So for running DNN workload on the new raspberry pi5 , I need to flash the SD card with the OS. I have downloaded and installed the official Raspberry Pi Imager. I have used a 64 GB microSD card ,a card reader and installed the Raspberry Pi (64 bit) operating system using RPi Imager on Windows . After writing the Raspberry Pi OS to SD card ,I booted Rpi by inserting SD card into it and connecting Rpi to monitor ,keyboard,mouse for accessability ,Ethernet cable for internet and Plug the Rpi in to power it on .

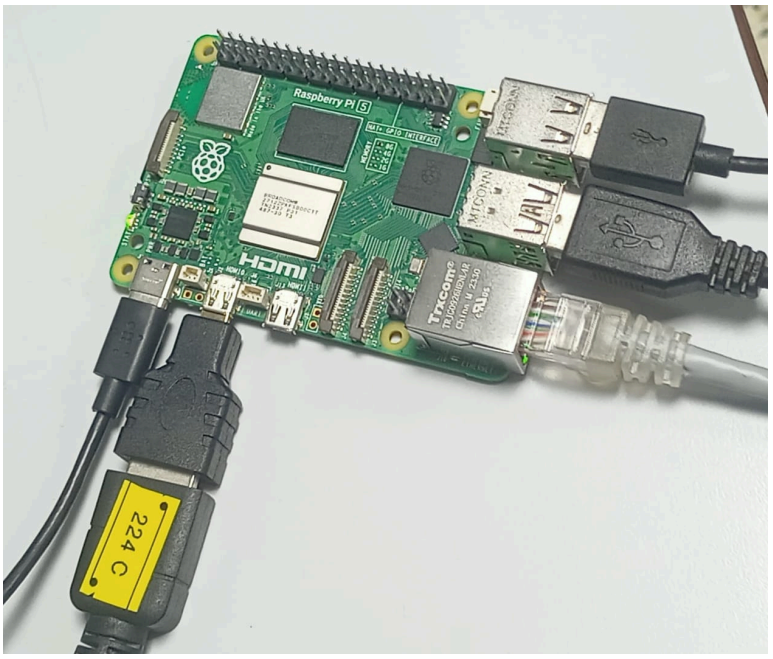


Fig : Raspberry Pi 5 setup

I had used the Raspberry Pi Imager settings to create a username and password,so I was able to go straight into the desktop environment.We have used nmcli linux command utility for managing network connections in Rpi .We setup a Static IP Address via nmcli to connect with LAN and seamlessly work in remote also .For Ex:

```
vboxuser@itslinuxfoss:~$ sudo nmcli con mod "Wired connection 1" ipv4.addresses 192.
168.1.200/24 ipv4.gateway 192.168.1.0 ipv4.dns 0.0.0.0 ipv4.method manual
vboxuser@itslinuxfoss:~$
```

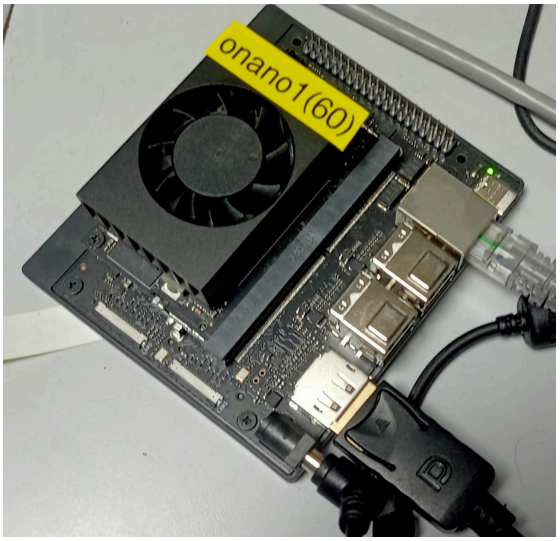
```
$ nmcli con show
```

```
vboxuser@itslinuxfoss:~$ nmcli con show
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  188904bd-0b93-4759-a0ca-8a04827cc226  ethernet  enp0s3
lo                  77dde0a3-eaa9-44a4-b728-bba0b36e5db2  loopback  lo
vboxuser@itslinuxfoss:~$
```

- Jetson AGX Orin :

The NVIDIA Jetson AGX Orin Developer Kit and all Jetson Orin modules share one System-on-Chip (SoC) architecture. This enables the developer kit to emulate any of the modules and makes it easy to start developing the next AI-powered product. Jetson AGX Orin modules deliver up to 275 TOPS of AI performance with power configurable between 15W and 60W. This gives up to 8X the performance of Jetson AGX Xavier in the same compact form factor. Jetson AGX Orin is available in 64GB ,32GB, and Industrial versions.

- Jetson Orin Nano:



The NVIDIA Jetson Orin Nano Developer Kit sets a new standard for creating entry-level AI-powered robots, smart drones, and intelligent cameras,it also simplifies the process of starting with the Jetson Orin Nano series. Jetson Orin Nano series modules deliver up to 40 TOPS of AI performance in the smallest Jetson form-factor, with power options between 7W and 15W. This gives you up to 80X the performance of NVIDIA Jetson Nano. Jetson Orin Nano is available in 8GB and 4GB versions.

