PLANETNIX

NixOS on the NVIDIA DGX Spark

Presented by

Graham Bennett

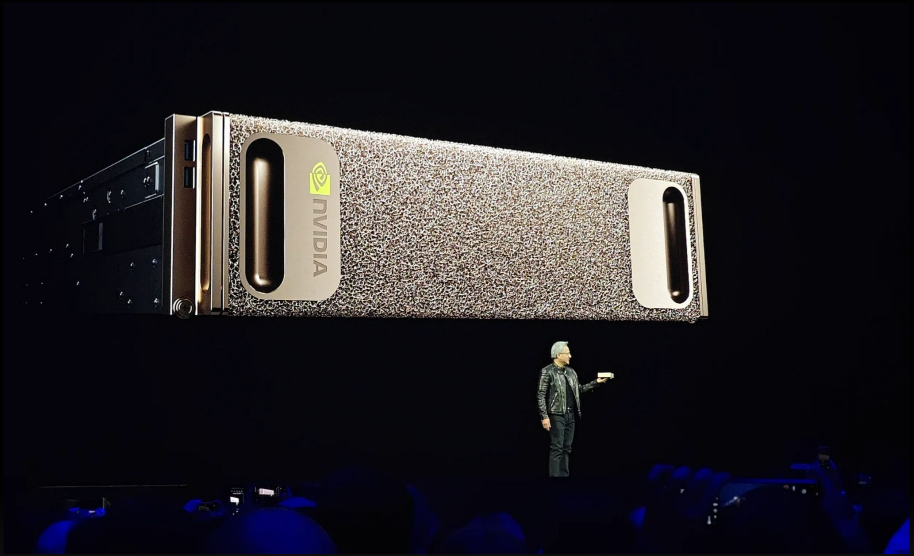
Powered by

FLOX

What is the DGX Spark?

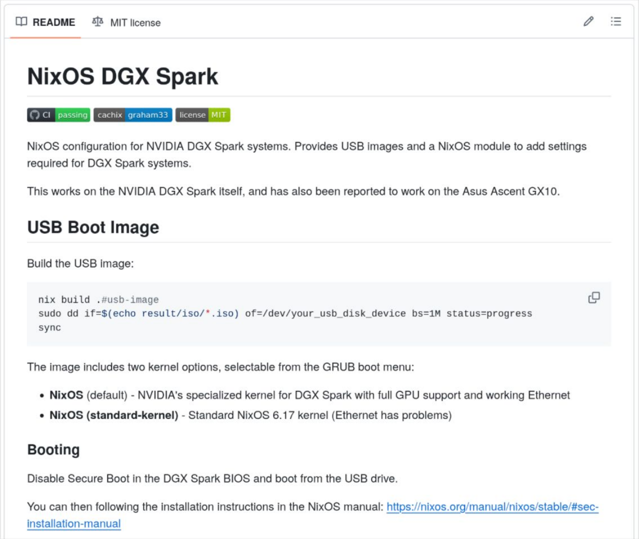
- NVIDIA Desktop AI workstation
- GB10 Blackwell GPU
- 20 ARM Cortex CPU cores
- 128GB unified CPU+GPU memory

- Runs **NVIDIA DGX OS** (Ubuntu derivative)
- **But we want NixOS!**



NixOS!

- USB image, NixOS module and config template
- NVIDIA custom Linux 6.17 kernel
- NVIDIA drivers + various NixOS options
- Support for **NVIDIA DGX Spark Playbooks…**



NixOS DGX Spark

NixOS configuration for NVIDIA DGX Spark systems. Provides USB images and a NixOS module to add settings required for DGX Spark systems.

This works on the NVIDIA DGX Spark itself, and has also been reported to work on the Asus Ascent GX10.

