



Nvidia GPU 세팅

| | |
|--------------|---------------|
| Category | gpu |
| Status | Done |
| Last Updated | @May 31, 2025 |

Ubuntu 24.04 및 RTX 6000 Ada 환경에서 NVIDIA 드라이버, CUDA 12.9 및 AI 환경 설정 종합 가이드

이 보고서는 NVIDIA RTX 6000 Ada Generation 그래픽 카드가 장착된 Ubuntu 24.04 워크스테이션에 NVIDIA 드라이버 및 CUDA Toolkit 12.9를 설치하는 체계적인 가이드를 제공합니다. 또한 GPU 가속 워크로드를 위한 Docker 구성 방법을 자세히 설명하고 NVIDIA Inference Microservices (NIM)를 소개합니다. 이 설정은 고급 AI 코딩 및 개발의 기초가 됩니다.

I. 서론

인공 지능(AI) 및 머신 러닝(ML) 개발 환경은 강력한 하드웨어와 세심하게 구성된 소프트웨어 스택을 필요로 합니다. NVIDIA RTX 6000 Ada Generation 그래픽 카드는 48GB GDDR6 ECC 메모리와 18,176개의 CUDA 코어를 통해 AI, 렌더링 및 고성능 컴퓨팅 작업에 탁월한 성능을 제공합니다. Ubuntu 24.04 LTS는 이러한 까다로운 워크로드를 위한 안정적이고 널리 채택된 플랫폼 역할을 합니다. 이 가이드는 NVIDIA 드라이버, CUDA Toolkit 버전 12.9 및 필요한 cuDNN 라이브러리를 올바르게 설치하여 안정적인 AI 개발 환경을 구축하는 데 중점을 둡니다. 또한 Docker 컨테이너 내에서 GPU 가속을 활용하는 점점 더 일반적인 요구 사항을 해결하고 NVIDIA NIM을 사용하여 모델을 배포하는 초기 단계를 제공합니다.

여기에 설명된 절차는 호환성을 보장하고 최신 안정 소프트웨어 버전에 대한 액세스를 위해 공식 NVIDIA 리포지토리를 사용하는 것을 우선시합니다. 각 주요 설치 단계에는 각 구성 요소의 성공적인 설정을 확인하기 위한 검증 절차가 함께 제공됩니다.

II. 시스템 요구 사항 및 초기 점검

설치 프로세스를 시작하기 전에 시스템이 필요한 하드웨어 및 소프트웨어 요구 사항을 충족하고 기존 환경이 준비되었는지 확인하는 것이 중요합니다.

A. 하드웨어 요구 사항

- **GPU:** NVIDIA RTX 6000 Ada Generation. 이 카드는 NVIDIA Ada Lovelace 아키텍처, ECC 기능이 있는 48GB GDDR6 메모리, 4세대 Tensor 코어 568개, 3세대 RT 코어 142개, CUDA 코어 18,176개를 갖추고 있으며 최대 전력 소비량은 300W입니다.
- **CPU:** Intel Core i5/i7/i9/Xeon 또는 AMD Ryzen/Epyc 클래스 프로세서.
- **시스템 메모리:** GPU 메모리 이상; GPU 메모리의 두 배 권장 (즉, >= 48GB, 96GB 권장).
- **확장 슬롯:** PCI Express 4.0 x16 (선택).
- **전원 공급 장치:** RTX 6000 Ada의 300W TDP 및 기타 시스템 구성 요소를 지원하기에 충분한 전력량. 이 카드는 1x PCIe CEM5 16핀 전원 커넥터를 사용합니다.

B. 소프트웨어 요구 사항

- **운영 체제:** Ubuntu 24.04 LTS (Noble Numbat) x86_64. NVIDIA RTX 6000 Ada는 Ubuntu 24.04 LTS에서 테스트되었습니다. CUDA 12.9도 Ubuntu 24.04를 지원합니다.

- **커널 헤더:** 드라이버 설치에 필요합니다. 현재 실행 중인 커널과 일치해야 합니다.
- **gcc 컴파일러:** CUDA 샘플 및 애플리케이션 컴파일에 필요합니다. Ubuntu 24.04에는 일반적으로 호환되는 버전이 포함되어 있습니다.
- **Secure Boot:** 시스템의 UEFI/BIOS에서 Secure Boot가 활성화된 경우 드라이버 설치 중에 추가 단계(MOK 등록)가 필요합니다. 설치 프로세스를 위해 Secure Boot를 일시적으로 비활성화하는 것이 더 간단한 경우가 많지만, 일부 가이드에서는 키 등록 방법을 제안합니다. 이 가이드는 Secure Boot가 활성화되어 있을 수 있다고 가정하며 MOK 등록이 요청될 수 있는 시점을 명시합니다.

C. 설치 전 시스템 점검 및 업데이트

1. GPU 감지 확인:

시스템이 NVIDIA RTX 6000 Ada를 인식하는지 확인합니다.Bash

```
lspci | grep -i nvidia
```

출력에 NVIDIA 그래픽 카드가 표시되어야 합니다. 그렇지 않은 경우 물리적 설치 및 BIOS 설정을 확인하십시오. RTX 6000 Ada의 장치 ID가 출력에 나타날 수 있습니다.

2. 시스템 패키지 업데이트:

최신 상태의 시스템으로 시작하는 것이 좋습니다.Bash

```
sudo apt update
sudo apt upgrade -y
```

3. 커널 헤더 및 개발 도구 설치:

이는 드라이버 설치 중 커널 모듈 빌드에 필수적입니다.Bash

```
sudo apt install linux-headers-$(uname -r) build-essential
```

`build-essential` 패키지에는 `gcc`, `g++`, `make` 및 기타 필요한 도구가 포함되어 있습니다.

4. 충돌하는 드라이버 처리 (Nouveau):

오픈 소스 Nouveau 드라이버는 NVIDIA의 독점 드라이버와 충돌할 수 있습니다. 블랙리스트에 추가하는 것이 종종 권장됩니다. 최신 NVIDIA 설치 프로그램이 이를 처리하지만 명시적인 블랙리스트 추가는 문제를 예방할 수 있습니다.

`/etc/modprobe.d/blacklist-nouveau.conf` 파일을 만듭니다: Bash

```
sudo nano /etc/modprobe.d/blacklist-nouveau.conf
```

다음 줄을 추가합니다:

```
blacklist nouveau
options nouveau modeset=0
```

파일을 저장한 다음 initramfs를 업데이트하고 재부팅합니다:

```
sudo update-initramfs -u
sudo reboot
```

재부팅 후 드라이버 설치를 진행합니다. 이 단계는 NVIDIA 독점 드라이버를 위한 더 깨끗한 환경을 보장하여 설치 실패 또는 불안정한 그래픽 환경으로 이어질 수 있는 잠재적인 충돌을 최소화하는 데 도움이 됩니다.

III. Ubuntu 24.04에서 RTX 6000 Ada용 NVIDIA 드라이버 설치

올바른 NVIDIA 드라이버를 설치하는 것은 AI 워크로드를 위해 RTX 6000 Ada를 활용하기 위한 기본 단계입니다. 권장되는 방법은 공식 NVIDIA CUDA 네트워크 리포지토리를 사용하는 것입니다. 이는 드라이버와 CUDA Toolkit 간의 최적의 호환성을 보장하기 때문입니다. 이 접근 방식은 NVIDIA가 소프트웨어 스택을 위해 이러한 리포지토리를 관리하므로 일반적인 Ubuntu 드라이버나 타사 PPA를 사용하는 것보다 전문 AI/ML 개발에 일반적으로 더 강력합니다.

A. NVIDIA CUDA 네트워크 리포지토리 구성

NVIDIA CUDA 리포지토리는 드라이버, CUDA Toolkit 및 기타 관련 소프트웨어에 대한 액세스를 제공합니다. Ubuntu 24.04의 경우 리포지토리를 다음과 같이 구성해야 합니다. 이 명령은 네트워크 리포지토리 설정에 대한 NVIDIA 문서에 제공된 구조를 기반으로 하며 Ubuntu 24.04에 맞게 조정되었으며, 전용 패키지 목록이 있는 것으로 확인되었습니다.

