Protopia 影像轉換的負責任的 AI

NetApp artificial intelligence solutions

NetApp
February 12, 2026

This PDF was generated from https://docs.netapp.com/zh-tw/netapp-solutions-ai/software/ai-conf-protopia-overview.html on February 12, 2026. Always check docs.netapp.com for the latest.

目錄

Protopia 影像轉換的負責任的 AI . . . 1
TR-4928：負責任的 AI 和機密推理 - NetApp AI 與 Protopia 影像和資料轉換 . . . 1
目標受眾 . . . 1
解決方案架構 . . . 2
解決方案領域 . . . 3
環境情報 . . . 3
邊緣設備穿戴裝置 . . . 3
非戰鬥人員撤離行動 . . . 3
醫療保健和生物醫學研究 . . . 3
AI/ML分析的雲端遷移 . . . 3
技術概述 . . . 4
普羅托邦 . . . 4
NetApp ONTAP AI . . . 4
NetApp ONTAP . . . 5
NetApp DataOps 工具包 . . . 5
NVIDIA Triton 推理伺服器 . . . 6
PyTorch . . . 6
NetApp Astra控制 . . . 7
NetApp Trident . . . 7
NetApp BlueXP複製與同步 . . . 7
NetApp BlueXP分類 . . . 7
測試和驗證計劃 . . . 7
測試配置 . . . 8
測試程式 . . . 8
先決條件 . . . 8
場景 1 – JupyterLab 中的按需推理 . . . 8
場景 2 – Kubernetes 上的批次推理 . . . 13
場景 3 – NVIDIA Triton 推理伺服器 . . . 18
推理精度比較 . . . 22
混淆速度 . . . 23
結論 . . . 23
在哪裡可以找到更多資訊和致謝 . . . 24
致謝 . . . 24

Protopia 影像轉換的負責任的 AI

TR-4928：負責任的 AI 和機密推理 - NetApp AI 與 Protopia 影像和資料轉換

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

隨著影像捕捉和影像處理技術的出現，視覺解讀已成為溝通不可或缺的一部分。數位影像處理中的人工智慧 (AI) 帶來了新的商業機會，例如在醫療領域用於癌症和其他疾病的識別、在地理空間視覺化分析中用於研究環境危害、在模式識別中、在視訊處理中用於打擊犯罪等等。然而，這一機會也伴隨著非凡的責任。

組織交給人工智慧的決策越多，他們承擔的與資料隱私和安全以及法律、道德和監管問題相關的風險就越大。負責任的人工智慧使公司和政府組織能夠建立信任和治理的實踐，這對於大型企業大規模應用人工智慧至關重要。本文檔介紹了NetApp在三種不同情境下驗證的 AI 推理解決方案，該解決方案使用NetApp資料管理技術與 Protopia 資料混淆軟體來私有化敏感資料並降低風險和道德問題。

消費者和商業實體每天都會使用各種數位設備產生數百萬張影像。隨之而來的數據和運算工作量的激增使得企業轉向雲端運算平台來實現規模和效率。同時，隨著影像資料轉移到公有雲，人們對其所含敏感資訊的隱私問題也產生了擔憂。缺乏安全和隱私保障成為影像處理人工智慧系統部署的主要障礙。

此外，還有 "刪除權"根據 GDPR，個人有權要求組織刪除其所有個人資料。還有 "隱私權法"，該法案製定了公平資訊實踐準則。根據 GDPR，照片等數位影像可以構成個人數據，GDPR 規定了資料的收集、處理和刪除方式。不這樣做就是不遵守 GDPR，這可能會導致違反合規性的巨額罰款，這可能會對組織造成嚴重損害。隱私權原則是實施負責任的人工智慧的支柱之一，它確保機器學習 (ML) 和深度學習 (DL) 模型預測的公平性，並降低違反隱私或法規遵循相關的風險。

本文檔描述了在三種不同場景下經過驗證的設計解決方案，包括有和沒有影像混淆，這些場景與保護隱私和部署負責任的 AI 解決方案有關：

- 場景 1. Jupyter 筆記本中的按需推理。
- 場景 2.在 Kubernetes 上進行批次推理。
- 場景 3. NVIDIA Triton 推理伺服器。

對於該解決方案，我們使用人臉偵測資料集和基準（FDDB），這是一個為研究無約束人臉偵測問題而設計的人臉區域資料集，結合 PyTorch 機器學習框架來實現 FaceBoxes。此資料集包含 2845 張不同解析度影像中 5171 張臉的註釋。此外，本技術報告還介紹了從NetApp客戶和現場工程師收集的一些適用於此解決方案的解決方案領域和相關用例。

