**GPU Based VLLM Deployment**

First of all in ubuntu(Thinkstation) I have run commands to look some thing which include:

* Lsblk
* sudo fdisk -l
* df -h
* lspci | grep -i vga  
  nvidia-smi

In a result of these commands I am getting this detail about the system:

**Conclusion: Key Takeaways**

|  |  |
| --- | --- |
| Component | Summary |
| CPU | Dual Intel Xeon Gold 6138 (40 cores total) |
| RAM | 128 GiB DDR4 |
| Disk | 2 TB via MegaRAID, 1.1 TB used |
| GPU | NVIDIA RTX 3090 (24 GB VRAM) |
| GPU Active? | Yes – actively used by a Python process |
| OS Drive | /dev/sda3 mounted as / |
| Boot Mode | UEFI (seen from /boot/efi) |

**What nvidia-smi command is doing and what info it gaves us?**

### **Driver and CUDA Info**

* **NVIDIA Driver Version**: 550.144.03
* **CUDA Version**: 12.4
  + This means your system can run recent CUDA-based applications, like PyTorch or TensorFlow.

### 🎮 **GPU Details**

* **Model**: NVIDIA GeForce RTX 3090
* **Total VRAM (Memory)**: 24,576 MiB (24 GB)
* **Used VRAM**: 16,588 MiB (≈16.2 GB used)
* **Available VRAM**: ~8 GB
* **Temperature**: 45°C (cool, healthy)
* **Power Usage**: 28W out of 350W max (very low, idle state)
* **Performance State (Perf)**: P8 = Idle/Low power mode
* **GPU Utilization**: 0% (not actively running calculations at this moment)

### 🧠 **Running GPU Processes**

|  |  |  |  |
| --- | --- | --- | --- |
| PID | Type | Program/Process Name | VRAM Used |
| 1933 | G | /usr/lib/xorg/Xorg | 105 MiB |
| 2288 | C+G | gnome-remote-desktop-daemon | 379 MiB |
| 2355 | G | gnome-shell | 39 MiB |
| 696967 | G | Google Chrome | 4 MiB |
| 697017 | G | Chrome-related (possibly video) | 41 MiB |
| 711097 | C | /usr/bin/python3 | **15,992 MiB** (≈15.6 GB) |

**Directory creation:**

I have created directory for my name in ubuntu thinkstation in desktop I will be doing all my working in this. Then in this I have created my project root directory.  
Now first I will create a virtual environment using conda in my project directory.

**Virtual environment creation by conda in ssh.  
Step 1: Install Conda on Your SSH Server**

wget <https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh>

**2. Run the Installer**

bash Miniconda3-latest-Linux-x86\_64.sh  
Miniconda is installed at \home\ubuntu\miniconda

**3.Restart the shell (or reload your profile)**

exec "$SHELL"  
Also you have to activate the conda as well by the command:

source ~/miniconda3/etc/profile.d/conda.sh  
Then verify by conda –version.

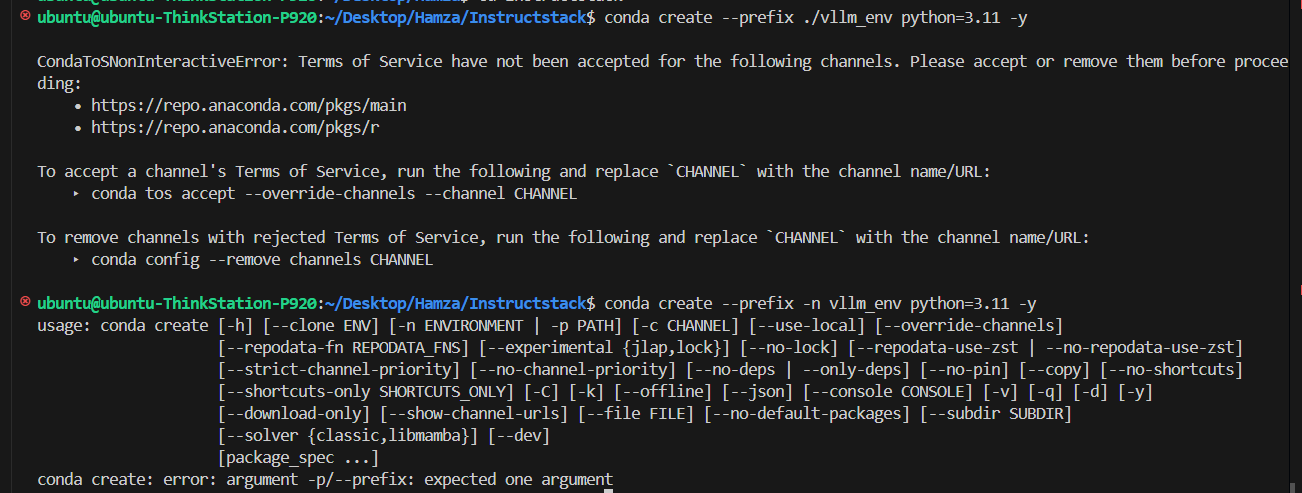
**Step 2: Create a Conda Virtual Environment**

conda create -n vllm\_env python=3.11 -y

I want conda environment in my project directory then run this:

conda create --prefix ./vllm\_env python=3.11 -y

**In the creation of virtual environement by conda you may face these errors:**



This happens because Conda **needs you to accept the Terms of Service** for official Anaconda channels.

