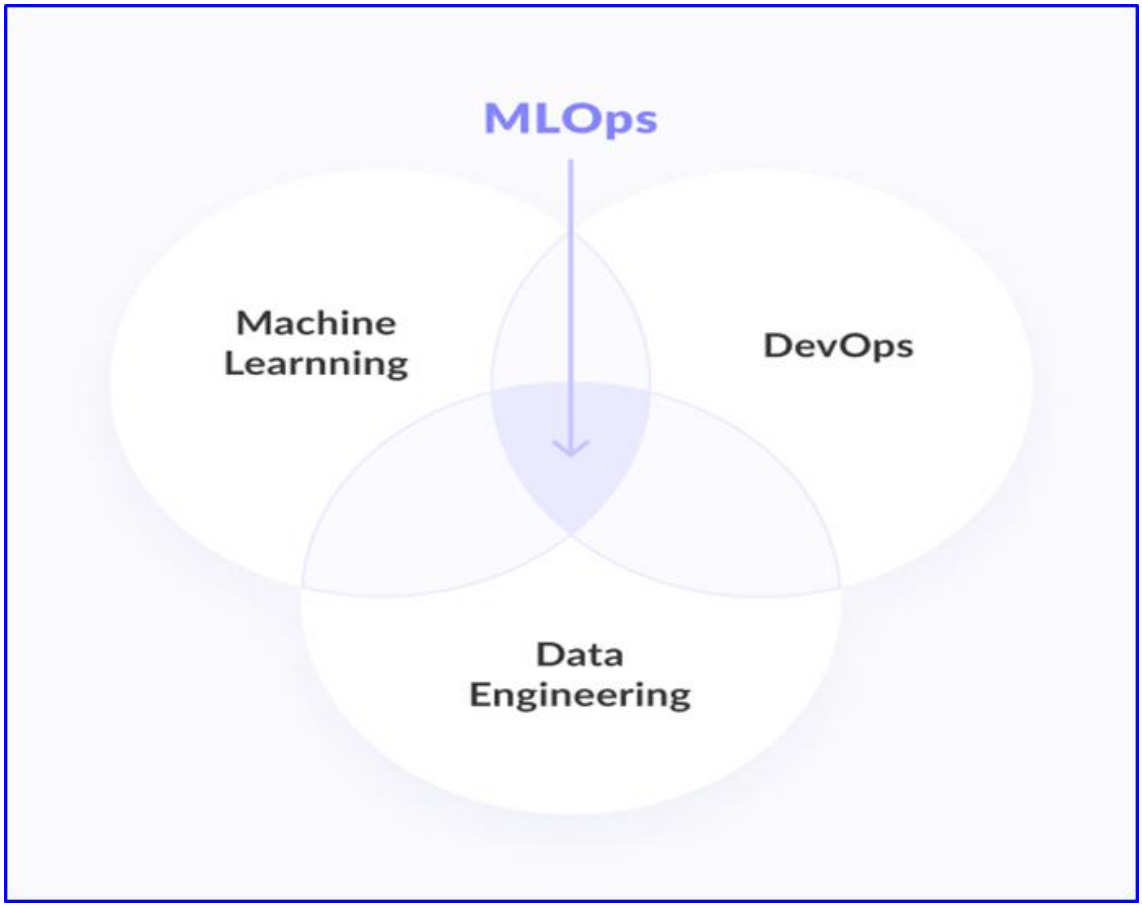
部署到生产环境可以是手动、半自动化或自动化的过程，部署一般在开发流程中的预生产环境中成功执行测试之后进行。