USB Boot Image

Build the USB image:

```
nix build .#usb-image
sudo dd if=$(echo result/iso/*.iso) of=/dev/your_usb_disk_device bs=1M status=progress
sync
```

The image includes two kernel options, selectable from the GRUB boot menu:

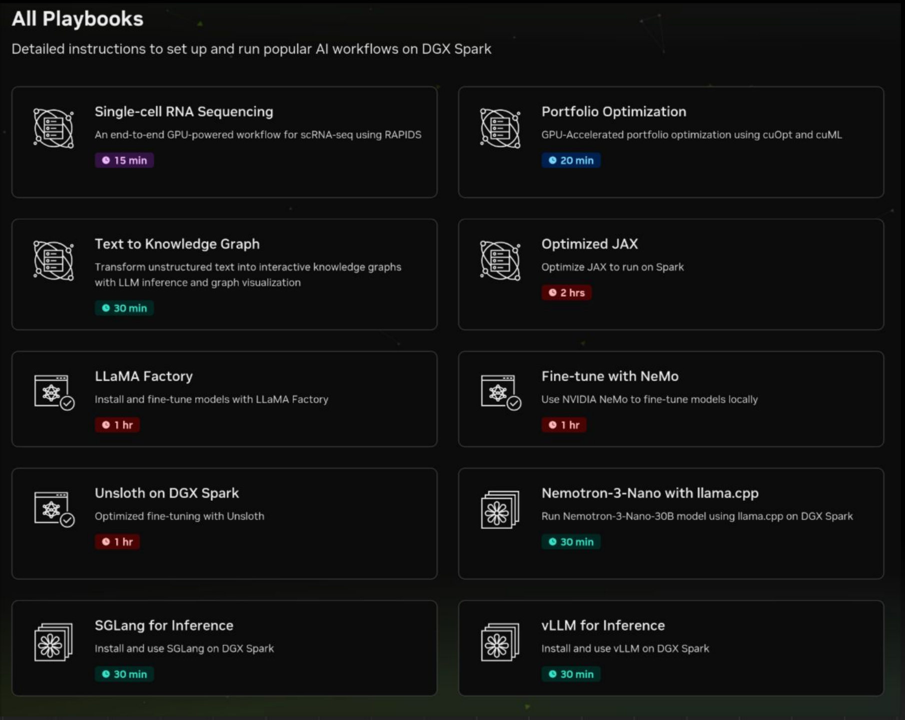
- **NixOS** (default) - NVIDIA's specialized kernel for DGX Spark with full GPU support and working Ethernet
- **NixOS (standard-kernel)** - Standard NixOS 6.17 kernel (Ethernet has problems)

Booting

Disable Secure Boot in the DGX Spark BIOS and boot from the USB drive.

You can then following the installation instructions in the NixOS manual: https://nixos.org/manual/nixos/stable/#sec-installation-manual

https://github.com/graham33/nixos-dgx-spark/



All Playbooks

Detailed instructions to set up and run popular AI workflows on DGX Spark

Single-cell RNA Sequencing
An end-to-end GPU-powered workflow for scRNA-seq using RAPIDS
15 min

Portfolio Optimization
GPU-Accelerated portfolio optimization using cuOpt and cuML
20 min

Text to Knowledge Graph
Transform unstructured text into interactive knowledge graphs with LLM inference and graph visualization
30 min

Optimized JAX
Optimize JAX to run on Spark
2 hrs

LLaMA Factory
Install and fine-tune models with LLaMA Factory
1 hr

Fine-tune with NeMo
Use NVIDIA NeMo to fine-tune models locally
1 hr

Unsloth on DGX Spark
Optimized fine-tuning with Unsloth
1 hr

Nemotron-3-Nano with llama.cpp
Run Nemotron-3-Nano-30B model using llama.cpp on DGX Spark
30 min