### 🔧 Fix: Accept the Terms of Service

**Run the following two commands:**

conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/main

conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/r

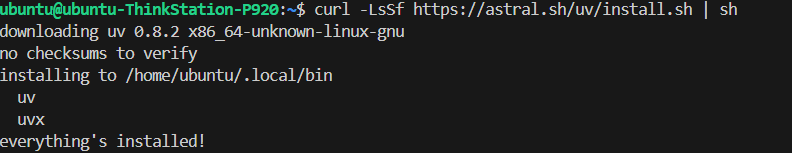
This accepts the ToS for the default channels. After that, you can create environments again.  
After that the virtual environment will be successfully installed in project directory and then you can activate it by conda activate ./vllm\_env.  
 **Setting up the VLLM for models:**

Pip can be used for all the installations but uv can be a best option for me now although I have used pip in the setup of cpu. Why see below:

To automatically select the appropiate pytorch index at runtime.by inspecting the installed CUDA driver version via --torch-backend=auto (or UV\_TORCH\_BACKEND=auto). To select a specific backend (e.g., cu126), set --torch-backend=cu126 (or UV\_TORCH\_BACKEND=cu126). If this doesn't work, try running uv self update to update uv first.  
We can install the uv by the following commands:

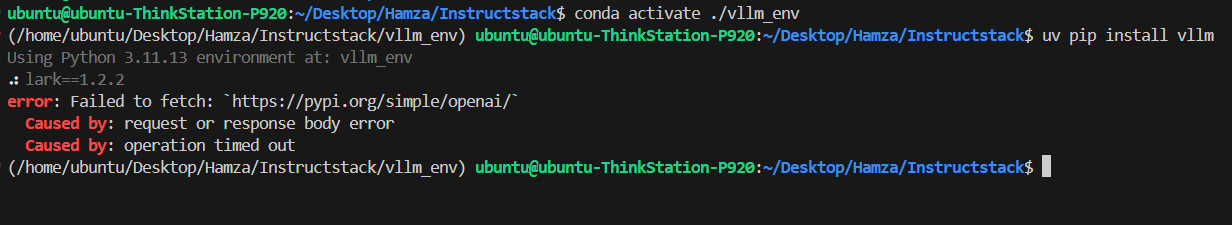
wget -qO- https://astral.sh/uv/install.sh | sh

curl -LsSf https://astral.sh/uv/install.sh | sh



Now installing the vllm by uv:

Running this command now First time I run it getting this error:

uv pip install vllm  


**Fix: Increase UV Download Timeout**

uv uses the environment variable UV\_HTTP\_TIMEOUT (default is 30 seconds). To give downloads more time, increase it to something like **300 seconds** or **500 seconds**.

1. **Set timeout temporarily for this session:**

export UV\_HTTP\_TIMEOUT=500

1. **(Optional) Make it permanent by adding to your ~/.bashrc:**

echo "export UV\_HTTP\_TIMEOUT=500" >> ~/.bashrc

source ~/.bashrc

This adjustment is often recommended when uv fails during package installation due to slow networks

**Concurrency Testing of Model with vllm:**

### **✅ Libraries**

|  |  |
| --- | --- |
| Library | Purpose |
| asyncio | Python's built-in library for running **concurrent asynchronous tasks** using coroutines. It lets multiple users "wait" without blocking others. |
| aiohttp | Async version of requests. Used for **non-blocking HTTP calls** (e.g., to your VLLM server). |
| time | For measuring response time (how long each request takes). |
| random | Used to randomly select different prompts from the pool for each user. |

### **✅ Key Functions**

|  |  |
| --- | --- |
| Function | Purpose |
| async def | Defines an **asynchronous function**, which runs without blocking other tasks. |
| asyncio.create\_task() | Starts a coroutine in the background — like launching a thread, but lightweight. |
| await | Tells Python to wait **asynchronously** (not blocking) until a task finishes. |
| aiohttp.ClientSession() | Manages a connection session for efficient multiple HTTP requests. |
| session.post() | Sends a POST request to the VLLM server with your prompt and parameters. |

**time.perf\_counter():**

In Python, time.perf\_counter() is a function from the time module that returns the value of a performance counter. This counter is a clock with the highest available resolution, specifically designed for measuring short durations and evaluating the performance of code.

Here's a breakdown of what time.perf\_counter() does:

**High-Resolution, Monotonic Clock**: It utilizes a monotonic clock, meaning it always moves forward and never goes backward, unlike the system clock which can be adjusted. This makes it ideal for accurately measuring elapsed time intervals.