目標受眾

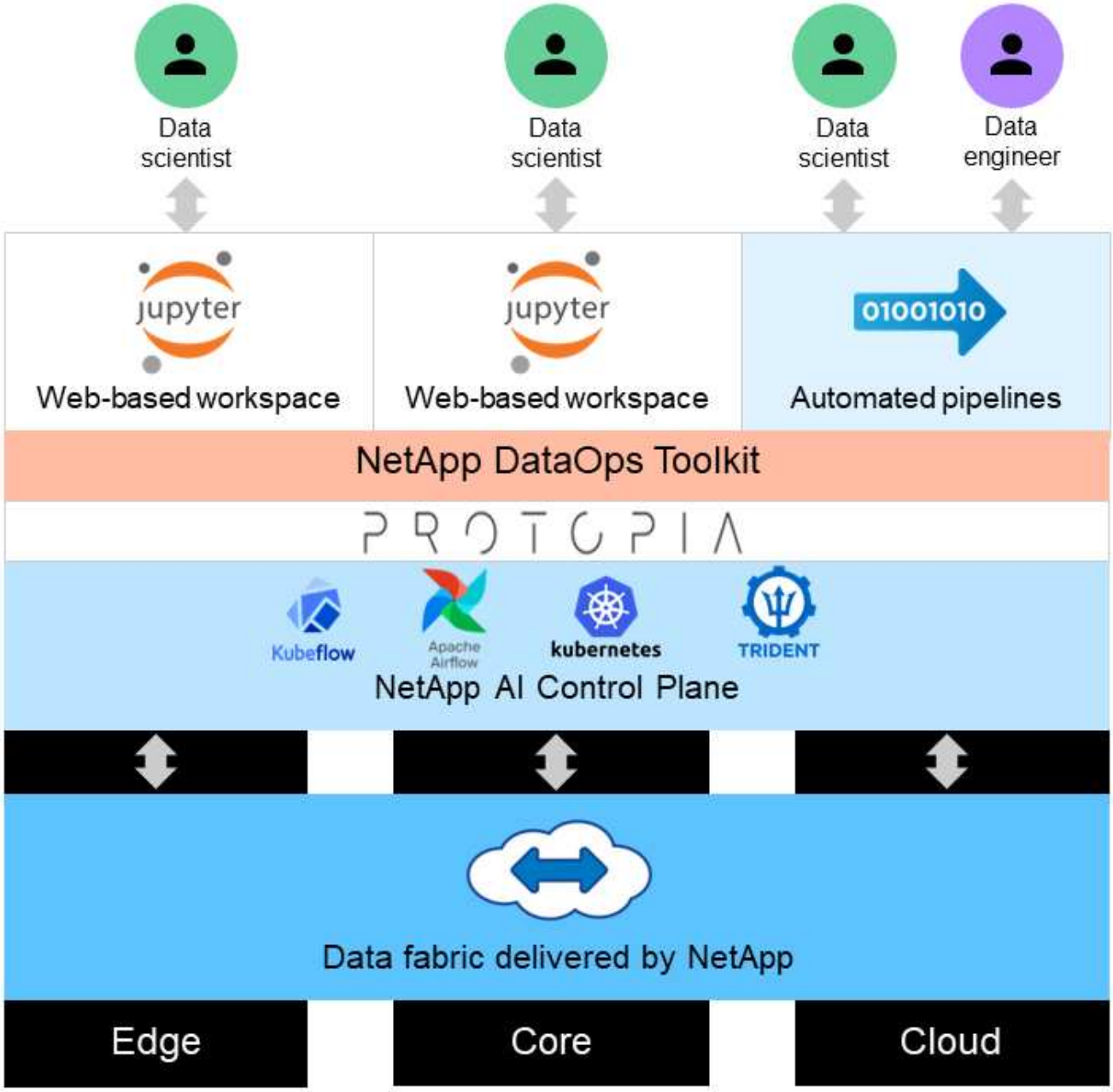
本技術報告面向以下受眾：

- 希望設計和部署負責任的人工智慧並解決公共場所臉部影像處理的資料保護和隱私問題的商業領袖和企業架構師。
- 旨在保護和維護隱私的資料科學家、資料工程師、人工智慧/機器學習 (ML) 研究人員以及人工智慧/機器學習系統開發人員。

- 為符合 GDPR、CCPA 或國防部 (DoD) 和政府組織的隱私法等監管標準的 AI/ML 模型和應用程式設計資料混淆解決方案的企業架構師。
- 資料科學家和人工智慧工程師正在尋找有效的方法來部署深度學習 (DL) 和 AI/ML/DL 推理模型來保護敏感資訊。
- 負責邊緣推理模型的部署和管理的邊緣設備管理員和邊緣伺服器管理員。

解決方案架構

該解決方案旨在利用 GPU 和傳統 CPU 的處理能力來處理大型資料集上的即時和批量推理 AI 工作負載。此驗證證明了尋求負責任的 AI 部署的組織所需的 ML 隱私保護推理和最佳資料管理。該解決方案提供了適用於單節點或多節點 Kubernetes 平台的架構，用於邊緣和雲端運算，並透過 Jupyter Lab 和 CLI 介面與核心本地的NetApp ONTAP AI、 NetApp DataOps Toolkit 和 Protopia 混淆軟體互連。下圖顯示了由NetApp支援、採用 DataOps Toolkit 和 Protopia 的資料結構的邏輯架構概覽。



Protopia 混淆軟體在NetApp DataOps Toolkit 上無縫運行，並在離開儲存伺服器之前轉換資料。

解決方案領域

數位影像處理具有許多優點，使許多組織能夠充分利用與視覺表示相關的數據。 NetApp和Protopia 解決方案提供了獨特的 AI 推理設計，以在整個 ML/DL 生命週期中保護和私有化 AI/ML 資料。它使客戶能夠保留敏感資料的所有權，透過緩解與隱私相關的擔憂，使用公共或混合雲部署模型實現規模和效率，並在邊緣部署人工智慧推理。

環境情報

在環境危害領域，各行各業可以多種方式利用地理空間分析。政府和公共工程部門可以就公共衛生和天氣狀況獲得可行的見解，以便在流行病或野火等自然災害期間更好地為公眾提供建議。例如，您可以在公共場所（例如機場或醫院）識別 COVID 陽性患者，而不會損害受影響個人的隱私，並提醒相關部門和附近公眾採取必要的安全措施。

邊緣設備穿戴裝置

在軍事和戰場上，您可以使用邊緣 AI 推理作為可穿戴設備來追蹤士兵的健康狀況、監控駕駛員行為並向當局發出接近軍用車輛的安全和相關風險警報，同時保護士兵的隱私。軍隊的未來將走向高科技，戰場物聯網 (IoBT) 和軍事物聯網 (IoMT) 將應用於可穿戴作戰裝備，幫助士兵透過使用快速邊緣運算識別敵人並在戰鬥中表現得更好。保護和保存從無人機和穿戴式裝置等邊緣裝置收集的視覺資料對於阻止駭客和敵人至關重要。

非戰鬥人員撤離行動

非戰鬥人員撤離行動 (NEO) 由國防部執行，旨在協助將生命受到威脅的美國公民和國民、國防部文職人員以及指定人員（東道國 (HN) 和第三國國民 (TCN)）撤離到適當的安全避難所。現有的行政控制措施主要採用手動的撤離人員篩選流程。然而，透過使用高度自動化的 AI/ML 工具結合 AI/ML 視訊混淆技術，可以提高撤離人員識別、撤離人員追蹤和威脅篩選的準確性、安全性和速度。

醫療保健和生物醫學研究

影像處理用於根據電腦斷層掃描 (CT) 或磁振造影 (MRI) 獲得的 3D 影像診斷手術規劃的病理。 HIPAA 隱私規則規定了組織如何收集、處理和刪除所有個人資訊和照片等數位影像。根據 HIPAA 安全港規定，要使資料符合可共享條件，必須刪除正面照片和任何類似影像。用於從結構 CT/MR 影像中隱藏個人臉部特徵的去識別或顱骨剝離演算法等自動化技術已成為生物醫學研究機構資料共享過程的重要組成部分。

AI/ML分析的雲端遷移

企業客戶傳統上在本地訓練和部署 AI/ML 模型。出於規模經濟和效率的原因，這些客戶正在擴展以將 AI/ML 功能轉移到公共、混合或多雲部署。然而，它們受到可以向其他基礎設施公開的數據的限制。 NetApp解決方案可應對各種網路安全威脅， "資料保護"和安全評估，並與 Protopia 資料轉換結合，最大限度地降低將影像處理 AI/ML 工作負載遷移到雲端所帶來的風險。