SGLang for Inference
Install and use SGLang on DGX Spark
30 min

vLLM for Inference
Install and use vLLM on DGX Spark
30 min

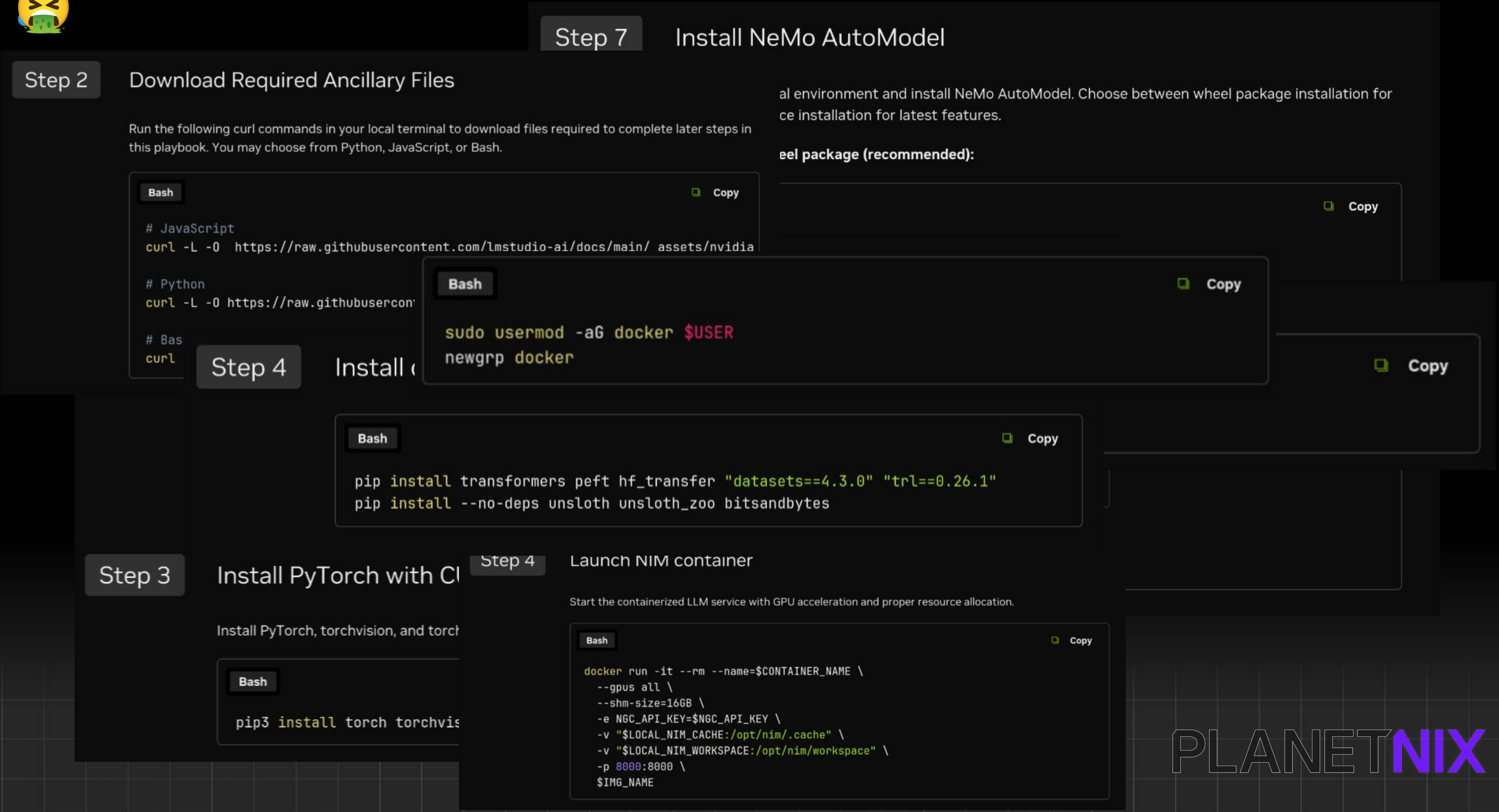
What are Playbooks?

- Recipes to do useful things on your DGX Spark
- Lots of great content
- **But… there’s a problem**

WARNING!

