负责的AI和机密推理—NetApp AI与Protopia Image Transformation

NetApp Solutions

NetApp
April 12, 2024

This PDF was generated from https://docs.netapp.com/zh-cn/netapp-solutions/ai/ai-protopia-overview.html on April 12, 2024. Always check docs.netapp.com for the latest.

目录

负责的AI和机密推理—NetApp AI与Protopia Image Transformation . . . 1
TR-4928：《责任AI和机密推理—NetApp AI与Protopia映像和数据转型》 . . . 1
解决方案区域 . . . 3
技术概述 . . . 4
测试和验证计划 . . . 7
测试配置 . . . 7
测试操作步骤 . . . 8
推理准确性比较 . . . 22
模糊速度 . . . 23
结论 . . . 23
从何处查找追加信息和相关信息 . . . 24

负责的AI和机密推理—NetApp AI与Protopia Image Transformation

TR-4928：《责任AI和机密推理—NetApp AI与Protopia映像和数据转型》

Sathish Thyagarajan、Michael Oglesby、NetApp ByG Hoon Ahn、Jennifer Cwagenberg、Protopia

随着图像捕获和图像处理的出现、视觉解释已成为通信不可或缺的一部分。数字图像处理中的人工智能(AI)带来了新的业务机会、例如癌症和其他疾病的医学领域、用于研究环境危害的地理空间视觉分析、模式识别、用于打击犯罪的视频处理等。但是、这一机会也伴随着非凡的责任。

企业将越来越多的决策交由AI处理、他们就越能接受与数据隐私和安全以及法律、道德和法规问题相关的风险。通过负责任的AI、企业和政府组织可以建立信任和监管、这对于大型企业的大规模AI至关重要。本文档介绍了在三种不同情形下由NetApp验证的AI推理解决方案 、它使用NetApp数据管理技术与Protopia数据混淆软件将敏感数据私营化并降低风险和道德问题。

消费者和业务实体每天都使用各种数字设备生成数百万张图像。因此、数据和计算工作负载的大规模激增使企业转向云计算平台以实现规模和效率。同时、在传输到公有 云时、会出现有关图像数据中所含敏感信息的隐私问题。缺乏安全和隐私保证已成为部署图像处理AI系统的主要障碍。

此外、还有 "擦除权" 根据《一般数据保护条例》、个人有权要求组织擦除其所有个人数据。此外、还提供了 "隐私法案"、建立公平信息实践准则。根据《一般数据保护条例》、照片等数字图像可以构成个人数据、该条例规定了数据的收集、处理和擦除方式。如果不遵守《一般数据保护条例》、可能会因违反合规性而被处以巨额罚款、从而对组织造成严重损害。隐私原则是实施负责任AI的基石、它可以确保机器学习(ML)和深度学习(DL)模型预测的公平性、并降低与违反隐私或法规相关的风险。

本文档介绍了在三种不同场景下经过验证的设计解决方案 、其中包括与保护隐私和部署负责的AI解决方案 相关的图像混淆和不混淆：

- *场景1.*在Jupyter笔记本电脑中按需推理。
- 场景**2.** Kubernetes上的批处理推理。
- 场景**3.** NVIDIA Triton推理服务器。

对于此解决方案 、我们将使用面部检测数据集和基准测试(FDDB)、这是一个人脸区域数据集、用于研究无约束面部检测问题、并结合使用PyTorch机器学习框架来实施FaceBoxes。此数据集包含一组具有各种分辨率的2845图像中5171个面的标注。此外、本技术报告还介绍了解决方案 的一些领域以及在适用此解决方案 的情况下从NetApp客户和现场工程师收集的相关用例。

目标受众

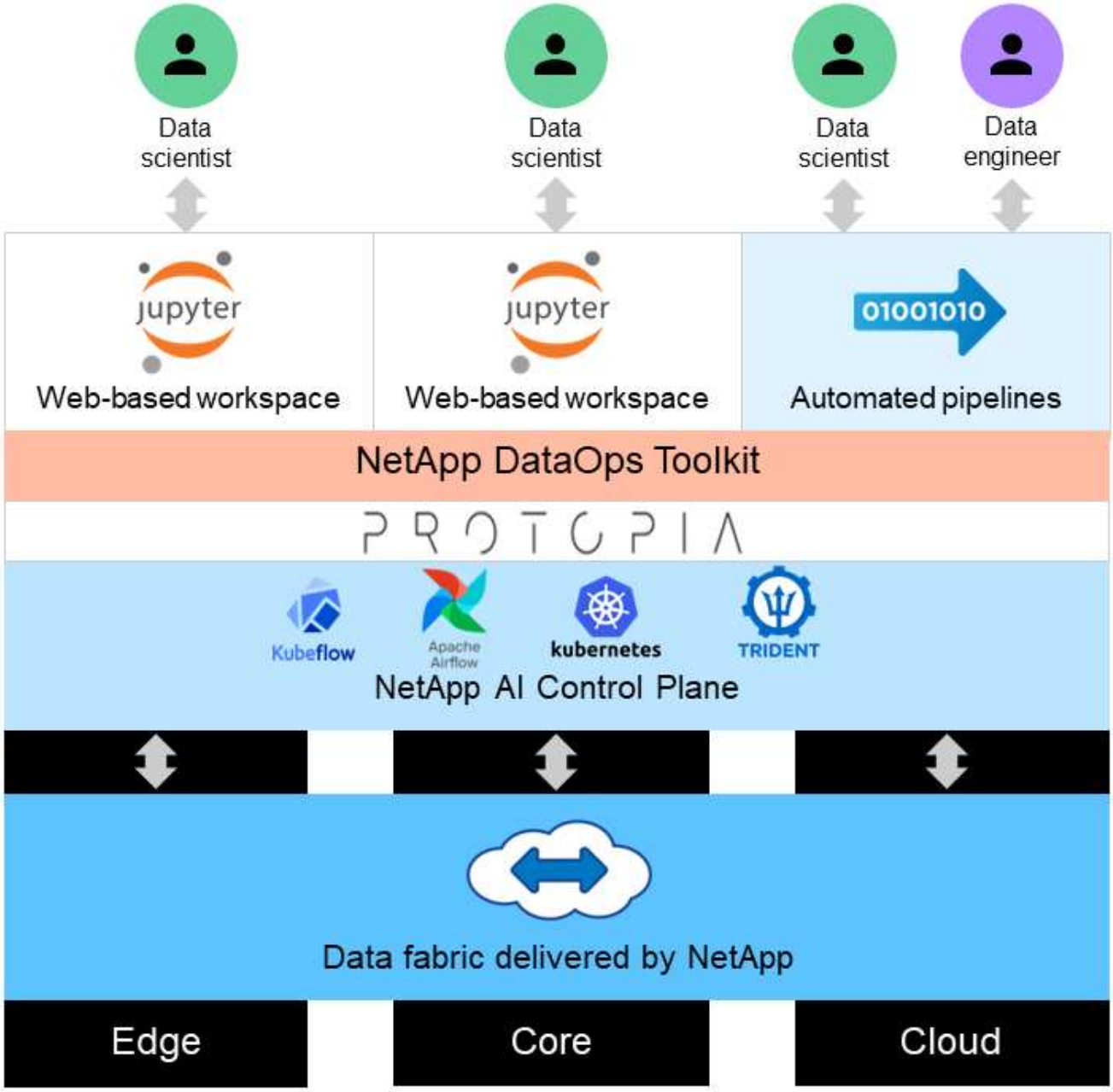
本技术报告面向以下受众：

- 业务主管和企业架构师、他们希望设计和部署负责的AI、并解决与公有 空间中面部图像处理相关的数据保护和隐私问题。
- 数据科学家、数据工程师、AI/机器学习(ML)研究人员以及旨在保护和维护隐私的AI/ML系统开发人员。

- 为符合GDPR、CCPA或国防部(DoD)隐私法案等法规标准的AI/ML模型和应用程序设计数据混淆解决方案的企业架构师。
- 数据科学家和AI工程师正在寻找高效的方法来部署深度学习(DL)和AI/ML/DL推理模型、以保护敏感信息。
- 边缘设备管理器和边缘服务器管理员，负责部署和管理边缘推理模型。