有關其他行業的邊緣運算和 AI 推理的更多用例，請參閱"TR-4886 邊緣人工智慧推理"以及NetApp AI 博客， "情報與隱私" 。

技術概述

本節概述了完成此解決方案所需的各種技術組件。

普羅托邦

Protopia AI 為當今市場上的機密推理提供了一種不引人注目的純軟體解決方案。 Protopia 解決方案透過最大限度地減少敏感資訊的暴露，為推理服務提供了無與倫比的保護。人工智慧僅接收資料記錄中對於執行手頭任務真正必要的信息，僅此而已。大多數推理任務不會使用每個資料記錄中存在的所有資訊。無論您的 AI 使用的是圖像、語音、視訊還是結構化表格數據，Protopia 都只提供推理服務所需的內容。該專利核心技術使用數學策劃的雜訊來隨機轉換資料並混淆給定 ML 服務不需要的資訊。該解決方案不會掩蓋數據；相反，它通過使用精選的隨機噪聲來改變數據表示。

Protopia 解決方案將改變表示的問題表述為基於梯度的擾動最大化方法，該方法仍然保留與模型功能相關的輸入特徵空間中的信息。此發現過程在訓練 ML 模型結束時作為微調過程運行。在傳遞過程自動產生一組機率分佈之後，低開銷數據轉換會將這些分佈中的雜訊樣本應用於數據，並在將其傳遞給模型進行推理之前對其進行混淆。

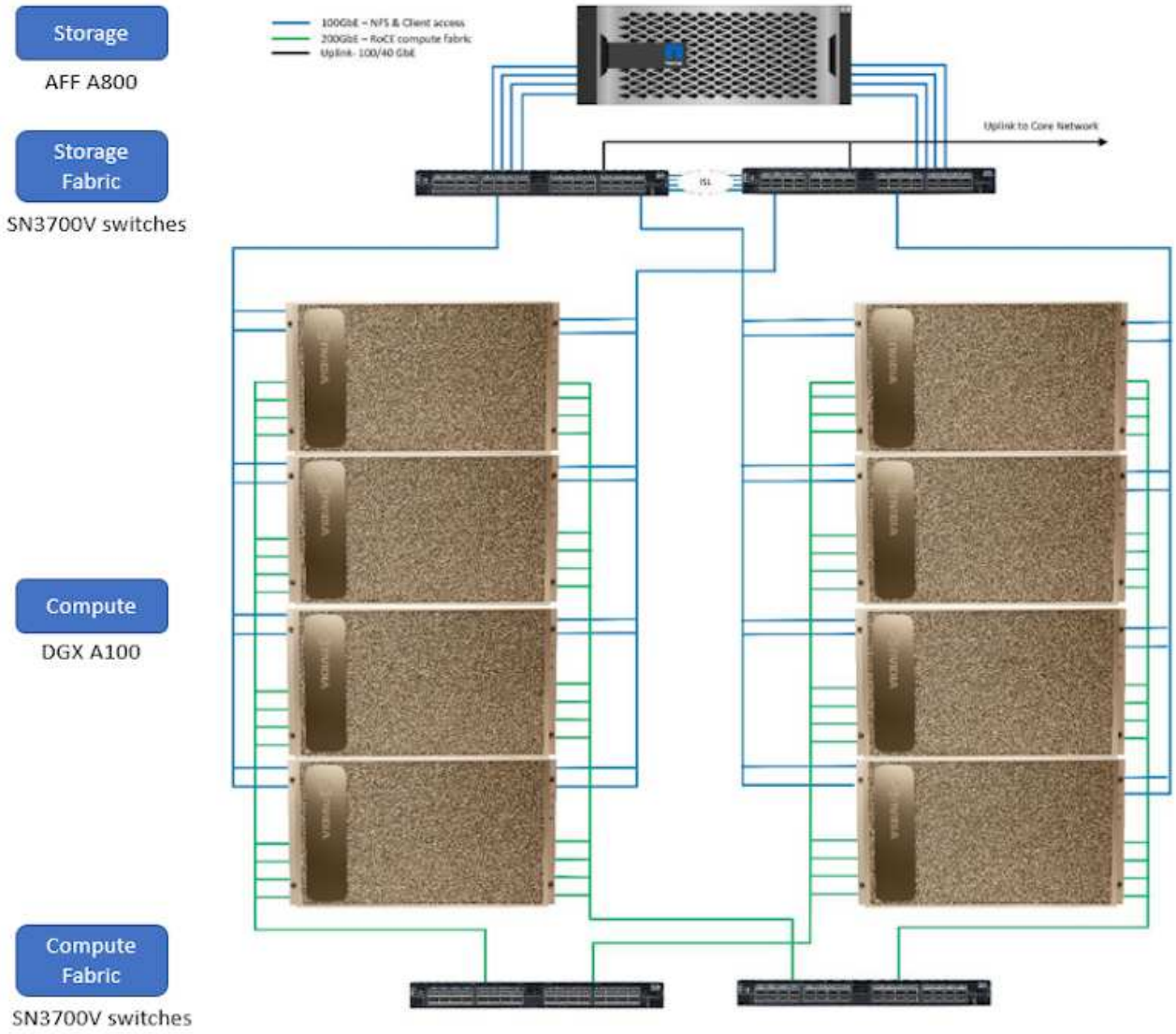
NetApp ONTAP AI

NetApp ONTAP AI 參考架構由 DGX A100 系統和NetApp雲端連接儲存系統提供支持，由NetApp和NVIDIA開發和驗證。它為 IT 組織提供了一個具有以下優勢的架構：

- 消除設計複雜性
- 允許獨立擴展計算和存儲
- 使客戶能夠從小規模開始並無縫擴展
- 提供一系列適合各種效能和成本點的儲存選項

ONTAP AI 將 DGX A100 系統和NetApp AFF A800儲存系統與最先進的網路緊密整合。 ONTAP AI 透過消除設計複雜性和猜測來簡化 AI 部署。客戶可以從小規模開始，然後無中斷地發展，同時智慧地管理從邊緣到核心到雲端再返回的資料。

下圖顯示了採用 DGX A100 系統的ONTAP AI 系列解決方案的幾種變體。 AFF A800系統性能已通過最多八個 DGX A100 系統驗證。透過向ONTAP叢集添加儲存控制器對，該架構可以擴展到多個機架，以支援許多 DGX A100 系統和具有線性性能的 PB 級儲存容量。這種方法可以靈活地根據所使用的 DL 模型的大小和所需的效能指標獨立地改變計算與儲存的比率。



有關ONTAP AI 的更多信息，請參閱 "NVA-1153：配備NVIDIA DGX A100 系統和 Mellanox Spectrum 乙太網路交換器的NetApp ONTAP AI。"

NetApp ONTAP

ONTAP 9.11 是NetApp最新一代儲存管理軟體，它支援企業將基礎架構現代化並過渡到雲端就緒資料中心。ONTAP利用業界領先的數據管理功能，只需一套工具即可管理和保護數據，無論數據位於何處。您也可以將資料自由移動到任何需要的地方：邊緣、核心或雲端。 ONTAP 9.11 包含許多功能，可簡化資料管理、加速和保護關鍵數據，並支援跨混合雲架構的下一代基礎架構功能。