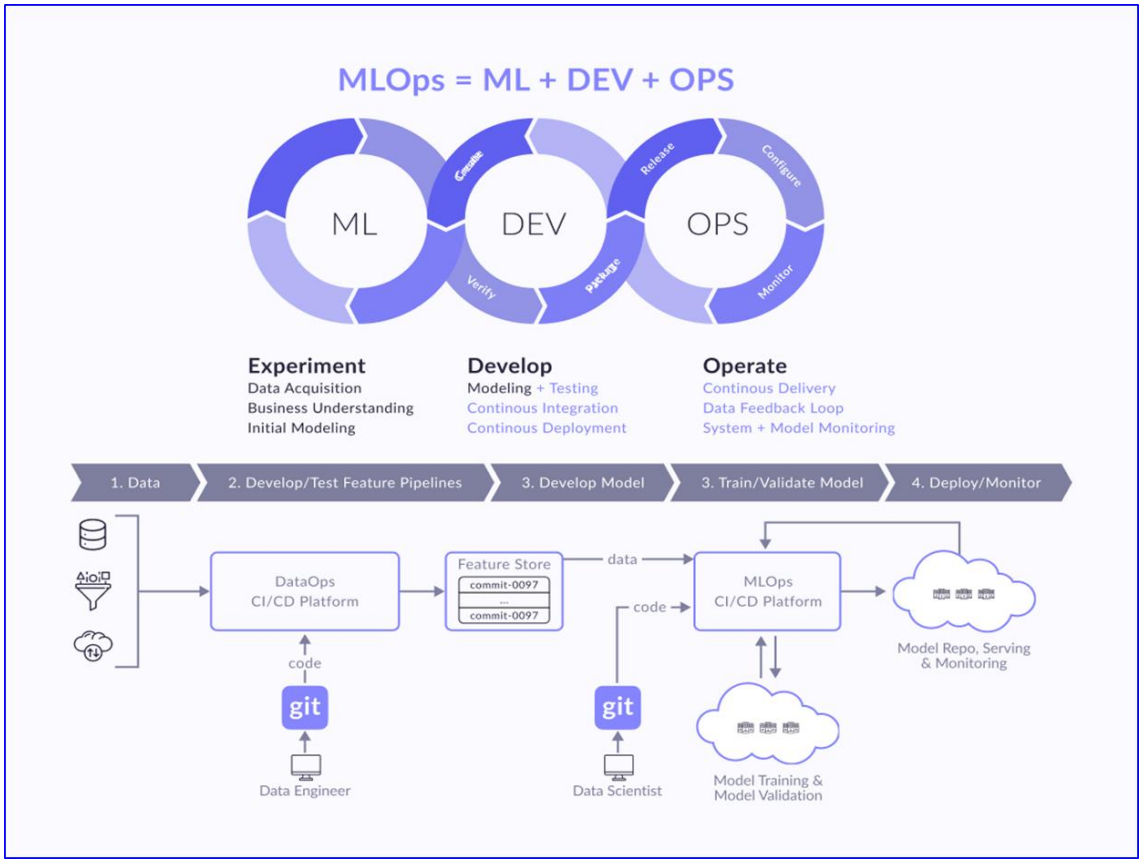
Ops 阶段还**通过提供预测来确保 AI 组件的实时测试，同时遵守严格的延迟要求**。在此步骤中，**开发人员还根据实时数据收集模型的执行情况信息。这些信息可以用来触发模型的再训练**。