解决方案架构

此解决方案 可通过利用GPU的处理能力以及传统CPU来处理大型数据集中的实时和批量推理AI工作负载。此验证证明了ML的隐私保护推理以及寻求负责AI部署的组织所需的最佳数据管理。此解决方案 提供的架构适合单节点或多节点Kubernetes平台、用于与核心内部部署的NetApp ONTAP AI、NetApp DataOps工具包以及使用Jupyter实验室和CLI界面的Protopia混淆软件互连的边缘和云计算。下图显示了由NetApp提供支持的Data Fabric与DataOps Toolkit和Protopia的逻辑架构概述。



Protopia混淆软件可在NetApp DataOps工具包上无缝运行、并在离开存储服务器之前转换数据。

解决方案区域

数字图像处理具有许多优势、可以使许多组织充分利用与可视化表示相关的数据。NetApp和Protopia解决方案 提供了一种独特的AI推理设计、用于在整个ML/DL生命周期内保护AI/ML数据并将其私营化。它可以帮助客户保留敏感数据的所有权、通过减轻与隐私相关的顾虑来使用公有 或混合云部署模式以实现规模和效率、并在边缘部署AI推理。

环境智能

行业可以通过多种方式在环境危害领域利用地理空间分析。政府和公有 Works部门可以获得有关公有 运行状况和天气状况的切实可行的见解、以便在发生大流行病或野火等自然灾害时更好地为公有 提供建议。例如、您可以在机场或医院等公有 空间识别出COVID-阳性患者、而不会影响受影响患者的隐私、并提醒相关主管部门和附近的公有 采取必要的安全措施。

边缘设备可穿戴设备

在军队和战场上、您可以使用边缘的人工智能推理作为可穿戴设备来跟踪军人的健康状况、监控司机行为、并就接近军车的安全和相关风险向当局发出警报、同时保护和保护军人的隐私。随着战场物联网(Internet of Battlefield Things、IoBT)和军事物联网(Internet of militar Things、IoMT)的发展、军队的未来将采用高科技设备、通过快速边缘计算帮助军人识别敌人并提高作战性能。保护和保留从无人机和可穿戴设备等边缘设备收集的视觉数据对于防止黑客和敌人入侵至关重要。

非作战人员疏散操作

非作战人员后送行动(Noncombatant evelevation operations、NEO)由国防部执行、旨在帮助将生命面临危险的美国公民和公民、国防部文职人员以及指定人员(东道国(HN)和第三国公民(TCN))疏散到适当的安全天堂。所实施的管理控制主要采用手动疏散人员筛查流程。但是、使用高度自动化的AI/ML工具以及AI/ML视频模糊技术、可能会提高疏散人员识别、疏散人员跟踪和威胁筛选的准确性、安全性和速度。

医疗保健和生物医学研究

图像处理用于通过计算机体层成像(CT)或磁场共振成像(MRI)获得的3D图像诊断用于手术计划的病理学。HIPAA隐私规则规定了组织必须如何收集、处理和擦除所有个人信息和照片等数字图像的数据。要使数据符合HIPAA《安全港》规定的可共享性、必须删除全面照片图像和任何类似图像。用于从结构化的CT/MR图像中模糊个人面部特征的除名或头骨‐剥离算法等自动化技术已成为生物医学研究机构数据共享过程的重要组成部分。

人工智能/机器学习分析的云迁移

传统上、企业客户已经在内部培训和部署了AI/ML模式。出于规模经济和效率方面的考虑、这些客户正在进行扩展、将AI/ML功能迁移到公有 、混合云或多云云部署中。但是、它们受数据可以公开到其他基础架构的约束。NetApp解决方案可解决所需的各种网络安全威胁 "数据保护" 和安全性评估、结合使用Protopia数据转换、最大程度地降低将图像处理AI/ML工作负载迁移到云的相关风险。

有关其他行业的边缘计算和AI推理的其他用例、请参见 "TR-4886人工智能在边缘进行推理" 以及NetApp AI博客、 "智能与隐私"。

技术概述

本节概述了完成此解决方案 所需的各种技术组件。

Protopia

Protopia AI提供了一种不引人注目的纯软件解决方案 、用于在当今市场中进行保密推理。通过最大限度地减少敏感信息的暴露、Protopia解决方案 可为推理服务提供无与伦比的保护。AI只会馈送数据记录中对执行手头任务真正至关重要的信息、而不会再提供更多信息。大多数推理任务不会使用每个数据记录中的所有信息。无论您的AI是使用图像、语音、视频还是结构化表格数据、Protopia都能提供推理服务所需的功能。获得专利的核心技术利用数学精心设计的噪声来随机转换数据、并收集给定ML服务不需要的信息。此解决方案 不会屏蔽数据、而是会使用随机筛选的噪声来更改数据表示形式。

Protopia解决方案 将更改表示形式的问题表述为基于梯度的最大干扰方法、该方法仍会在输入功能空间中保留与模型功能相关的信息。此发现过程将在训练ML模型结束时作为微调过程运行。在传递自动生成一组概率分布之后、低开销数据转换会将这些分布中的噪声样本应用于数据、并在将其传递到模型进行推理之前使其发生混淆。

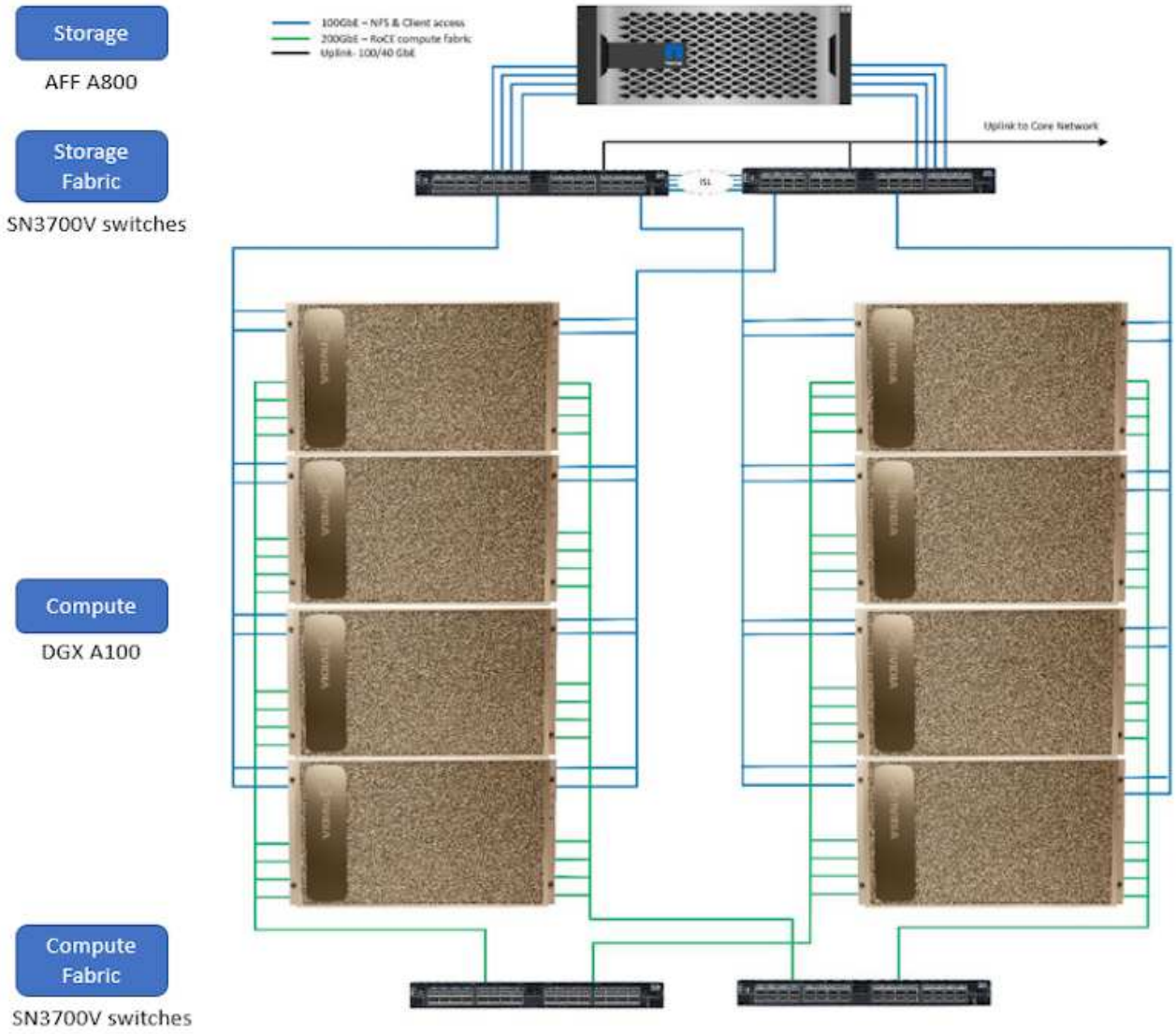
NetApp ONTAP AI

NetApp ONTAP AI参考架构由DGX A100系统和NetApp云互联存储系统提供支持、由NetApp和NVIDIA开发并验证。它为 IT 组织提供了一个架构，可提供以下优势：