NetApp DataOps 工具包

NetApp DataOps Toolkit 是一個 Python 函式庫，可協助開發人員、資料科學家、DevOps 工程師和資料工程師輕鬆執行各種資料管理任務，例如近乎即時地配置新的資料磁碟區或 JupyterLab 工作區、近乎即時地複製資料磁碟區或 JupyterLab 工作區，以及近乎即時快照資料磁碟區或 JupyterLab 工作區的基準測試區以進行可測試區的基準測試區或 JupyterLab 工作區以測試區。這個 Python 庫可以作為命令列實用程式或函數庫，您可以將其匯入到任何 Python 程式或 Jupyter 筆記本中。

NVIDIA Triton 推理伺服器

NVIDIA Triton 推理伺服器是一款開源推理服務軟體，可協助標準化模型部署和執行，以在生產中提供快速且可擴展的 AI。 Triton Inference Server 透過讓團隊能夠在任何基於 GPU 或 CPU 的基礎架構上從任何框架部署、運行和擴展經過訓練的 AI 模型，簡化了 AI 推理。 Triton Inference Server 支援所有主流框架，例如 TensorFlow、 NVIDIA TensorRT、PyTorch、MXNet、OpenVINO 等。 Triton 與 Kubernetes 集成，可進行編排和擴展，您可以在所有主要的公有雲 AI 和 Kubernetes 平台中使用它。它還與許多 MLOps 軟體解決方案整合。

PyTorch

"PyTorch"是一個開源的 ML 框架。它是一個針對使用 GPU 和 CPU 的深度學習而最佳化的張量庫。 PyTorch 套件包含多維張量的資料結構，它提供了許多實用程序，用於高效序列化張量以及其他有用的實用程式。它還有一個 CUDA 對應物，可讓您在具有運算能力的NVIDIA GPU 上執行張量運算。在本次驗證中，我們使用 OpenCV-Python (cv2) 函式庫來驗證我們的模型，同時利用 Python 最直覺的電腦視覺概念。

簡化資料管理

資料管理對於企業 IT 營運和資料科學家至關重要，以便將適當的資源用於 AI 應用程式和訓練 AI/ML 資料集。以下有關NetApp技術的附加資訊超出了本次驗證的範圍，但可能與您的部署相關。

ONTAP資料管理軟體包括以下功能，可簡化操作並降低總營運成本：

- 內聯資料壓縮和擴展重複資料刪除。資料壓縮減少了儲存區塊內部浪費的空間，重複資料刪除顯著增加了有效容量。這適用於本地儲存的資料和分層到雲端的資料。
- 最小、最大和自適應服務品質 (AQoS)。細粒度的服務品質 (QoS) 控制有助於維持高度共享環境中關鍵應用程式的效能水準。
- NetApp FabricPool。提供冷資料到公有和私有雲儲存選項的自動分層，包括 Amazon Web Services (AWS)、Azure 和NetApp StorageGRID儲存解決方案。有關FabricPool的更多信息，請參閱 "TR-4598：FabricPool最佳實踐"。

加速並保護數據

ONTAP提供卓越等級的效能和資料保護，並透過以下方式擴展這些功能：

- 性能和更低的延遲。 ONTAP以盡可能低的延遲提供盡可能高的吞吐量。
- 資料保護。 ONTAP提供內建資料保護功能，並在所有平台上提供通用管理。
- NetApp磁碟區加密 (NVE)。 ONTAP提供原生磁碟區級加密，同時支援板載和外部金鑰管理。
- 多租戶和多因素身份驗證。 ONTAP支援以最高等級的安全性共用基礎架構資源。

面向未來的基礎設施

ONTAP具有以下功能，可協助滿足嚴苛且不斷變化的業務需求：

- 無縫擴展和無中斷運行。 ONTAP支援無中斷地向現有控制器和橫向擴展叢集添加容量。客戶可以升級到最新技術，例如 NVMe 和 32Gb FC，而無需昂貴的資料遷移或中斷。
- 雲端連線。 ONTAP是與雲端連接最緊密的儲存管理軟體，在所有公有雲中均提供軟體定義儲存（ONTAP Select）和Google Cloud NetApp Volumes Volumes ）的選項。
- 與新興應用程式的整合。 ONTAP使用支援現有企業應用的相同基礎架構，為下一代平台和應用（如自動駕駛汽車、智慧城市和工業 4.0）提供企業級資料服務。

NetApp Astra控制

NetApp Astra產品系列由NetApp儲存和資料管理技術提供支持，為本地和公有雲中的 Kubernetes 應用程式提供儲存和應用程式感知資料管理服務。它使您能夠輕鬆備份 Kubernetes 應用程序，將資料遷移到不同的集群，並立即創建可運行的應用程式克隆。如果您需要管理在公有雲中運行的 Kubernetes 應用程序，請參閱 "Astra控制服務"。 Astra Control Service 是一項NetApp託管服務，可為 Google Kubernetes Engine (GKE) 和 Azure Kubernetes Service (AKS) 中的 Kubernetes 叢集提供應用程式感知資料管理。

NetApp Trident

Astra "Trident"NetApp推出的一款適用於 Docker 和 Kubernetes 的開源動態儲存編排器，可簡化持久性儲存的建立、管理和使用。 Trident是一個 Kubernetes 原生應用程序，直接在 Kubernetes 叢集中運作。 Trident讓客戶能夠將 DL 容器映像無縫部署到NetApp儲存體上，並為 AI 容器部署提供企業級體驗。 Kubernetes 使用者（ML 開發人員、資料科學家等）可以建立、管理和自動化編排和克隆，以利用NetApp技術提供支援的高階資料管理功能。

NetApp BlueXP複製與同步

"BlueXP複製和同步"是NetApp 的一項快速、安全的資料同步服務。無論您需要在本機 NFS 或 SMB 檔案共用、NetApp StorageGRID、 NetApp ONTAP S3、 Google Cloud NetApp Volumes、 Azure NetApp Files、Amazon Simple Storage Service (Amazon S3)、Amazon Elastic File System (Amazon EFS)、Azure Blob、Google)傳輸都能快速安全地將文件移動到您需要的位置。資料傳輸完成後，可在來源端和目標端完全使用。 BlueXP Copy 和 Syncc 會根據您預先定義的計劃持續同步數據，僅移動增量，從而最大限度地減少數據複製所花費的時間和金錢。 BlueXP Copy and Sync 是一種軟體即服務 (SaaS) 工具，其設定和使用極為簡單。 BlueXP Copy 和 Sync 觸發的資料傳輸由資料代理執行。您可以在 AWS、Azure、Google Cloud Platform 或本機部署BlueXP Copy 和 Sync 資料代理程式。

