

SIT724

LLM-CCA (Large Language Model for Congestion Control Algorithms) is a project to optimise TCP congestion control using Large Language Model (LLM). It dynamically adjusts the Congestion Window (CWND) by analysing network state data (e.g., RTT, CWND, and throughput) to improve network performance in complex network environments and high load conditions. The goal of LLM-CCA is to improve the responsiveness and adaptability of traditional TCP congestion control algorithms to reduce latency and packet loss, while maintaining efficient operation on hardware resource constrained devices.

Hardware and software required for the project:

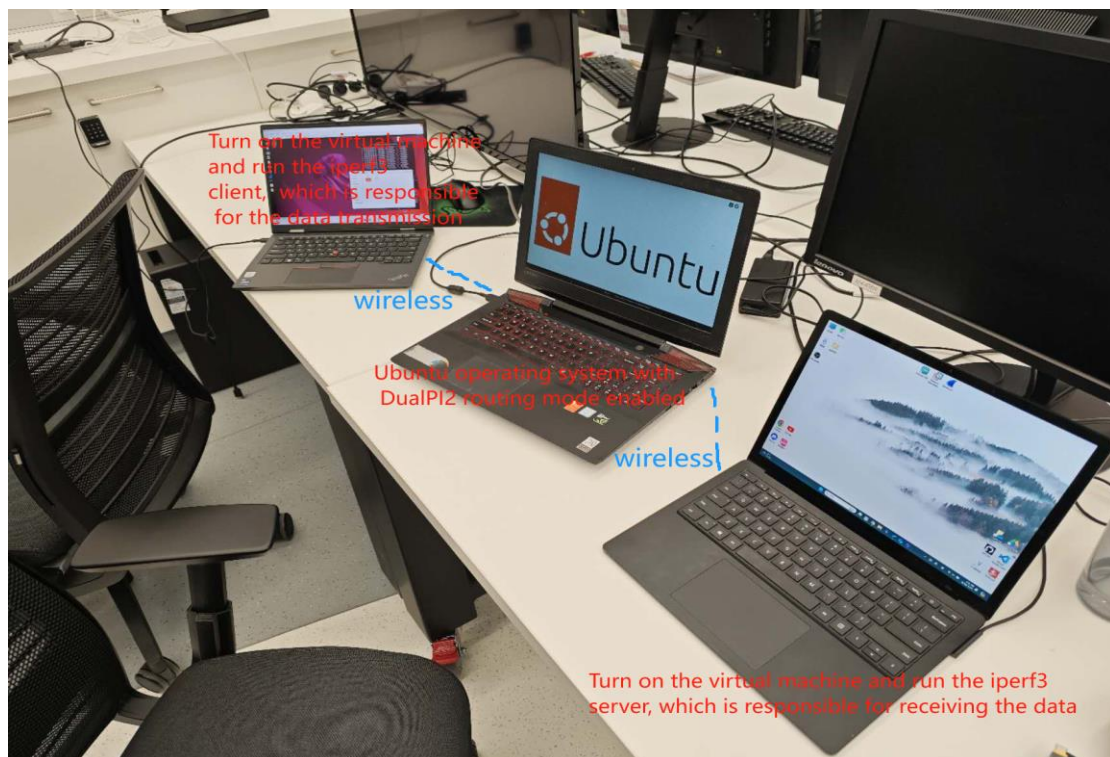
Hardware: 2 Windows computers, 1 Ubuntu system computer.

Software: VMware Pro 16, iperf3, L4S, Runpod, NetLLM

A virtual machine needs to be installed on 2 windows computers!

L4S needs to be installed on all 3 computers

Testbed



This experiment simulates the data transfer process and collects data by installing L4S for all 3 computers and using iperf3. Due to hardware constraints, we used Runpod to train and run our NetLLM.

How to reproduce experiment:

Step 0:

Client, Server, and Router within the tutorial refer to roles in the experiment structure, not items.

The SIT723 documentation has detailed how Client, Server and Router work with each other, and is currently replacing the L4S framework (replacing CCA and AQM) within the experiment structure of SIT723, which still has 2 Clients, 1 Server and 1 Router roles. tCP Prague is the CCA for L4S. DualPI2 is the AQM for L4S.

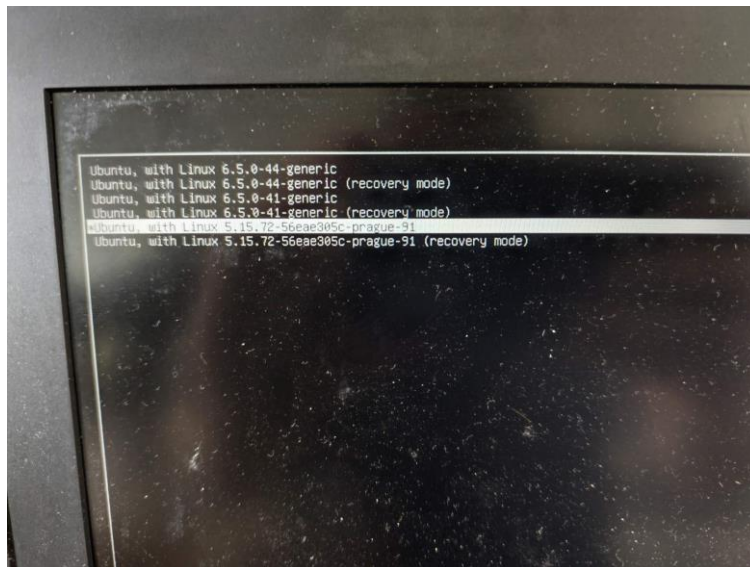
Step 1:

Installation of the L4S framework: <https://github.com/L4STeam/linux>

Please note that you need to install the L4S framework on the Client, Server, and Router.

The L4S framework involves modifying the kernel, so you must ensure that the kernel version of your Ubuntu virtual machine or Ubuntu system has a name similar to Ubuntu with Linux 5.15.72-56eae305c-prague-91.

After installing the L4S framework on the virtual machine, you will need to manually select the kernel boot option. During the virtual machine's startup, continuously press Esc until you are taken to a page where you can select the kernel. If you end up on the desktop, you'll need to reboot and try again. My laptop, which replaces the router, is a dual-boot system with Windows and Ubuntu, and it allows me to select the operating system at startup directly.



Checkpoint: At this point your Client, Server, and Router should have finished installing the L4S framework.

Please use `sysctl net.ipv4.tcp_available_congestion_control` on both Client and Server side to check if TCP Prague is enabled.

As the Router, you can either use a Linux-supported router or a Linux-based computer (but you will need to manually switch it to router mode).

At this point, use the command:

ip link show

to check how many network interfaces your device has. Linux laptops usually have two (corresponding to WiFi and Ethernet).

Next, use:

tc qdisc show dev [port name]

For example: `tc qdisc show dev eth0`

to check if AQM rules are enabled on your network interface.

You should see something like:

qdisc [AQM name] [number]: root refcnt 2 limit 1000p target 15ms ...

Here, **[AQM name]** should be **DualPI2**, but by default, it might be **fq**.

If you don't see **DualPI2**, check if **DualPI2** is enabled with:

lsmod | grep dualpi2

You should see the following output:

```
Sch_dualpi2      24576      3
```

Next, replace the AQM rule with:

sudo tc qdisc replace dev [port name] root dualpi2

Run the command again

tc qdisc show dev [port name]

You should now see **dualpi2** enabled on the network interface

Checkpoint: At this point, your Client, Server, and Router have all installed the L4S framework. 2 VMs will be enabled on the Client side, 1 VM supporting the L4S architecture, and the other VM not supporting the L4S architecture (Reno, CUBIC, and BBR VMs from SIT723). The VM that supports L4S architecture needs to verify that the TCP Prague algorithm is available. Secondly, your Router has enabled DualPI2 rules into the NIC and ensured that this NIC will be used by the accessed device (e.g., DualPI2 rules are enabled for the Ethernet NIC, but your device is still using a Wi-Fi connection).

Step 2:

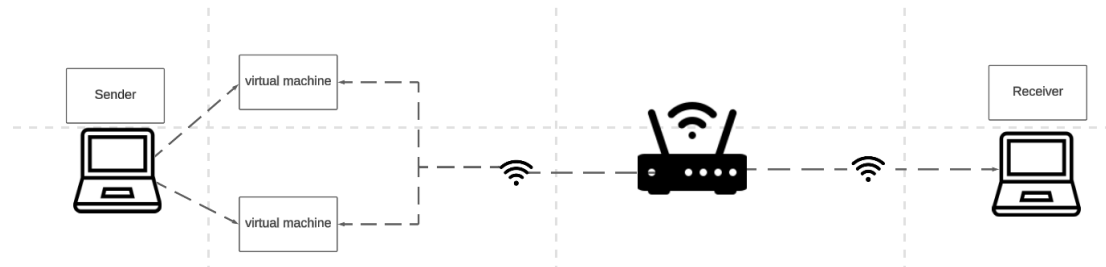
Begin verifying that the L4S architecture is working properly.

Re-establish the experimental architecture structure for SIT723.

Client X

<- - - - > **Router** <- - - - > **Server**

Client Y



Client X supports L4S architecture, Client Y does not support L4S architecture, Router supports L4S architecture. The reason Server also needs to install L4S architecture is that it needs to have ECN enabled and TCP Prague installed. Otherwise, Client X will show that Server does not support this congestion algorithm when it tries to connect.

Server:

iperf3 -s -p 3000

iperf3 -s -p 3001

Client X:

iperf3 -c [ip address] -p 3000 -C prague -tinf

Client Y:

iperf3 -c [ip address] -p 3001 -C cubic -tinf

[ip address] is the IP address of your Server side.

After iperf3 connects properly, please keep Clients connected to Server. Then start Wireshark on the Server side or Router side, after opening Wireshark enter `ip.dsfield.ecn == 3`

Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=1 Acks=1 Wm=582 Len=1436 Tsv=12748546778 TSecr=4279516352 E1B=1 ECER=0 EEOB=1
2	0.000160206	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294942885 Wm=24568 Len=0 Tsv=1279517944 TSecr=2748546766 E1B=5162110 ECER=0 EEOB=1
3	0.0002545514	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294945757 Wm=24568 Len=0 Tsv=1279517951 TSecr=2748546766 E1B=5164982 ECER=0 EEOB=1
4	0.0002703690	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294947193 Wm=24568 Len=0 Tsv=1279517952 TSecr=2748546767 E1B=5166488 ECER=0 EEOB=1
5	0.000398473	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=1437 Acks=1 Wm=582 Len=1436 Tsv=12748546781 TSecr=4279516352 E1B=1 ECER=0 EEOB=1
6	0.004399137	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294948629 Wm=24559 Len=0 Tsv=1279517952 TSecr=2748546768 E1B=5167854 ECER=0 EEOB=1
7	0.004707419	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=2873 Acks=1 Wm=582 Len=1436 Tsv=12748546781 TSecr=4279516352 E1B=1 ECER=0 EEOB=1
8	0.005715586	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=4309 Acks=1 Wm=582 Len=1436 Tsv=12748546782 TSecr=4279516359 E1B=1 ECER=0 EEOB=1
9	0.006545421	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=5745 Acks=1 Wm=582 Len=1436 Tsv=12748546782 TSecr=4279516359 E1B=1 ECER=0 EEOB=1
10	0.007196106	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294951581 Wm=24568 Len=0 Tsv=1279517955 TSecr=2748546768 E1B=5170726 ECER=0 EEOB=1
11	0.008083704	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294954373 Wm=24552 Len=0 Tsv=1279517955 TSecr=2748546770 E1B=5173598 ECER=0 EEOB=1
12	0.008346697	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294957245 Wm=24568 Len=0 Tsv=1279517958 TSecr=2748546771 E1B=5176470 ECER=0 EEOB=1
13	0.008877479	192.168.4.14	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=7181 Acks=1 Wm=582 Len=1436 Tsv=12748546783 TSecr=4279516359 E1B=1 ECER=0 EEOB=1
14	0.009734906	192.168.4.14	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=8617 Acks=1 Wm=582 Len=1436 Tsv=12748546786 TSecr=4279516362 E1B=1 ECER=0 EEOB=1
15	0.011517582	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=10053 Acks=1 Wm=582 Len=1436 Tsv=12748546786 TSecr=4279516363 E1B=1 ECER=0 EEOB=1
16	0.012007778	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=11489 Acks=1 Wm=582 Len=1436 Tsv=12748546787 TSecr=4279516363 E1B=1 ECER=0 EEOB=1
17	0.012908512	192.168.4.14	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=12925 Acks=1 Wm=582 Len=1436 Tsv=12748546787 TSecr=4279516363 E1B=1 ECER=0 EEOB=1
18	0.013655092	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294960811 Wm=24544 Len=0 Tsv=1279517958 TSecr=2748546772 E1B=5179342 ECER=0 EEOB=1
19	0.014174400	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=14361 Acks=1 Wm=582 Len=1436 Tsv=12748546788 TSecr=4279516364 E1B=1 ECER=0 EEOB=1
20	0.014745139	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294961553 Wm=24568 Len=0 Tsv=1279517961 TSecr=2748546773 E1B=5180778 ECER=0 EEOB=1
21	0.014898880	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=4294962989 Wm=24559 Len=0 Tsv=1279517962 TSecr=2748546774 E1B=5182214 ECER=0 EEOB=1
22	0.015048043	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=1 Wm=24568 Len=0 Tsv=1279517966 TSecr=2748546774 E1B=5186522 ECER=0 EEOB=1
23	0.015716264	192.168.4.14	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=15797 Acks=1 Wm=582 Len=1436 Tsv=12748546789 TSecr=4279516365 E1B=1 ECER=0 EEOB=1
24	0.016286967	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=1437 Wm=24552 Len=0 Tsv=1279517967 TSecr=2748546778 E1B=5187958 ECER=0 EEOB=1
25	0.017594134	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=2873 Wm=24568 Len=0 Tsv=1279517970 TSecr=2748546781 E1B=5189394 ECER=0 EEOB=1
26	0.018124203	192.168.4.14	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=17233 Acks=1 Wm=582 Len=1436 Tsv=12748546790 TSecr=4279516367 E1B=1 ECER=0 EEOB=1
27	0.018342389	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=5745 Wm=24568 Len=0 Tsv=1279517973 TSecr=2748546781 E1B=5192266 ECER=0 EEOB=1
28	0.019299999	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=18669 Acks=1 Wm=582 Len=1436 Tsv=12748546790 TSecr=4279516367 E1B=1 ECER=0 EEOB=1
29	0.021071638	192.168.4.10	TCP	1514	56048 → 3000 [ACK, ACEv5] Seq=20105 Acks=1 Wm=582 Len=1436 Tsv=12748546793 TSecr=4279516371 E1B=1 ECER=0 EEOB=1
30	0.021716630	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=7181 Wm=24568 Len=0 Tsv=1279517976 TSecr=2748546782 E1B=5193702 ECER=0 EEOB=1
31	0.022371666	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=8617 Wm=24559 Len=0 Tsv=1279517976 TSecr=2748546783 E1B=5195138 ECER=0 EEOB=1
32	0.023573703	192.168.4.14	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=10053 Wm=24568 Len=0 Tsv=1279517977 TSecr=2748546786 E1B=5196574 ECER=0 EEOB=1

Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp80, id 0
 Ethernet II, Src: Intel_82:eb:a2 (c8:46:28:bb:11:c9), Dst: Intel_82:eb:a2 (c8:34:8e:02:eb:a2)
 Internet Protocol Version 4, Src: 192.168.4.10, Dst: 192.168.4.14
 Transmission Control Protocol, Src Port: 56048, Dst Port: 3000, Seq: 1, Ack: 1, Len: 1436
 Data (1436 bytes)

```

0000  c8 34 8e 02 eb a2 dc 46 28 bb 11 c9 00 00 45 01  f-----F-----E
0010  05 cd 0d a9 40 00 40 00 06 9e 09 c8 a8 0a c8 a8  @-@-@-@-@-@-@-@-
0020  04 00 da f0 80 b8 c0 d9 50 c6 12 82 60 cd b1 50  00000000000000000000000000000000000000000000
0030  01 46 da 00 00 01 01 00 0a a0 d3 7d cd f4 14 00  00000000000000000000000000000000000000000000
0040  3c c0 aa 00 00 00 01 00 00 00 00 01 01 01 01 01  00000000000000000000000000000000000000000000
0050  ee 4a 45 83 fe ec 4e 9f 94 16 9d 9c de cd 55 14  JE-N-----U-----
0060  42 8c 1c 77 47 9f ae fa 89 95 82 aa d2 46 2a 41  B-----P-----A
0070  2a c2 da b0 73 7d 99 c2 54 97 68 71 35 8a 20 88  *--s--T--q--
0080  e1 58 40 ce f7 b5 ed b2 a6 da 65 85 17 2d 12  --X-----
0090  c8 bd dd 50 52 85 83 29 80 a2 84 16 79 6f 5a 36  --PR-----yoZF
0100  bd f3 af 09 51 4b 19 27 2a 1f a8 46 90 5c 66  --00000000000000000000000000000000000000000000
0110  7a 7c 7b 7d 7e 7f 80 81 82 83 84 85 86 87 88  00000000000000000000000000000000000000000000

```

As long as you can see the result after you enter it, it means that ECN=3 marked packets are appearing, and it also means that DualPI2 is marking the congested packets with ECN. please let iperf3's connection last longer, in my previous tests congestion appeared at a ratio of about 1:6000 or 20,000th packets before the first congestion marking occurs.

Time	Source	Destination	Protocol	Length	Info
1439..111.241267078	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62234593 Wm=24551 Len=0 Tsv=1279629176 TSecr=2748656101 E1B=312250 ECER=0 EEOB=1
1439..111.251783295	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62236029 Wm=24542 Len=0 Tsv=1279629176 TSecr=2748656102 E1B=313686 ECER=0 EEOB=1
1439..111.252620772	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62237465 Wm=24534 Len=0 Tsv=1279629176 TSecr=2748656104 E1B=315122 ECER=0 EEOB=1
1439..111.253461333	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62238901 Wm=24525 Len=0 Tsv=1279629176 TSecr=2748656105 E1B=316558 ECER=0 EEOB=1
1439..111.254070407	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62240337 Wm=24517 Len=0 Tsv=1279629176 TSecr=2748656106 E1B=317994 ECER=0 EEOB=1
1439..111.254248664	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62241773 Wm=24508 Len=0 Tsv=1279629176 TSecr=2748656107 E1B=319430 ECER=0 EEOB=1
1439..111.264561513	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62246081 Wm=24551 Len=0 Tsv=1279629180 TSecr=2748656111 E1B=323738 ECER=0 EEOB=1
1439..111.264981311	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62247517 Wm=24542 Len=0 Tsv=1279629180 TSecr=2748656112 E1B=325738 ECER=0 EEOB=1
1439..111.265041537	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=62248953 Wm=24534 Len=0 Tsv=1279629180 TSecr=2748656134 E1B=326618 ECER=0 EEOB=1
2425..222.559661377	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=97591133 Wm=24551 Len=0 Tsv=1277490060 TSecr=2748770605 E1B=1213550 ECER=0 EEOB=1
4714..400.752157491	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=191472505 Wm=24526 Len=0 Tsv=12779918700 TSecr=2748948559 E1B=12189508 ECER=0 EEOB=1
4714..400.752755491	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=191473941 Wm=24515 Len=0 Tsv=12779918700 TSecr=2748948562 E1B=12111886 ECER=0 EEOB=1
4813..408.432754213	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=195649829 Wm=24551 Len=0 Tsv=12779926382 TSecr=2748956343 E1B=16286974 ECER=0 EEOB=1
4813..408.432942194	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=195651265 Wm=24542 Len=0 Tsv=12779926382 TSecr=2748956345 E1B=16288418 ECER=0 EEOB=1
5964..497.892052799	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=242328445 Wm=24551 Len=0 Tsv=124280015634 TSecr=2749045869 E1B=12633942 ECER=0 EEOB=1
5964..497.892599086	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=242329081 Wm=24542 Len=0 Tsv=124280015634 TSecr=2749045871 E1B=12635378 ECER=0 EEOB=1
5968..498.190218473	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=242505073 Wm=24551 Len=0 Tsv=124280016138 TSecr=2749046358 E1B=12810978 ECER=0 EEOB=1
5968..498.190368758	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=242506509 Wm=24542 Len=0 Tsv=124280016138 TSecr=2749046360 E1B=12812006 ECER=0 EEOB=1
6823..565.428054656	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278484893 Wm=24524 Len=0 Tsv=124280083373 TSecr=2749113270 E1B=15235158 ECER=0 EEOB=1
6823..565.430038593	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278485489 Wm=24513 Len=0 Tsv=124280083373 TSecr=2749113272 E1B=15236554 ECER=0 EEOB=1
6823..565.430424723	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278486925 Wm=24502 Len=0 Tsv=124280083373 TSecr=2749113275 E1B=15237990 ECER=0 EEOB=1
6823..565.430630927	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278488361 Wm=24491 Len=0 Tsv=124280083373 TSecr=2749113277 E1B=15239426 ECER=0 EEOB=1
6823..565.433640967	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278492669 Wm=24551 Len=0 Tsv=124280083377 TSecr=2749113287 E1B=15243754 ECER=0 EEOB=1
6823..565.434608155	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278494105 Wm=24542 Len=0 Tsv=124280083377 TSecr=2749113289 E1B=15245170 ECER=0 EEOB=1
6823..565.434604383	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278495541 Wm=24534 Len=0 Tsv=124280083377 TSecr=2749113291 E1B=15246606 ECER=0 EEOB=1
6823..565.435316795	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278496977 Wm=24525 Len=0 Tsv=124280083377 TSecr=2749113294 E1B=15248042 ECER=0 EEOB=1
6823..565.435997253	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278498413 Wm=24517 Len=0 Tsv=124280083377 TSecr=2749113308 E1B=15249478 ECER=0 EEOB=1
6823..565.437608700	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=278499849 Wm=24508 Len=0 Tsv=124280083377 TSecr=2749113328 E1B=15250914 ECER=0 EEOB=1
7378..608.160721618	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=381737565 Wm=24551 Len=0 Tsv=124280126101 TSecr=2749155912 E1B=14932390 ECER=0 EEOB=1
7378..608.166851692	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=381737201 Wm=24542 Len=0 Tsv=124280126101 TSecr=2749155915 E1B=14933834 ECER=0 EEOB=1
7642..630.401002073	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=312869073 Wm=24551 Len=0 Tsv=124280148335 TSecr=2749178018 E1B=16065706 ECER=0 EEOB=1
7642..630.40181173	192.168.4.14	192.168.4.10	TCP	78	3000 → 56048 [ACK, ACEv5] Seq=1 Acks=312870599 Wm=24542 Len=0 Tsv=124280148335 TSecr=2749178021 E1B=16067142 ECER=0 EEOB=1

Frame 143939: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface wlp80, id 0
 Ethernet II, Src: Intel_82:eb:a2 (c8:34:8e:02:eb:a2), Dst: Intel_bb:11:c9 (dc:46:28:bb:11:c9)
 Internet Protocol Version 4, Src: 192.168.4.14, Dst: 192.168.4.10
 Transmission Control Protocol, Src Port: 3000, Dst Port: 56048, Seq: 1, Ack: 62234593, Len: 0

```

0000  dc 46 28 bb 11 c9 c8 34 8e 02 eb a2 00 00 45 03  f-----F-----E
0010  00 40 bd 15 40 00 40 06 f4 36 c8 a8 8a c8 a8 a8  @-@-@-@-@-@-@-@-
0020  04 00 08 da f0 12 82 60 cd ca 8a f0 a6 b1 50 00  00000000000000000000000000000000000000000000
0030  5f e7 11 d9 00 00 01 01 08 0a ff 15 f5 78 a3 05  00000000000000000000000000000000000000000000
0040  29 e5 aa 00 04 c3 ba 00 00 00 00 00 01 01 01 01 
```

You can check if there are L4S packets by typing `ip.dsfield.ecn == 1` in the filter section, if there are, then L4S is enabled properly. Of course, you can also double-click the packet to see if it is an L4S packet instead of using the filter.

```
> Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface ens33, id 0
> Ethernet II, Src: VMware_2e:29:97 (00:0c:29:2e:29:97), Dst: Intel_02:eb:a2 (c8:34:8e:02:eb:a2)
< Internet Protocol Version 4, Src: 192.168.4.10, Dst: 192.168.4.14
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  < Differentiated Services Field: 0x01 (DSCP: CS0, ECN: ECT(1))
    0000 00.. = Differentiated Services Codepoint: Default (0)
    .... 01 = Explicit Congestion Notification: ECN-Capable Transport codepoint '01' (1)
  Total Length: 60
  Identification: 0xd904 (55556)
  > 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0xd84d [validation disabled]
```

↓
L4S packet

```
0000 c8 34 8e 02 eb a2 00 0c 29 2e 29 97 08 00 45 01 4.....).)---E
0010 00 3c d9 04 40 00 06 d8 4d c0 a8 04 0a c0 a8 .c.g.g.M.....
0020 04 0e e6 aa 0b b8 7f 9b c6 3b 00 00 00 00 a1 c2 .....;
0030 fa f0 89 97 00 00 02 04 05 b4 04 02 08 0a a2 2c .....
0040 ac 3d 00 00 00 00 01 03 03 07 .....=.....
```

```
> Frame 260508: 2962 bytes on wire (23696 bits), 2962 bytes captured (23696 bits) on interface ens33, id 0
> Ethernet II, Src: VMware_13:e5:65 (00:0c:29:13:e5:65), Dst: Intel_02:eb:a2 (c8:34:8e:02:eb:a2)
< Internet Protocol Version 4, Src: 192.168.4.6, Dst: 192.168.4.14
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  < Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    0000 00.. = Differentiated Services Codepoint: Default (0)
    .... 00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 2948
  Identification: 0x176a (5994)
  > 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: TCP (6)
  Header Checksum: 0x8ea5 [validation disabled]
```

↓
Non-L4S packet

```
0000 c8 34 8e 02 eb a2 00 0c 29 13 e5 65 08 00 45 00 4.....).)---E
0010 0b 84 17 6a 40 00 06 8e a5 c0 a8 04 06 c0 a8 ...jg.g.....
0020 04 0e e5 a2 0b b9 4a d6 55 5c b7 1e 23 e0 80 18 .....J.U\.#...
0030 01 f6 94 db 00 00 01 01 08 0a 74 66 da 6e 4c 16 .....tf.nL
0040 4b b5 a5 8b 86 fa 44 e0 c2 77 f5 4d bc b3 84 3b @....D..w.H...;
0050 db 27 72 a0 7c 26 ae 04 f3 df 77 f2 c7 7f 32 9c 'r|&...w.w...2
0060 ae d4 84 7e fc f8 b8 3a c7 50 a8 72 dc e9 01 59 .....:..P.n...Y
0070 35 89 bf 33 35 54 60 6b d1 f9 82 48 87 eb 4a c6 5..35T'k...H..J
0080 b7 51 b9 0c 6f 9e 7f 0e a1 24 8b 7c 29 af 9a 5e Q..o...$..}...^
0090 d5 59 ec 33 af a3 ff 63 16 9b 67 b9 5f f9 05 0f Y.3...c...g....
00a0 6c 44 54 39 62 2d c4 43 75 24 bc fc cf 51 39 20 1DT9b-C u$...Q9
00b0 1d f4 88 c6 e4 5b 50 a8 85 0a 9c a3 bc dc c6 21 .....[P.....]
00c0 43 1d 02 be a5 f7 8d 61 fd 49 8f c8 2a 0f b4 c1 C.....a..I..*...
00d0 1f 49 ef 09 21 24 5c c9 b3 4a c9 b7 a5 71 6c 15 I..i$\\...j...ql
00e0 79 81 d5 69 c7 27 dd 28 58 f3 22 df 20 fa bf 34 y..i'({X"....4
00f0 e3 d3 c1 df 49 72 5b 48 0e 6f f1 79 79 08 e6 16 ..Ir[H..o..yy...
0100 73 25 7c b9 a3 9f 2f 3a 02 d2 8d b7 8c fb 6d a9 s%|.../:.....m
0110 9b 6e b8 bc 30 2d c4 0c 31 27 eb 69 6f 06 1e 6f ..n~o...1'io..o
0120 eb 37 a1 07 9f ca f4 0d af 2d 1f 79 7e 20 a0 1d 7.....-y...w
0130 47 34 cc 82 be eb c5 1a ca 60 f2 2b bf 2d 95 c3 G4.....+.....
```

Step 2:

Collecting L4S Data

Since this experiment requires collecting data for **CWND**, **RTT**, and **throughput**, and we are using **iperf3** to simulate data transmission, we can directly write a script that includes the **iperf3** command. This script will save all the data during the iperf3 data transmission in a **JSON** file.