- 消除设计复杂性
- 支持独立扩展计算和存储
- 支持客户从小规模入手，无缝扩展
- 为各种性能和成本点提供了一系列存储选项

ONTAP AI将DGX A100系统和NetApp AFF A800存储系统与一流的网络紧密集成在一起。ONTAP AI可消除设计复杂性和猜测性工作、从而简化AI部署。客户可以从小规模入手、实现无中断增长、同时智能管理从边缘到核心再到云再到云的数据。

下图显示了采用DGX A100系统的ONTAP AI解决方案系列中的多种变体。最多可使用八个DGX A100系统验证AFF A800系统性能。通过向ONTAP 集群添加存储控制器对、该架构可以扩展到多个机架、以支持多个DGX A100系统和数PB的存储容量、并实现线性性能。这种方法可以灵活地根据所使用的DL型号的大小以及所需的性能指标独立更改计算与存储的比率。



有关追加信息 关于ONTAP AI的信息、请参见 "NVA-1153：采用NVIDIA DGX A100系统和Mellanox频谱以太网交换机的NetApp ONTAP AI。"

NetApp ONTAP

ONTAP 9.11是NetApp推出的最新一代存储管理软件、可帮助企业打造现代化的基础架构并过渡到云就绪数据中心。借助行业领先的数据管理功能，无论数据位于何处， ONTAP 都可以通过一组工具来管理和保护数据。您还可以将数据自由移动到需要的任何位置：边缘，核心或云。ONTAP 9.11包含许多功能、可简化数据管理、加快和保护关键数据、并在混合云架构中实现下一代基础架构功能。

NetApp DataOps 工具包

NetApp DataOps工具包是一个Python库、可使开发人员、数据科学家、开发运营工程师和数据工程师轻松执行各种数据管理任务、例如近乎即时地配置新的数据卷或JupyterLab工作空间、近乎即时地克隆数据卷或JupyterLab工作空间、 并近乎即时地为数据卷或JupyterLab工作空间创建快照、以实现可追溯性或基线化。此Python库可以用作命令行实用程序或函数库、您可以将其导入到任何Python程序或Jupyter笔记本电脑中。

NVIDIA Triton 推理服务器

NVIDIA Triton推理服务器是一款开源推理服务软件、可帮助实现模型部署和执行标准化、从而在生产环境中提供快速且可扩展的AI。Triton推理服务器支持团队从任何基于GPU或CPU的基础架构上的任何框架部署、运行和扩展经过培训的AI模型、从而简化了AI推理过程。Triton推理服务器支持所有主要框架、例如TensorFlow 、NVIDIA TensorRT、PyTorch、MXNet、 OpenVINO等。Triton与Kubernetes集成、用于协调和扩展、您可以在所有主要公有 云AI和Kubernetes平台中使用。它还与许多MLOps软件解决方案集成在一起。

PyTorch

"PyTorch" 是开源ML框架。它是一个经过优化的用于深度学习的tensor库、使用GPU和CPU。PyTorch软件包包含多维感应器的数据结构、可提供许多实用程序来高效地序列化其他有用实用程序中的感应器。它还具有一个CUDA对应项、可用于在具有计算功能的NVIDIA GPU上运行Tensor计算。在此验证中、我们使用OpenCV-Python (CV2)库来验证我们的型号、同时利用Python最直观的计算机视觉概念。

简化数据管理

数据管理对于企业IT运营和数据科学家至关重要、这样才能将适当的资源用于AI应用程序和训练AI/ML数据集。以下有关NetApp技术的追加信息 不在此验证范围内、但可能与您的部署相关。

ONTAP 数据管理软件包括以下功能、可简化操作并降低总运营成本：

- 实时数据缩减和扩展的重复数据删除。数据缩减可减少存储块中浪费的空间、重复数据删除可显著提高有效容量。此适用场景数据存储在本地，并分层到云。
- 最低、最高和自适应服务质量(AQoS)。精细的服务质量(QoS)控制有助于在高度共享的环境中保持关键应用程序的性能水平。
- NetApp FabricPool。可将冷数据自动分层到公有 和私有云存储选项、包括Amazon Web Services (AWS) 、Azure和NetApp StorageGRID Storage解决方案。有关 FabricPool 的详细信息，请参见 "TR-4598 ：FabricPool 最佳实践"。

加速和保护数据

ONTAP 可提供卓越的性能和数据保护、并通过以下方式扩展这些功能：

- 性能和更低的延迟。ONTAP 可提供尽可能高的吞吐量和尽可能低的延迟。
- 数据保护ONTAP 可提供内置数据保护功能、并在所有平台之间进行通用管理。
- NetApp卷加密(NVE)。ONTAP 提供原生 卷级加密、并支持板载和外部密钥管理。
- 多租户和多因素身份验证。ONTAP 支持以最高的安全性级别共享基础架构资源。

Future-Proof 基础架构

ONTAP 可通过以下功能满足不断变化的苛刻业务需求：

- 无缝扩展和无中断运行。ONTAP 支持无中断地向现有控制器和横向扩展集群添加容量。客户可以升级到 NVMe 和 32 Gb FC 等最新技术，而无需进行成本高昂的数据迁移或中断。
- 云连接。ONTAP 是云互联程度最高的存储管理软件、可在所有公有 云中选择软件定义的存储(ONTAP Select)和云原生实例(NetApp Cloud Volumes Service)。
- 与新兴应用程序集成。ONTAP 通过使用支持现有企业应用程序的相同基础架构、为下一代平台和应用程序(例如自动驾驶汽车、智能城市和行业4.0)提供企业级数据服务。

NetApp Astra Control

NetApp Astra 产品系列为内部和公有 云中的 Kubernetes 应用程序提供存储和应用程序感知型数据管理服务，并采用 NetApp 存储和数据管理技术。通过它、您可以轻松备份Kubernetes应用程序、将数据迁移到其他集群以及即时创建有效的应用程序克隆。如果您需要管理在公有 云中运行的Kubernetes应用程序、请参见的文档 "Astra 控制服务"。Astra Control Service 是一项由 NetApp 管理的服务，可在 Google Kubernetes Engine （ GKEE ）和 Azure Kubernetes Service （ AKS ）中为 Kubernetes 集群提供应用程序感知型数据管理。

NetApp Astra Trident

Astra "Trident" NetApp 是适用于 Docker 和 Kubernetes 的开源动态存储编排程序，可简化永久性存储的创建，管理和使用。Trident是Kubernetes本机应用程序、直接在Kubernetes集群中运行。借助 Trident ，客户可以将 DL 容器映像无缝部署到 NetApp 存储上，并为 AI 容器部署提供企业级体验。Kubernetes用户(ML开发人员、数据科学家等)可以创建、管理和自动化流程编排和克隆、从而利用NetApp技术提供的高级数据管理功能。