1. **NVIDIA CUDA 리포지토리 키 및 리포지토리 소스 추가:** `cuda-keyring` 패키지는 NVIDIA 패키지 서명에 사용되는 GPG 키를 설치하여 패키지의 신뢰성을 보장합니다.

Bash

이 시퀀스는 먼저 키링 패키지를 다운로드하고

`dpkg` 를 사용하여 설치한 다음(NVIDIA GPG 키 등록) 새로 추가된 NVIDIA 리포지토리의 패키지를 포함하도록 로컬

`apt` 패키지 목록을 업데이트합니다.

```
# cuda-keyring 패키지 다운로드 및 설치
```

```
# 참고: 정확한 URL은 변경될 수 있습니다. 항상 Ubuntu 24.04용 최신 NVIDIA CUDA 다운로드 페이지를 참조하십시오.
```

```
# 다음 구조가 일반적입니다:
```

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/x86_64/cuda-keyring_1.1-1_all.deb
```

```
sudo dpkg -i cuda-keyring_1.1-1_all.deb
```

```
sudo apt-get update
```

```
# 다운로드한 deb 파일 정리
```

```
rm cuda-keyring_1.1-1_all.deb
```

B. NVIDIA 드라이버 설치

리포지토리가 구성되면 NVIDIA 드라이버를 설치할 수 있습니다.

1. **`cuda-drivers` 메타 패키지 설치:**

이 메타 패키지는 리포지토리에서 사용 가능한 CUDA 브랜치와 호환되는 최신 적절한 드라이버 시리즈를 설치하도록 설계되었습니다. Bash

```
sudo apt-get install cuda-drivers
```

NVIDIA NIM 문서는 최적의 성능을 위해 드라이버 버전 550 이상을 권장합니다. CUDA 12.x 리포지토리 브랜치의

`cuda-drivers` 패키지는 적합한 버전을 제공해야 합니다. NVIDIA의 오픈 커널 모듈을 사용하는 드라이버 버전 560.35.03으로 RTX 6000 Ada를 사용하여 Ubuntu 24.04에서 테스트한 내용이 언급되었습니다. `cuda-drivers` 패키지는 종종 설치 중에 옵션을 제공하거나 감지된 하드웨어에 대해 독점 또는 오픈 커널 모듈을 모두 사용할 수 있는 경우 둘 중 하나를 사용하도록 구성할 수 있습니다.

2. **시스템 재부팅:**

새 커널 모듈이 로드되고 드라이버가 활성화되려면 재부팅이 필요합니다. Bash

```
sudo reboot
```

Secure Boot 참고: Secure Boot가 활성화된 경우 시스템은 재부팅 프로세스 중에 MOK(Machine Owner Key) 등록을 요청할 가능성이 높습니다. 화면의 지시에 따라 NVIDIA 드라이버용 키를 등록하십시오. 여기에는 일반적으로 "Enroll MOK"를 선택하고 확인한 다음 드라이버 사전 설치 단계(DKMS에서 요청한 경우)에서 설정한 암호를 입력하는 과정이 포함됩니다. MOK를 등록하지 않으면 NVIDIA 드라이버가 로드되지 않습니다.

C. 드라이버 설치 확인

시스템이 재부팅된 후 NVIDIA 드라이버가 올바르게 설치되었고 RTX 6000 Ada가 인식되는지 확인합니다.

1. **`nvidia-smi` 실행:**

이 명령줄 유틸리티는 설치된 NVIDIA 드라이버 및 GPU에 대한 정보를 제공합니다.

Bash

성공적인 출력은 다음을 표시합니다:

```
nvidia-smi
```

- **드라이버 버전:** 설치된 NVIDIA 드라이버의 버전입니다.
- **CUDA 버전:** 설치된 드라이버에서 지원하는 *최대 CUDA 버전*을 나타냅니다. 이는 아직 시스템에 설치된 CUDA Toolkit의 버전이 아니라 드라이버의 호환성 수준이라는 점을 이해하는 것이 중요합니다. 실제 CUDA Toolkit은 별도의 설치입니다. 사용자가 이 값을 활성 툴킷 버전으로 잘못 해석하는 경우가 많으므로 이 구분이 중요합니다.
- **GPU 이름:** "NVIDIA RTX 6000 Ada Generation"이 표시되어야 합니다.
- **GPU 메모리:** 총 메모리, 할당된 메모리 및 사용 가능한 메모리 (예: RTX 6000 Ada의 경우 총 48GB).
- GPU 사용률, 온도 및 실행 중인 프로세스(있는 경우)와 같은 기타 세부 정보.

다음 표는 드라이버 설치 명령을 요약한 것입니다:

| 단계 | 명령 | 예상 결과/참고 |
|---------------------------|---|---|
| 1. NVIDIA 리포지토리 키 및 소스 추가 | <code>wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/x86_64/cuda-keyring_1.1-1_all.deb</code> <code>sudo dpkg -i cuda-keyring_1.1-1_all.deb</code> <code>sudo apt-get update</code> <code>rm cuda-keyring_1.1-1_all.deb</code> | <code>apt</code> 소스가 NVIDIA 리포지토리로 업데이트되었습니다. |
| 2. NVIDIA 드라이버 설치 | <code>sudo apt-get install cuda-drivers</code> | 리포지토리에서 최신 호환 NVIDIA 드라이버가 설치되었습니다. |
| 3. 시스템 재부팅 | <code>sudo reboot</code> | 시스템이 재부팅됩니다. Secure Boot가 활성화된 경우 MOK 등록이 필요할 수 있습니다. |
| 4. 드라이버 설치 확인 | <code>nvidia-smi</code> | 드라이버 버전, CUDA 호환성 버전 및 RTX 6000 Ada GPU 세부 정보를 표시합니다. 드라이버가 로드되어 작동하는지 확인합니다. |

Export to Sheets

`nvidia-smi` 가 실패하거나 GPU를 표시하지 않으면 문제 해결 섹션을 참조하십시오.

IV. Ubuntu 24.04에 CUDA Toolkit 12.9 설치

NVIDIA 드라이버가 올바르게 설치되고 확인되면 다음 단계는 CUDA Toolkit 버전 12.9를 설치하는 것입니다. CUDA Toolkit은 컴파일러, 라이브러리 및 도구를 포함하여 GPU 가속 애플리케이션을 만들기 위한 개발 환경을 제공합니다. 이전 섹션에서 구성한 NVIDIA 네트워크 리포지토리가 이 설치에 사용되며, Ubuntu 24.04 및 CUDA 12.9용 패키지를 특별히 제공합니다.

A. CUDA Toolkit 12.9 설치

1. CUDA Toolkit 패키지 설치:

개발 환경의 예측 가능성을 보장하기 위해 특정 버전의 CUDA Toolkit을 설치하는 것이 좋습니다.

`cuda-toolkit-12-9` 패키지(또는 `cuda-12-9` 와 유사한 이름의 메타 패키지)는 12.9 버전을 대상으로 합니다. 명시적인 버전을 사용하는 것이 일반적인 `sudo apt-get install cuda` 보다 선호됩니다. 후자는 리포지토리 기본값이 변경될 경우 다른 버전을 설치할 수 있기 때문입니다.

Bash

이 명령은 컴파일러(

`nvcc`), 라이브러리 및 헤더를 포함한 CUDA Toolkit 12.9를 다운로드하고 설치합니다. Ubuntu 24.04 x86_64용 NVIDIA 리포지토리에는 CUDA 12.9용 특정 패키지가 나열되어 있어 이 방법을 통해 사용 가능함을 확인합니다. Ubuntu 24.04 기반 컨테이너에 NVIDIA CUDA 12.9.0이 존재한다는 점도 간접적인 지원을 나타냅니다.