1. First, open the virtual machine and create a folder.
2. Open the terminal and create a script file using the command

Command: **nano file_name.sh**

3. Open the .sh file and enter:

Code:

```
JSON_OUTPUT_FILE="iperf3_bbr_output.json"
```

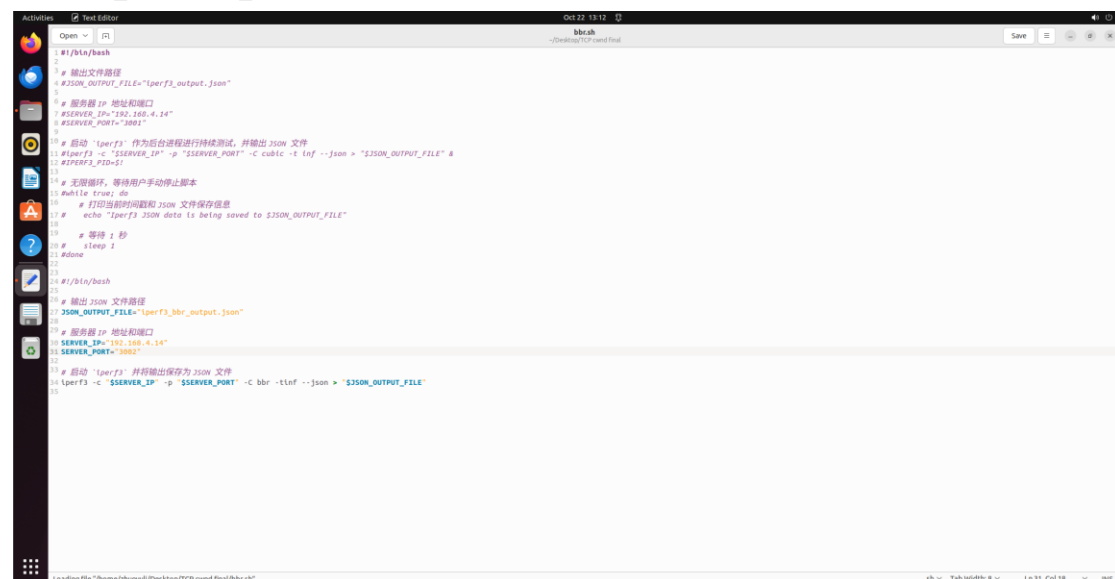
```
SERVER_IP="192.168.4.14"
```

```
SERVER_PORT="3002"
```

```
iperf3 -c "$SERVER_IP" -p "$SERVER_PORT" -C bbr -tinf --json > "$JSON_OUTPUT_FILE"
```

Please replace **SERVER_IP** and **SERVER_PORT** with the server's IP and port. Currently, the script runs using the **BBR congestion control algorithm**. If you want to use a different algorithm, simply modify the **bbr** after the **-C** flag in the last line of the code to the desired congestion control algorithm, such as **Cubic**:

```
iperf3 -c "$SERVER_IP" -p "$SERVER_PORT" -C cubic -tinf --json > "$JSON_OUTPUT_FILE"
```



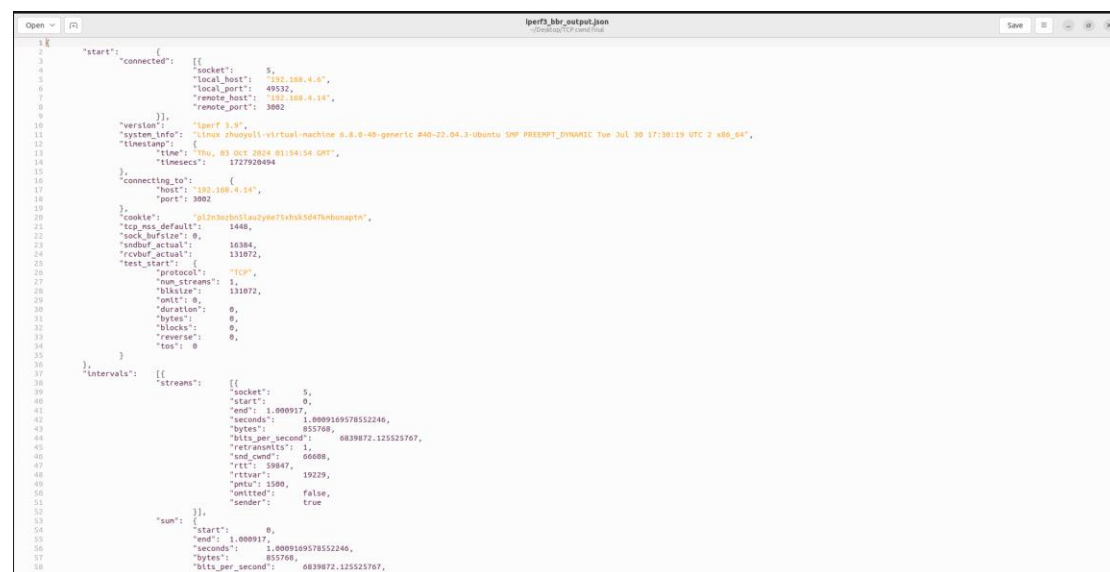
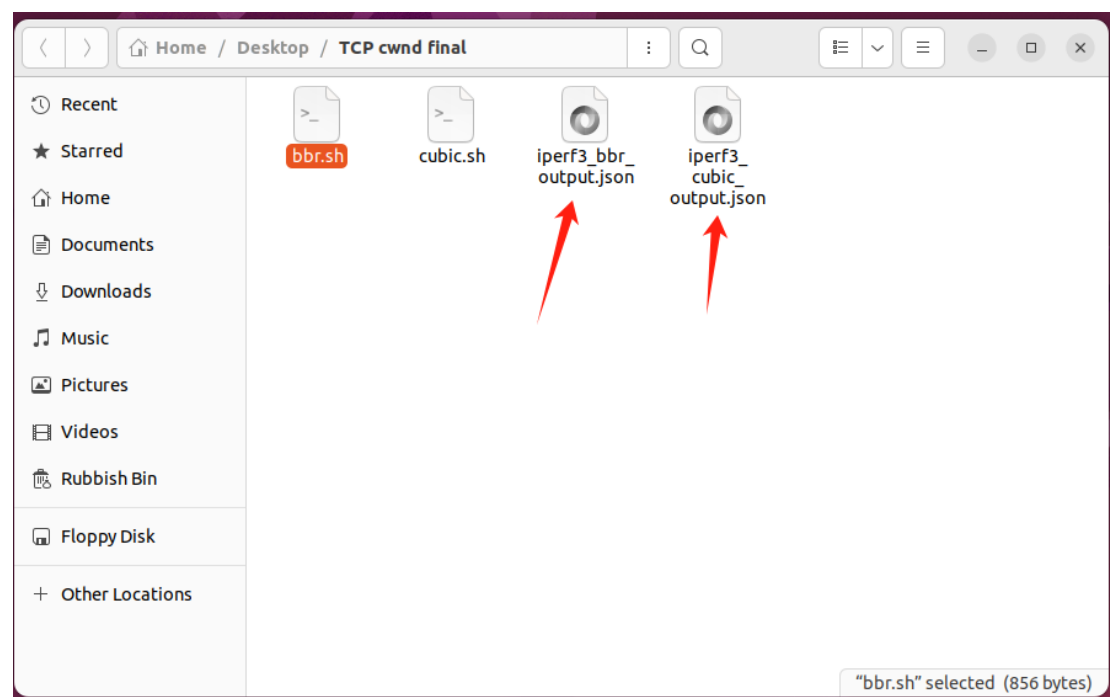
- After completing the script, remember to save it and then enter the following command in the terminal to add execution permissions to the script:

Command: **chmod +x file_name.sh**

- Once everything is set up, you just need to ensure that **iperf3** is running on the server, and then you can run the script with the following command

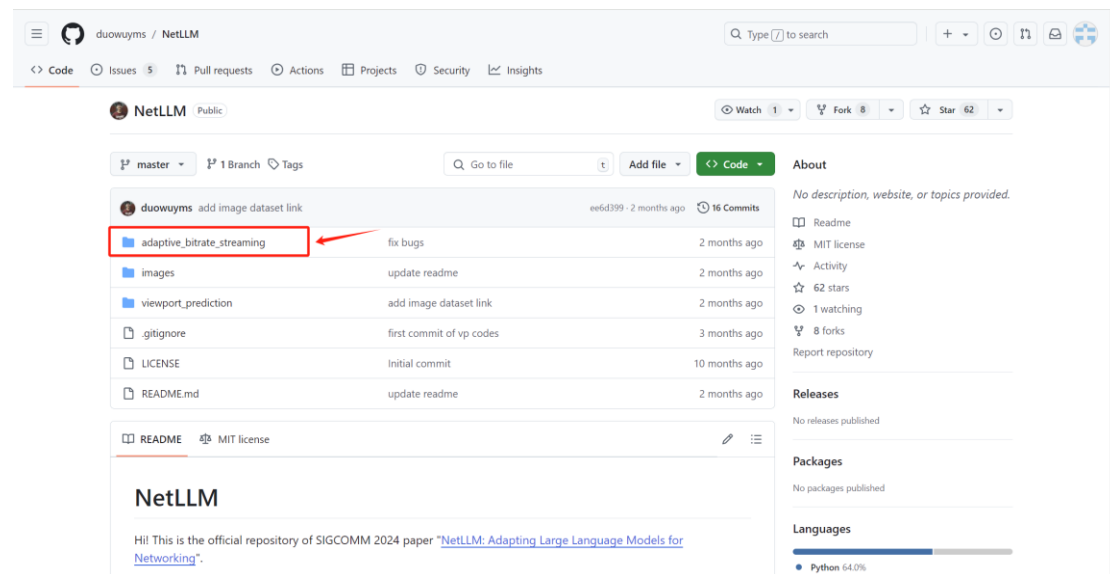
Command: **./file_name.sh**

- The JSON file containing all the information from the **iperf3** data transmission will be saved in the same folder where the script is located.



Step 3: How to Download Llama2 and the Original NetLLM

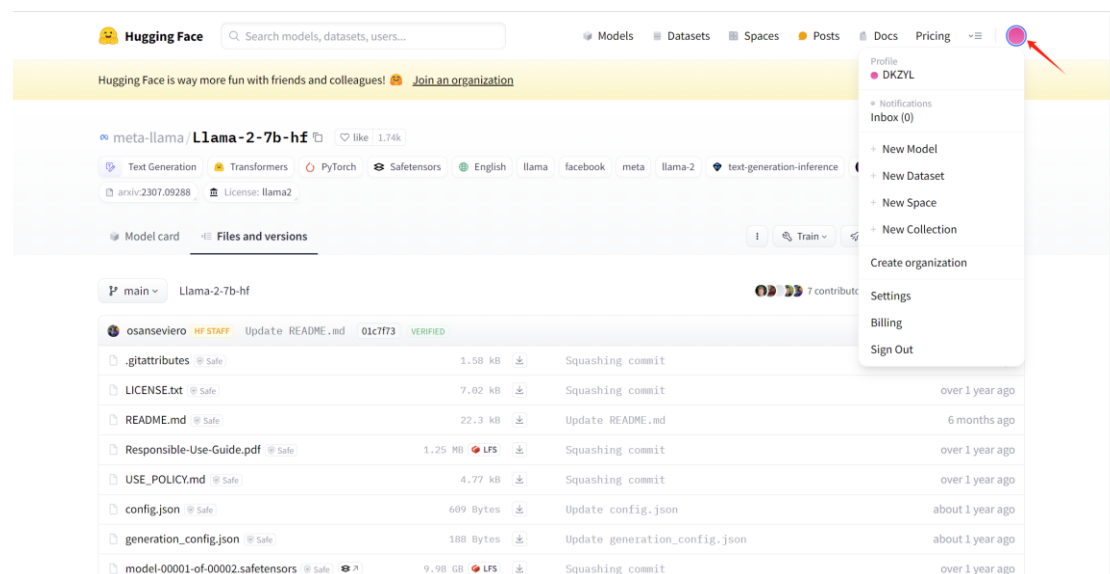
1. First, go to the official NetLLM GitHub and download the adaptive folder. Official GitHub link: <https://github.com/duowuymys/NetLLM/tree/master>



2. The LLM used in this experiment is **meta-llama/Llama-2-7b-hf**, so we will go to **Hugging Face** to download it.

Link: <https://huggingface.co/meta-llama/Llama-2-7b-hf/tree/main>

Due to Hugging Face's requirements, you need to log in or sign up before downloading Llama-2. If you encounter any issues during the download, make sure to sign Hugging Face's community guidelines and request permission to download Llama-2. All of this can be done directly on their website.