NetApp BlueXP复制和同步

"BlueXP复制和同步" 是一项 NetApp 服务，用于快速安全地同步数据。无论您是需要在内部NFS还是SMB文件共享、NetApp StorageGRID、NetApp ONTAP S3、NetApp Cloud Volumes Service、Azure NetApp Files 、Amazon Simple Storage Service (Amazon S3)、Amazon Elelic File System (Amazon EFS)、Azure Blb 、Google Cloud Storage、 或者IBM Cloud Object Storage、BlueXP Copy and Sync可将文件快速安全地移动到您需要的位置。数据传输完成后，即可在源和目标上完全使用。BlueXP Copy和Syncc会根据您预定义的计划持续同步数据、仅移动增量、从而最大限度地减少数据复制所需的时间和资金。BlueXP Copy and Sync是一款软件即服务(SaaS)工具、设置和使用极其简单。由BlueXP复制和同步触发的数据传输由数据代理执行。您可以在AWS、Azure、Google Cloud Platform或内部部署BlueXP Copy and Sync数据代理。

NetApp BlueXP分类

由强大的AI算法驱动、 "NetApp BlueXP分类" 在您的整个数据资产中提供自动化控制和数据监管。您可以轻松确定成本节省、发现合规性和隐私问题、并找到优化机会。BlueXP分类信息板可让您深入了解如何识别重复数据以消除冗余、映射个人数据、非个人数据和敏感数据、以及针对敏感数据和异常情况启用警报。

测试和验证计划

在此解决方案 设计中、我们对以下三种情形进行了验证：

- 在JupyterLab工作空间中使用适用于Kubernetes的NetApp DataOps工具包进行编排的推理任务、包括使用和不使用Protopia混淆。
- 在Kubernetes上使用使用NetApp DataOps Toolkit for Kubernetes编排的数据卷进行批量推理作业、包括使用和不使用Protopia混淆。
- 使用NVIDIA Triton推理服务器实例的推理任务、该实例使用适用于Kubernetes的NetApp DataOps工具包进行编排。在调用Triton推理API之前、我们会对映像应用程序混淆、以模拟通过网络传输的任何数据必须被混淆的常见要求。此工作流适用于在受信任区域内收集数据、但必须传递到该受信任区域之外以进行推理的情形。如果没有Protopia混淆、则在敏感数据不离开受信任区域的情况下、无法实施此类工作流。

测试配置

下表概括了解决方案 设计验证环境。

组件	version
Kubernetes	1.21.6
NetApp Astra Trident CSI驱动程序	22.01.0
适用于Kubernetes的NetApp DataOps工具包	2.3.0
NVIDIA Triton 推理服务器	21.11-py3.

测试操作步骤

本节介绍完成验证所需的任务。

前提条件

要执行本节所述的任务、您必须能够访问安装并配置了以下工具的Linux或macOS主机：

- Kubectl (配置为访问现有Kubernetes集群)
  - 可参见安装和配置说明 "此处"。
- 适用于Kubernetes的NetApp DataOps工具包
  - 可以找到安装说明 "此处"。

场景1—JupyterLab中的按需推理

1. 为AI/ML推理工作负载创建Kubernetes命名空间。

```
$ kubectl create namespace inference
namespace/inference created
```

2. 使用NetApp DataOps工具包配置永久性卷、以存储要执行推理的数据。

```
$ netapp_dataops_k8s_cli.py create volume --namespace=inference --pvc
-name=inference-data --size=50Gi
Creating PersistentVolumeClaim (PVC) 'inference-data' in namespace
'inference'.
PersistentVolumeClaim (PVC) 'inference-data' created. Waiting for
Kubernetes to bind volume to PVC.
Volume successfully created and bound to PersistentVolumeClaim (PVC)
'inference-data' in namespace 'inference'.
```

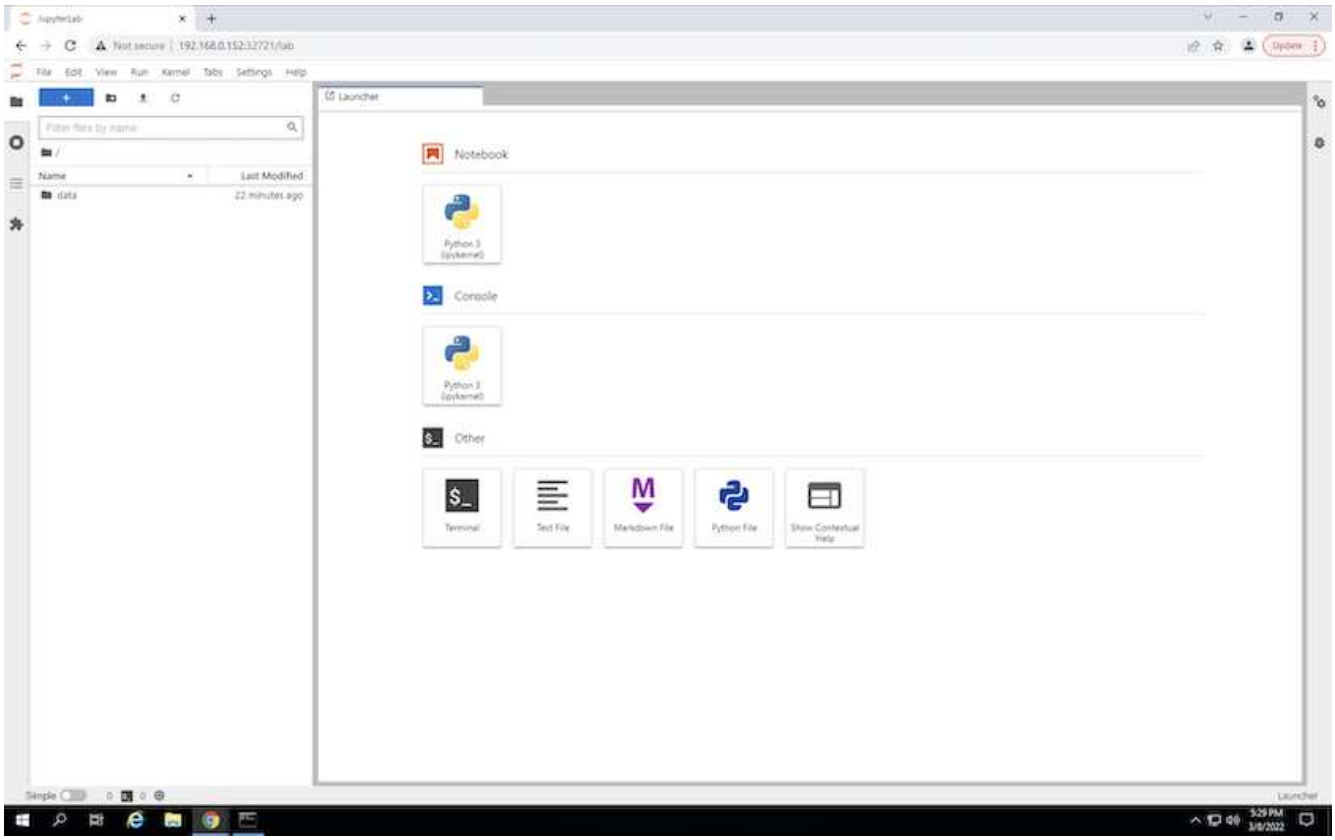
3. 使用NetApp DataOps工具包创建新的JupyterLab工作空间。使用`-mount- PVC`选项挂载上一步中创建的永久性卷。根据需要使用`- nvidia-GPU`选项将NVIDIA GPU分配给工作空间。

   在以下示例中、永久性卷`推理-data`会挂载到JupyterLab工作空间容器中、该容器位于`home/jovyon/data`。使用Project Jupyter官方容器映像时、`/home/jovyan`将作为JupyterLab Web界面中的