DISTURBING CONTENT

Viewer discretion is advised



Step 7 Install NeMo AutoModel

al environment and install NeMo AutoModel. Choose between wheel package installation for ce installation for latest features.

eel package (recommended):

Step 2 Download Required Ancillary Files

Run the following curl commands in your local terminal to download files required to complete later steps in this playbook. You may choose from Python, JavaScript, or Bash.

```
# JavaScript
curl -L -O  https://raw.githubusercontent.com/lmstudio-ai/docs/main/_assets/nvidia

# Python
curl -L -O https://raw.githubuserco

# Bas
curl
```

```
sudo usermod -aG docker $USER
newgrp docker
```

Step 4 Install

```
pip install transformers peft hf_transfer "datasets==4.3.0" "trl==0.26.1"
pip install --no-deps unsloth unsloth_zoo bitsandbytes
```

Step 3 Install PyTorch with CU

Install PyTorch, torchvision, and torch

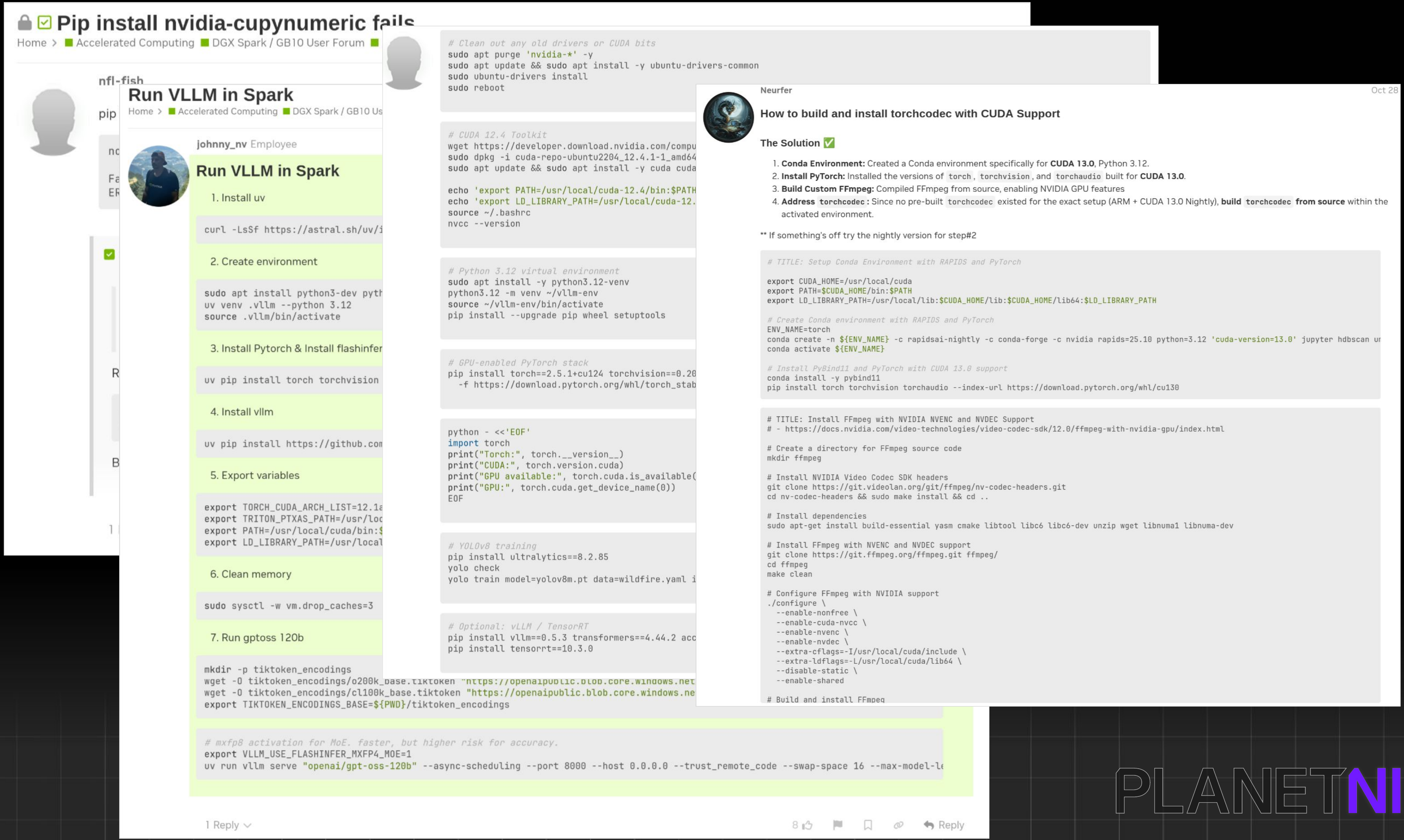
```
pip3 install torch torchvis
```

Step 4 Launch NIM container

Start the containerized LLM service with GPU acceleration and proper resource allocation.

```
docker run -it --rm --name=$CONTAINER_NAME \
  --gpus all \
  --shm-size=16GB \
  -e NGC_API_KEY=$NGC_API_KEY \
  -v "$LOCAL_NIM_CACHE:/opt/nim/.cache" \
  -v "$LOCAL_NIM_WORKSPACE:/opt/nim/workspace" \
  -p 8000:8000 \
  $IMG_NAME
```

PLANETNIX



Pip install nvidia-cupynumeric fails

Home > Accelerated Computing DGX Spark / GB10 User Forum

nfl-fish

pip

```
nc

Fa
ER
```

R

B

1

Run VLLM in Spark

Home > Accelerated Computing DGX Spark / GB10 Us

johnny_nv Employee

Run VLLM in Spark

1. Install uv

```
curl -LsSf https://astral.sh/uv/i
```

2. Create environment

```
sudo apt install python3-dev pyth
uv venv .vllm --python 3.12
source .vllm/bin/activate
```

3. Install Pytorch & Install flashinfer

```
uv pip install torch torchvision
```

4. Install vllm

```
uv pip install https://github.com
```

5. Export variables

```
export TORCH_CUDA_ARCH_LIST=12.1a