7. MLOps



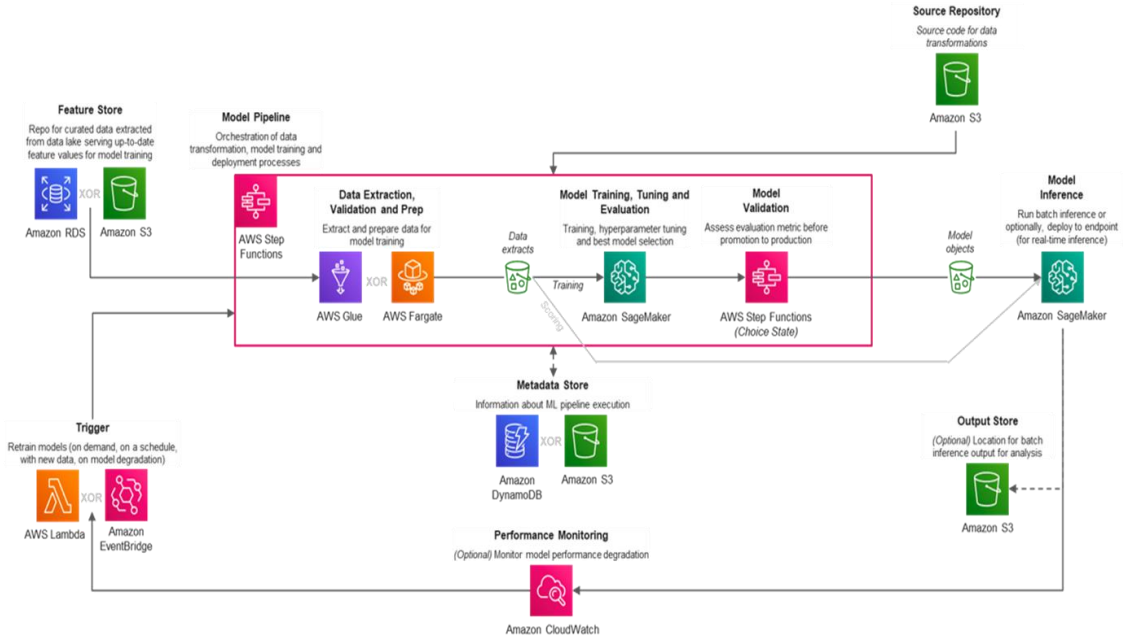
MLOps 是机器学习和运营的复合词，它是数据科学家和运维专业人员之间协作和沟通的一种实践，以帮助管理生产机器学习（或深度学习）生命周期。与 DevOps 或 DataOps 方法类似，MLOps 希望提高自动化程度并提高生产 ML 的质量，同时还关注业务和监管要求。 虽然 MLOps 也是作为一组最佳实践开始的，但它正在慢慢演变为一种独立的 ML 生命周期管理方法。 