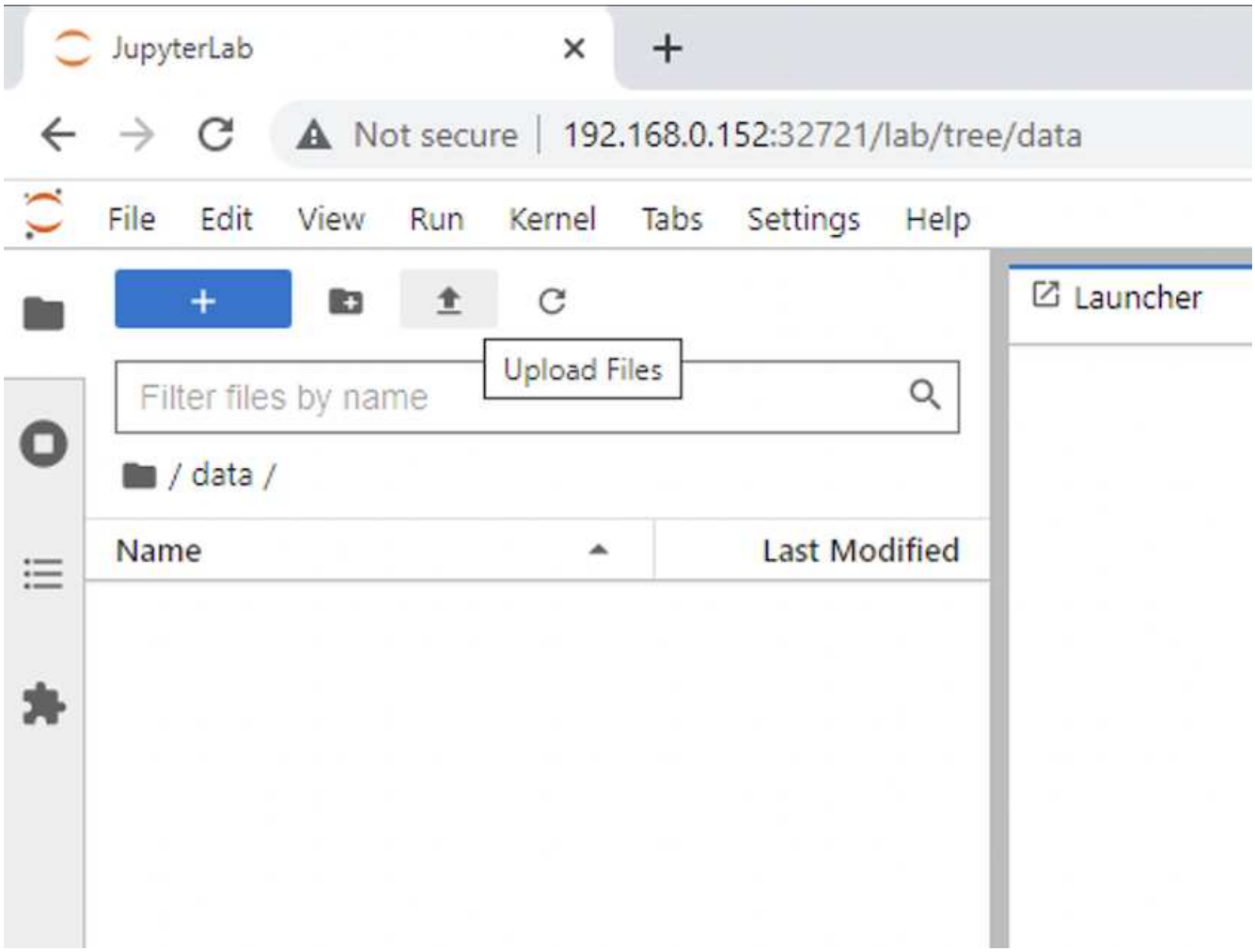
顶级目录提供。

```
$ netapp_dataops_k8s_cli.py create jupyterlab --namespace=inference
--workspace-name=live-inference --size=50Gi --nvidia-gpu=2 --mount
-pvc=inference-data:/home/jovyan/data
Set workspace password (this password will be required in order to
access the workspace):
Re-enter password:
Creating persistent volume for workspace...
Creating PersistentVolumeClaim (PVC) 'ntap-dsutil-jupyterlab-live-
inference' in namespace 'inference'.
PersistentVolumeClaim (PVC) 'ntap-dsutil-jupyterlab-live-inference'
created. Waiting for Kubernetes to bind volume to PVC.
Volume successfully created and bound to PersistentVolumeClaim (PVC)
'ntap-dsutil-jupyterlab-live-inference' in namespace 'inference'.
Creating Service 'ntap-dsutil-jupyterlab-live-inference' in namespace
'inference'.
Service successfully created.
Attaching Additional PVC: 'inference-data' at mount_path:
'/home/jovyan/data'.
Creating Deployment 'ntap-dsutil-jupyterlab-live-inference' in namespace
'inference'.
Deployment 'ntap-dsutil-jupyterlab-live-inference' created.
Waiting for Deployment 'ntap-dsutil-jupyterlab-live-inference' to reach
Ready state.
Deployment successfully created.
Workspace successfully created.
To access workspace, navigate to http://192.168.0.152:32721
```

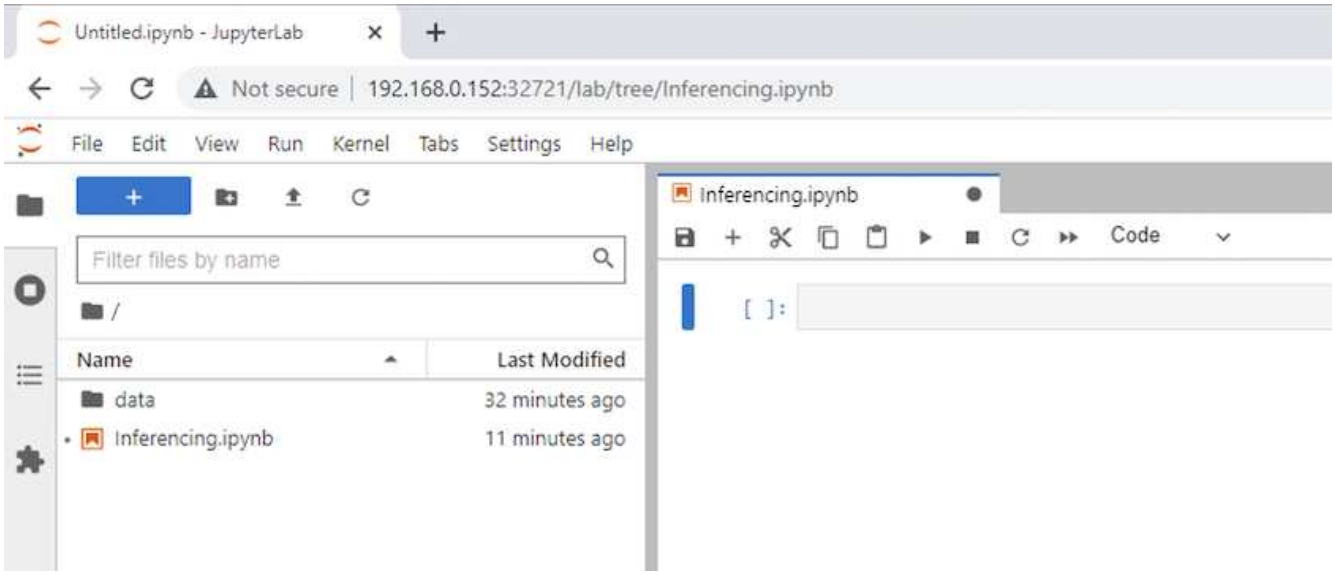
4. 使用`create jupyterlab`命令输出中指定的URL访问JupyterLab工作空间。数据目录表示已挂载到工作空间的永久性卷。



5. 打开`data`目录并上传要执行推理的文件。将文件上传到数据目录时、这些文件会自动存储在挂载到工作空间的永久性卷上。要上传文件、请单击上传文件图标、如下图所示。



6. 返回顶级目录并创建新的笔记本。



7. 向笔记本电脑添加推理代码。以下示例显示了图像检测用例的推理代码。

Launcher | image-demo-pytorch.ipynb

Code | Python 3 (ipykernel)