NetApp BlueXP分類

在強大的AI演算法驅動下， "NetApp BlueXP分類"為您的整個資料資產提供自動化控制和資料治理。您可以輕鬆找到節省成本的方法、識別合規性和隱私問題並找到最佳化機會。 BlueXP分類儀表板可讓您洞察重複數據以消除冗餘，映射個人、非個人和敏感數據，並針對敏感數據和異常情況發出警報。

測試和驗證計劃

對於此解決方案設計，驗證了以下三個場景：

- 在 JupyterLab 工作區內，使用NetApp DataOps Toolkit for Kubernetes 進行編排的推理任務（有和沒有 Protopia 混淆）。
- 在 Kubernetes 上執行批次推理作業（有和沒有 Protopia 混淆），其資料磁碟區是使用NetApp DataOps Toolkit for Kubernetes 進行編排的。
- 使用NVIDIA Triton 推理伺服器實例的推理任務，該實例是透過使用NetApp DataOps Toolkit for Kubernetes 進行編排的。在呼叫 Triton 推理 API 之前，我們對圖像應用了 Protopia 混淆，以模擬任何透過網路傳輸的資料都必須混淆的常見要求。此工作流程適用於在受信任區域內收集資料但必須傳遞到該受信任區域之外進行推理的用例。如果沒有 Protopia 混淆，就不可能在敏感資料不離開受信任區域的情況下實現這種類型的工作流程。

測試配置

下表概述了解決方案設計驗證環境。

成分	版本
Kubernetes	1.21.6
NetApp Trident CSI 驅動程式	22.01.0
適用於 Kubernetes 的NetApp DataOps 工具包	2.3.0
NVIDIA Triton 推理伺服器	21.11-py3

測試程式

本節描述完成驗證所需的任務。

先決條件

要執行本節中概述的任務，您必須能夠存取安裝並配置了以下工具的 Linux 或 macOS 主機：

- Kubectl（設定用於存取現有的 Kubernetes 叢集）
  - 可以找到安裝和設定說明 "這裡"。
- 適用於 Kubernetes 的NetApp DataOps 工具包
  - 安裝說明可以找到 "這裡"。

場景 1 – JupyterLab 中的按需推理

1. 為 AI/ML 推理工作負載建立 Kubernetes 命名空間。

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

2. 使用NetApp DataOps Toolkit 配置持久卷，用於儲存您將執行推理的資料。

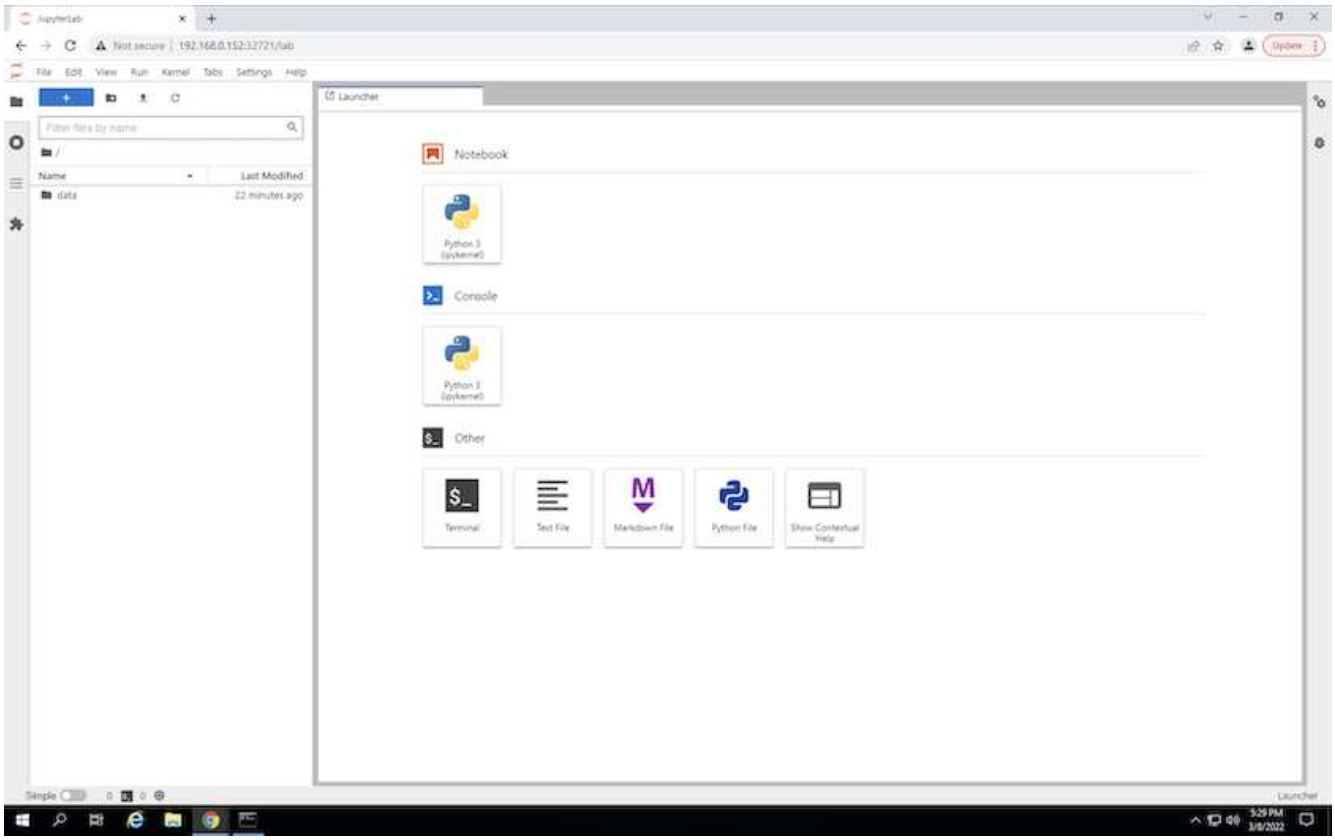
```
$ netapp_dataops_k8s_cli.py create volume --namespace=inference --pvc
-name=inference-data --size=50Gi
Creating PersistentVolumeClaim (PVC) 'inference-data' in namespace
'inference'.
PersistentVolumeClaim (PVC) 'inference-data' created. Waiting for
Kubernetes to bind volume to PVC.
Volume successfully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

3. 使用NetApp DataOps Toolkit 建立新的 JupyterLab 工作區。使用上一步建立的持久性卷 `--mount- pvc`選項。根據需要NVIDIA `-- nvidia-gpu`選項。

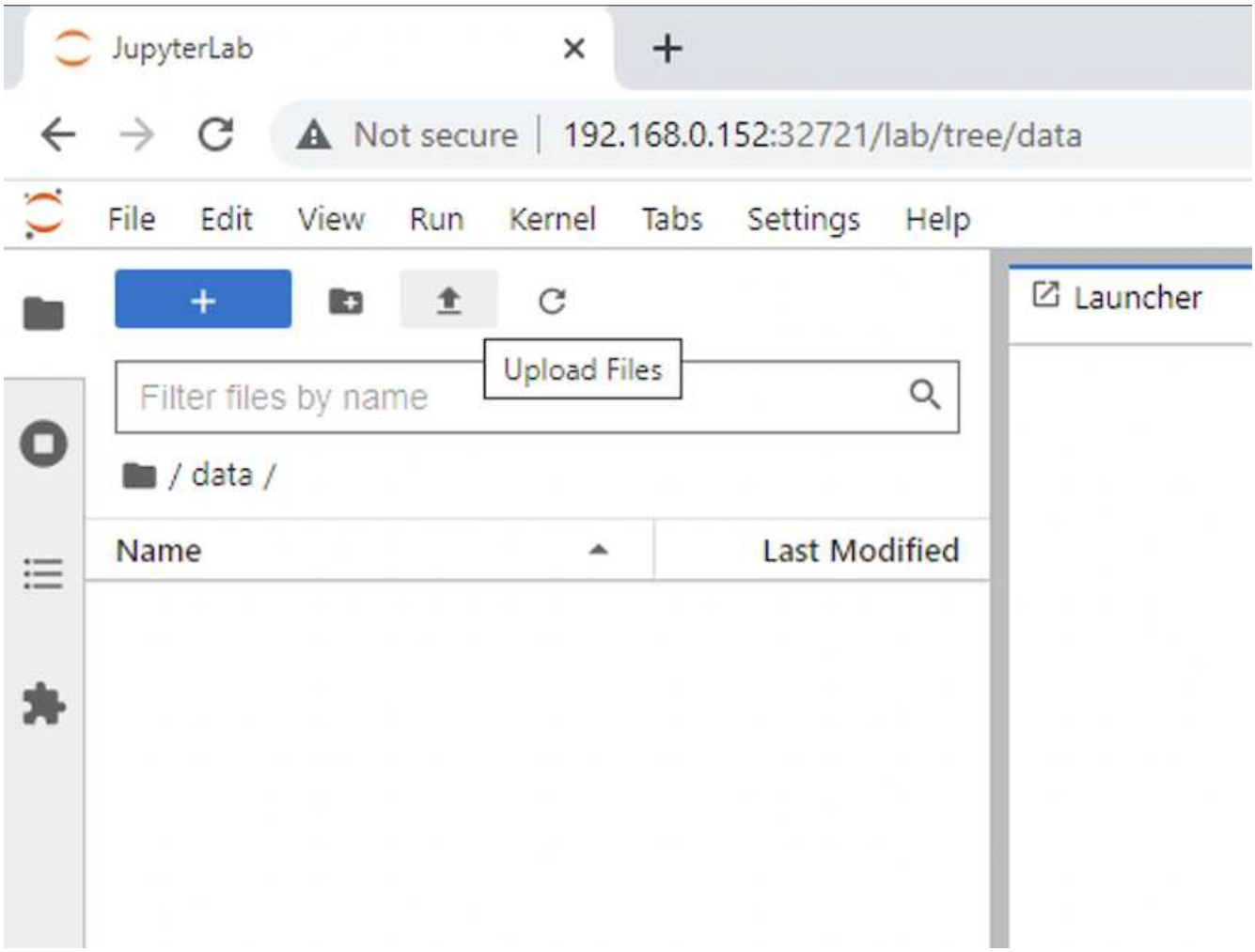
   在以下範例中，持久卷 `inference-data`被掛載到 `JupyterLab` 工作區容器中 `/home/jovyan/data`。使用官方 Project Jupyter 容器鏡像時， `/home/jovyan`在 JupyterLab Web 介面中顯示為頂級目錄。