After obtaining permission, please download all the files from that page

main

Llama-2-7b-hf

7 contributors

History: 36 commits

osanseviero

HF STAFF

Update README.md

01c7f73

VERIFIED

6 months ago

.gitattributes

@ Safe

1.58 kB

Squashing commit

over 1 year ago

LICENSE.txt

@ Safe

7.82 kB

Squashing commit

over 1 year ago

README.md

@ Safe

22.3 kB

Update README.md

6 months ago

Responsible-Use-Guide.pdf

@ Safe

1.25 MB

LFS

Squashing commit

over 1 year ago

USE_POLICY.md

@ Safe

4.77 kB

Squashing commit

over 1 year ago

config.json

@ Safe

609 Bytes

Update config.json

about 1 year ago

generation_config.json

@ Safe

188 Bytes

Update generation_config.json

about 1 year ago

model-00001-of-00002.safetensors

@ Safe

9.98 GB

LFS

Squashing commit

over 1 year ago

model-00002-of-00002.safetensors

@ Safe

3.5 GB

LFS

Squashing commit

over 1 year ago

model.safetensors.index.json

@ Safe

26.8 kB

Squashing commit

over 1 year ago

pytorch_model-00001-of-00002.bin

@ Safe

pickle

9.98 GB

LFS

Upload LlamaForCausalLM

over 1 year ago

pytorch_model-00002-of-00002.bin

@ Safe

pickle

3.5 GB

LFS

Upload LlamaForCausalLM

over 1 year ago

pytorch_model.bin.index.json

@ Safe

26.8 kB

Upload LlamaForCausalLM

over 1 year ago

special_tokens_map.json

@ Safe

414 Bytes

Upload tokenizer

over 1 year ago

tokenizer.json

@ Safe

1.84 MB

Upload tokenizer

over 1 year ago

tokenizer.model

@ Safe

500 kB

LFS

Squashing commit

over 1 year ago

tokenizer_config.json

@ Safe

776 Bytes

Upload tokenizer

about 1 year ago

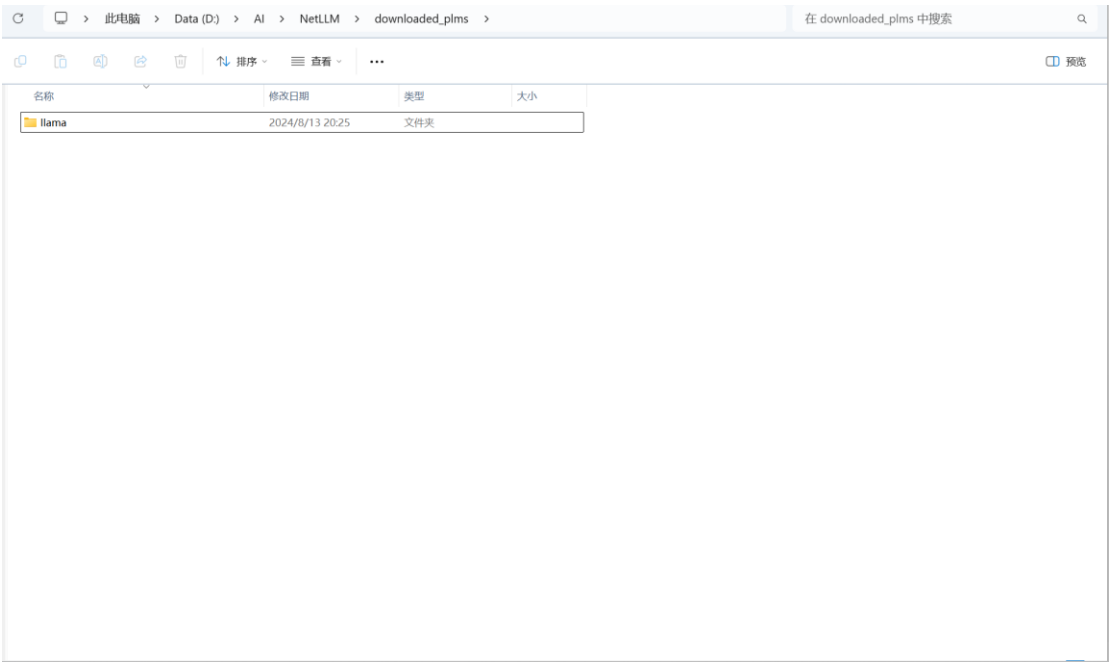
huggingface.co/meta-llama/._fdd2ef0b8f1248112241aaee4598b5b0bbb9

After downloading the **Llama-2** and **adaptive** folders, it's recommended to first create a larger folder to contain both of these folders. This will make it easier to manage and upload to Runpod later, as you'll only need to upload one large folder.

Next, place the **adaptive** folder inside this larger folder. Then, create a folder named **downloaded_plms** within it

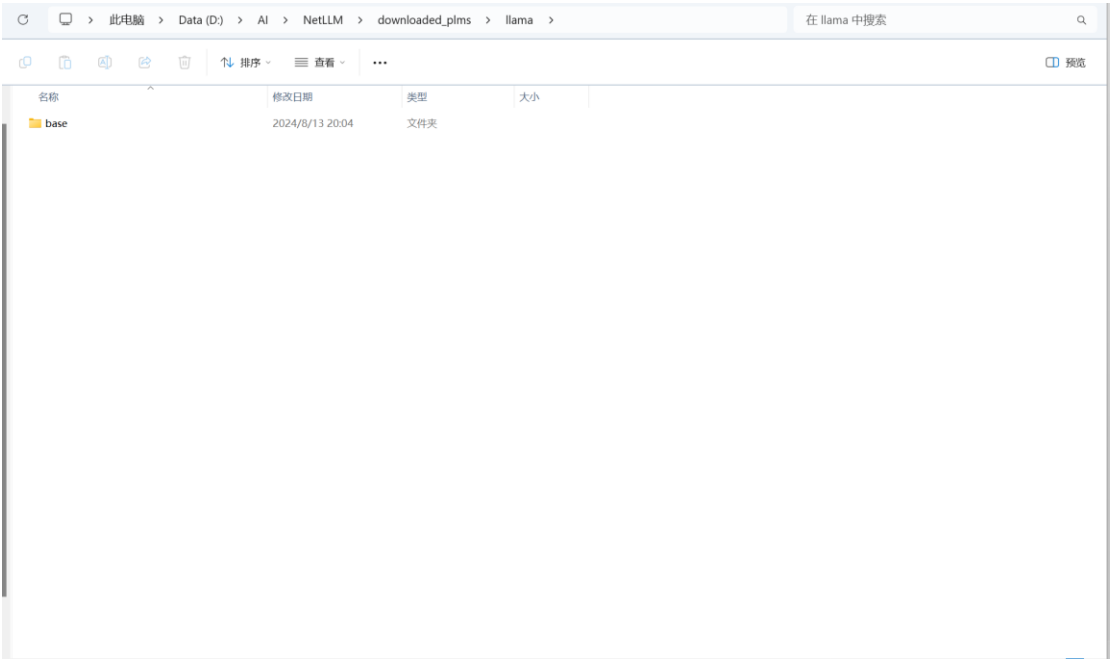
	adaptive_bitrate_streaming	2024/8/26 14:54	文件夹
	downloaded_plms	2024/8/12 12:25	文件夹

In this downloaded_plms folder we need to create a llama folder.

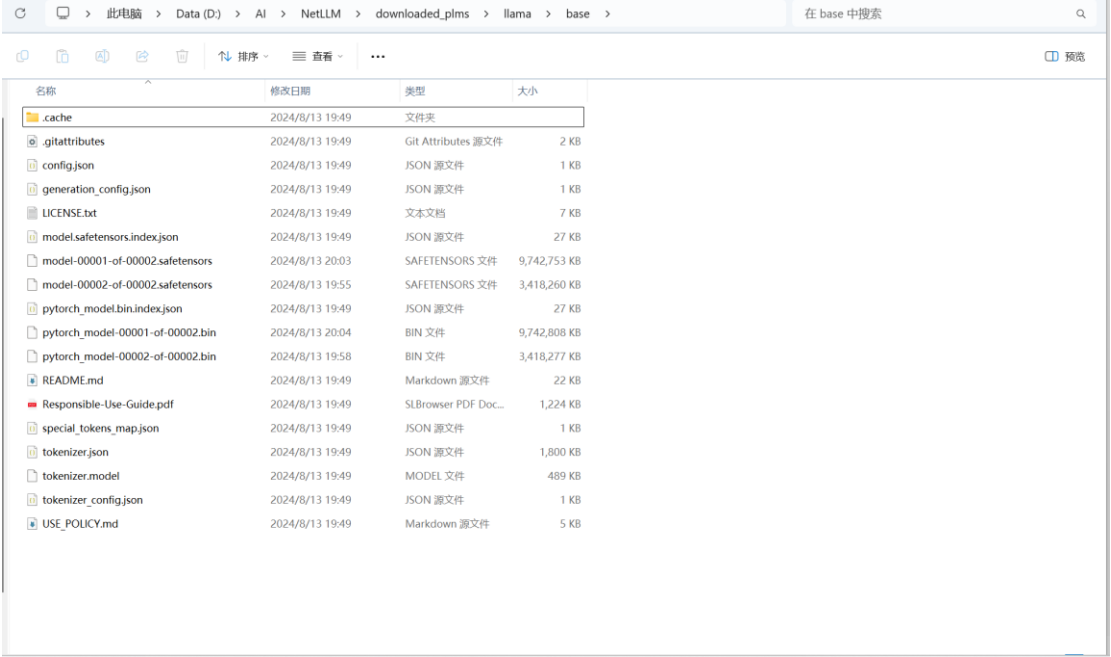


此电脑 > Data (D:) > AI > NetLLM > downloaded_plms				在 downloaded_plms 中搜索	
排序 · 查看 · ...				预览	
名称	修改日期	类型	大小		
	llama	2024/8/13 20:25	文件夹		

In this llama folder we need to create another base folder.



Then put all the Llama-2 files you just downloaded in the base folder.

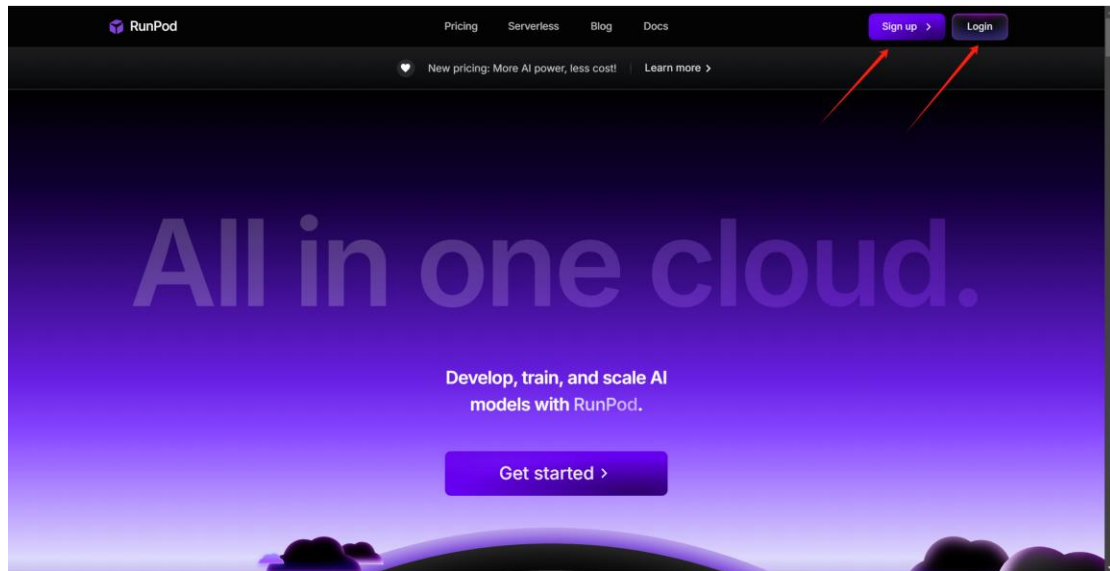


Step 4: How to Use Runpod to Train NetLLM

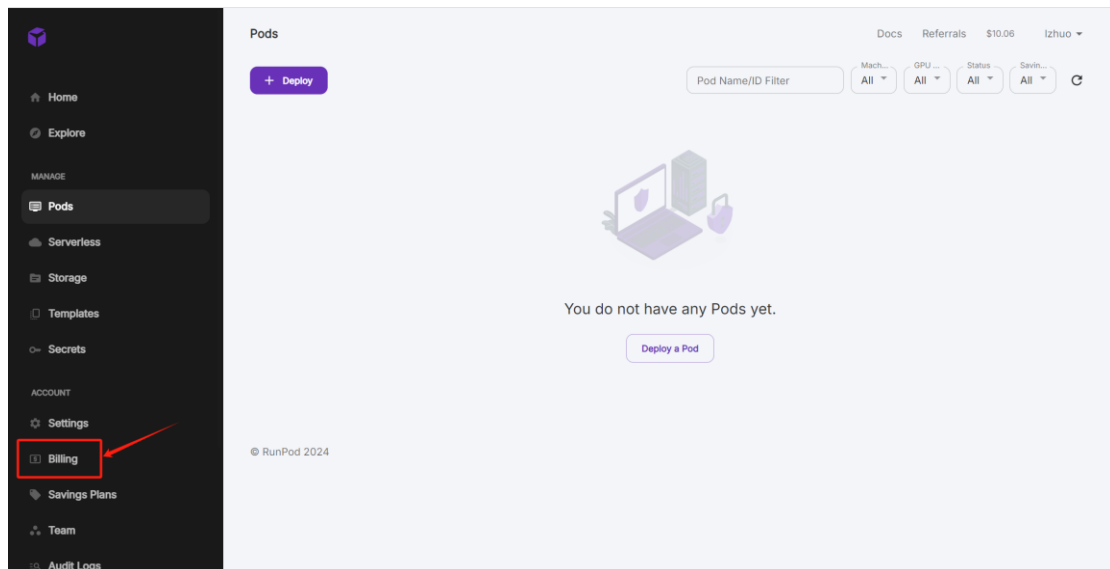
In this experiment, we use **Runpod** to run the **NetLLM** architecture.