STEP 3-1: Clean (Without obfuscation) detection

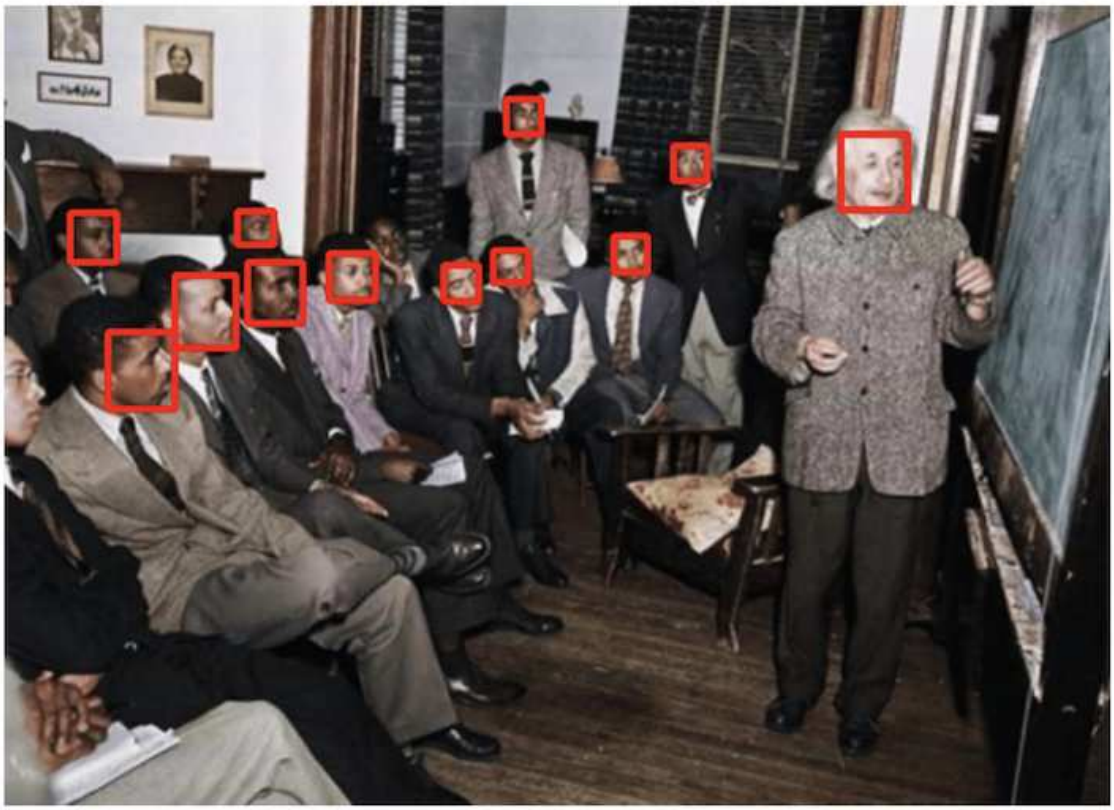
[9]:
```
# get current frame
frame = input_image

# preprocess input
preprocessed_input = preprocess_input(frame)
preprocessed_input = torch.Tensor(preprocessed_input).to(device)

# run forward pass
clean_activation = clean_model.forward_head(preprocessed_input)  # runs the first few layers
loc, pred = clean_model.forward_tail(clean_activation)  # runs rest of the layers

# postprocess output
clean_pred = (loc.detach().cpu().numpy(), pred.detach().cpu().numpy())
clean_outputs = postprocess_outputs(
    clean_pred, [[input_image_width, input_image_height]], priors, THRESHOLD
)

# draw rectangles
clean_frame = copy.deepcopy(frame)  # needs to be deep copy
for (x1, y1, x2, y2, s) in clean_outputs[0]:
    x1, y1 = int(x1), int(y1)
    x2, y2 = int(x2), int(y2)
    cv2.rectangle(clean_frame, (x1, y1), (x2, y2), (0, 0, 255), 4)
```

Launcher | image-demo-pytorch.ipynb

Code | Python 3 (ipykernel)

Visualize Clean (Without obfuscation) detection

[10]:
```
show_cv2_image(clean_frame, scale=2)
```



8. 将Protopia混淆添加到推理代码中。Protopia直接与客户合作、提供特定于使用情形的文档、不在本技术报告的范围内。以下示例显示了添加了Protopia混淆的图像检测用例的推理代码。

Launcher | image-demo-pytorch.ipynb

Code | Python 3 (ipykernel)

STEP 3-2: Protopia AI (With obfuscation) detection

[11]:
```python
# get current frame
frame = input_image

# preprocess input
preprocessed_input = preprocess_input(frame)
preprocessed_input = torch.Tensor(preprocessed_input).to(device)

# run forward pass
not_noisy_activation = noisy_model.forward_head(preprocessed_input)  # runs the first few layers
##################################################################
#          SINGLE ADITIONAL LINE FOR PRIVATE INFERENCE           #
##################################################################
noisy_activation = noisy_model.forward_noise(not_noisy_activation)
##################################################################
loc, pred = noisy_model.forward_tail(noisy_activation)  # runs rest of the layers

# postprocess output
noisy_pred = (loc.detach().cpu().numpy(), pred.detach().cpu().numpy())
noisy_outputs = postprocess_outputs(
    noisy_pred, [[input_image_width, input_image_height]], priors, THRESHOLD * 0.5
)

# get reconstruction of the noisy activation
noisy_reconstruction = decoder_function(noisy_activation)
noisy_reconstruction = noisy_reconstruction.detach().cpu().numpy()[0]
noisy_reconstruction = unpreprocess_output(
    noisy_reconstruction, (input_image_width, input_image_height), True
).astype(np.uint8)

# draw rectangles
for (x1, y1, x2, y2, s) in noisy_outputs[0]:
    x1, y1 = int(x1), int(y1)
    x2, y2 = int(x2), int(y2)
    cv2.rectangle(noisy_reconstruction, (x1, y1), (x2, y2), (0, 0, 255), 4)
```

Launcher | image-demo-pytorch.ipynb

Code | Python 3 (ipykernel)

Visualize Protopia AI (With obfuscation) detection

[12]:
```python
show_cv2_image(noisy_reconstruction, scale=2)
```

场景2—Kubernetes上的批处理推理

1. 为AI/ML推理工作负载创建Kubernetes命名空间。

```
$ kubectl create namespace inference
namespace/inference created
```

2. 使用NetApp DataOps工具包配置永久性卷、以存储要执行推理的数据。

```
$ netapp_dataops_k8s_cli.py create volume --namespace=inference --pvc
-name=inference-data --size=50Gi
Creating PersistentVolumeClaim (PVC) 'inference-data' in namespace
'inference'.
PersistentVolumeClaim (PVC) 'inference-data' created. Waiting for
Kubernetes to bind volume to PVC.
Volume successfully created and bound to PersistentVolumeClaim (PVC)
'inference-data' in namespace 'inference'.
```

3. 使用要执行推理的数据填充新的永久性卷。

   可以通过多种方法将数据加载到PVC上。如果您的数据当前存储在与S3兼容的对象存储平台中、例如NetApp StorageGRID 或Amazon S3、则可以使用 "NetApp DataOps工具包S3 Data Mover功能"。另一种简单的方法是创建JupyterLab工作空间、然后通过JupyterLab Web界面上传文件、如""一节中的步骤3到5所述场景1—JupyterLab中的按需推理。 "

4. 为批处理推理任务创建Kubernetes作业。以下示例显示了一个图像检测用例的批处理推理作业。此作业会对一组映像中的每个映像执行推理、并将推理准确性指标写入到stdout。

```
$ vi inference-job-raw.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-inference-raw
  namespace: inference
spec:
  backoffLimit: 5
  template:
    spec:
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: inference-data
      - name: dshm
        emptyDir:
          medium: Memory
      containers:
      - name: inference
        image: netapp-protopia-inference:latest
        imagePullPolicy: IfNotPresent
        command: ["python3", "run-accuracy-measurement.py", "--dataset",
"/data/netapp-face-detection/FDDB"]
        resources:
          limits:
            nvidia.com/gpu: 2
        volumeMounts:
        - mountPath: /data
          name: data
        - mountPath: /dev/shm
          name: dshm
      restartPolicy: Never
$ kubectl create -f inference-job-raw.yaml
job.batch/netapp-inference-raw created
```