```
sudo apt-get install cuda-toolkit-12-9
```

B. 환경 변수 설정 (설치 후 필수)

시스템 및 개발 도구가 CUDA Toolkit 바이너리 및 라이브러리를 찾을 수 있도록 `PATH` 및 `LD_LIBRARY_PATH` 환경 변수를 업데이트해야 합니다. 이러한 변경 사항은 일반적으로 지속성을 위해 사용자의 `~/.bashrc` 파일에 추가됩니다.

1. `~/.bashrc` 에 CUDA 경로 추가:Bash

```
echo 'export PATH=/usr/local/cuda-12.9/bin${PATH:+:${PATH}}' >> ~/.bashrc
echo 'export LD_LIBRARY_PATH=/usr/local/cuda-12.9/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}' >> ~/.bashrc
```

2. 현재 셸 세션에 변경 사항 적용:Bash

```
source ~/.bashrc
```

3. 환경 변수 확인:

경로가 올바르게 추가되었는지 확인합니다:

Bash

출력에는

`PATH` 에 `/usr/local/cuda-12.9/bin` 이, `LD_LIBRARY_PATH` 에 `/usr/local/cuda-12.9/lib64` 가 표시되어야 합니다.

```
echo $PATH
echo $LD_LIBRARY_PATH
```

C. CUDA Toolkit 설치 확인

철저한 확인을 통해 툴킷이 올바르게 설치되었고 GPU와 통신할 수 있는지 확인합니다.

1. `nvcc` 버전 확인:

NVIDIA CUDA 컴파일러(`nvcc`)는 툴킷의 핵심 구성 요소입니다. 해당 버전은 설치된 툴킷과 일치해야 합니다.

Bash

```
nvcc --version
```

출력에는 "Cuda compilation tools, release 12.9"를 포함한 `nvcc` 에 대한 정보가 표시되어야 합니다.

2. CUDA 샘플 컴파일 및 실행: <이 부분은 실패>

NVIDIA는 툴킷의 기능을 테스트하기 위한 CUDA 샘플을 제공합니다. 이러한 샘플은 이제 GitHub에서 호스팅되며 일반적으로 툴킷 패키지와 함께 기본적으로 설치되지 않습니다.

- **CUDA 샘플 리포지토리 복제:** Bash `git` 이 설치되어 있지 않으면 `sudo apt install git` 으로 설치합니다.

```
git clone https://github.com/NVIDIA/cuda-samples.git
cd cuda-samples
```

- **`deviceQuery` 빌드 및 실행:** `deviceQuery` 샘플은 CUDA 지원 장치와 해당 속성을 열거합니다.

Bash

```
cd Samples/1_Uutilities/deviceQuery
make
```

`./deviceQuery`

...

성공적으로 실행되면 NVIDIA RTX 6000 Ada에 대한 자세한 정보가 표시되고

`Result = PASS` 로 끝납니다. 이는 CUDA 런타임이 GPU를 올바르게 식별하고 상호 작용할 수 있음을 확인합니다. 샘플이 GitHub로 마이그레이션됨에 따라 개발자는 이제 이러한 예제를 적극적으로 가져와야 하며, 이는 샘플이 종종 로컬 설치 경로에 포함되었던 이전 CUDA Toolkit 배포판과 달라진 점입니다.

다음 표는 CUDA Toolkit 12.9 설치 및 확인 단계를 간략하게 설명합니다:

| 단계 | 명령 | 목적/예상 결과 |
|---|--|--|
| 1. CUDA Toolkit 12.9 설치 | <code>sudo apt-get install cuda-toolkit-12-9</code> | CUDA Toolkit 12.9 (컴파일러, 라이브러리, 헤더) 설치됨. |
| 2. 환경 변수 설정 | <code>echo 'export PATH=/usr/local/cuda-12.9/bin\${PATH:+:\${PATH}}' >> ~/.bashrc</code> <code>echo 'export LD_LIBRARY_PATH=/usr/local/cuda-12.9/lib64\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}' >> ~/.bashrc</code> <code>source ~/.bashrc</code> | <code>PATH</code> 및 <code>LD_LIBRARY_PATH</code> 가 CUDA 12.9 디렉토리를 포함하도록 업데이트됨. |
| 3. <code>nvcc</code> 버전 확인 | <code>nvcc --version</code> | CUDA Toolkit 버전 표시, 12.9인지 확인. |
| 4. <code>deviceQuery</code> 샘플 컴파일 및 실행 | <code>git clone https://github.com/NVIDIA/cuda-samples.git</code> <code>cd cuda-samples/Samples/1_Uutilities/deviceQuery</code> <code>make</code> <code>./deviceQuery</code> | <code>deviceQuery</code> 가 성공적으로 컴파일 및 실행되어 RTX 6000 Ada 세부 정보와 <code>Result = PASS</code> 를 표시함. 툴킷 기능 및 GPU 통신 확인. |

Export to Sheets

V. CUDA 12.9용 cuDNN 설치

NVIDIA CUDA Deep Neural Network 라이브러리(cuDNN)는 심층 신경망을 위한 GPU 가속 기본 요소 라이브러리입니다. 딥 러닝 프레임워크에서 고성능을 달성하는 데 필수적입니다. Ubuntu 24.04의 CUDA 12.9용 cuDNN은 드라이버 및

CUDA Toolkit에 사용된 것과 동일한 NVIDIA 네트워크 리포지토리에서 설치해야 합니다. 이 방법은 호환성을 보장하고 수동 tarball 방법에 비해 설치를 단순화합니다.

A. cuDNN 라이브러리 설치

Ubuntu 24.04 x86_64용 NVIDIA 리포지토리에는 CUDA 12.9용 특정 빌드를 포함하여 CUDA 12.x와 호환되는 cuDNN 9 패키지가 포함되어 있습니다. 설치에는 런타임 및 개발자 패키지를 가져오는 과정이 포함됩니다.

1. cuDNN 런타임 및 개발자 라이브러리 설치:

cuDNN의 패키지 이름은 특정할 수 있습니다. 의 목록(예:

`libcudnn9-cuda-12.9.10.1.4-1_amd64.deb`, `libcudnn9-cuda-12.9.10.1.4-1_amd64.deb`, `libcudnn9-dev-cuda-12.9.10.1.4-1_amd64.deb`) 및 일반적인 cuDNN 설치 패턴을 기반으로 다음 명령은 CUDA 12.x용 cuDNN 9를 대상으로 합니다. 설치된 CUDA 버전에 해당하는 메타 패키지를 사용하는 것이 가장 좋습니다. CUDA 12.x의 경우 일반적으로 `sudo apt-get -y install cudnn9-cuda-12` 명령이 제안됩니다. CUDA 12.9가 설치되어 있으므로 `apt` 는 올바른 종속성을 해결해야 합니다. `cudnn9-cuda-12-9` 와 같은 더 구체적인 메타 패키지를 사용할 수 있고 NVIDIA에서 이 목적을 위해 명확하게 문서화한 경우 더욱 정확할 것입니다.

Bash

cuDNN 설치에

`apt` 를 사용하는 것은 수동 방법보다 강력하게 권장됩니다. `apt` 는 파일 배치, 권한 및 종속성을 올바르게 처리하여 더 강력하고 유지 관리 가능한 시스템을 만들기 때문입니다. 사용자가 CUDA 12.8용 cuDNN을 설치하려고 할 때 종속성으로 인해 12.9 패키지가 선호되었던 예서 암시된 버전 불일치와 같은 잠재적인 문제는 설치된 CUDA 12.9에 대한 올바른 메타 패키지를 대상으로 하여 완화됩니다.

```
# 최신 cuDNN 버전을 볼 수 있도록 패키지 목록 업데이트
sudo apt-get update
```

```
# CUDA 12.x용 cuDNN 9 런타임 라이브러리 설치
sudo apt-get install libcudnn9-cuda-12
```

```
# CUDA 12.x용 cuDNN 9 개발자 라이브러리 설치 (헤더 포함)
sudo apt-get install libcudnn9-dev-cuda-12
```

B. cuDNN 설치 확인

확인을 통해 cuDNN 라이브러리가 올바르게 설치되고 액세스 가능한지 확인합니다.

1. 설치된 패키지 및 라이브러리/헤더 파일 확인:

- 설치된 cuDNN 패키지 나열:

Bash

`libcudnn9` 및 `libcudnn9-dev` 와 같은 패키지가 표시되어야 합니다.