export TRITON_PTXAS_PATH=/usr/loc
export PATH=/usr/local/cuda/bin:$
export LD_LIBRARY_PATH=/usr/local
```

6. Clean memory

```
sudo sysctl -w vm.drop_caches=3
```

7. Run gptoss 120b

```
mkdir -p tiktoken_encodings
wget -O tiktoken_encodings/o200k_base.tiktoken "https://openaipublic.blob.core.windows.net
wget -O tiktoken_encodings/cl100k_base.tiktoken "https://openaipublic.blob.core.windows.ne
export TIKTOKEN_ENCODINGS_BASE=${PWD}/tiktoken_encodings
```

```
# mxfp8 activation for MoE. faster, but higher risk for accuracy.
export VLLM_USE_FLASHINFER_MXFP4_MOE=1
uv run vllm serve "openai/gpt-oss-120b" --async-scheduling --port 8000 --host 0.0.0.0 --trust_remote_code --swap-space 16 --max-model-le
```

1 Reply 8 Reply

```
# Clean out any old drivers or CUDA bits
sudo apt purge 'nvidia-*' -y
sudo apt update && sudo apt install -y ubuntu-drivers-common
sudo ubuntu-drivers install
sudo reboot
```

```
# CUDA 12.4 Toolkit
wget https://developer.download.nvidia.com/compu
sudo dpkg -i cuda-repo-ubuntu2204_12.4.1-1_amd64
sudo apt update && sudo apt install -y cuda cuda

echo 'export PATH=/usr/local/cuda-12.4/bin:$PATH
echo 'export LD_LIBRARY_PATH=/usr/local/cuda-12.
source ~/.bashrc
nvcc --version
```

```
# Python 3.12 virtual environment
sudo apt install -y python3.12-venv
python3.12 -m venv ~/vllm-env
source ~/vllm-env/bin/activate
pip install --upgrade pip wheel setuptools
```

```
# GPU-enabled PyTorch stack
pip install torch==2.5.1+cu124 torchvision==0.20
  -f https://download.pytorch.org/whl/torch_stab
```

```
python - <<'EOF'
import torch
print("Torch:", torch.__version__)
print("CUDA:", torch.version.cuda)
print("GPU available:", torch.cuda.is_available(
print("GPU:", torch.cuda.get_device_name(0))
EOF
```

```
# YOLOv8 training
pip install ultralytics==8.2.85
yolo check
yolo train model=yolov8m.pt data=wildfire.yaml i
```

```
# Optional: vLLM / TensorRT
pip install vllm==0.5.3 transformers==4.44.2 acc
pip install tensorrt==10.3.0
```