5. 确认推理作业已成功完成。

```
$ kubectl -n inference logs netapp-inference-raw-255sp
100%|██████████| 89/89 [00:52<00:00,  1.68it/s]
Reading Predictions : 100%|██████████| 10/10 [00:01<00:00,  6.23it/s]
Predicting ... : 100%|██████████| 10/10 [00:16<00:00,  1.64s/it]
==================== Results ====================
FDDB-fold-1 Val AP: 0.9491256561145955
FDDB-fold-2 Val AP: 0.9205024466101926
FDDB-fold-3 Val AP: 0.9253013871078468
FDDB-fold-4 Val AP: 0.9399781485863011
FDDB-fold-5 Val AP: 0.9504280149478732
FDDB-fold-6 Val AP: 0.9416473519339292
FDDB-fold-7 Val AP: 0.9241631566241117
FDDB-fold-8 Val AP: 0.9072663297546659
FDDB-fold-9 Val AP: 0.9339648715035469
FDDB-fold-10 Val AP: 0.9447707905560152
FDDB Dataset Average AP: 0.9337148153739079
==================================================
mAP: 0.9337148153739079
```

6. 将Protopia混淆添加到推理作业。您可以从Protopia中找到直接添加Protopia混淆的使用案例专用说明、该说明不在本技术报告的讨论范围内。以下示例显示了一个人脸检测用例的批处理推理作业、该用例使用0.8的字母值添加了质子模糊。此作业会在对一组图像中的每个图像执行推理之前应用程序对象模糊、然后将推理准确性指标写入stdout。

   对于alpha值0.05%、0.1、0.2、0.4、0.6、 0.8、0.9和0.95。您可以在中查看结果 ""推理准确性比较。""

```
$ vi inference-job-protopia-0.8.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-inference-protopia-0.8
  namespace: inference
spec:
  backoffLimit: 5
  template:
    spec:
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: inference-data
      - name: dshm
        emptyDir:
          medium: Memory
      containers:
      - name: inference
        image: netapp-protopia-inference:latest
        imagePullPolicy: IfNotPresent
        env:
        - name: ALPHA
          value: "0.8"
        command: ["python3", "run-accuracy-measurement.py", "--dataset",
"/data/netapp-face-detection/FDDB", "--alpha", "$(ALPHA)", "--noisy"]
        resources:
          limits:
            nvidia.com/gpu: 2
        volumeMounts:
        - mountPath: /data
          name: data
        - mountPath: /dev/shm
          name: dshm
      restartPolicy: Never
$ kubectl create -f inference-job-protopia-0.8.yaml
job.batch/netapp-inference-protopia-0.8 created
```

7. 确认推理作业已成功完成。

```
$ kubectl -n inference logs netapp-inference-protopia-0.8-b4dkz
100%|██████████| 89/89 [01:05<00:00,  1.37it/s]
Reading Predictions : 100%|██████████| 10/10 [00:02<00:00,  3.67it/s]
Predicting ... : 100%|██████████| 10/10 [00:22<00:00,  2.24s/it]
==================== Results ====================
FDDB-fold-1 Val AP: 0.8953066115834589
FDDB-fold-2 Val AP: 0.8819580264029936
FDDB-fold-3 Val AP: 0.8781107458462862
FDDB-fold-4 Val AP: 0.9085731346308461
FDDB-fold-5 Val AP: 0.9166445508275378
FDDB-fold-6 Val AP: 0.9101178994188819
FDDB-fold-7 Val AP: 0.8383443678423771
FDDB-fold-8 Val AP: 0.8476311547659464
FDDB-fold-9 Val AP: 0.8739624502111121
FDDB-fold-10 Val AP: 0.8905468076424851
FDDB Dataset Average AP: 0.8841195749171925
==================================================
mAP: 0.8841195749171925
```

场景3—NVIDIA Triton推理服务器

1. 为AI/ML推理工作负载创建Kubernetes命名空间。

```
$ kubectl create namespace inference
namespace/inference created
```

2. 使用NetApp DataOps工具包配置永久性卷、以用作NVIDIA Triton推理服务器的型号存储库。

```
$ netapp_dataops_k8s_cli.py create volume --namespace=inference --pvc
-name=triton-model-repo --size=100Gi
Creating PersistentVolumeClaim (PVC) 'triton-model-repo' in namespace
'inference'.
PersistentVolumeClaim (PVC) 'triton-model-repo' created. Waiting for
Kubernetes to bind volume to PVC.
Volume successfully created and bound to PersistentVolumeClaim (PVC)
'triton-model-repo' in namespace 'inference'.
```

3. 将您的型号存储在中的新永久性卷上 "格式。" 这可由NVIDIA Triton推理服务器识别。

   可以通过多种方法将数据加载到PVC上。一种简单的方法是创建JupyterLab工作空间、然后通过JupyterLab Web界面上传文件、如中的步骤3到5所述场景1—JupyterLab中的按需推理。"

4. 使用NetApp DataOps工具包部署新的NVIDIA Triton推理服务器实例。

```
$ netapp_dataops_k8s_cli.py create triton-server --namespace=inference
--server-name=netapp-inference --model-repo-pvc-name=triton-model-repo
Creating Service 'ntap-dsutil-triton-netapp-inference' in namespace
'inference'.
Service successfully created.
Creating Deployment 'ntap-dsutil-triton-netapp-inference' in namespace
'inference'.
Deployment 'ntap-dsutil-triton-netapp-inference' created.
Waiting for Deployment 'ntap-dsutil-triton-netapp-inference' to reach
Ready state.
Deployment successfully created.
Server successfully created.
Server endpoints:
http: 192.168.0.152: 31208
grpc: 192.168.0.152: 32736
metrics: 192.168.0.152: 30009/metrics
```

5. 使用Triton客户端SDK执行推理任务。以下Python代码摘录使用Triton Python客户端SDK为人脸检测用例执行推理任务。此示例调用Triton API并传递图像以进行推理。然后、Triton推理服务器接收请求、调用模型、并在API结果中返回推理输出。

```
# get current frame
frame = input_image
# preprocess input
preprocessed_input = preprocess_input(frame)
preprocessed_input = torch.Tensor(preprocessed_input).to(device)
# run forward pass
clean_activation = clean_model_head(preprocessed_input)  # runs the
first few layers
#######################################################################
##############
#          pass clean image to Triton Inference Server API for
inferencing           #
#######################################################################
##############
triton_client =
httpclient.InferenceServerClient(url="192.168.0.152:31208",
verbose=False)
model_name = "face_detection_base"
inputs = []
outputs = []
inputs.append(httpclient.InferInput("INPUT__0", [1, 128, 32, 32],
"FP32"))
inputs[0].set_data_from_numpy(clean_activation.detach().cpu().numpy(),
binary_data=False)
```

```
outputs.append(httpclient.InferRequestedOutput("OUTPUT__0",
binary_data=False))
outputs.append(httpclient.InferRequestedOutput("OUTPUT__1",
binary_data=False))
results = triton_client.infer(
    model_name,
    inputs,
    outputs=outputs,
    #query_params=query_params,
    headers=None,
    request_compression_algorithm=None,
    response_compression_algorithm=None)
#print(results.get_response())
statistics =
triton_client.get_inference_statistics(model_name=model_name,
headers=None)
print(statistics)
if len(statistics["model_stats"]) != 1:
    print("FAILED: Inference Statistics")
    sys.exit(1)

loc_numpy = results.as_numpy("OUTPUT__0")
pred_numpy = results.as_numpy("OUTPUT__1")
#######################################################################
##############
# postprocess output
clean_pred = (loc_numpy, pred_numpy)
clean_outputs = postprocess_outputs(
    clean_pred, [[input_image_width, input_image_height]], priors,
THRESHOLD
)
# draw rectangles
clean_frame = copy.deepcopy(frame)  # needs to be deep copy
for (x1, y1, x2, y2, s) in clean_outputs[0]:
    x1, y1 = int(x1), int(y1)
    x2, y2 = int(x2), int(y2)
    cv2.rectangle(clean_frame, (x1, y1), (x2, y2), (0, 0, 255), 4)
```

6. 将Protopia混淆添加到推理代码中。您可以从Protopia中找到直接添加Protopia混淆的使用案例专用说明；但是、此过程不在本技术报告的讨论范围内。以下示例显示了与上一步5中显示的相同的Python代码、但添加了Protopia obfuscation。