```
dpkg -l | grep libcudnn
```

- 헤더 파일 존재 및 버전 확인 (경로는 패키지 세부 정보에 따라 약간 다를 수 있지만 일반적으로 표준 포함 디렉터리에 있음):

기본 헤더

`cuda_version.h` 에는 버전 매크로가 포함되어 있습니다. `apt` 설치 후 일반적인 위치는 `/usr/include/x86_64-linux-gnu/` 내입니다.

Bash

이는 cuDNN 애플리케이션 컴파일에 필요한 헤더 파일이 있는지 확인하고 설치된 cuDNN 버전을 검사할 수 있도록 합니다.

```
# 버전 헤더 확인
ls -l /usr/include/x86_64-linux-gnu/cudnn_version.h
# CUDNN_MAJOR, CUDNN_MINOR, CUDNN_PATCHLEVEL 정의를 보려면 내용 표시
cat /usr/include/x86_64-linux-gnu/cudnn_version.h
```

2. cuDNN 샘플 컴파일 및 실행 (mnistCUDNN):<이 부분 실패>

NVIDIA는 리포지토리의 패키지를 통해 설치할 수 있는 cuDNN 샘플을 제공합니다.

- cuDNN 샘플 설치: `libcudnn9-samples` 패키지를 사용할 수 있습니다.Bash

```
sudo apt-get install libcudnn9-samples
```

- **mnistCUDNN 컴파일 및 실행:**

샘플은 일반적으로

`/usr/src/cudnn_samples_v9/` 또는 유사한 위치에 설치됩니다. 이전 가이드에서는 때때로 `$HOME/cudnn_samples_v9/` 를 참조합니다. 패키지 버전의 위치를 사용해야 합니다.

Bash

```
# mnistCUDNN 샘플 디렉터리로 이동
# 정확한 경로는 패키지 버전에 따라 다를 수 있으므로 /usr/src/ 확인
cd /usr/src/cudnn_samples_v9/mnistCUDNN

# 샘플 컴파일
sudo make clean && sudo make
# (빌드 아티팩트에 대해 /usr/src가 사용자 쓰기 가능하지 않은 경우 sudo가 필요할 수 있음)
# 또는 샘플 디렉터리를 사용자 쓰기 가능한 위치로 먼저 복사:
# cp -r /usr/src/cudnn_samples_v9 ~/my_cudnn_samples
# cd ~/my_cudnn_samples/mnistCUDNN
# make clean && make
```

샘플 실행

`./mnistCUDNN`

...

성공적으로 실행되면 일반적으로

Test passed! 가 인쇄됩니다. 이는 cuDNN이 설치된 CUDA Toolkit 및 GPU와 올바르게 작동하는지 종합적으로 확인합니다.

다음 표는 네트워크 리포지토리를 사용한 cuDNN 설치를 요약한 것입니다:

| 단계 | 명령 | 목적 |
|------------------------|---|--|
| 1. 패키지 목록 업데이트 | <code>sudo apt-get update</code> | 로컬 패키지 인덱스 새로 고침. |
| 2. cuDNN 런타임 설치 | <code>sudo apt-get install libcudnn9-cuda-12</code> | CUDA 12.x와 호환되는 cuDNN 9 런타임 라이브러리를 설치합니다. |
| 3. cuDNN 개발자 패키지 설치 | <code>sudo apt-get install libcudnn9-dev-cuda-12</code> | CUDA 12.x용 cuDNN 9 헤더 파일 및 개발 라이브러리를 설치합니다. |
| 4. (선택 사항) cuDNN 샘플 설치 | <code>sudo apt-get install libcudnn9-samples</code> | cuDNN 샘플 코드를 설치합니다. |
| 5. 설치 확인 | <code>`dpkg -I \</code> | <code>grep libcudnn cat /usr/include/x86_64-linux-gnu/cudnn_version.h (샘플이 설치된 경우) cd /usr/src/cudnn_samples_v9/mnistCUDNN && make && ./mnistCUDNN`</code> |

Export to Sheets

VI. GPU 가속을 위한 Docker 설정

Docker 컨테이너는 휴대 가능하고 재현 가능한 환경을 만들기 위해 AI 개발에 널리 사용됩니다. Docker 컨테이너가 NVIDIA RTX 6000 Ada GPU에 액세스할 수 있도록 하려면 NVIDIA Container Toolkit을 설치하고 Docker를 올바르게 구성해야 합니다. 이 툴킷은 이전의 `nvidia-docker2` 를 대체하고 보다 모듈화된 접근 방식을 제공합니다.

A. Docker Engine 설치

Docker가 아직 설치되어 있지 않으면 공식 Docker 설명서를 따르십시오. Ubuntu에서의 일반적인 설치와 같습니다:

1. **Docker의 `apt` 리포지토리 설정 (공식 Docker CE에 대해 아직 수행하지 않은 경우):**

가장 최신 명령은

<https://docs.docker.com/engine/install/ubuntu/>를 참조하십시오. 여기에는 일반적으로 Docker의 GPG 키와 리포지토리를 추가하는 과정이 포함됩니다.

2. **Docker Engine 설치:**Bash

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

3. 루트가 아닌 사용자로 Docker 관리 (권장):Bash

그런 다음 로그아웃했다가 다시 로그인하거나 현재 터미널 세션에서

`newgrp docker` 를 실행하여 그룹 변경 사항을 적용합니다. 이렇게 하면 `sudo` 없이 `docker` 명령을 실행할 수 있습니다.

```
sudo groupadd docker # 그룹이 아직 존재하지 않는 경우
sudo usermod -aG docker $USER
```

B. NVIDIA Container Toolkit 설치

NVIDIA Container Toolkit은 Docker가 NVIDIA GPU와 상호 작용하는 데 필요한 구성 요소를 제공합니다.

1. NVIDIA Container Toolkit 리포지토리 구성:

다음 명령은 NVIDIA Container Toolkit용 GPG 키와 리포지토리를 추가합니다.Bash

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \
&& curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

2. 툴킷 패키지 설치:Bash

```
sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit
```

C. NVIDIA 런타임을 사용하도록 Docker 구성

Docker 데몬은 NVIDIA 런타임을 인식하도록 구성해야 합니다.

1. `nvidia-ctk` 를 사용하여 Docker 구성: `nvidia-ctk` (NVIDIA Container Toolkit CLI) 유틸리티는 Docker의 `daemon.json` 파일을 수정합니다.

Bash

```
sudo nvidia-ctk runtime configure --runtime=docker
```

2. Docker 데몬 다시 시작:

Docker 서비스를 다시 시작하여 구성 변경 사항을 적용합니다.

Bash

```
sudo systemctl restart docker
```

D. Docker에서 GPU 액세스 확인

Docker 컨테이너가 GPU에 액세스할 수 있는지 확인하려면 `nvidia-smi` 를 실행하는 테스트 컨테이너를 실행합니다.

1. CUDA 기본 컨테이너 실행:

공식 NVIDIA CUDA 이미지를 사용합니다. CUDA 12.9 설정을 확인하려면 CUDA 12.9가 포함된 이미지가 이상적입니다. NVIDIA는 Docker Hub에

`nvidia/cuda` 이미지를 제공하고 보다 전문화된 컨테이너를 위해 `nvcr.io` 를 제공합니다. 예서는 CUDA 12.9.0이 포함된 Ubuntu 24.04 기반 컨테이너를 언급하지만 Docker Hub에서 간단한 기본 이미지로 공개적으로 사용할 수 있는지 여부는 다를 수 있습니다. 일반적인 대체 방법은 Ubuntu 22.04와 같은 최신 Ubuntu LTS를 기반으로 하는 CUDA 12.9 이미지입니다.

Bash

`-gpus all` 플래그는 Docker에 사용 가능한 모든 GPU를 컨테이너에 액세스할 수 있도록 지시합니다. 설정이 올바르면 이 명령은 지정된 이미지를 가져오고(아직 없는 경우) 컨테이너 내에서 `nvidia-smi` 를 실행합니다. 출력은 호스트의 `nvidia-smi` 출력과 유사해야 하며 RTX 6000 Ada 및 드라이버 세부 정보를 표시합니다. 이는 NVIDIA Container Toolkit이 올바르게 작동하고 GPU 리소스가 Docker 환경으로 전달되고 있음을 확인합니다. CUDA 12.9 라이브러리가 포함된 컨테이너를 사용하면 이 CUDA 버전에 맞게 빌드된 Docker 내 애플리케이션이 예상대로 작동할 수 있도록 더욱 보장됩니다.