Runpod link: <https://www.runpod.io/>

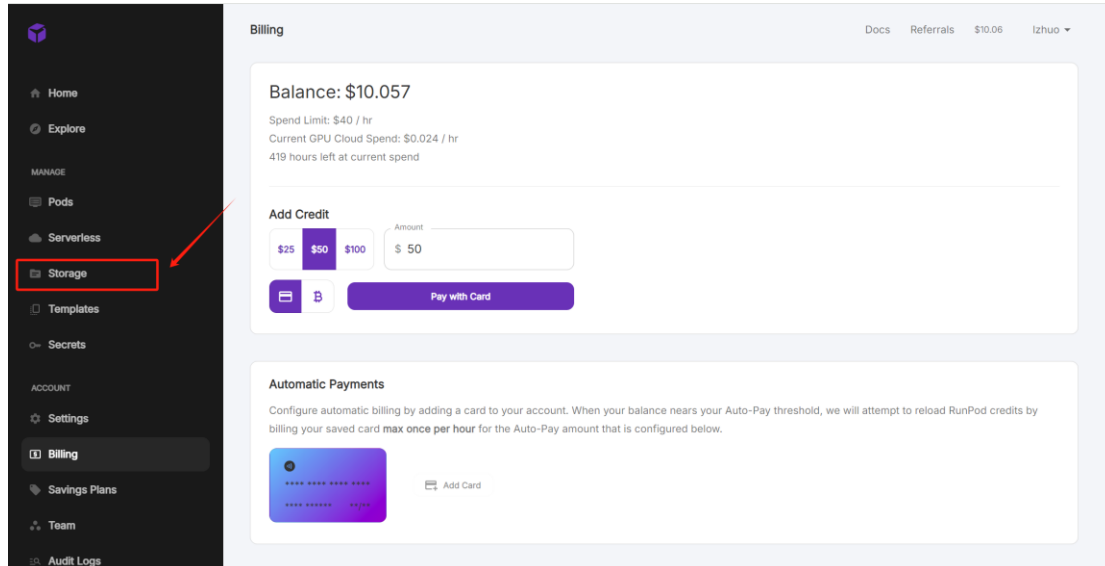
1. Click **Sign Up** or **Login** to register or log in



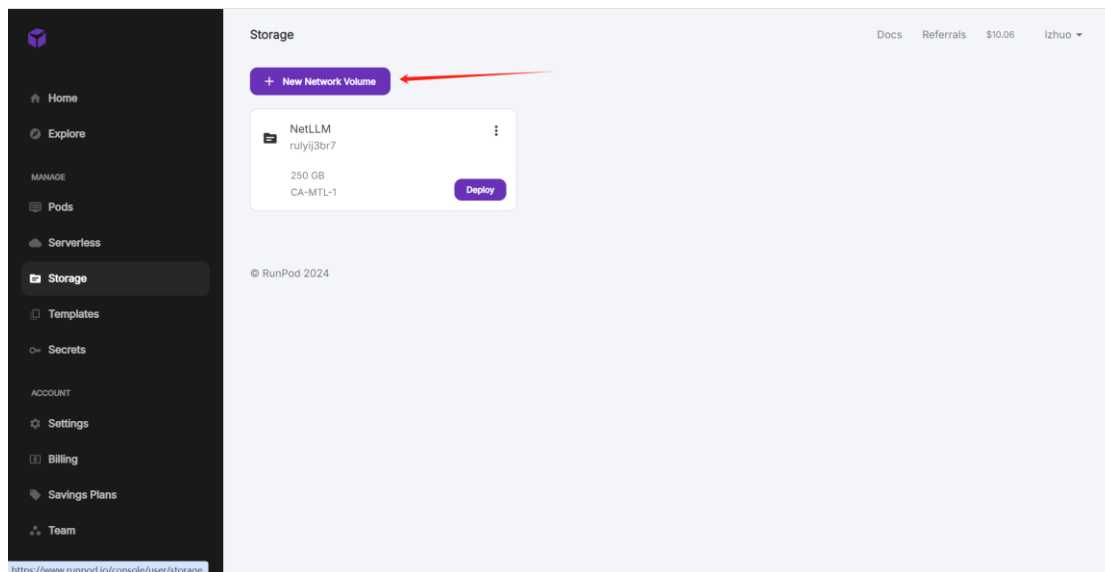
2. After logging in, first go to the **Billing** section to add funds, as Runpod cannot be used without a balance.



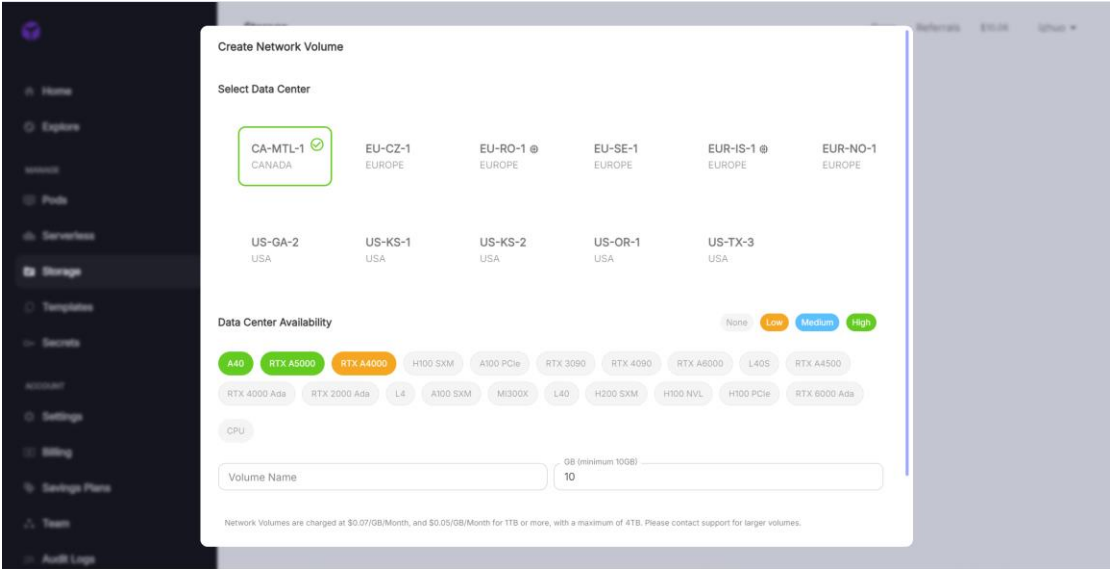
- Click on the **Storage** section and create a storage. The storage is used to save the **LLM model** and **NetLLM architecture**. If you don't create storage and use only the container in Runpod, all data will be lost after the pod is shut down. By using storage, you only need to upload the **NetLLM** framework and **LLM** once, and you can reuse them multiple times afterward



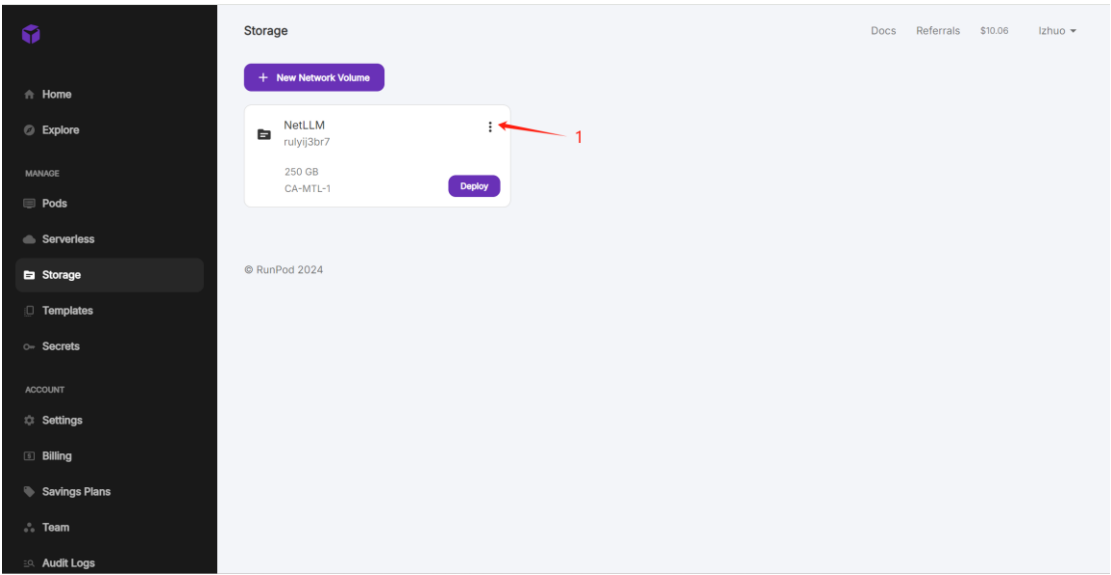
I've already created one, so you can ignore this step. Click on **New Network Volume**

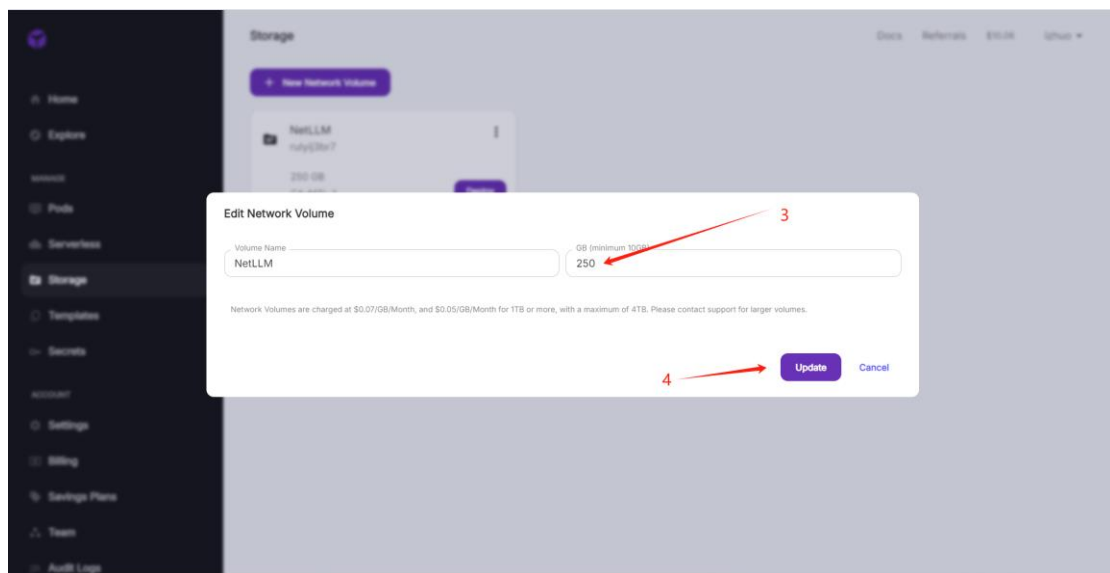
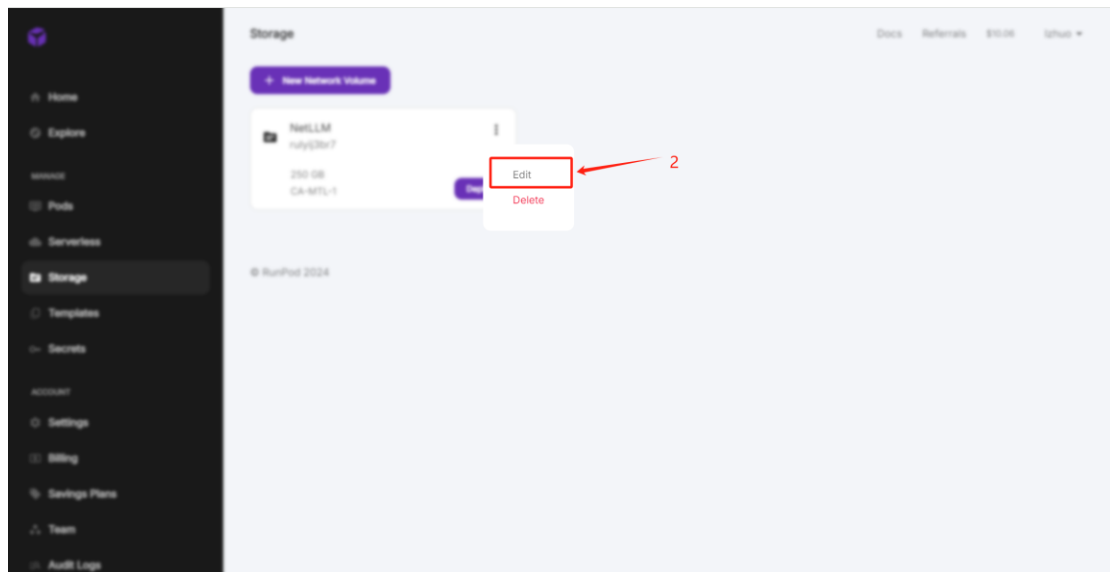


I recommend selecting **CA-MTL-1**, as this is the Runpod server closest to Australia. **Data Center Availability** shows the available GPU servers in the area. For our NetLLM project, we only need to rent 2 A40 GPUs (servers), making **CA-MTL-1** the best choice. Please remember to modify the storage name and adjust the storage size, with 50GB to 100GB being the recommended size. Once the modifications are done, click **Create** to create the storage

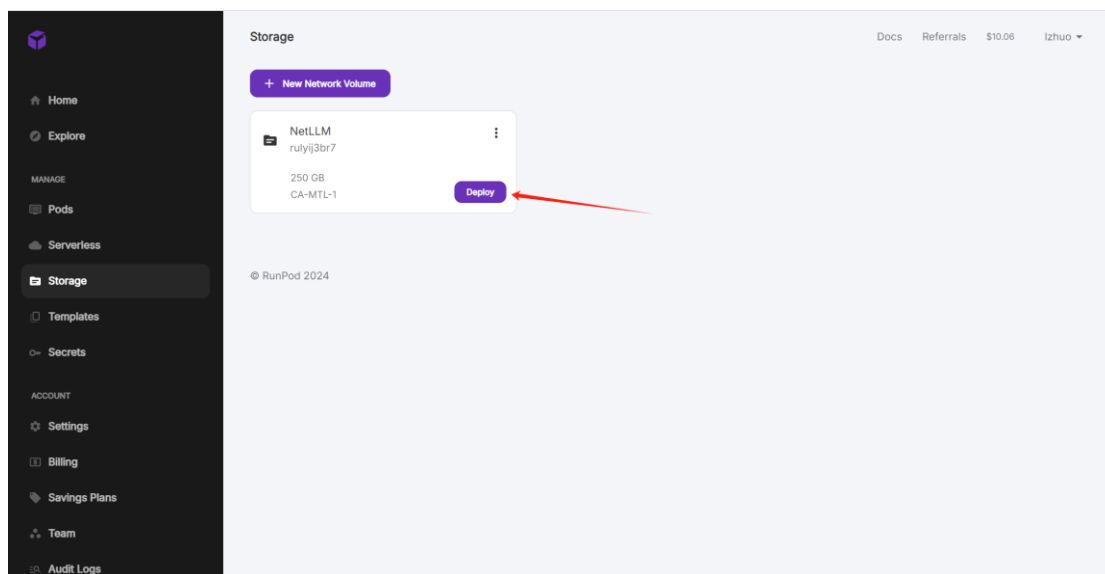


If you subsequently find that the storage space is insufficient, you can return to storage to expand the storage space