As we discussed in the Performance Metrics section,below is the implemented Pseudocode snippet of training function for a given DNN workload in which Fetch stall, GPU compute and end-to-end times for every training minibatch is

```
def train(model: torch.nn.Module, train_loader: DataLoader, epoch_fname: Any, fetch_fname: Any,
compute_fname: Any, vmtouch_fname: Any, dataset_outer_folder: str, reference_time: float, epochs:
int) -> None:

    model.train()
    criterion = CrossEntropyLoss()  # type: ignore
    optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)  # type: ignore
    vmtouch_dir = join_path(dataset_outer_folder, 'vmtouch_output/')  # type: ignore
    start1, end1, start2, end2, start3, end3 = create_cuda_events()  # type: ignore

    start_time = get_current_time()  # type: ignore
    epoch_count = 0
    img_idx = batch_size - 1  # type: ignore
    stabilization_time = 15
    num_epochs = 5
    for epoch in range(num_epochs):
        print('Epoch:', epoch, 'Begins')
        record_event(start1)
        record_event(start2)
        epoch_start_time = get_current_time()
        batch_count = 0

        for batch_idx, (images, labels) in enumerate(train_loader):
            current_time = get_current_time()
            print('Current time:', current_time)
            if current_time - epoch_start_time > stabilization_time:
                batch_count += 1
            if batch_count == MAX_BATCH_COUNT:
                break

            print('Batch index =', batch_idx)
            images, labels = move_to_device(images, DEVICE), move_to_device(labels, DEVICE)
            record_event(end2)
            record_event(start3)

            optimizer.zero_grad()
            loss = criterion(model(images), labels)
            loss.backward()
            optimizer.step()
            record_event(end3)
            synchronize_cuda()
            record_event(end1)
            synchronize_cuda()
            fetch_time = elapsed_time(start2, end2)
            compute_time = elapsed_time(start3, end3)
            epoch_time = elapsed_time(start1, end1)

            log_info(fetch_fname, epoch, batch_idx, fetch_time, current_time - reference_time)
            log_info(compute_fname, epoch, batch_idx, compute_time, current_time - reference_time)
            log_info(epoch_fname, epoch, epoch_time, current_time - reference_time)

            record_event(start1)
            record_event(start2)
            img_idx += batch_size
        print('Epoch:', epoch, 'Ends')
        epoch_count += 1
```

measured.

4.4) Generalizability to another device

The Jetson devices come with a number of predefined power modes that users can choose from.We use Unix utility used to read information about the frequency scaling .Specifically, it lists the available frequencies for the first CPU (cpu0) on a Linux system for Jetson Device . We configured a custom power mode by specifying the number of CPU cores enabled, and the frequencies of CPU, GPU and RAM (External Memory Controller (EMC)). The power mode can be changed on the fly, without any system downtime.

To save results for each power mode, we iteratively change the CPU frequency rather than running all power modes in one execution. This way, if an error occurs, we can identify which CPU frequency caused the error, discard it, and restart with that particular CPU frequency again.Below I have provided a sample script file as pseudo code for training.

```
# Define parameters
prefetch_factor = 2
no_workers = [4]
external_device = "nvme0n1"
batch_size = 16

# Remove previous log and result files
remove_files("mn_nw*")
remove_files("log_file_*")

#Available CPU frequencies fro next run
#available_frequencies = [c2,c3,c4,c4,c5,c6,c7,c8,c9.c10]

# Define CPU, GPU, and memory frequency values
cpu_cores_values = [2, 4, 6]
cpu_frq_values = [c1]
gpu_frq_values = [g1, g2, g3]
mem_frq_values = [m1, m2]

# Iterate over all combinations of CPU, GPU, and memory frequencies
for cpu_frq in cpu_frq_values:
    for gpu_frq in gpu_frq_values:
        for cpu_cores in cpu_cores_values:
            for mem_frq in mem_frq_values:

                # Generate and apply the power mode configuration
                generate_power_mode(cpu_cores, cpu_frq, gpu_frq, mem_frq)
                apply_power_mode()
                enable_fan_and_clocks()
                log_current_configuration()

                # Run the training script and log the output
                run_training_script(no_workers, prefetch_factor, external_device)
                kill_running_python_processes()

                # Save results and logs for the current configuration
                save_results_and_logs(cpu_cores, cpu_frq, gpu_frq, mem_frq)
```

CHAPTER - 5 Result and Analysis

5.1) Performance Analysis of Hardware Devices

Each DNN workload runs on all Hardware devices except BERT on Raspberry pi5. The minibatch time,fetch time and GPU compute time are reported . Every training workload is run for 5 epochs except raspberry pi 5 run for epoch 0 (we discussed the reason in the performance metrics section).We calculated Average epoch times across all devices as a summation of minibatch time for all epochs divided by number of epochs and report the average epoch times .Since we were running few epochs we were iterating through all the batches.We configure PyTorch's Dataloader to run with 4 worker processes to enable pipelined and parallel data fetch and pre-processing . We use a batch size of 16 for all training workloads .