MLOps 适用于整个 ML 生命周期--从模型生成（软件开发生命周期、持续集成/持续交付）、编排和部署的集成，到健康、诊断、治理和业务指标。简单来说，**MLOps=ML+DEV+OPS**



下面，我们来看下业界典型的 MLOps 平台。

8. MLOps 平台案例一：AWS

第一类 MLOps 平台主要是有一些云厂商提供，如谷歌云平台、微软云平台、亚马逊云平台上提供的 MLOps 解决方案。



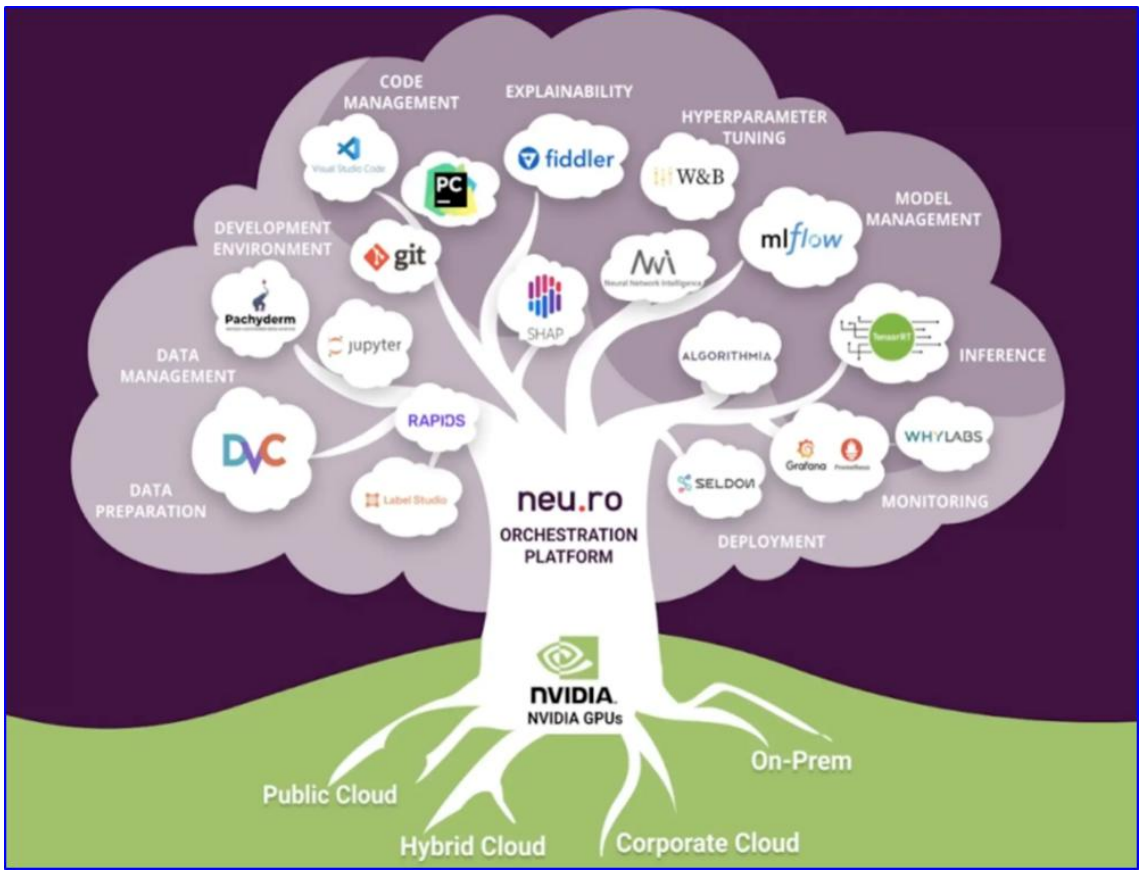
图中是 AWS 中自动化 MLOps 流水线的示例，我们可以看到 AWS 的 MLOps 平台关键组件有如下几个：