```
$ 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
```

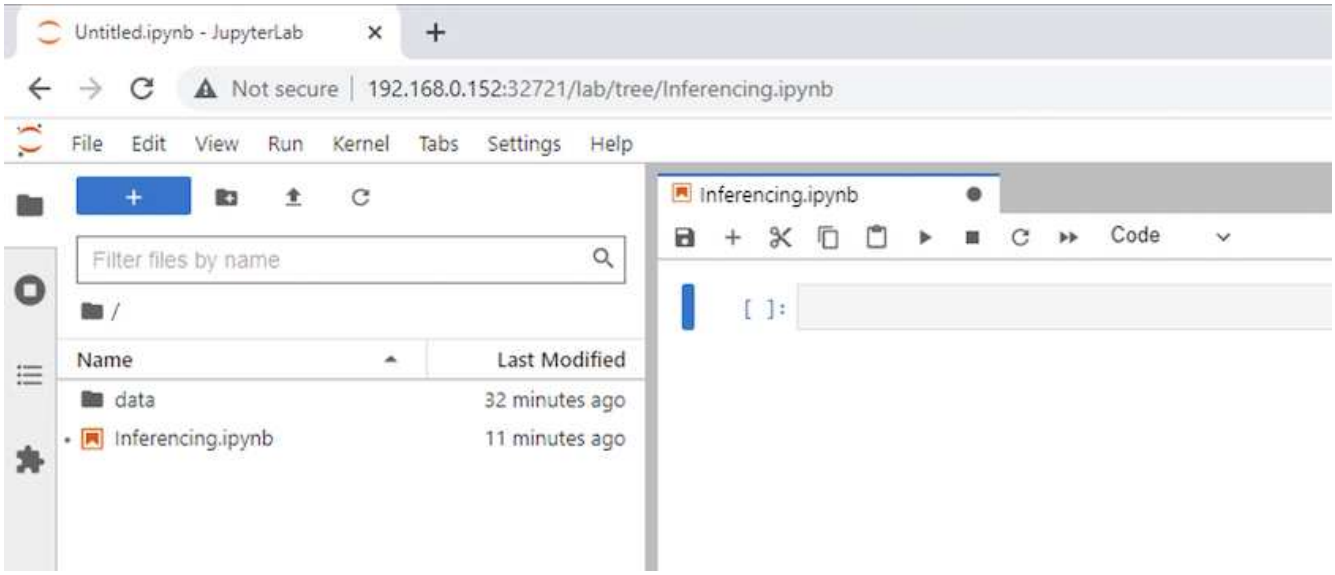
4. 使用輸出中指定的 URL 存取 JupyterLab 工作區 `create jupyterlab`命令。資料目錄代表掛載到工作區的持久卷。



5. 打開 `data`目錄並上傳要執行推理的檔案。當檔案上傳到資料目錄時，它們會自動儲存在掛載到工作區的持久性磁碟區上。要上傳文件，請點擊上傳文件圖標，如下圖所示。



6. 返回頂級目錄並建立一個新的筆記本。



7. 將推理代碼加入筆記本中。以下範例顯示了影像檢測用例的推理程式碼。

Launcher | image-demo-pytorch.ipynb

Code | Python 3 (ipykernel)

STEP 3-1: Clean (Without obfuscation) detection

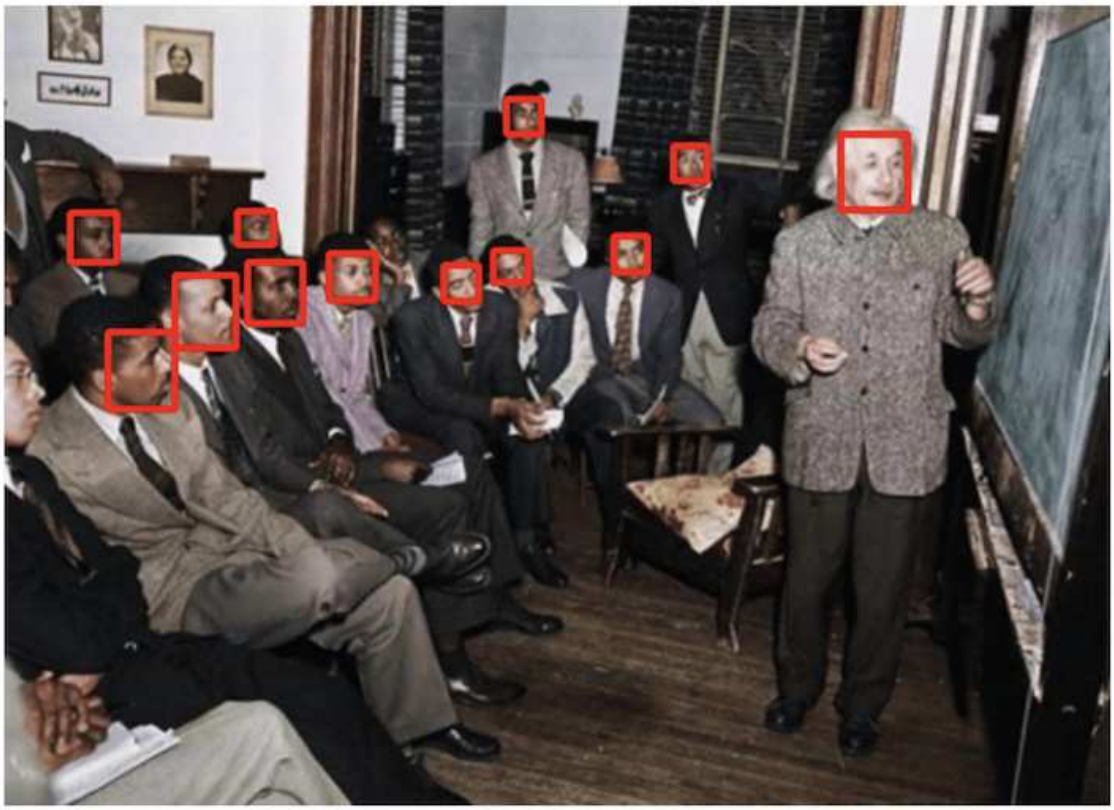
[9]:
```
# get current frame
frame = input_image