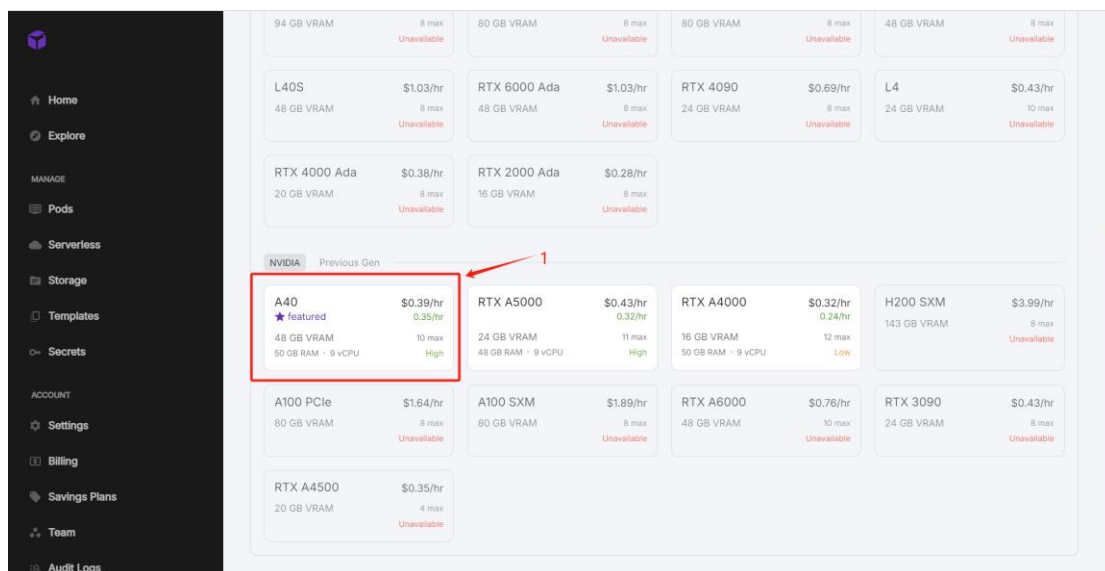




4. After setting up the storage, click on deploy.



Find A40 and click on it.



Increase GPU Count from 1 to 2.

runpod/pytorch:2.1.0-py3.10-cuda11.8.0-devel-ubuntu22.04

Edit Template

GPU Count: 1

Instance Pricing

Plan	Hourly Cost	Weekly Cost	Monthly Cost	3 Month Cost	6 Month Cost
On-Demand (Non-Interruptible)	\$0.39/hr				
1 Week Savings Plan	\$0.35/hr	\$58.80			
1 Month Savings Plan	\$0.35/hr		\$260.40		
3 Month Savings Plan	\$0.35/hr			\$772.80	
6 Month Savings Plan	\$0.35/hr				\$1529.15
Spot (Interruptible)	\$0.28/hr				

Network Volume: NetLLM 250 GB

☒ SSH Terminal Access

☒ Start Jupyter Notebook

runpod/pytorch:2.1.0-py3.10-cuda11.8.0-devel-ubuntu22.04

Edit Template

GPU Count: 2

Instance Pricing

Plan	Hourly Cost	Weekly Cost	Monthly Cost	3 Month Cost	6 Month Cost
On-Demand (Non-Interruptible)	\$0.78/hr				
1 Week Savings Plan	\$0.70/hr	\$117.60			
1 Month Savings Plan	\$0.70/hr		\$520.80		
3 Month Savings Plan	\$0.70/hr			\$1545.60	
6 Month Savings Plan	\$0.70/hr				\$3058.30
Spot (Interruptible)	\$0.56/hr				

Network Volume: NetLLM 250 GB

☒ SSH Terminal Access

☒ Start Jupyter Notebook

Remember to tick the following two boxes

On-Demand (Non-Interruptible) \$0.78/hr

1 Week Savings Plan \$0.70/hr \$117.60

1 Month Savings Plan \$0.70/hr \$520.80

3 Month Savings Plan \$0.70/hr \$1545.60

6 Month Savings Plan \$0.70/hr \$3058.30

Spot (Interruptible) \$0.56/hr

Network Volume: NetLLM 250 GB

☒ SSH Terminal Access

☒ Start Jupyter Notebook

Pricing Summary

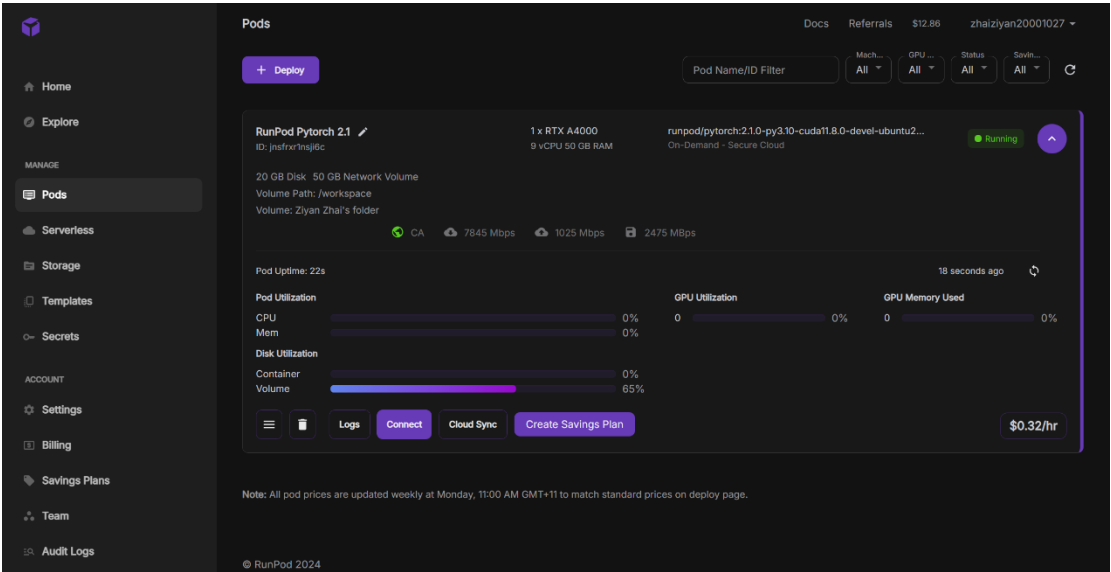
GPU Cost: \$0.78 / hr
Running Disk Cost: \$0.003 / hr
Network Volume Cost: \$0.024 / hr

Pod Summary

2x A40 (96 GB VRAM)
100 GB RAM • 18 vCPU
Total Disk: 270 GB

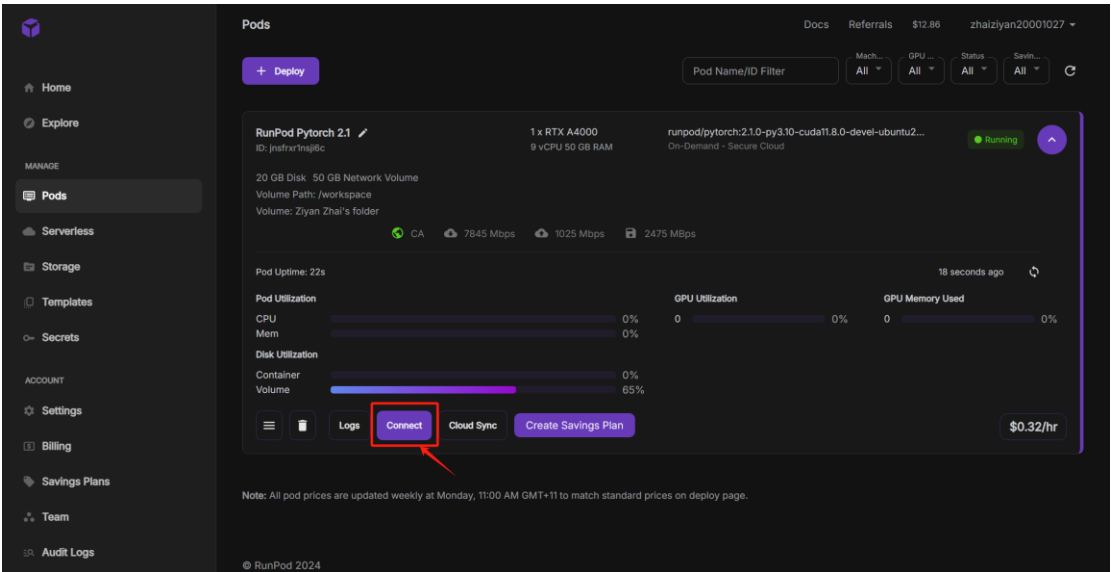
Deploy On-Demand

After the deployment is complete, you will see this page. My volume is at 65% because I have already uploaded the NetLLM-related files. If you are using newly created storage, the volume will be at 0.

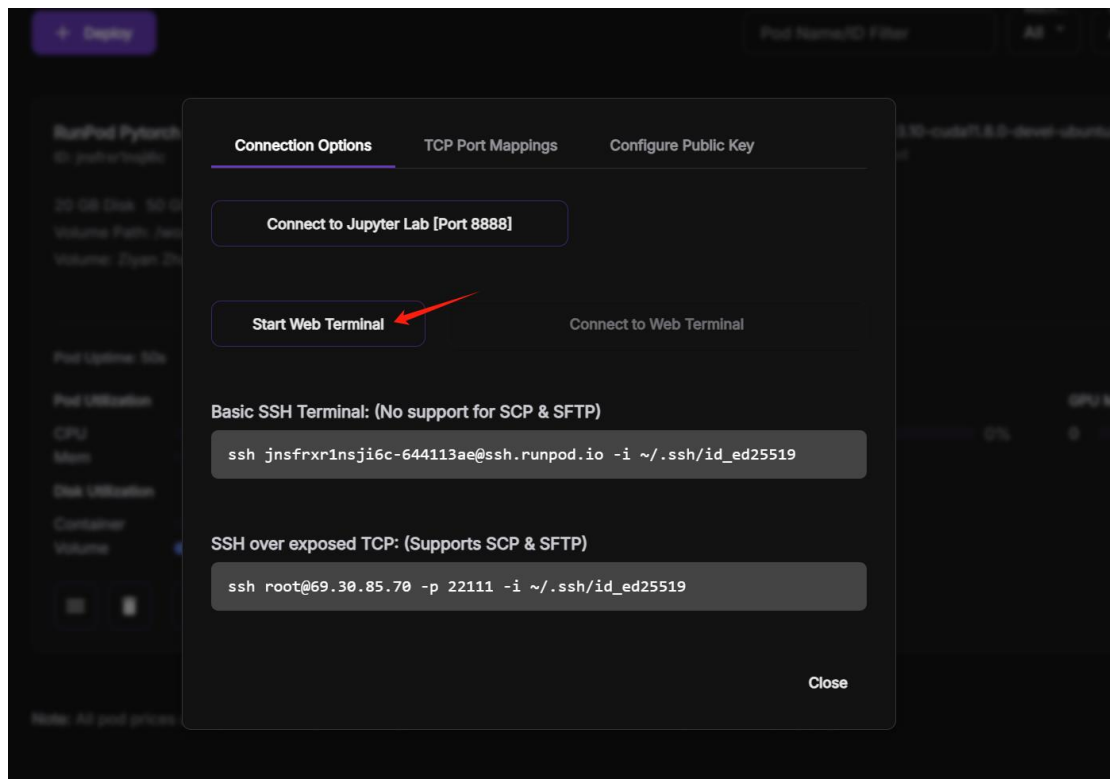


5. Using Runpod

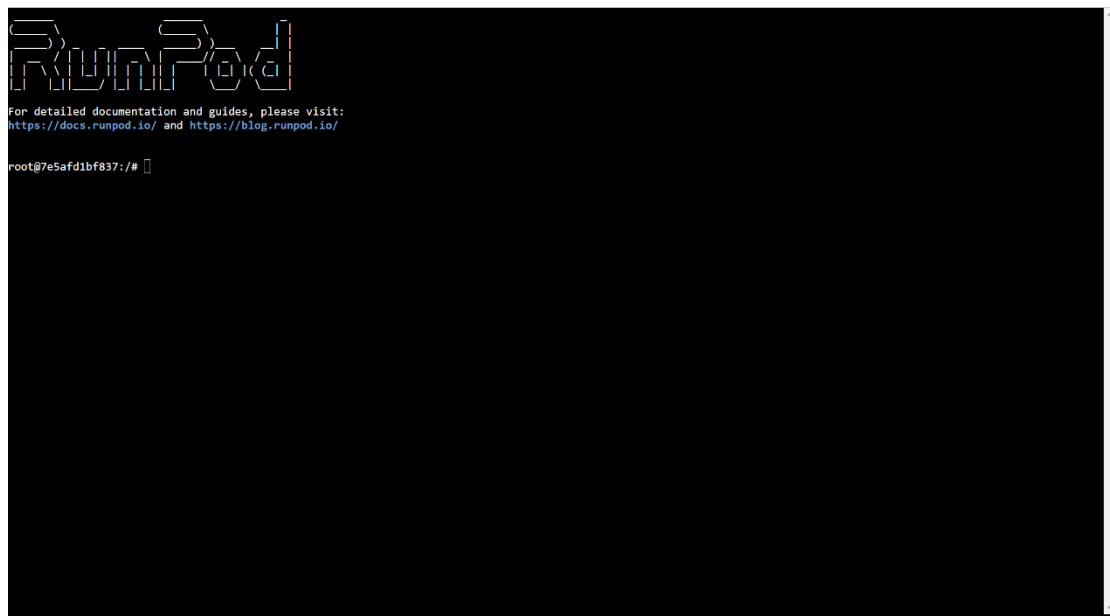
After the deployment is complete, click **Connect**.



