RTX GPU to Jetson Orin Nano: CIFAR-10 CUDA Project Manual

Step-by-step instructions from MacBook VS Code Remote-SSH to RTX training, ONNX export, inference test, and file transfer to Jetson Nano

Prepared for Brojo / ML GPU Edge Nano workflow | Date: 07 May 2026

Core idea: MacBook is only the controller. The actual CUDA training runs on the RTX GPU desktop. The Jetson Orin Nano is used later for edge deployment and TensorRT optimization.

Component	Correct value used in this workflow
Controller computer	MacBook with VS Code Remote-SSH
GPU training machine	Windows RTX desktop; hostname seen as BrojoGame
RTX GPU	NVIDIA GeForce RTX 5080
Working CUDA Python	C:\Users\brojo\miniconda3\envs\ml\python.exe
Project folder on GPU PC	D:\ml_projects\cifar10_cuda_edge
Jetson SSH user	brojonano
Jetson IP used	100.74.213.67
Jetson target folder	/home/brojonano/Documents/ai_lab
Model files produced	cifar10_cnn_cuda.pth and cifar10_cnn_cuda.onnx

Contents

1. Overall workflow

```
MacBook VS Code
   ↓ Remote-SSH
Windows RTX GPU desktop (BrojoGame)
   ↓ PyTorch CUDA training
cifar10_cnn_cuda.pth
   ↓ ONNX export
cifar10_cnn_cuda.onnx
   ↓ scp file transfer
Jetson Orin Nano (brojonano@100.74.213.67)
   ↓ TensorRT FP16 conversion and benchmark
cifar10_cnn_fp16.engine
```

Important: Do not train on the MacBook for this project. The MacBook controls the GPU PC remotely. The GPU PC

performs CUDA training.

2. Open VS Code on MacBook and connect to the RTX GPU PC

- Open VS Code on the MacBook.
- Install the extension named Remote - SSH if it is not already installed.
- Press Command + Shift + P.
- Search for Remote-SSH: Connect to Host.
- Connect to your GPU PC using its Windows SSH username and IP address.

```
ssh brojogpu@<GPU_PC_IP>
```

Expected remote identity: In this session the remote terminal showed hostname BrojoGame and whoami as brojogame\brojogpu. This is okay because the working Python environment under C:\Users\brojo was still accessible.

3. Verify correct remote machine, folder, Python, and CUDA

3.1 Check that the VS Code terminal is running on the GPU PC

```
hostname
whoami
Expected example output:
BrojoGame
brojogame\brojogpu
```

3.2 Go to the working project folder

```
D:
cd D:\ml_projects\cifar10_cuda_edge
```

Folder used in the successful run: Your successful terminal prompt was PS D:\ml_projects\cifar10_cuda_edge>. Use this folder for this project to avoid confusion.

3.3 Verify the correct existing Python environment

```
Test-Path "C:\Users\brojo\miniconda3\envs\ml\python.exe"
Expected output:
True
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" -c "import torch; print('CUDA available:',
torch.cuda.is_available()); print('GPU:', torch.cuda.get_device_name(0)); print('CUDA:',
torch.version.cuda)"
Expected successful output:
CUDA available: True
GPU: NVIDIA GeForce RTX 5080
CUDA: 12.8
```

Do not reinstall everything: If the above command works, do not reinstall PyTorch or CUDA. Use the exact python.exe path directly when Conda activation is inconvenient.

4. Create or open the training script in D drive

- In the VS Code Remote-SSH terminal connected to the GPU PC, go to the project folder.
- Use the code command to open a .py file in the MacBook VS Code editor. Do not use Notepad for Remote-SSH work because Notepad may open on the GPU PC screen.

```
D:
cd D:\ml_projects\cifar10_cuda_edge
code train_cifar10_cuda.py
```

- Paste the training code into the opened editor.

- Save with Command + S on the MacBook keyboard.
- If you accidentally paste extra words, remove them in the editor, then save again.

4.1 Check syntax before running

```
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" -m py_compile train_cifar10_cuda.py
```

Syntax check result: If there is no output, the Python syntax is okay. If there is an error, VS Code will show the line number; fix that line and run the syntax check again.

5. Run CUDA training on RTX 5080

```
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" train_cifar10_cuda.py
```

5.1 Expected successful output

```
Fresh CIFAR-10 CUDA Training
Using device: cuda
GPU: NVIDIA GeForce RTX 5080
CUDA version: 12.8
Epoch 1/8 | Loss: ... | Train Acc: ... | Test Acc: ... | Time: ...s
...
Epoch 8/8 | Loss: 111.3177 | Train Acc: 80.44% | Test Acc: 80.16% | Time: 10.17s
Saved PyTorch model: cifar10_cnn_cuda.pth
```

Result achieved: The project successfully trained on the RTX 5080 GPU and reached about 80% CIFAR-10 test accuracy in 8 epochs. The exact accuracy can vary slightly between runs.

5.2 Optional: monitor GPU usage during training

- Open a second terminal in the same VS Code Remote-SSH window and run:

```
nvidia-smi -l 1
```

What to look for: You should see python.exe using GPU memory and nonzero GPU utilization while training is running.