# 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. 將 Protopia 混淆加入到您的推理程式碼中。 Protopia 直接與客戶合作提供特定用例的文檔，這超出了本技術報告的範圍。以下範例展示了新增了 Protopia 混淆的影像偵測用例的推理程式碼。

Launcher | image-demo-pytorch.ipynb

Code | Python 3 (ipykernel)

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

```
[11]: # 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

```
show_cv2_image(noisy_reconstruction, scale=2)
```

場景 2 – Kubernetes 上的批次推理

1. 為 AI/ML 推理工作負載建立 Kubernetes 命名空間。

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

2. 使用NetApp DataOps Toolkit 配置持久卷，用於儲存您將執行推理的資料。

```
$ 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. 使用您將執行推理的資料填入新的持久卷。

   有幾種方法可以將資料載入到 PVC 上。如果您的資料目前儲存在與 S3 相容的物件儲存平台（例如NetApp StorageGRID或 Amazon S3）中，那麼您可以使用 "NetApp DataOps Toolkit S3 Data Mover 功能"。另一種簡單的方法是建立一個 JupyterLab 工作區，然後透過 JupyterLab Web 介面上傳文件，如“場景 1 – JupyterLab 中的按需推理 “

4. 為您的批次推理任務建立一個 Kubernetes 作業。以下範例展示了影像偵測用例的批次推理作業。此作業對一組影像中的每個影像執行推理，並將推理準確度指標寫入標準輸出。

```
$ 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. 確認推理作業已成功完成。

```
$ 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. 將 Protopia 混淆加入到您的推理工作中。您可以直接從 Protopia 找到有關新增 Protopia 混淆的用例特定說明，這超出了本技術報告的範圍。以下範例展示了針對人臉偵測用例的批量推理作業，其中新增了 Protopia 混淆，並使用 ALPHA 值 0.8。此作業在對一組影像中的每個影像執行推理之前應用 Protopia 混淆，然後將推理準確度指標寫入標準輸出。

   我們對 ALPHA 值 0.05、0.1、0.2、0.4、0.6、0.8、0.9 和 0.95 重複了此步驟。您可以在"推理準確度比較。"

```
$ 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. 確認推理作業已成功完成。

```
$ 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
```

場景 3 – NVIDIA Triton 推理伺服器

1. 為 AI/ML 推理工作負載建立 Kubernetes 命名空間。

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

2. 使用NetApp DataOps Toolkit 配置持久性卷，用作NVIDIA Triton 推理伺服器的模型儲存庫。

```
$ 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. 將您的模型儲存在新的持久性卷中 "格式"NVIDIA Triton 推理伺服器可以識別它。

   有幾種方法可以將資料載入到 PVC 上。一個簡單的方法是建立一個 JupyterLab 工作區，然後透過 JupyterLab Web 介面上傳文件，如“場景 1 – JupyterLab 中的按需推理 。“

4. 使用NetApp DataOps Toolkit 部署新的NVIDIA Triton Inference Server 實例。

```
$ 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 客戶端 SDK 執行推理任務。以下 Python 程式碼摘錄使用 Triton Python 用戶端 SDK 執行人臉偵測用例的推理任務。此範例呼叫 Triton API 並傳入圖像進行推理。然後，Triton 推理伺服器接收請求，呼叫模型，並將推理輸出作為 API 結果的一部分傳回。

```
# 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. 將 Protopia 混淆加入到您的推理程式碼中。您可以直接從 Protopia 找到有關新增 Protopia 混淆的特定用例說明；但是，此過程超出了本技術報告的範圍。以下範例顯示了與前面步驟 5 中所示的相同的 Python 程式碼，但添加了 Protopia 混淆。