We observe that the NVIDIA RTX 3090 is significantly faster than the NVIDIA RTX A5000, which can be explained by the 3090 having more CUDA cores of the same Ampere generation (10, 496 vs. 8, 192). The increased number of CUDA cores in the 3090 allows for more parallel processing power

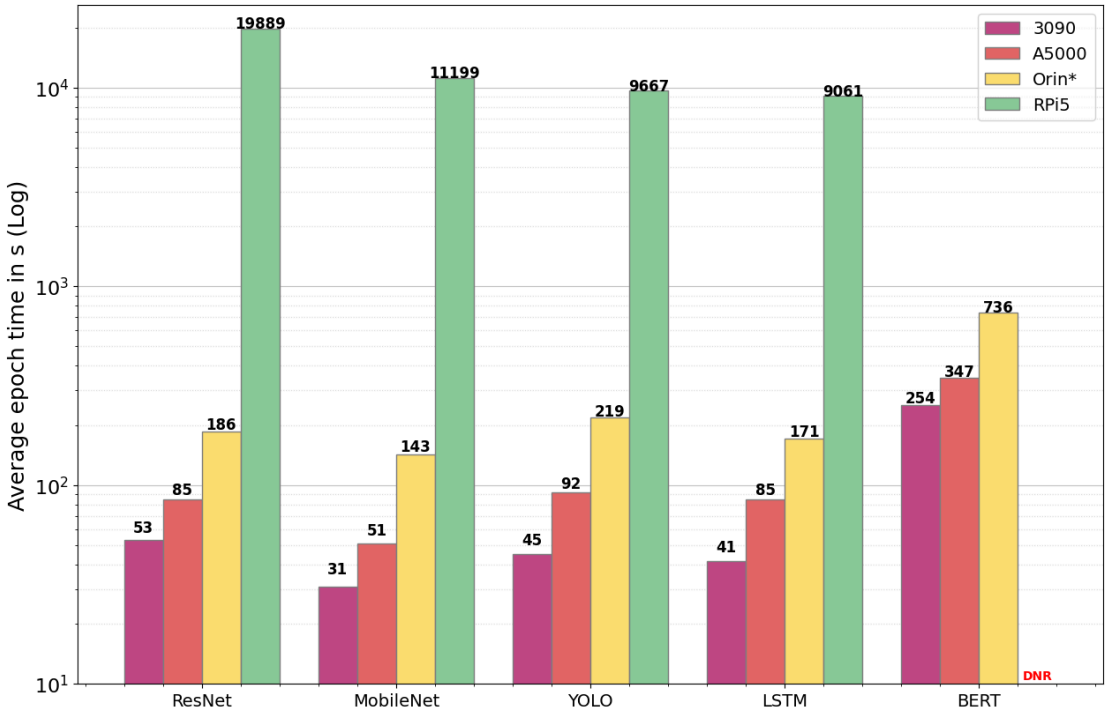


Fig: Average epoch times across various edge and server devices.

The NVIDIA Jetson AGX Orin is slower than both the A5000 and the 3090 as this is reflected in its longer average epoch time shown in figure. Despite being based on the same Ampere architecture, the reduced core count 2048 of Orin AGX , resulted in lower overall performance .

The Raspberry Pi 5 has the lowest average epoch time among all the hardware devices tested.It trains using just the ARM CPU cores, and is two orders of magnitude slower than the Orin. BERT was unable to run on the RPi5 (DNR in Figure) as it ran out of memory in 8GB of memory available in the RPi

	3090	A5000	RPi5	Orin AGX
CPU	16 core 12th Gen Intel i9-12900K	32 core AMD EPYC 7532	4 core ARM Cortex-A76	12 core ARM A78AE
Max CPU frequency (MHz)	5200	2400	2400	2200
GPU(arch/CUDA cores)	Ampere ,10496	Ampere,8192	VideoCore VII (graphics only)	Ampere , 2048
Max GPU frequency (MHz)	2115	1695	800	1300
RAM (GB)	128 (CPU) + 24 (GPU)	512 (CPU) + 24 (GPU)	8	64 (shared)
Peak Power (W)	350	230	27	60

.

Table: Specifications of Server GPUs and Edge Devices

5.2) Training on Jetson Orin Nano

We ran two DNN workloads ResNet and MobileNet on Jetson Orin Nano. We configured a custom power mode by specifying the number of CPU cores enabled, and the frequencies of CPU, GPU and RAM (External Memory Controller (EMC)). We choose 180 of the available 1800 power modes for profiling data for ResNet and MobileNet workloads .

Using the ResNet workload trained on Orin AGX as the reference model for prediction, we transfer-learn to Orin Nano by retraining using 50 random power modes for ResNet (or MobileNet).We validate the predictions of the transferred model using all 180 power modes we profile for the workload.we observe low median time and power errors, with MAPEs of 7.85% and 5.96% for ResNet, and 8.98% and 4.72% for MobileNet. This confirms the strong

generalizability of PowerTrain to even Jetson accelerators that are 6.9× less powerful than the Orin AGX.