```
# 사용 가능하고 알려진 경우 Ubuntu 24.04 기반 이미지로 시도
# sudo docker run --rm --gpus all nvcr.io/nvidia/cuda:12.9.0-base-ubuntu24.04 nvidia-smi
# 그렇지 않으면 쉽게 사용할 수 있는 CUDA 12.9 이미지 사용:
sudo docker run --rm --gpus all nvidia/cuda:12.9.0-base-ubuntu22.04 nvidia-smi
```

다음 표는 NVIDIA Container Toolkit 설치를 요약한 것입니다:

| 단계 | 하위 단계 | 명령 | 참고 |
|--------------------------------|---|---|--|
| 1. Docker Engine 설치 | 1a. Docker 리포지토리 설정 | (공식 Docker 문서 참조) | 공식 Docker CE가 설치되었는지 확인합니다. |
| | 1b. Docker 패키지 설치 | <code>sudo apt-get update sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin</code> | Docker 구성 요소를 설치합니다. |
| | 1c. 사용자를 <code>docker</code> 그룹에 추가 | <code>sudo usermod -aG docker \$USER newgrp docker</code> (또는 로그아웃/로그인) | <code>sudo</code> 없이 Docker를 실행할 수 있도록 합니다. |
| 2. NVIDIA Container Toolkit 설치 | 2a. 툴킷 리포지토리 구성 | <code>`curl -fsSL...</code> | <code>sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list` (전체 명령은 텍스트 참조)</code> |
| | 2b. 툴킷 패키지 설치 | <code>sudo apt-get update sudo apt-get install -y nvidia-container-toolkit</code> | 툴킷을 설치합니다. |
| 3. Docker 런타임 구성 | 3a. <code>nvidia-ctk</code> 로 Docker 구성 | <code>sudo nvidia-ctk runtime configure --runtime=docker</code> | NVIDIA 런타임을 사용하도록 Docker의 <code>daemon.json</code> 을 업데이트합니다. |
| | 3b. Docker 서비스 다시 시작 | <code>sudo systemctl restart docker</code> | 런타임 구성 변경 사항을 적용합니다. |
| 4. GPU 액세스 확인 | 4a. 테스트 컨테이너 실행 | <code>sudo docker run --rm --gpus all nvidia/cuda:12.9.0-base-ubuntu22.04 nvidia-smi</code> | Docker 컨테이너 내에서 GPU에 액세스할 수 있는지 확인합니다. 출력에 RTX 6000 Ada가 표시되어야 합니다. |

Export to Sheets

VII. Nvidia NIM (NVIDIA Inference Microservices) 소개

NVIDIA Inference Microservices (NIM)는 AI 모델 배포를 단순화하여 확장 가능하고 프로덕션 준비가 된 마이크로서비스로 전환하도록 설계된 사전 빌드되고 최적화된 컨테이너입니다. NIM은 배포 복잡성 감소, 모델별 최적화를 통한 성능 향상, 손쉬운 통합을 위한 표준화된 API와 같은 이점을 제공합니다.

A. Nvidia NIM의 전제 조건

NIM을 배포하기 전에 다음 전제 조건이 충족되었는지 확인하십시오:

- **NVIDIA GPU:** 충분한 메모리가 있는 NVIDIA GPU가 필요합니다. RTX 6000 Ada는 많은 NIM에 적합합니다.
- **운영 체제:** 지원되는 Linux 배포판. `glibc >= 2.35` 가 포함된 Ubuntu 24.04는 호환됩니다.
- **NVIDIA 드라이버:** 섹션 III에서 완료한 대로 최신 NVIDIA 드라이버를 설치해야 합니다.
- **Docker 및 NVIDIA Container Toolkit:** 섹션 VI에 자세히 설명된 대로 설치하고 구성해야 합니다.
- **NVIDIA AI Enterprise 라이선스:** LLM 및 기타 모델용 NVIDIA NIM을 자체 호스팅하려면 일반적으로 NVIDIA AI Enterprise 라이선스가 필요합니다. 사용자는 라이선스 조건을 확인하고 사용 가능한 경우 평가판 옵션을 살펴봐야 합니다. 이는 기술적 설정 외에 중요한 고려 사항입니다.
- **NGC 계정 및 API 키:** NIM 컨테이너 이미지를 가져오려면 NVIDIA의 NGC 카탈로그에 액세스해야 합니다.

B. NIM 설정 단계

1. NGC API 키 생성:

NGC 컨테이너 레지스트리로 인증하려면 API 키가 필요합니다.

- NGC 웹사이트(일반적으로 <https://org.ngc.nvidia.com/setup/api-keys> 또는 <https://org.ngc.nvidia.com/setup/personal-keys>)로 이동합니다.
- 새 API 키를 생성합니다. 키 권한에 대해 "NGC Catalog" 서비스(또는 이를 포함하는 더 넓은 범위)가 선택되었는지 확인합니다.
- 생성된 API 키를 복사하여 안전하게 보관합니다.

2. API 키를 환경 변수로 내보내기:

NIM 컨테이너는 이 환경 변수를 사용하여 모델 자산 다운로드를 위해 NGC로 인증합니다.Bash

```
export NGC_API_KEY="YOUR_API_KEY_HERE"
```

"YOUR_API_KEY_HERE" 를 실제 키로 바꿉니다. 프로덕션 환경의 보안을 위해 이 키를 관리하는 더 안전한 방법(예: Docker 시크릿 또는 버전 제어에 커밋되지 않은 환경 파일)을 고려하십시오.

3. NGC Docker 레지스트리에 로그인:

API 키를 사용하여 NVIDIA의 컨테이너 레지스트리인

nvc.io 에 로그인합니다.Bash

```
echo "$NGC_API_KEY" | sudo docker login nvc.io --username '$oauthtoken' --password-stdin
```

사용자 이름 `$oauthtoken` 은 API 키가 인증에 사용되고 있음을 나타내는 특수 문자열입니다.

C. Nvidia NIM 실행

설정이 완료되면 NIM 컨테이너를 시작할 수 있습니다. 특정 모델 및 이미지 태그는 다를 수 있습니다.

1. 실행할 NIM 식별 (선택 사항이지만 유익함):

사용 가능한 NIM은 NVIDIA API 카탈로그(<https://build.nvidia.com>)

)를 탐색하거나 NGC CLI 도구(설치된 경우)를 `ngc registry image list --format_type csv nvc.io/nim/*` 와 같은 명령으로 사용하여 찾을 수 있습니다.

2. 예시: Llama 3 NIM 시작:

다음은 예시입니다.

`MODEL_NIM_PATH` 및 `NIM_IMAGE_TAG` 를 NGC 카탈로그에서 원하는 모델의 실제 경로 및 태그로 바꾸십시오.Bash