- AWS Step Functions：用于编排流水线中的各种作业并结合模型验证逻辑。
- Amazon S3 ：用于初始数据存储/数据湖，数据提取存储文件、源代码、模型对象、推理输出和元数据存储（初始特征存储也可能是关系数据库，例如：Amazon RDS 或者 Amazon DynamoDB 可以是用于存储某些用例的元数据）。
- Amazon SageMaker ：用于模型训练、超参数调整和模型推理（批处理或实时）。
- AWS Glue 或 ECS/Fargate ：用于为训练作业提取、验证和准备数据。
- AWS Lambda ：用于执行函数并充当重新训练模型的触发器。
- Amazon CloudWatch：用于监控 SageMaker 的调整和训练作业。

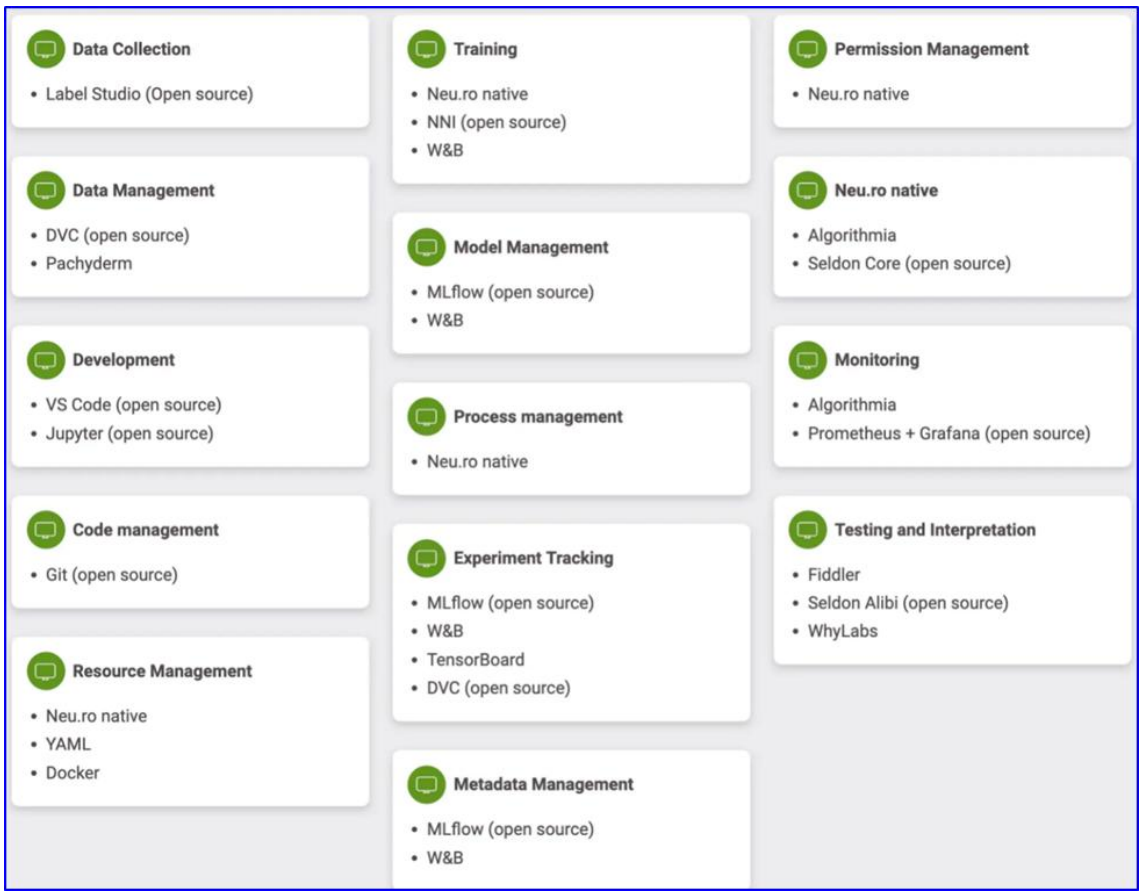
9. MLOps 平台案例二：Neu.ro

第二类 MLOps 平台主要是通过集成开源工具和自研的方式，将您需要的所有工具集成在一起，提供一个成熟的机器学习开发环境。如：Neu.ro。

Neu.ro MLOps 平台，用于在公有云、混合云和私有云上进行全周期 ML/DL 应用程序开发和部署。



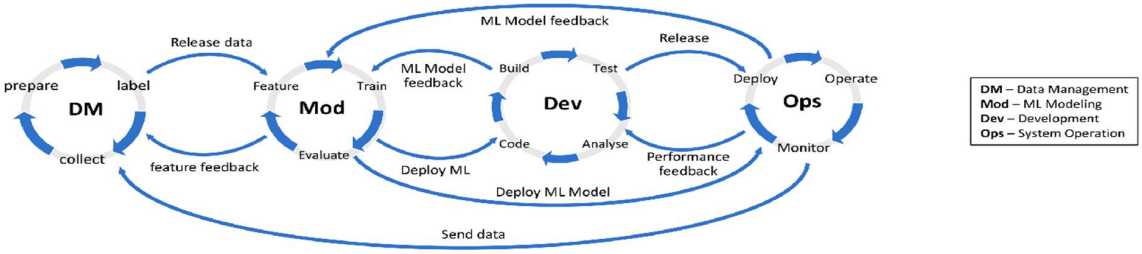
Neuro 平台使用户能够自动执行从数据收集到准备、训练、测试和部署的 ML 管道。使用 Neuro，打包、扩展、微调、检测和持续交付的步骤都可以完全自动化，解决每个组织的两个主要挑战：上线生产的时间和资源管理。



10. 结论

本次课题分享了 ML 工作流和 DevOps 流程的集成的方法以及市面上两类典型的 MLOps 平台，以帮助大家了解如何系统地构建基于 AI 的软件系统。 在开发基于 ML 的软件系统时，该方法还有助于解决一些已确定的挑战。特别是 ML 模型实验和部署期间的手动步骤，包括解决 ML 制品的版本控制和依赖管理问题。

一个完善的 MLOps 平台主要包括如下几个模块：



数据管理：收集数据、数据预处理、数据标注、数据（版本）管理、数据分析

ML 建模：算法管理、特征工程、模型训练、模型评估、模型（实验）管理

软件开发：模型构建、集成和系统测试、识别和分析模型更新导致其他组件出现的Bug

系统运营：模型（版本）管理、模型部署、模型推理、模型监控