On the **Connect** page, click **Start Web Terminal**, then click **Connect to Web Terminal**.

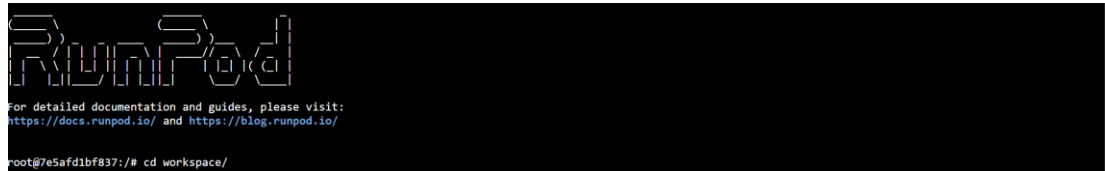


After clicking **Connect to Web Terminal**, Runpod will launch a web terminal. If you see this web terminal, it means you have successfully accessed the Runpod cloud GPU terminal.



Next, use the command: **cd workspace/**

to enter the storage folder. Please ensure that all files are stored in the **workspace** folder. If you store them elsewhere, all files and data will be automatically deleted when you shut down the Pod.



6. Install the necessary components to run NetLLM (Note: Every time you start a new Pod, you will need to reinstall these components, as they won't be saved in the storage.)

You can either install them one by one:

Command:

```
python -m pip install --upgrade pip
pip install openprompt==1.0.1
pip install numpy==1.24.4
pip install peft==0.6.2
pip install transformers==4.34.1
pip install --upgrade huggingface_hub
pip install scikit-learn
pip install munch
```

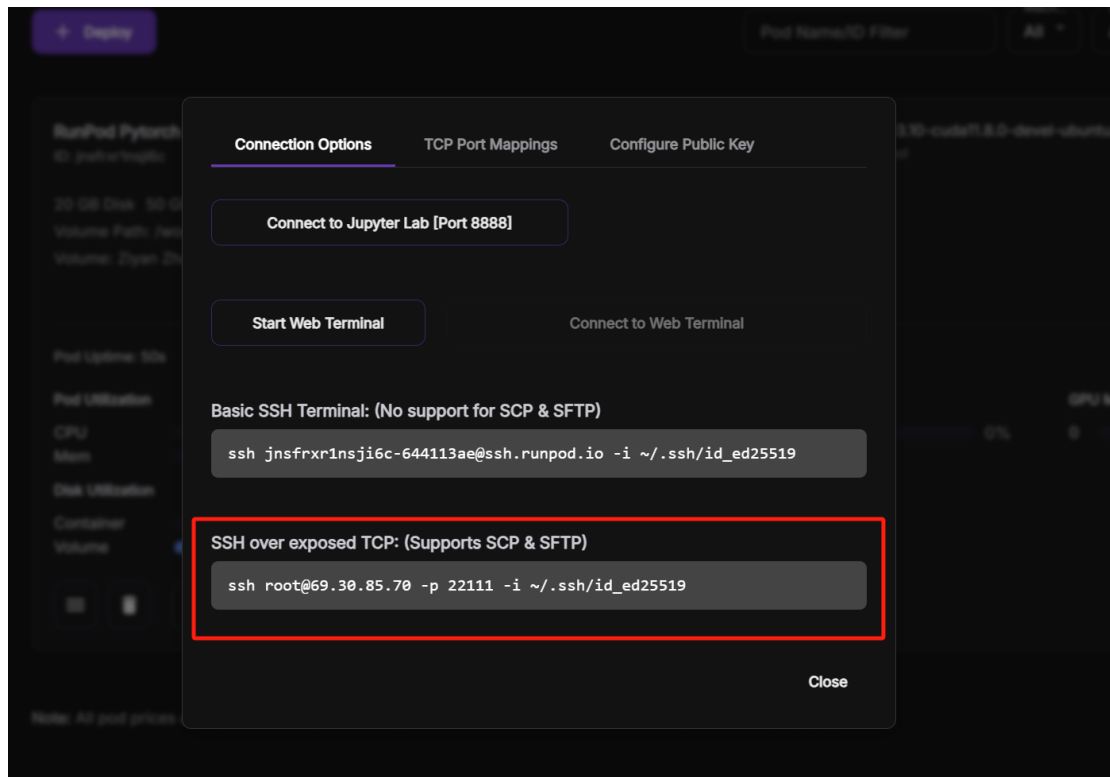
Or run a single command to install them all at once:

Command:

```
python -m pip install --upgrade pip && pip install openprompt==1.0.1 && pip install numpy==1.24.4 && pip install peft==0.6.2 && pip install transformers==4.34.1 && pip install --upgrade huggingface_hub && pip install scikit-learn && pip install munch
```

7. Upload the NetLLM architecture files to Runpod's storage

Let's return to the Runpod interface and click **Connect** again to go back to this page. We need to use the SSH provided by the Runpod server to upload the files. Each server will provide a different SSH connection.



Using this SSH as an example:

- **root**: Username for the SSH connection
- **69.30.85.70**: Runpod server's IP address
- **-p 22111**: SSH port number (22111)
- **-i ~/.ssh/id_ed25519**: Specifies the path to your SSH private key file

Command:

```
scp -i ~/.ssh/id_ed25519 -r -P port_number /path/to/local/folder  
root@ip_address:/path/to/remote/directory
```

Now that we have the SSH information, to upload the NetLLM files from your local machine, you just need to enter the command in your local terminal (cmd), replacing **port_number**, **/path/to/local/folder**, **ip_address**, and **/path/to/remote/directory** with your own details. For example, in this case, it should be modified to:

Command:

```
scp -P 22111 -i ~/.ssh/id_ed25519 -r "D:/NetLLM" root@69.30.85.70:/workspace/
```

8. How to Use NetLLM

After the upload is complete, first navigate to the **adaptive** folder within the NetLLM folder. The **adaptive** folder contains the main files for the NetLLM architecture.

Command:

```
cd NetLLM/
```

```
cd adaptive_bitrate_streaming
```

A terminal window with a black background. At the top, the 'RunPod' logo is displayed in a stylized, blocky font. Below the logo, a line of text reads: 'For detailed documentation and guides, please visit: https://docs.runpod.io/ and https://blog.runpod.io/'. The terminal shows a series of commands and their outputs: 'root@7e5afd1bf837:/# cd workspace/' followed by 'root@7e5afd1bf837:/workspace# cd NetLLM/' followed by 'root@7e5afd1bf837:/workspace/NetLLM# cd adaptive_bitrate_streaming' followed by 'root@7e5afd1bf837:/workspace/NetLLM/adaptive_bitrate_streaming# |'.

If the PKL file is included in the NetLLM folder, use the following command to start training NetLLM:

Command:

```
python run_plm.py --adapt --grad-accum-steps 32 --plm-type llama --plm-size base  
--rank 128 --device cuda:0 --device-out cuda:1 --lr 0.0001 --warmup-steps 2000 --  
num-epochs 20 --eval-per-epoch 2 --exp-pool-path your_exp_pool_path
```

Detailed Explanation of the Command Parameters:

python run_plm.py:

- Runs the run_plm.py script, which is the Python file responsible for training the NetLLM model. This file contains the core code for training the language model.

--adapt:

- Enables model adaptation or fine-tuning. This flag indicates that some adaptive training techniques are being used, or the language model is being fine-tuned for a specific task.

--grad-accum-steps 32:

- Specifies that gradient accumulation will occur over 32 steps. Gradient accumulation is often used when GPU memory is limited, meaning the model updates its parameters only after processing 32 mini-batches. This technique allows for larger batch sizes with less memory usage.

--plm-type llama:

- Specifies the type of Pretrained Language Model (PLM) as llama. In this case, we are using Llama2 as the LLM.

--plm-size base:

- Sets the size of the pretrained language model to base. This typically indicates a model with a standard number of parameters. Different sizes (e.g., small, base, large) represent different model complexities and capabilities.

--rank 128:

- Refers to the rank used in low-rank decomposition, which is set to 128. This parameter is used to reduce the computational complexity of model training, especially in techniques like matrix factorization.

--device cuda:0:

- Specifies that the training will run on GPU device cuda:0. This is the first GPU (indexed from 0) that will be used.

--device-out cuda:1:

- Specifies that output-related operations will take place on GPU device cuda:1. Since we are renting 2 A40 GPUs, both cuda:0 and cuda:1 will be used for training.

--lr 0.0001:

- Sets the learning rate to 0.0001. The learning rate controls how much the model's parameters are updated at each step, making it one of the most important hyperparameters for training.

--warmup-steps 2000:

- Specifies 2000 warmup steps at the beginning of the training. This means the learning rate will gradually increase from 0 over the first 2000 steps to avoid instability at the start of training.

--num-epochs 20:

- Sets the total number of epochs to 20. An epoch represents one complete pass through the entire training dataset. You can modify this as needed.

--eval-per-epoch 2:

- Specifies that the model will be evaluated twice per epoch, meaning performance will be assessed twice during each training cycle (e.g., on a validation set).

--exp-pool-path your_exp_pool_path:

- This parameter specifies the location of the PKL file.
-

Since we are currently using the original version of NetLLM, there is no need to provide our own experience pool file. You just need to run this command.

Command:

```
python run_plm.py --adapt --grad-accum-steps 32 --plm-type llama --plm-size base --rank 128 --device cuda:0 --lr 0.0001 --warmup-steps 2000 --num-epochs 80 --eval-per-epoch 2
```


If the command runs successfully, you will see information like this (the number of steps depends on the amount of data in your PKL file; the more data, the more steps):

```
Step 0 - mean train loss 1.985231
Step 100 - mean train loss 2.213923
Step 200 - mean train loss 2.189049
Step 300 - mean train loss 2.185949
Step 400 - mean train loss 2.221947
Step 500 - mean train loss 2.228495
Step 600 - mean train loss 2.236243
Step 700 - mean train loss 2.228206
Step 800 - mean train loss 2.220431
Step 900 - mean train loss 2.215590
===== Training Iteration #0 =====
>>>>>>>> Training Information:
{'time/training': 339.8703103065491,
 'training/train_loss_mean': 2.217822276684175,
 'training/train_loss_std': 0.3987985239587319}
Checkpoint saved at: data/ft_plms/llama_base/artifacts_exp_pools_ss_Nonearly_stop_-1_checkpoint/0
```

Please note that completing **iteration 0** does not necessarily mean that there are no issues with your PKL file or the NetLLM program. Be patient and wait for the program to enter **iteration 1**. If **iteration 1** runs smoothly without errors, the rest of the training process is likely to proceed without issues. However, one important point to keep in mind: since we are using **Runpod** for training and the server is located in Canada rather than Australia, there may be **lost connection** issues during the training process. Because NetLLM, unlike other LLMs or deep learning models, cannot resume training from where it was interrupted, it is not recommended to run the NetLLM training program overnight. It's advisable to check the training progress every hour to ensure it hasn't been interrupted.

```
/usr/local/lib/python3.10/dist-packages/transformers/generation_utils.py:24: FutureWarning: Importing GenerationMixin from src/transformers/generation_utils.py is deprecated and will be removed in Transformers v5. Import as 'from transformers import GenerationMixin' instead.
  warnings.warn(
Arguments:
Namespace(exp_pool_path='./prague_exp_pool.pkl', sample_step=None, trace='fcc-test', trace_num=100, video='video', fixed_order=False, plm_type='llama', plm_size='base', rank=1, feature_dim=256, w=20, gamma=1.0, lr=0.0001, weight_decay=0.0001, warmup_steps=2000, num_epochs=5, eval_per_epoch=2, save_checkpoint_per_epoch=2, target_return_scale=1.0, adapt=True, test=False, grad_accum_steps=32, seed=100003, scale=1000, model_dir=None, device='cuda:0', device_out='cuda:1', device_mid=None)
Loading traces from data/traces/test/fcc-test/
Experience dataset info:
  Launch('max_reward': 8.5759033476188, 'min_reward': 6.5074270790278925, 'max_return': 0.9107929413961323, 'min_return': 0.00021968027992012382, 'min_timestep': 0, 'max_timestep': 1000000, 'max_action': 3, 'min_action': 1)
Loading checkpoint shards: 100% |#####| 2/2 [00:12<00:00, 6
normalizer.cc(51) LOG(INFO) precompiled charmap is empty. use identity normalization.
If tokenizer is loaded: [1, 22172, 3186]

Step 0 - mean train loss 2.940735
Step 100 - mean train loss 2.629836
===== Training Iteration #0 =====
>>>>>>>> Training Information:
{'time/training': 30.821568965911865,
 'training/train_loss_mean': 2.648814216666265,
 'training/train_loss_std': 0.42323751098136914}
Checkpoint saved at: data/ft_plms/llama_base/_ss_None/rank_128_w_20_gamma_1.0_sfd_256_lr_0.0001_wd_0.0001_warm_2000_epochs_5_seed_100003/early_stop_-1_checkpoint/0
>>>>>>>> Evaluation Information
{'best_return': 0.0,
 'episodes_len': 4700,
 'episodes_return': 0.0,
 'time/evaluation': 556.1744978427887}
Step 0 - mean train loss 2.094530
Step 100 - mean train loss 2.641876
===== Training Iteration #1 =====
>>>>>>>> Training Information:
{'time/training': 29.65080665420532,
 'training/train_loss_mean': 2.6438361023544172,
 'training/train_loss_std': 0.4226540673253825}
Step 0 - mean train loss 2.531451
Step 100 - mean train loss 2.649771
===== Training Iteration #2 =====
>>>>>>>> Training Information:
{'time/training': 29.55081205987102,
 'training/train_loss_mean': 2.6337102640659436,
 'training/train_loss_std': 0.4222205221205113}
Checkpoint saved at: data/ft_plms/llama_base/_ss_None/rank_128_w_20_gamma_1.0_sfd_256_lr_0.0001_wd_0.0001_warm_2000_epochs_5_seed_100003/early_stop_-1_checkpoint/2
```