```
# 예시: NGC의 실제 모델 경로 및 태그로 바꾸십시오.
```

```
export MODEL_NIM_PATH="meta/llama3-8b-instruct"
```

```
export NIM_IMAGE_TAG="1.0.0" # 예시 태그, 최신 버전은 NGC 확인
```

```
export NIM_IMAGE_NAME="nvc.io/nim/${MODEL_NIM_PATH}:${NIM_IMAGE_TAG}"
```

```
export LOCAL_NIM_CACHE="$HOME/.cache/nim" # 모델 아티팩트용 로컬 캐시
```

```
mkdir -p "$LOCAL_NIM_CACHE"
```

```
export CONTAINER_NAME="nim_llama3_8b_instruct"
```

```
sudo docker run -it --rm --name ${CONTAINER_NAME} \
  --gpus all \
  --runtime=nvidia \
  -e NGC_API_KEY \
  -v "${LOCAL_NIM_CACHE}:/opt/nim/.cache" \
  -u $(id -u):$(id -g) \
  -p 8000:8000 \
  "${NIM_IMAGE_NAME}"
```

주요 Docker 실행 매개변수 설명 :

- `-gpus all` : 사용 가능한 모든 GPU를 컨테이너에 노출합니다.
- `--runtime=nvidia` : NVIDIA 컨테이너 런타임을 지정합니다.
- `-e NGC_API_KEY` : 모델 다운로드를 위해 NGC API 키를 컨테이너에 전달합니다.
- `-v "${LOCAL_NIM_CACHE}:/opt/nim/.cache"` : 다운로드한 모델 가중치 및 아티팩트용 캐시로 로컬 디렉터리를 마운트하여 후속 실행 시 재다운로드를 방지합니다.
- `-u $(id -u):$(id -g)` : 마운트된 캐시와의 권한 문제를 방지하기 위해 현재 사용자의 ID 및 그룹 ID로 컨테이너를 실행합니다.
- `-p 8000:8000` : 호스트의 포트 8000을 NIM 서비스가 일반적으로 수신 대기하는 컨테이너의 포트 8000에 매핑합니다.
- `${NIM_IMAGE_NAME}` : `nvc.io` 의 NIM 컨테이너 이미지 전체 이름 및 태그입니다.

D. NIM과 상호 작용

NIM 컨테이너가 실행되면 모델을 다운로드하고(아직 캐시되지 않은 경우) 추론 서버를 시작합니다.

1. 서버 준비 대기:

컨테이너 로그(`sudo docker logs -f ${CONTAINER_NAME}`)에서 서버가 준비되었음을 나타내는 메시지(예: "NIM server is ready" 또는 유사 메시지)를 모니터링합니다.

2. 상태 엔드포인트 확인:Bash

성공적인 응답(예: 상태 "ready"인 HTTP 200)은 서버가 작동 중임을 나타냅니다.

```
curl http://localhost:8000/v1/health/ready
```

3. NIM에서 제공하는 모델 나열:Bash

그러면 이 특정 NIM 인스턴스에서 제공하는 모델이 표시됩니다.

```
curl http://localhost:8000/v1/models
```

4. 추론 요청 보내기:

LLM NIM의 경우 API는 종종 OpenAI와 호환됩니다. 채팅 완료 모델 예시 :

Bash

많은 LLM에 OpenAI 호환 API를 사용하면 개발자가 기존 클라이언트 라이브러리를 종종 활용할 수 있으므로 통합이 단 순화됩니다.

```
curl -X POST http://localhost:8000/v1/chat/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer YOUR_NGC_API_KEY" \
-d '{
  "model": "meta/llama3-8b-instruct", // v1/models 출력의 모델 이름 사용
  "messages":,
  "temperature": 0.7,
  "max_tokens": 100
}'
```

E. NIM 컨테이너 중지

Bash

```
sudo docker stop ${CONTAINER_NAME}
```

다음 표는 NVIDIA NIM의 주요 Docker 실행 매개변수를 강조 표시합니다:

| 매개변수 | 예시 값 | 목적 |
|---|--|--|
| <code>--gpus all</code> | <code>all</code> | 모든 호스트 GPU를 컨테이너에서 사용할 수 있도록 합니다. |
| <code>--runtime=nvidia</code> | <code>nvidia</code> | GPU 액세스를 위한 NVIDIA 컨테이너 런타임을 지정합니다. |
| <code>-e NGC_API_KEY</code> | (환경 변수 값) | NGC에서 모델/자산 다운로드 인증을 위해 NGC API 키를 컨테이너에 전달합니다. |
| <code>-v LOCAL_NIM_CACHE:/opt/nim/.cache</code> | <code>"\$HOME/.cache/nim:/opt/nim/.cache"</code> | 다운로드한 모델을 유지하고 재다운로드를 방지하기 위해 컨테이너 내 캐시로 호스트 디렉토리를 마운트합니다. |
| <code>-p HOST_PORT:CONTAINER_PORT</code> | <code>8000:8000</code> | 호스트의 포트를 NIM 서비스가 수신 대기하는 컨테이너 내부의 포트에 매핑합니다. |
| <code>NIM_IMAGE_NAME</code> | <code>nvcr.io/nim/meta/llama3-8b-instruct:1.0.0</code> | 실행할 특정 NIM의 전체 Docker 이미지 이름 및 태그입니다. |

Export to Sheets

VIII. (선택 사항 보너스) CUDA 12.9를 지원하는 AI 프레임워크 설치

드라이버, CUDA Toolkit, cuDNN 및 Docker를 설정한 후 개발자는 일반적으로 PyTorch 또는 TensorFlow와 같은 AI 프레임워크를 설치합니다. 이는 종속성을 효과적으로 관리하기 위해 일반적으로 Python 가상 환경 내에서 `pip`를 사용하여 설치됩니다.

시스템의 NVIDIA 드라이버는 CUDA 12.9와 호환되고 CUDA Toolkit 12.9가 설치되어 있지만, AI 프레임워크용 사전 컴파일된 `pip` 패키지는 약간 이전 CUDA 버전(예: CUDA 12.1, 12.3 또는 12.8)에 대해 빌드될 수 있다는 점에 유의해야 합니다.

이러한 패키지는 일반적으로 순방향 호환되며 드라이버의 호환성으로 인해 올바르게 실행되지만 프레임워크 자체는 핵심 작업을 위해 컴파일된 CUDA 버전을 활용합니다.

A. PyTorch

PyTorch는 종종 최신 CUDA 버전과 호환되는 `pip` 휠을 제공합니다.

1. Python 가상 환경 생성 및 활성화:Bash

```
python3 -m venv ~/pytorch_env
source ~/pytorch_env/bin/activate
```

2. PyTorch 설치:

최신 명령은 공식 PyTorch 웹사이트(<https://pytorch.org/get-started/locally/>)를 참조하십시오. CUDA 12.x의 경우 다음과 유사한 명령이 일반적입니다.

최근 정보에 따르면 Linux용 `cu128` (CUDA 12.8) 휠이 포함된 PyTorch 2.7을 사용할 수 있으며 CUDA 12.9 환경과 호환되어야 합니다. 특정 `cu129` 휠을 사용할 수 있게 되면 해당 휠이 선호됩니다. 그렇지 않으면 `cu121` 또는 `cu128` 이 일반적인 선택입니다.

Bash

(설치 시점에 PyTorch에서 CUDA 12.9에 대해 더 직접적이거나 최신 호환 휠(예: `cu129` 또는 `cu121`)을 권장하는 경우 `cu128` 을 조정하십시오).

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu128
```

3. PyTorch 설치 확인:Python

Python 파일(예:

`verify_pytorch.py`)로 저장하고 `python verify_pytorch.py` 를 실행합니다. `torch.cuda.is_available()` 는 `True` 를 반환해야 합니다.

```
import torch
print(f"PyTorch Version: {torch.__version__}")
print(f"CUDA Available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"CUDA Version (PyTorch built with): {torch.version.cuda}")
    print(f"GPU Name: {torch.cuda.get_device_name(0)}")
```

B. TensorFlow

TensorFlow는 Python 환경 내에 호환되는 CUDA 및 cuDNN 라이브러리를 번들로 제공하여 GPU 설정을 단순화하려는 `tensorflow[and-cuda]` `pip` 패키지를 제공합니다.