Neurfer Oct 28

How to build and install torchcodec with CUDA Support

The Solution ✅

1. **Conda Environment:** Created a Conda environment specifically for **CUDA 13.0**, Python 3.12.
2. **Install PyTorch:** Installed the versions of `torch`, `torchvision`, and `torchaudio` built for **CUDA 13.0**.
3. **Build Custom FFmpeg:** Compiled FFmpeg from source, enabling NVIDIA GPU features
4. **Address `torchcodec`:** Since no pre-built `torchcodec` existed for the exact setup (ARM + CUDA 13.0 Nightly), **build `torchcodec` from source** within the activated environment.

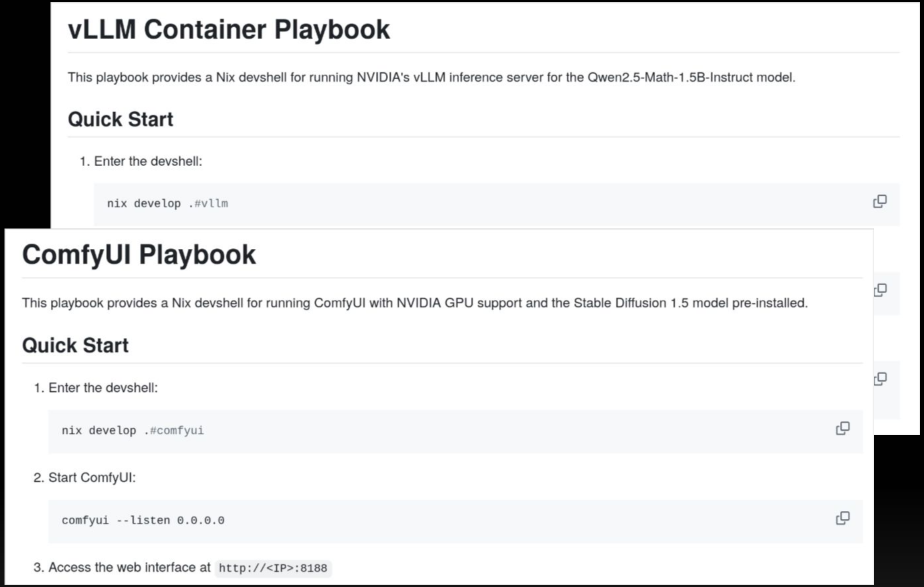
** If something's off try the nightly version for step#2

```
# TITLE: Setup Conda Environment with RAPIDS and PyTorch

export CUDA_HOME=/usr/local/cuda
export PATH=$CUDA_HOME/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/lib:$CUDA_HOME/lib:$CUDA_HOME/lib64:$LD_LIBRARY_PATH

# Create Conda environment with RAPIDS and PyTorch
ENV_NAME=torch
conda create -n ${ENV_NAME} -c rapidsai-nightly -c conda-forge -c nvidia rapids=25.10 python=3.12 'cuda-version=13.0' jupyter hdbscan ur
conda activate ${ENV_NAME}

# Install PyBind11 and PyTorch with CUDA 13.0 support
conda install -y pybind11
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu130
```

```
# TITLE: Install FFmpeg with NVIDIA NVENC and NVDEC Support
# - https://docs.nvidia.com/video-technologies/video-codec-sdk/12.0/ffmpeg-with-nvidia-gpu/index.html

# Create a directory for FFmpeg source code
mkdir ffmpeg

# Install NVIDIA Video Codec SDK headers
git clone https://git.videolan.org/git/ffmpeg/nv-codec-headers.git
cd nv-codec-headers && sudo make install && cd ..

# Install dependencies
sudo apt-get install build-essential yasm cmake libtool libc6 libc6-dev unzip wget libnuma1 libnuma-dev

# Install FFmpeg with NVENC and NVDEC support
git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg/
cd ffmpeg
make clean

# Configure FFmpeg with NVIDIA support
./configure \
  --enable-nonfree \
  --enable-cuda-nvcc \
  --enable-nvenc \
  --enable-nvdec \
  --extra-cflags=-I/usr/local/cuda/include \
  --extra-ldflags=-L/usr/local/cuda/lib64 \
  --disable-static \
  --enable-shared

# Build and install FFmpeg
```

We know a better way!