6. Fix ONNX export dependency and export ONNX

Observed issue: Training succeeded, but ONNX export initially failed with ModuleNotFoundError: No module named onnxscript. This is only an ONNX export dependency problem, not a GPU problem.

6.1 Install only the missing export packages

```
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" -m pip install onnxscript onnx
```

6.2 Create export-only script

```
code export_onnx_cifar10.py
```

- Paste the export script from Appendix B and save with Command + S.

6.3 Run ONNX export only

```
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" export_onnx_cifar10.py
Expected output:
Using device: cuda
Saved ONNX model: cifar10_cnn_cuda.onnx
```

DeprecationWarning is okay: The legacy TorchScript-based ONNX export warning can be ignored for this first project because the ONNX file was saved successfully.

7. Run PyTorch GPU inference test on RTX

```
code infer_cifar10_cuda.py
```

- Paste the inference script from Appendix C and save. Then run:

```
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" infer_cifar10_cuda.py
Actual successful output from this session:
Using device: cuda
GPU: NVIDIA GeForce RTX 5080
True class: cat
Predicted class: cat
Average PyTorch GPU inference time: 0.44554972648620605 ms
```

Meaning: This confirms the trained model can be loaded from cifar10_cnn_cuda.pth and run on the RTX GPU for inference.

8. Transfer ONNX model from RTX GPU PC to Jetson Orin Nano

- Run the scp command from the RTX GPU Remote-SSH terminal.
- The correct Jetson username in the successful transfer was brojonano, not jetson.
- The correct target folder was /home/brojonano/Documents/ai_lab, not /home/jetson.

```
scp cifar10_cnn_cuda.onnx brojonano@100.74.213.67:/home/brojonano/Documents/ai_lab
Expected successful output:
cifar10_cnn_cuda.onnx                                         100% 3184KB   2.7MB/s   00:01
```

If prompted about host authenticity: Type yes once. This adds the Jetson host key to the known hosts list. This is normal for the first SSH/SCP connection.

9. Connect to Jetson and prepare TensorRT step

- Open a second VS Code window on MacBook.
- Use Remote-SSH: Connect to Host.
- Connect to the Jetson using the correct username and IP address.

```
ssh brojonano@100.74.213.67
cd ~/Documents/ai_lab
ls -lh
Expected file:
cifar10_cnn_cuda.onnx
```

9.1 Find TensorRT trtexec on Jetson

```
which trtexec
find /usr -name trtexec 2>/dev/null
Common Jetson path:
/usr/src/tensorrt/bin/trtexec
```

9.2 Convert ONNX to TensorRT FP16 engine

```
/usr/src/tensorrt/bin/trtexec \
  --onnx=cifar10_cnn_cuda.onnx \
  --saveEngine=cifar10_cnn_fp16.engine \
  --fp16
Expected ending:
&&&& PASSED TensorRT.trtexec
ls -lh
Expected files:
cifar10_cnn_cuda.onnx
cifar10_cnn_fp16.engine
```

9.3 Benchmark TensorRT engine

```
/usr/src/tensorrt/bin/trtexec --loadEngine=cifar10_cnn_fp16.engine
```

9.4 Optional performance mode before benchmarking

```
sudo nvpmodel -m 0
sudo jetson_clocks
/usr/src/tensorrt/bin/trtexec --loadEngine=cifar10_cnn_fp16.engine
```

10. Troubleshooting notes from this exact session

Problem seen	Reason	Correct fix
Conda activation confusion	SSH user showed brojogame\brojogpu but Miniconda exists under C:\Users\brojo	Use full Python path: C:\Users\brojo\miniconda3\envs\ml\python.exe
FileNotFoundError: cifar10_cnn_cuda.pth	Inference code was run before training created the .pth file	Run the fresh training script first
No module named onnxscript	Missing ONNX export dependency	Install onnxscript and onnx in the same ml environment
Permission denied for jetson@100.74.213.67	Wrong Jetson username	Use brojonano@100.74.213.67
dest open /home/jetson failure	Wrong target home folder	Use /home/brojonano/Documents/ai_lab
VisibleDeprecationWarning from torchvision CIFAR	Library warning during CIFAR loading	Ignore for now; training and inference worked
Cannot paste into Notepad/editor	Notepad can open on the remote Windows screen or focus may be wrong	Use code filename.py in VS Code Remote-SSH editor

Final checklist before moving to next project

- VS Code Remote-SSH from MacBook to GPU PC works.
- GPU PC terminal shows hostname BrojoGame.
- CUDA check shows NVIDIA GeForce RTX 5080.
- Training script produces cifar10_cnn_cuda.pth.
- Export script produces cifar10_cnn_cuda.onnx.
- Inference script predicts a CIFAR-10 class on GPU.
- SCP transfers cifar10_cnn_cuda.onnx to /home/brojonano/Documents/ai_lab on Jetson.
- Jetson can see the ONNX file with ls -lh.
- TensorRT conversion is the next edge-deployment step.