1. Python 가상 환경 생성 및 활성화:Bash

```
python3 -m venv ~/tensorflow_env
source ~/tensorflow_env/bin/activate
```

2. GPU 지원 TensorFlow 설치: `tensorflow[and-cuda]`

패키지는 특정 버전의 CUDA 및 cuDNN을 번들로 제공합니다. 예를 들어, 최신 TensorFlow 버전은 CUDA 12.3 및 cuDNN 8.9.7을 번들로 제공할 수 있습니다. 이는 TensorFlow가 시스템에 설치된 CUDA 12.9 및 해당 cuDNN을 직접 사용하지 않고 이러한 번들 라이브러리를 작업에 사용하지만 CUDA 12.9 호환 드라이버에서 실행된다는 의미입니다. Bash

```
pip install -U pip # pip 업그레이드
pip install tensorflow[and-cuda]
```

3. TensorFlow 설치 확인:Python

```
import tensorflow as tf
print(f"TensorFlow Version: {tf.__version__}")
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print(f"Num GPUs Available: {len(gpus)}")
    for gpu in gpus:
        print(f"GPU Name: {gpu.name}")
else:
    print("No GPU detected by TensorFlow")
```

Python 파일(예: `verify_tensorflow.py`)로 저장하고 `python verify_tensorflow.py` 를 실행합니다. 출력에 RTX 6000 Ada가 나열되어야 합니다.

TensorFlow에 대해 시스템 CUDA 12.9와의 정확한 정렬이 엄격하게 필요한 경우 구성 프로세스 중에 CUDA 12.9 및 호환되는 cuDNN 버전을 지정하여 소스에서 TensorFlow를 빌드해야 합니다. 이는 고급 절차이며 TensorFlow 자체에

서 해당 특정 CUDA 버전에 연결된 특정 기능이나 최적화가 필요한 경우가 아니면 일반적으로 필요하지 않습니다.

다음 표는 AI 프레임워크 설치에 대한 빠른 참조를 제공합니다:

| 프레임워크 | 권장 설치 명령 (CUDA 12.x 호환성용) | 확인 명령 (Python) | CUDA 12.9 참고 사항 |
|------------|---|---|---|
| PyTorch | <code>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu128</code> | <code>import torch; print(torch.__version__); print(torch.cuda.is_available()); print(torch.version.cuda if torch.cuda.is_available() else "")</code> | 사전 빌드된 휠(예: <code>cu128</code> 또는 <code>cu121</code>)은 일반적으로 CUDA 12.9 드라이버/툴킷과 순방향 호환됩니다. 인쇄된 <code>torch.version.cuda</code> 는 PyTorch가 빌드된 CUDA 버전을 보여줍니다. |
| TensorFlow | <code>pip install tensorflow[and-cuda]</code> | <code>import tensorflow as tf; print(tf.__version__); print(tf.config.list_physical_devices('GPU'))</code> | <code>tensorflow[and-cuda]</code> 패키지는 자체 CUDA/cuDNN(예: 최신 TF의 경우 CUDA 12.3/cuDNN 8.9.7)을 번들로 제공합니다. 시스템의 CUDA 12.9를 직접 사용하지 않고 이를 사용하지만 12.9 호환 드라이버에서 실행됩니다. 정확한 12.9 연결을 위해서는 소스에서 빌드하십시오. |

Export to Sheets

IX. 일반적인 문제 해결

신중하게 설치하더라도 문제가 발생할 수 있습니다. 이 섹션에서는 일반적인 문제와 기본 문제 해결 단계를 다룹니다.

• NVIDIA 드라이버 문제:

- **증상:** `nvidia-smi` 명령 실패, GPU가 표시되지 않거나 시스템이 저해상도 디스플레이로 부팅됩니다.
- **가능한 원인 및 해결 방법:**
 - **Nouveau 충돌:** Nouveau가 블랙리스트에 있는지 확인합니다 (섹션 II.C.4).
 - **Secure Boot MOK:** Secure Boot가 활성화된 경우 NVIDIA 드라이버의 MOK가 올바르게 등록되지 않았을 수 있습니다. 재부팅하고 MOK 관리 옵션을 찾거나 Secure Boot를 일시적으로 비활성화합니다.
 - **설치 오류:** 드라이버 설치 중 오류에 대해 `apt` 로그(`/var/log/apt/term.log`) 및 커널 메시지(`dmesg | grep -i nvidia`)를 확인합니다.
 - **드라이버 재설치:** 완전히 새로 설치해야 할 수 있습니다. 재설치하기 전에 기존 NVIDIA 패키지를 제거합니다 (`sudo apt purge '*nvidia*' && sudo apt autoremove`).
 - **"GPU has fallen off the bus":** 이는 하드웨어 문제(GPU 장착, 전원 공급 장치, PCIe 슬롯), 과열 또는 심각한 드라이버/커널 비호환성을 나타낼 수 있는 심각한 오류입니다. 먼저 물리적 연결을 확인하십시오. 소프트웨어 비호환성인 경우 다른 드라이버 버전(이전 또는 최신 안정 버전)을 시도하면 도움이 될 수 있습니다.

• CUDA Toolkit 문제:

- **증상:** `nvcc: command not found` .
- **가능한 원인 및 해결 방법:**
 - **환경 변수:** `PATH` 및 `LD_LIBRARY_PATH` 가 올바르게 설정되지 않았거나 현재 터미널 세션에서 `~/.bashrc` 변경 사항이 적용되지 않았습니다 (섹션 IV.B). `echo $PATH` 및 `echo $LD_LIBRARY_PATH` 로 확인합니다.
 - **증상:** CUDA 샘플(예: `deviceQuery`)이 컴파일 또는 실행되지 않습니다.

- 가능한 원인 및 해결 방법:
 - **드라이버-툴킷 불일치:** 리포지토리 설치 시 이를 최소화하지만 드라이버가 CUDA 툴킷 버전을 지원하는지 확인합니다.
 - **누락된 종속성:** `build-essential` 또는 기타 개발 도구가 누락되었을 수 있습니다.
 - **잘못된 툴킷 버전:** `nvcc --version` 이 예상과 다른 버전을 표시합니다.
- **cuDNN 문제:**
 - **증상:** AI 프레임워크 또는 애플리케이션에서 누락된 cuDNN 라이브러리(예: `libcudnn.so.9: cannot open shared object file`)를 보고합니다.
 - 가능한 원인 및 해결 방법:
 - **설치 미완료:** cuDNN 런타임(`libcudnn9-cuda-12`) 또는 개발자 패키지(`libcudnn9-dev-cuda-12`)가 올바르게 설치되지 않았을 수 있습니다. `dpkg -l | grep libcudnn` 으로 확인합니다.
 - **라이브러리 경로:** `LD_LIBRARY_PATH` 에 cuDNN 라이브러리가 설치된 디렉터리가 포함되어 있지 않을 수 있습니다(`apt` 는 일반적으로 시스템 라이브러리 경로를 통해 이를 처리하지만). `sudo ldconfig -p | grep libcudnn` 은 동적 링커가 라이브러리를 찾는지 보여줄 수 있습니다.
- **Docker GPU 액세스 문제:**
 - **증상:** `docker run --gpus all...` 이 NVIDIA 런타임 또는 GPU 액세스와 관련된 오류로 실패합니다.
 - 가능한 원인 및 해결 방법:
 - **NVIDIA Container Toolkit:** 설치되지 않았거나 `nvidia-ctk runtime configure --runtime=docker` 가 실행되지 않았거나 구성 후 Docker 데몬이 다시 시작되지 않았습니다 (섹션 VI.C).
 - **Docker 데몬:** Docker 서비스가 실행 중이고 구성 변경 후 다시 시작되었는지 확인합니다.
 - **증상:** 호스트에서는 `nvidia-smi` 가 작동하지만 GPU 지원 컨테이너 내부에서는 작동하지 않습니다.
 - 가능한 원인 및 해결 방법:
 - Docker의 `daemon.json` 에 있는 NVIDIA 런타임 구성 문제.
 - 잘못된 `-gpus` 플래그 사용 또는 호환되지 않는 기본 이미지.
- **일반적인 문제 해결 팁:**
 - **시스템 로그 확인:**
 - `dmesg` : 커널 링 버퍼, 하드웨어 및 모듈 로드 문제에 유용합니다.
 - `journalctl -xe` : 자세한 시스템 로그, 종종 서비스 오류를 표시합니다.
 - `/var/log/syslog` : 일반 시스템 메시지.
 - `/var/log/nvidia-installer.log` : NVIDIA의 `.run` 파일을 사용하여 드라이버를 설치한 경우 (여기서는 권장하지 않음).
 - **재부팅:** 중요한 드라이버 또는 커널 모듈 변경 후에는 종종 재부팅이 필수적입니다.
 - **NVIDIA 개발자 포럼:** 특정 문제에 대한 도움을 받을 수 있는 귀중한 리소스입니다. 시스템, 설치된 소프트웨어 버전 및 오류 메시지에 대한 자세한 정보를 제공하십시오.