   请注意、在将图像传递到Triton API之前、系统会对该映像应用程序模糊。因此、非混淆映像永远不会离开本地计算机。仅通过网络传递模糊映像。此工作流适用于以下情形：在受信任区域内收集数据、但随后需要传递到该受信任区域以外以进行推理。如果没有Protopia混淆、则在敏感数据不离开受信任区域的情况下、无法实施此类工作流。

```
# get current frame
frame = input_image
# preprocess input
preprocessed_input = preprocess_input(frame)
preprocessed_input = torch.Tensor(preprocessed_input).to(device)
# run forward pass
not_noisy_activation = noisy_model_head(preprocessed_input)  # runs the
first few layers
#################################################################
#          obfuscate image locally prior to inferencing          #
#          SINGLE ADITIONAL LINE FOR PRIVATE INFERENCE           #
#################################################################
noisy_activation = noisy_model_noise(not_noisy_activation)
#################################################################
#######################################################################
####################
#          pass obfuscated image to Triton Inference Server API for
inferencing           #
#######################################################################
####################
triton_client =
httpclient.InferenceServerClient(url="192.168.0.152:31208",
verbose=False)
model_name = "face_detection_noisy"
inputs = []
outputs = []
inputs.append(httpclient.InferInput("INPUT__0", [1, 128, 32, 32],
"FP32"))
inputs[0].set_data_from_numpy(noisy_activation.detach().cpu().numpy(),
binary_data=False)
outputs.append(httpclient.InferRequestedOutput("OUTPUT__0",
binary_data=False))
outputs.append(httpclient.InferRequestedOutput("OUTPUT__1",
binary_data=False))
results = triton_client.infer(
    model_name,
    inputs,
    outputs=outputs,
    #query_params=query_params,
    headers=None,
    request_compression_algorithm=None,
    response_compression_algorithm=None)
#print(results.get_response())
statistics =
triton_client.get_inference_statistics(model_name=model_name,
headers=None)
```

```
print(statistics)
if len(statistics["model_stats"]) != 1:
    print("FAILED: Inference Statistics")
    sys.exit(1)

loc_numpy = results.as_numpy("OUTPUT__0")
pred_numpy = results.as_numpy("OUTPUT__1")
#####################################################################
###################

# postprocess output
noisy_pred = (loc_numpy, pred_numpy)
noisy_outputs = postprocess_outputs(
    noisy_pred, [[input_image_width, input_image_height]], priors,
THRESHOLD * 0.5
)
# get reconstruction of the noisy activation
noisy_reconstruction = decoder_function(noisy_activation)
noisy_reconstruction = noisy_reconstruction.detach().cpu().numpy()[0]
noisy_reconstruction = unpreprocess_output(
    noisy_reconstruction, (input_image_width, input_image_height), True
).astype(np.uint8)
# draw rectangles
for (x1, y1, x2, y2, s) in noisy_outputs[0]:
    x1, y1 = int(x1), int(y1)
    x2, y2 = int(x2), int(y2)
    cv2.rectangle(noisy_reconstruction, (x1, y1), (x2, y2), (0, 0, 255),
4)
```

推理准确性比较

在此验证中、我们使用一组原始映像为图像检测用例执行推理。然后、我们对同一组图像执行了相同的推理任务、并在推理之前添加了质子模糊。我们对Protopia混淆组件使用不同的alpha值重复执行此任务。在Protopia混淆的上下文中、alpha值表示应用的混淆量、较高的alpha值表示较高的混淆级别。然后、我们对这些不同的运行中的推理准确性进行了比较。

以下两个表提供了有关我们使用情形的详细信息并概述了结果。

Protopia直接与客户合作、为特定使用情形确定适当的alpha值。

组件	详细信息
型号	面板(PyTorch)-
数据集	FDDB数据集

前倾混淆	字母	准确性
否	不适用	0.9337148153739079
是的。	0.05	0.9028766627325002
是的。	0.1	0.9024301009661478
是的。	0.2	0.9081836283186224
是的。	0.4	0.9073066107482036
是的。	0.6	0.8847816568680239
是的。	0.8	0.8841195749171925
是的。	0.9	0.8455427675252052
是的。	0.95	0.8455427675252052

模糊速度

在此验证中、我们对1920 x 1080像素图像应用了五次Protopia混淆、并测量了每次完成混淆步骤所需的时间。

我们使用在一个NVIDIA V100 GPU上运行的PyTorch来应用混淆、并在两次运行之间清除了GPU缓存。在五次运行中、模糊步骤分别需要5.47毫秒、5.27毫秒、4.54毫秒、5.24毫秒和4.84毫秒才能完成。平均速度为5.072毫秒。

结论

数据有三种状态：空闲、传输和计算。任何AI推理服务的一个重要部分都应该是在整个过程中保护数据免受威胁。在推理期间保护数据至关重要、因为该过程可以公开有关外部客户和提供推理服务的企业的私有信息。Protopia AI是一种在当今市场中用于机密人工智能推理的纯软件解决方案。借助Protopia、AI只会从数据记录中传输经过转换的信息、而这些信息对于执行手头的AI/ML任务至关重要、而且不会再提供更多信息。这种随机转型不是一种屏蔽形式、而是基于使用精心设计的噪声以数学方式更改数据表示形式。

具有ONTAP 功能的NetApp存储系统可提供与本地SSD存储相同或更好的性能、并且与NetApp DataOps工具包相结合、可为数据科学家、数据工程师、AI/ML开发人员以及业务或企业IT决策者带来以下优势：

- 在 AI 系统，分析和其他关键业务系统之间轻松共享数据。这种数据共享可减少基础架构开销，提高性能并简化整个企业的数据管理。
- 可独立扩展的计算和存储，最大限度地降低成本并提高资源利用率。
- 利用集成的 Snapshot 副本和克隆简化开发和部署工作流，实现瞬时且节省空间的用户工作空间，集成版本控制和自动化部署。
- 针对灾难恢复、业务连续性和法规要求的企业级数据保护和数据监管。
- 简化了数据管理操作的调用；从Jupyter笔记本电脑中的NetApp DataOps工具包快速获取数据科学家工作空间的Snapshot副本、以实现备份和可追溯性。

NetApp和Protopia解决方案 提供了一个灵活的横向扩展架构、非常适合企业级AI推理部署。它支持数据保护、

并为敏感信息提供隐私保护、可通过内部部署和混合云部署中的负责AI实践满足机密AI推理要求。

从何处查找追加信息和相关信息

要了解有关本文档中所述信息的更多信息，请参见以下文档和 / 或网站：

- NetApp ONTAP 数据管理软件—ONTAP 信息库

  http://mysupport.netapp.com/documentation/productlibrary/index.html?productID=62286

- 适用于容器的 NetApp 持久存储— NetApp Trident

  "https://netapp.io/persistent-storage-provisioner-for-kubernetes/"

- NetApp DataOps 工具包

  "https://github.com/NetApp/netapp-dataops-toolkit"

- 适用于容器的NetApp持久存储—NetApp Astra Trident

  "https://netapp.io/persistent-storage-provisioner-for-kubernetes/"

- Protopia AI—机密推理

  "https://protopia.ai/blog/protopia-ai-takes-on-the-missing-link-in-ai-privacy-confidential-inference/"

- NetApp BlueXP复制和同步

  "https://docs.netapp.com/us-en/occm/concept_cloud_sync.html#how-cloud-sync-works"

- NVIDIA Triton 推理服务器

  "https://developer.nvidia.com/nvidia-triton-inference-server"

- NVIDIA Triton推理服务器文档

  "https://docs.nvidia.com/deeplearning/triton-inference-server/index.html"

- PyTorch中的FaceBoxes

  "https://github.com/zisianw/FaceBoxes.PyTorch"

致谢

- NetApp首席产品经理Mark Cates
- NetApp技术营销工程师Sufian Ahmad
- 首席技术官兼Protopia AI教授Hadi Essmaeilzadeh

版权信息

版权所有 © 2024 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。 美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 http://www.netapp.com/TM 上所列的商标是 NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。