- Playbooks defined as Nix flake devShells and apps
- Packages from Nixpkgs and projects like nixified.ai
- One command setup
- Everyone gets the exact same environment



vLLM Container Playbook

This playbook provides a Nix devshell for running NVIDIA's vLLM inference server for the Qwen2.5-Math-1.5B-Instruct model.

Quick Start

1. Enter the devshell:

```
nix develop .#vllm
```

ComfyUI Playbook

This playbook provides a Nix devshell for running ComfyUI with NVIDIA GPU support and the Stable Diffusion 1.5 model pre-installed.

Quick Start

1. Enter the devshell:

```
nix develop .#comfyui
```

2. Start ComfyUI:

```
comfyui --listen 0.0.0.0
```

3. Access the web interface at `http://<IP>:8188`

NixOS Wins (and rough edges)

Wins

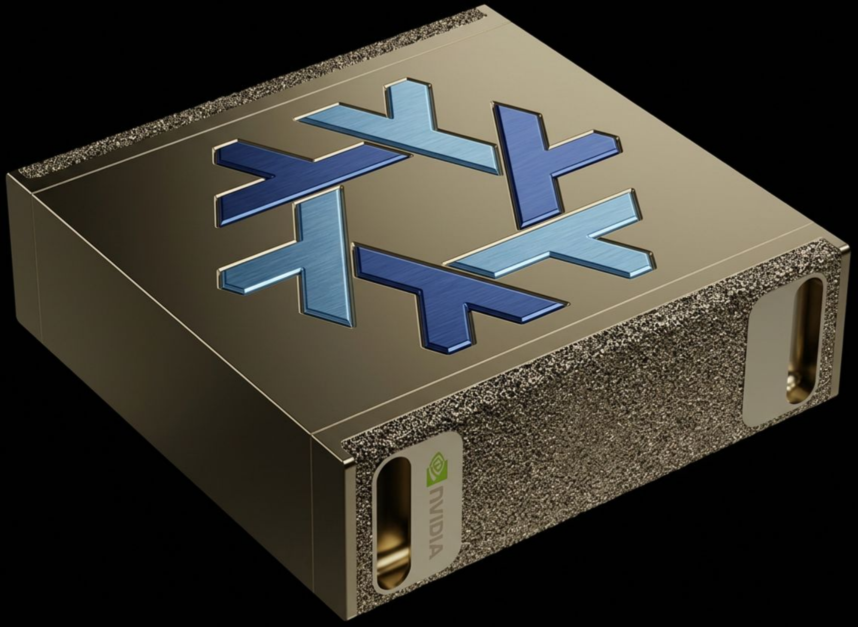
- Nixpkgs CUDA aarch64 ecosystem is quite healthy!
- Building a custom kernel is easy
- Write once, works for everyone
- Flakes make a good user interface

Rough edges

- Lack of cached aarch64 CUDA builds
- Uneven freshness of packages
- aarch64-linux is a less common target
- Community lacks GPU test hardware

Opportunity

- Shipping an AI package ecosystem to users is **hard**
- Nix is really good at this!
- DGX Spark is a great use-case for Nix/NixOS
  - Reproducible, complexity hidden
  - → Better out-of-the-box user experience
  - → Reduced support burden
- NVIDIA should consider supporting Nix natively!



Thank you!

(especially Nix CUDA team, nixified.ai and all Nix/Nixpkgs/NixOS contributors!)

Presented by

Graham Bennett

graham@grahambennett.org

@graham33