`nvidia-smi` 로 드라이버를 확인하고, `nvcc` 및 샘플로 CUDA를 확인하고, cuDNN을 확인하고, 마지막으로 Docker를 확인하는 체계적인 접근 방식은 문제가 있을 수 있는 부분을 격리하는 데 도움이 됩니다. 특히 CUDA 및 cuDNN의 많은 문제는 잘못된 환경 변수 설정이나 조정된 리포지토리에서 신중하게 설치하지 않은 경우 구성 요소 버전 간의 불일치로 인해 발생합니다.

X. 결론 및 모범 사례

이 보고서는 NVIDIA RTX 6000 Ada 그래픽 카드를 갖춘 Ubuntu 24.04 워크스테이션에서 강력한 AI 개발 환경을 구축하는 과정을 자세히 설명했습니다. 공식 NVIDIA 리포지토리를 사용하여 NVIDIA 드라이버, CUDA Toolkit 12.9 및 해당

cuDNN 라이브러리를 체계적으로 설치함으로써 개발자는 안정적인 고성능의 기반을 확보할 수 있습니다. 또한 NVIDIA Container Toolkit을 사용한 Docker 구성은 휴대 가능하고 재현 가능한 GPU 가속 워크플로를 가능하게 하며, NVIDIA NIM 소개는 간소화된 AI 모델 배포 경로를 제공합니다.

건강한 AI 개발 환경 유지를 위한 주요 모범 사례:

1. **공식 리포지토리 활용:** 가능하면 NVIDIA의 공식 `apt` 리포지토리에서 NVIDIA 드라이버, CUDA Toolkit 및 cuDNN을 설치하십시오. 이 방법은 일반적으로 구성 요소 호환성을 보장하고 업데이트를 단순화합니다.
2. **버전 고정 (고려 사항):** 프로덕션 또는 중요한 연구 환경의 경우 `apt` 기본 설정을 사용하여 드라이버 및 CUDA 구성 요소의 버전을 고정하여 기존 워크플로와 비호환성을 유발할 수 있는 의도하지 않은 업그레이드를 방지하는 것을 고려하십시오.
3. **Python 가상 환경:** AI 프로젝트에는 항상 Python 가상 환경(예: `venv`, `conda`)을 사용하십시오. 이렇게 하면 종속성을 격리하여 PyTorch 또는 TensorFlow와 같은 라이브러리의 다른 버전이 필요한 프로젝트 간의 충돌을 방지합니다.
4. **정기적인 문서 참조:** AI 및 GPU 컴퓨팅 환경은 빠르게 발전합니다. 최신 호환성 정보, 기능 및 모범 사례를 위해 CUDA, cuDNN, 드라이버 및 AI 프레임워크에 대한 공식 NVIDIA 문서를 정기적으로 확인하십시오.
5. **GPU 사용률 모니터링:** 개발 및 배포 중에 `nvidia-smi` 또는 NVIDIA Nsight와 같은 도구를 사용하여 GPU 사용률, 메모리 사용량 및 성능을 모니터링하십시오. 이는 병목 현상을 식별하고 GPU 리소스가 효율적으로 사용되고 있는지 확인하는 데 도움이 됩니다.
6. **NGC API 키 보안:** NVIDIA NIM 또는 기타 NGC 리소스를 사용할 때 NGC API 키를 기밀로 유지하고 안전하게 관리하십시오. 스크립트에 키를 하드코딩하거나 버전 제어에 커밋하지 마십시오.
7. **체계적인 업데이트:** 중요한 워크스테이션에 적용하기 전에 가능하면 스테이징 환경에서 테스트하여 드라이버 및 CUDA 구성 요소 업데이트를 신중하게 계획하십시오.
8. **커뮤니티 및 지원:** 문제 해결 및 커뮤니티 지원을 위해 NVIDIA 개발자 포럼과 같은 리소스를 활용하십시오.

이러한 지침과 제공된 자세한 설치 단계를 따르면 사용자는 Ubuntu 24.04에서 최첨단 AI 코딩 및 연구를 위해 NVIDIA RTX 6000 Ada 워크스테이션의 잠재력을 극대화할 수 있습니다.

[https://developer.nvidia.com/cuda-downloads?](https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=24.04&target_type=deb_local)

[target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=24.04&target_type=deb_local](https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=24.04&target_type=deb_local)

CUDA Toolkit Installer

Installation Instructions:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/x86_64/cuda-ubuntu2404.pin
sudo mv cuda-ubuntu2404.pin /etc/apt/preferences.d/cuda-repository-pin-600
wget https://developer.download.nvidia.com/compute/cuda/12.9.0/local_installers/cuda-repo-ubuntu2404-12-9-local_12.9.0-575.51.03-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2404-12-9-local_12.9.0-575.51.03-1_amd64.deb
sudo cp /var/cuda-repo-ubuntu2404-12-9-local/cuda-*-keyring.gpg /usr/share/keyrings/
sudo apt-get update
sudo apt-get -y install cuda-toolkit-12-9
```

Additional installation options are detailed [here](#).

Driver Installer

NVIDIA Driver Instructions (choose one option)

To install the open kernel module flavor:

```
sudo apt-get install -y nvidia-open
```

To install the proprietary kernel module flavor:

```
sudo apt-get install -y cuda-drivers
```


To switch between NVIDIA Driver kernel module flavors see [here](#).

The CUDA Toolkit contains Open-Source Software. The source code can be found [here](#). The checksums for the installer and patches can be found in [Installer Checksums](#). For further information, see the [Installation Guide for Linux](#) and the [CUDA Quick Start Guide](#).

Ubuntu 20.04/22.04/24.04, Debian 12

To switch from proprietary to open:

```
# apt install --autoremove --purge nvidia-open
```

 To switch from open to proprietary:

```
# apt install --autoremove --purge cuda-drivers
```



참고자료

<https://velog.io/@scuderia/%EB%A6%AC%EB%88%85%EC%8A%A4-24.04-cuda-cudnn-%EC%84%A4%EC%B9%98>

[https://developer.nvidia.com/cuda-downloads?](https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=24.04&target_type=deb_network)

[target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=24.04&target_type=deb_network](https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=24.04&target_type=deb_network)

- nvidia driver 설치
- `sudo apt install nvidia-cuda-toolkit`
- llama-cpp-python version available for CUDA 11.7.

cuda

<https://www.cherryservers.com/blog/install-cuda-ubuntu>

[https://developer.nvidia.com/cuda-12-2-2-download-archive?](https://developer.nvidia.com/cuda-12-2-2-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=22.04&target_type=deb_local)

[target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=22.04&target_type=deb_local](https://developer.nvidia.com/cuda-12-2-2-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=22.04&target_type=deb_local)

Nvidia NIM

<https://docs.nvidia.com/nim/large-language-models/latest/getting-started.html>

<https://linuxconfig.org/how-to-install-the-nvidia-drivers-on-ubuntu-22-04>

nvidia GPU on Docker

<https://www.howtogeek.com/devops/how-to-use-an-nvidia-gpu-with-docker-containers/>

<https://www.cyberciti.biz/faq/how-to-install-nvidia-driver-on-centos-7-linux/>

<https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#rhel-7-centos-7>