Appendix A. Full training script: train_cifar10_cuda.py

```
import time
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

EPOCHS = 8
BATCH_SIZE = 256
LEARNING_RATE = 0.001

class SmallCIFARCNN(nn.Module):
    def __init__(self):
        super().__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),

            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x


def main():
    print("Fresh CIFAR-10 CUDA Training")

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("Using device:", device)

    if device.type == "cuda":
        print("GPU:", torch.cuda.get_device_name(0))
        print("CUDA version:", torch.version.cuda)

    transform_train = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomCrop(32, padding=4),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=(0.4914, 0.4822, 0.4465),
            std=(0.2470, 0.2435, 0.2616)
```

```
    )
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(
        mean=(0.4914, 0.4822, 0.4465),
        std=(0.2470, 0.2435, 0.2616)
    )
])

train_dataset = datasets.CIFAR10(
    root="./data",
    train=True,
    download=True,
    transform=transform_train
)

test_dataset = datasets.CIFAR10(
    root="./data",
    train=False,
    download=True,
    transform=transform_test
)

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2,
    pin_memory=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)

model = SmallCIFARCNN().to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

for epoch in range(1, EPOCHS + 1):
    start_time = time.time()

    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images = images.to(device, non_blocking=True)
        labels = labels.to(device, non_blocking=True)

        optimizer.zero_grad()
        outputs = model(images)
        loss = loss_fn(outputs, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        predicted = outputs.argmax(dim=1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_acc = 100.0 * correct / total

    model.eval()
    test_correct = 0
    test_total = 0

    with torch.no_grad():
        for images, labels in test_loader:
```

```
                images = images.to(device, non_blocking=True)
                labels = labels.to(device, non_blocking=True)

                outputs = model(images)
                predicted = outputs.argmax(dim=1)

                test_correct += (predicted == labels).sum().item()
                test_total += labels.size(0)

        test_acc = 100.0 * test_correct / test_total
        epoch_time = time.time() - start_time

        print(
            f"Epoch {epoch}/{EPOCHS} | "
            f"Loss: {running_loss:.4f} | "
            f"Train Acc: {train_acc:.2f}% | "
            f"Test Acc: {test_acc:.2f}% | "
            f"Time: {epoch_time:.2f}s"
        )

    torch.save(model.state_dict(), "cifar10_cnn_cuda.pth")
    print("Saved PyTorch model: cifar10_cnn_cuda.pth")

    model.eval()
    dummy_input = torch.randn(1, 3, 32, 32).to(device)

    torch.onnx.export(
        model,
        dummy_input,
        "cifar10_cnn_cuda.onnx",
        input_names=["input"],
        output_names=["output"],
        opset_version=17
    )

    print("Saved ONNX model: cifar10_cnn_cuda.onnx")


if __name__ == "__main__":
    main()
```

Appendix B. Full ONNX export script: export_onnx_cifar10.py

```
import torch
import torch.nn as nn

class SmallCIFARCNN(nn.Module):
    def __init__(self):
        super().__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),

            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x


def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("Using device:", device)

    model = SmallCIFARCNN().to(device)
    model.load_state_dict(torch.load("cifar10_cnn_cuda.pth", map_location=device))
    model.eval()

    dummy_input = torch.randn(1, 3, 32, 32).to(device)

    torch.onnx.export(
        model,
        dummy_input,
        "cifar10_cnn_cuda.onnx",
        input_names=["input"],
        output_names=["output"],
        opset_version=17,
        dynamo=False
    )

    print("Saved ONNX model: cifar10_cnn_cuda.onnx")


if __name__ == "__main__":
    main()
```

Appendix C. Full inference script: infer_cifar10_cuda.py

```
import time
import torch
import torch.nn as nn
from torchvision import datasets, transforms

CLASSES = [
    "airplane", "automobile", "bird", "cat", "deer",
    "dog", "frog", "horse", "ship", "truck"
]

class SmallCIFARCNN(nn.Module):
    def __init__(self):
        super().__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),

            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x


def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("Using device:", device)

    if device.type == "cuda":
        print("GPU:", torch.cuda.get_device_name(0))

    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(
            mean=(0.4914, 0.4822, 0.4465),
            std=(0.2470, 0.2435, 0.2616)
        )
    ])

    test_dataset = datasets.CIFAR10(
        root="./data",
        train=False,
```

```
        download=True,
        transform=transform_test
    )

    model = SmallCIFARCNN().to(device)
    model.load_state_dict(torch.load("cifar10_cnn_cuda.pth", map_location=device))
    model.eval()

    image, label = test_dataset[0]
    image = image.unsqueeze(0).to(device)

    with torch.no_grad():
        for _ in range(50):
            _ = model(image)

        if device.type == "cuda":
            torch.cuda.synchronize()

        start = time.time()

        for _ in range(1000):
            output = model(image)

        if device.type == "cuda":
            torch.cuda.synchronize()

        end = time.time()

    predicted = output.argmax(dim=1).item()

    print("True class:", CLASSES[label])
    print("Predicted class:", CLASSES[predicted])
    print("Average PyTorch GPU inference time:", ((end - start) / 1000) * 1000, "ms")


if __name__ == "__main__":
    main()
```