After the training is complete, you can enter the following command to run a test.:

```
python run_plm.py --test --grad-accum-steps 32 --plm-type llama --plm-size base --
rank 128 --device cuda:0 --lr 0.0001 --warmup-steps 2000 --num-epochs 80 --eval-
per-epoch 2
```

The output should look something like this.

```
pool06@ec3f82759:/workspace/NaTLM/adaptive_bilatrate_streaming$ python run_plm.py --test --grad-accum-steps 32 --plm-type llama --plm-size base --rank 128 --device cuda:0 --num-gpus 1 --lr 0.0001 --warmup-steps 2000 --num-epochs 80 --eval-per-epoch 2
/usr/local/lib/python3.10/dist-packages/transformers/generation_utils.py:24: FutureWarning: Importing 'GenerationMixin' from 'src/transformers/generation_utils.py' is deprecated and will be removed in transformers v5. Import as 'from transformers import GenerationMixin' instead.
warnings.warn(
Arguments:
Namespace(exp_pool_path='artifacts/exp_pools/exp_pool.pkl', sample_step=None, trace='fcc-test', trace_num=100, video='videol', fixed_order=False, plm_type='llama', plm_size='base', rank=128, state_feature_dim=256, w_20_gamma=1.0, lr=0.0001, weight_decay=0.0001, warmup_steps=2000, num_epochs=80, eval_per_epoch=2, save_checkpoint_per_epoch=2, target_return='which_layer=1, adapt_plm=True, trace=True, grad_accum_steps=32, seed=100003, scale=1000, model_dir=None, device='cuda:0', device_out='cuda:1', device_mid=None)
Loading traces from data/traces/test/fcc-test/
Experience dataset info:
Number of samples: 1.3, 'min_reward': -85.0127377757031, 'max_return': 0.0466198750491922, 'min_return': 0.0006304750495579298, 'min_timestep': 0, 'max_timestep': 46, 'train_loader': 0, 'validation_loader': 0, 'min_action': 0}
Loading checkpoint shards: 100%|██████████| 2/2 [00:00:00.00, 25.0it/s]
normalizer.cc(51) LOG(INFO) precompiled charmap is empty. use identity normalization.
If tokenizer is loaded: [1, 22172, 3186]

Load model from: data/ft_plms/llama_base/artifacts.exp_pools.ss.None/rank_128_w_20_gamma_1.0_sfd_256_lr_0.0001_wd_0.0001_warm_2000_epochs_80_seed_100003/early_stop_-_1_best_model_100
['time': 813.0310912132263, 'mean_reward': 0.759016151472371]
Test time: 813.0310912132263
Mean reward: 0.759016151472371
Results saved at: artifacts/results/test_video/rank_num_fixed_True/llama_base/early_stop_-_1_rank_128_w_20_gamma_1.0_tgt_scale_1.0_seed_100003
pool06@ec3f82759:/workspace/NaTLM/adaptive_bilatrate_streaming$
```

How to Use My NetLLM Program

If you want to use my NetLLM program, you can download my **adaptive** folder from GitHub and overwrite the original NetLLM folder on your local machine. Then, rerun the **scp** command to upload it to Runpod.

GitHub Link: <https://github.com/MPTCP-FreeBSD/LLM-CCA/tree/main>

Afterward, the rest of the process is similar to running the original NetLLM. We use the same command to start training, but now you can modify certain parts of the training command.

Suggested modifications:

num-epochs [number]

eval-per-epoch-[number]

```
--exp-pool-path your_exp_pool_path
```

For example,

Command:

```
python run_plm.py --adapt --grad-accum-steps 32 --plm-type llama --plm-size base
--rank 128 --device cuda:0 --device-out cuda:1 --lr 0.0001 --warmup-steps 2000 --
num-epochs 60 --eval-per-epoch 1 --exp-pool-path ./prague_exp_pool.pkl
```

Set the total number of training epochs to **60**.

Perform **1** evaluation per training epoch.

Use the **prague_exp_pool.pkl** file for the experience pool.

Step 5: Explanation of My Modifications to the Original NetLLM

1. Experience Pool Generation Script

In the official version of NetLLM, the experience pool files are generated using formulas and algorithms to simulate data. However, since we conducted real experiments and collected actual data, there is no need to generate the experience pool through simulation. To create our own experience pool file, I wrote a script. All you need to do is convert the data previously collected using **iperf3** from a **json** file into a **csv** file, ensuring it includes **RTT**, **throughput**, and **CWND** data.

In my experience pool file, I use **RTT**, **throughput**, and **CWND** as the **state**, and **CWND** as the **action** (since CWND controls the sender's bitrate, we want NetLLM to learn how to control CWND). The **reward** value is calculated as **throughput/RTT**, because when **throughput** is high and **RTT** is low, the reward increases; conversely, when **RTT** is high and **throughput** is low, the reward decreases. Then, we smooth the reward value using **log10**.

Additionally, we normalized the **CWND** values to ensure that NetLLM can correctly interpret the **action** values (without normalization, the **mean train loss** reported by NetLLM during training could be very high, which would hinder the model's learning).

```
def run(args):
    df = pd.read_csv(args.csv_file)
    exp_pool = ExperiencePool()

    # 找到最大和最小的 cwnd 值
    max_cwnd_value = df['cwnd'].max()
    min_cwnd_value = df['cwnd'].min()

    for index, row in df.iterrows():
        rtt = float(row['rtt']) if pd.notna(row['rtt']) else 0
        cwnd = float(row['cwnd']) if pd.notna(row['cwnd']) else 0
        throughput = float(row['throughput']) if pd.notna(row['throughput']) else 0

        # 将 state 转换为 Tensor, 只包含 [rtt, cwnd, throughput]
        state = torch.tensor([rtt, cwnd, throughput], dtype=torch.float32)

        # 将 cwnd 缩放到 [1, 2, 3] 范围
        # 先进行归一化, 再映射到 [1, 2, 3] 范围
        action_normalized = (cwnd - min_cwnd_value) / (max_cwnd_value - min_cwnd_value)
        action = int(action_normalized * 2) + 1 # 映射到 1, 2, 3

        # 计算 reward
        reward = calculate_reward(rtt, throughput)
        done = index == len(df) - 1

        # 将 state, action, reward, done 加入经验池
        exp_pool.add(state, action, reward, done)

    # 保存 ExperiencePool 实例
    exp_pool_path = os.path.join(args.output_dir, 'prague_exp_pool.pkl')

    with open(exp_pool_path, 'wb') as f:
        pickle.dump(exp_pool, f)

    print(f"经验池已保存至: {exp_pool_path}")
```

2. run_plm.py

Originally, run_plm used **cross entropy** as the loss function, which is typically used for

classification tasks. However, since our action is **CWND**, which is a continuous value, we needed to change the loss function to **MSELoss** (Mean Squared Error Loss).

```
# 适应函数 adapt() 中的修改
def adapt(args, model, exp_dataset, exp_dataset_info, eval_env_settings, checkpoint_dir, best_model_dir, eval_process_reward_fn):
    optimizer = AdamW(
        model.parameters(),
        lr=args.lr,
        weight_decay=args.weight_decay,
    )
    lr_scheduler = LambdaLR(
        optimizer,
        lambda steps: min((steps + 1) / args.warmup_steps, 1)
    )
    loss_fn = MSELoss() #MSELoss可以修改为Cross-Entropy Loss CE Loss | MSELoss can be replaced with Cross-Entropy Loss (CE Loss).
    trainer = Trainer(args, model=model, optimizer=optimizer, exp_dataset=exp_dataset, loss_fn=loss_fn, device=args.device, lr_scheduler=lr_scheduler,
                      grad_accum_steps=args.grad_accum_steps)

    target_return = exp_dataset_info.max_return * args.target_return_scale
    best_eval_return = 0.

    total_train_losses = []
    min_loss=np.inf
```

3. trainer.py

The trainer.py file was modified to calculate **MSE loss**. The **Trainer** class is responsible for managing the model's training process. It uses data from the experience pool to compute the loss via forward propagation, and then updates the model's parameters through backpropagation. Gradient accumulation is implemented to improve efficiency while maintaining small batch sizes.

```
adaptive_bitrate_streaming > plm_special > trainer.py > Trainer > train_step
11 class Trainer:
26 def train_epoch(self, report_loss_per_steps=100):
37
38     # perform gradient accumulation update
39     train_loss = train_loss / self.grad_accum_steps
40     train_loss.backward()
41     torch.nn.utils.clip_grad_norm_(self.model.parameters(), .25)
42     if ((step + 1) % self.grad_accum_steps == 0) or (step + 1 == dataset_size):
43         self.optimizer.step()
44         self.optimizer.zero_grad(set_to_none=True)
45         if self.lr_scheduler is not None:
46             self.lr_scheduler.step()
47
48     if step % 2 == 0:
49         mean_train_loss = np.mean(train_losses)
50         print(f'Step {step} - mean train loss {mean_train_loss:>9f}')
51
52     logs['time/training'] = time.time() - train_start
53     logs['training/train_loss_mean'] = np.mean(train_losses)
54     logs['training/train_loss_std'] = np.std(train_losses)
55
56     return logs, train_losses
57
58 def train_step(self, batch):
59     states, actions, returns, timesteps, labels = process_batch(batch, device=self.device)
60
61     # 前向传播, 获取预测的 actions_pred
62     actions_pred = self.model(states, actions, returns, timesteps)
63
64     # 调整 actions_pred 的形状, 使其与 labels 形状一致
65     actions_pred = actions_pred.squeeze(-1) # 将形状从 (1, 20, 1) 调整为 (1, 20)
66
67     labels = labels.float() # 确保 labels 是 float 类型
68
69     # 计算 MSE 损失
70     loss = self.loss_fn(actions_pred, labels)
71
72     return loss
```

4. state_encoder.py

The state_encoder was modified to encode numerical features such as **RTT**, **CWND**, and **throughput** into high-dimensional vectors. Each feature is mapped to the **embed_dim** dimension through a linear layer. These encoded vectors are then concatenated and passed through a fully connected layer to generate the final embedded vector output. This encoder can be part of a larger model, helping **NetLLM** or other models understand the relationships between these network performance indicators.

```
adaptive_bitrate_streaming > plm_special > models > state_encoder.py > EncoderNetwork > forward
1 import torch.nn as nn
2 import torch
3
4 class EncoderNetwork(nn.Module):
5     def __init__(self, embed_dim=128):
6         super().__init__()
7         self.embed_dim = embed_dim
8
9         # Linear layers for numerical features
10        self.rtt_fc = nn.Linear(1, embed_dim) # For RTT
11        self.cwnd_fc = nn.Linear(1, embed_dim) # For CWND
12        self.throughput_fc = nn.Linear(1, embed_dim) # For Throughput
13
14        # Final fully connected layer to combine all features
15        self.fc_final = nn.Linear(embed_dim * 3, embed_dim) # 3 inputs: RTT, CWND, Throughput
16
17    def forward(self, state):
18        # Extract the components from the state
19        rtt = state[..., 0].unsqueeze(-1) # RTT
20        cwnd = state[..., 1].unsqueeze(-1) # CWND
21        throughput = state[..., 2].unsqueeze(-1) # Throughput
22
23        # Process numerical features
24        rtt_encoded = torch.relu(self.rtt_fc(rtt))
25        cwnd_encoded = torch.relu(self.cwnd_fc(cwnd))
26        throughput_encoded = torch.relu(self.throughput_fc(throughput))
27
28        # Concatenate all encoded features
29        combined = torch.cat([rtt_encoded, cwnd_encoded, throughput_encoded], dim=-1)
30
31        # Final output
32        output = torch.relu(self.fc_final(combined))
33
34        return output
35
36
```


5. rl_policy.py

The original rl_policy was modified, and a policy network class for **Offline Reinforcement Learning (Offline RL)** was defined, named **OfflineRLPolicy**. This class uses a **Multimodal Encoder** to process network states (such as **RTT**, **CWND**, and **throughput**) along with **PLM** (Pretrained Language Model) embeddings. Through this network, it is possible to learn how to predict **CWND** (Congestion Window) values in network control tasks based on state and historical information.

```
adaptive_bitrate_streaming > plm_special > models > rl_policy.py > ...
9  INF = 1e5
10
11  class OfflineRLPolicy(nn.Module):
12      def __init__(
13          self,
14          state_feature_dim,
15          bitrate_levels,
16          state_encoder, # 用于处理 [RTT, CWND, Throughput]
17          plm,
18          plm_embed_size,
19          max_length=None,
20          max_ep_len=100,
21          device='cuda' if torch.cuda.is_available() else 'cpu',
22          device_out=None,
23          residual=False,
24          which_layer=-1, # 早停
25          **kwargs
26      ):
27          super().__init__()
28
29          if device_out is None:
30              device_out = device
31
32          self.bitrate_levels = bitrate_levels
33          self.max_length = max_length
34
35          self.plm = plm
36          self.plm_embed_size = plm_embed_size
37
38          # ===== Multimodal Encoder (start) =====
39          self.state_encoder = state_encoder
40          self.state_feature_dim = state_feature_dim
41          self.embed_timestep = nn.Embedding(max_ep_len + 1, plm_embed_size).to(device)
42          self.embed_return = nn.Linear(1, plm_embed_size).to(device)
43          self.embed_action = nn.Linear(1, plm_embed_size).to(device)
44          self.embed_state1 = nn.Linear(state_feature_dim, plm_embed_size).to(device) # 状态编码器
45
46          self.embed_ln = nn.LayerNorm(plm_embed_size).to(device)
47          # ===== Multimodal Encoder (end) =====
48
49          # @@@@ action head: 0.1-1.0 @@@@ end @@@@
```