   請注意，在將影像傳遞給 Triton API 之前，會先進行 Protopia 混淆。因此，未混淆的影像永遠不會離開本機。只有經過混淆的圖像才會在網路上傳遞。此工作流程適用於在受信任區域內收集資料但隨後需要傳遞到該受信任區域之外進行推理的用例。如果沒有 Protopia 混淆技術，就不可能實現這種類型的工作流程，因為敏感資料永遠不會離開受信任的區域。

```
# 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)
```

推理精度比較

為了進行此驗證，我們使用一組原始影像對影像檢測用例進行了推理。然後，我們對同一組圖像執行相同的推理任務，並在推理之前添加了 Protopia 混淆。我們使用 Protopia 混淆組件的不同 ALPHA 值重複了這個任務。在 Protopia 混淆的背景下，ALPHA 值表示所應用的混淆量，ALPHA 值越高表示混淆等級越高。然後，我們比較了這些不同運行的推理準確性。

以下兩個表格提供了有關我們的用例的詳細資訊並概述了結果。

Protopia 直接與客戶合作，確定特定用例的適當 ALPHA 值。

成分	細節
模型	FaceBoxes（PyTorch）-
數據集	FDDB資料集

原始托邦的混淆	阿爾法	準確性
不	不適用	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

混淆速度

為了進行此驗證，我們將 Protopia 混淆應用於 1920 x 1080 像素影像五次，並測量每次完成混淆步驟所需的時間。

我們使用在單一NVIDIA V100 GPU 上運行的 PyTorch 來應用混淆，並在運行之間清除了 GPU 快取。在五次運行中，混淆步驟分別花費 5.47ms、5.27ms、4.54ms、5.24ms 和 4.84ms 完成。平均速度為5.072毫秒。

結論

數據有三種狀態：靜止、傳輸、計算。任何人工智慧推理服務的一個重要部分應該是在整個過程中保護資料免受威脅。在推理過程中保護資料至關重要，因為該流程可能會暴露有關外部客戶和提供推理服務的企業的私人資訊。 Protopia AI 是當今市場上用於機密 AI 推理的非侵入式純軟體解決方案。借助 Protopia，AI 僅接收資料記錄中對於執行手頭上的 AI/ML 任務至關重要的轉換訊息，僅此而已。這種隨機變換不是一種掩蔽形式，而是基於透過使用精心策劃的雜訊以數學方式改變資料的表示。

具有ONTAP功能的NetApp儲存系統可提供與本地 SSD 儲存相同或更好的效能，並且與NetApp DataOps Toolkit 結合使用，可為資料科學家、資料工程師、AI/ML 開發人員以及業務或企業 IT 決策者帶來以下優勢：

- 在人工智慧系統、分析系統和其他關鍵業務系統之間輕鬆共享資料。這種資料共享減少了基礎設施開銷，提高了效能，並簡化了整個企業的資料管理。
- 獨立可擴展的計算和存儲，以最大限度地降低成本並提高資源利用率。
- 使用整合的 Snapshot 副本和複製來簡化開發和部署工作流程，以實現即時且節省空間的使用者工作區、整合版本控制和自動部署。
- 針對災難復原、業務連續性和監管要求的企業級資料保護和資料治理。
- 簡化資料管理作業的呼叫；從 Jupyter 筆記本中的NetApp DataOps Toolkit 快速取得資料科學家工作區的 Snapshot 副本以進行備份和追溯。

NetApp和 Protopia 解決方案提供了靈活的橫向擴展架構，非常適合企業級 AI 推理部署。它可以實現資料保護並為敏感資訊提供隱私，其中機密的 AI 推理要求可以透過內部部署和混合雲端部署中的負責任的 AI 實踐來滿

足。

在哪裡可以找到更多資訊和致謝

要了解有關本文檔中描述的信息的更多信息，請參閱以下文檔和/或網站：

- NetApp ONTAP資料管理軟體 — ONTAP資訊庫

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

- NetApp容器持久性儲存 — NetApp Trident

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

- NetApp DataOps 工具包

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

- NetApp容器持久性儲存 — NetApp Trident

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

- Protopia AI—機密推理

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

- NetApp BlueXP複製與同步

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

- NVIDIA Triton 推理伺服器

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

- NVIDIA Triton 推理伺服器文檔

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

- PyTorch 中的 FaceBoxes

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

致謝

- NetApp首席產品經理 Mark Cates
- Sufian Ahmad， NetApp技術行銷工程師
- Protopia AI 技術長兼教授 Hadi Esmaeilzadeh

版權資訊

Copyright © 2026 NetApp, Inc. 版權所有。台灣印製。非經版權所有人事先書面同意，不得將本受版權保護文件的任何部分以任何形式或任何方法（圖形、電子或機械）重製，包括影印、錄影、錄音或儲存至電子檢索系統中。

由 NetApp 版權資料衍伸之軟體必須遵守下列授權和免責聲明：

此軟體以 NETAPP「原樣」提供，不含任何明示或暗示的擔保，包括但不限於有關適售性或特定目的適用性之擔保，特此聲明。於任何情況下，就任何已造成或基於任何理論上責任之直接性、間接性、附隨性、特殊性、懲罰性或衍生性損害（包括但不限於替代商品或服務之採購；使用、資料或利潤上的損失；或企業營運中斷），無論是在使用此軟體時以任何方式所產生的契約、嚴格責任或侵權行為（包括疏忽或其他）等方面，NetApp 概不負責，即使已被告知有前述損害存在之可能性亦然。

NetApp 保留隨時變更本文所述之任何產品的權利，恕不另行通知。NetApp 不承擔因使用本文所述之產品而產生的責任或義務，除非明確經過 NetApp 書面同意。使用或購買此產品並不會在依據任何專利權、商標權或任何其他 NetApp 智慧財產權的情況下轉讓授權。

本手冊所述之產品受到一項（含）以上的美國專利、國外專利或申請中專利所保障。

有限權利說明：政府機關的使用、複製或公開揭露須受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中的「技術資料權利 - 非商業項目」條款 (b)(3) 小段所述之限制。

此處所含屬於商業產品和 / 或商業服務（如 FAR 2.101 所定義）的資料均為 NetApp, Inc. 所有。根據本協議提供的所有 NetApp 技術資料和電腦軟體皆屬於商業性質，並且完全由私人出資開發。 美國政府對於該資料具有非專屬、非轉讓、非轉授權、全球性、有限且不可撤銷的使用權限，僅限於美國政府為傳輸此資料所訂合約所允許之範圍，並基於履行該合約之目的方可使用。除非本文另有規定，否則未經 NetApp Inc. 事前書面許可，不得逕行使用、揭露、重製、修改、履行或展示該資料。美國政府授予國防部之許可權利，僅適用於 DFARS 條款 252.227-7015(b)（2014 年 2 月）所述權利。

商標資訊

NETAPP、NETAPP 標誌及 http://www.netapp.com/TM 所列之標章均為 NetApp, Inc. 的商標。文中所涉及的所有其他公司或產品名稱，均為其各自所有者的商標，不得侵犯。