**Performance Measurement**: Its primary use case is to measure the execution time of code segments or functions. You call time.perf\_counter() at the beginning and end of the code you want to time, and the difference between the two returned values represents the elapsed time in fractional seconds.

**The Python code I provided uses asyncio with aiohttp to send concurrent (non-blocking) requests — perfect for your use case: testing how multiple users simultaneously interact with your vLLM server.**

After Giving response Vllm server generates some fields with the help of loggers these are:

INFO 07-30 11:37:25 [logger.py:43] Received request cmpl-91a7c5fc7f2348c2bd76bd225a284317-0: prompt: '### Database Schema:\nTable: employees\nColumns: id, name, department\_id, salary, hire\_date\n\nTable: departments\nColumns: id, name\n\n### Question:\nShow the average salary of employees in each department who were hired after 2019. List only departments with more than 5 employees. Order the results by average salary in descending order.\n\n### SQL:\n', params: SamplingParams(n=1, presence\_penalty=0.0, frequency\_penalty=0.0, repetition\_penalty=1.1, temperature=0.3, top\_p=0.8, top\_k=20, min\_p=0.0, seed=None, stop=[';'], stop\_token\_ids=[], bad\_words=[], include\_stop\_str\_in\_output=False, ignore\_eos=False, max\_tokens=150, min\_tokens=0, logprobs=None, prompt\_logprobs=None, skip\_special\_tokens=True, spaces\_between\_special\_tokens=True, truncate\_prompt\_tokens=None, guided\_decoding=None, extra\_args=None), prompt\_token\_ids: [14374, 9994, 12539, 510, 2556, 25, 8256, 198, 13965, 25, 877, 11, 829, 11, 9292, 842, 11, 16107, 11, 17983, 4164, 271, 2556, 25, 25215, 198, 13965, 25, 877, 11, 829, 271, 14374, 15846, 510, 7812, 279, 5461, 16107, 315, 8256, 304, 1817, 9292, 879, 1033, 21446, 1283, 220, 17, 15, 16, 24, 13, 1759, 1172, 25215, 448, 803, 1091, 220, 20, 8256, 13, 7217, 279, 3059, 553, 5461, 16107, 304, 43084, 1973, 382, 14374, 7870, 510], prompt\_embeds shape: None, lora\_request: None, prompt\_adapter\_request: None.

INFO: 127.0.0.1:35814 - "POST /v1/completions HTTP/1.1" 200 OK

INFO 07-30 10:48:53 [loggers.py:118] Engine 000: Avg prompt throughput: 8.9 tokens/s, Avg generation throughput: 8.4 tokens/s, Running: 0 reqs, Waiting: 0 reqs, GPU KV cache usage: 0.0%, Prefix cache hit rate: 0.0%

This log means:

* ✅ **Your request was processed successfully.**
* ⚙️ **Prompt & generation speeds** are shown (tokens per second).
* 🧠 **KV cache usage** is how much GPU memory is used for storing internal model state (0% = idle).
* 🔁 **Prefix cache hit rate** is 0% (not reusing previous prompt parts).
* 📭 "Running: 0 reqs, Waiting: 0 reqs" → no active or queued jobs now.

### Quick breakdown:

* ✅ **Acceptable** for a single request, especially if the prompt is complex or batching is off.
* ⚠️ **Could be faster** — usually 20–100 tokens/s is expected on a 3090 for a 1.5B model, depending on:
  + Prompt length
  + max\_num\_seqs setting
  + gpu-memory-utilization
  + Whether you're using batching
  + Whether model is still warming up

### Tips to improve speed:

* Try lowering --max-num-seqs if you serve just one user at a time.
* Set --gpu-memory-utilization a bit higher (like 0.9).
* Use shorter prompts or fewer simultaneous requests during testing.

### --gpu-memory-utilization

* **Definition**: This sets **how much of your GPU’s total memory** VLLM is allowed to use.
* **Example**:
  + If your GPU has 24 GB and --gpu-memory-utilization=0.85, VLLM will use up to **85% of 24 GB = ~20.4 GB**.
* **Why it matters**:
  + Prevents VLLM from using **all** memory so other system processes (like display drivers) don’t crash.
  + Helps avoid CUDA OOM (Out Of Memory) errors.

### 🔸 --max-num-seqs

* **Definition**: This controls the **maximum number of sequences (requests)** VLLM will **process in parallel**.
* **Why it matters**:
  + Higher values = better **GPU utilization** and **throughput**, but use more memory.
  + Lower values = safer for **smaller GPUs** or when memory is tight.
* **Example**:
  + --max-num-seqs=32 → VLLM can run up to 32 generation tasks **in parallel**.
  + If set too high, you might get OOM errors.

### 🧠 TL;DR:

| **Flag** | **Controls** | **Tradeoff** |
| --- | --- | --- |
| --gpu-memory-utilization | % of GPU memory VLLM can use | Higher = more capacity, but riskier |
| --max-num-seqs | Number of parallel requests | Higher = faster, but uses more memory |

### **What is Batching in VLLM?**

**Batching** means combining multiple requests together and processing them **at the same time** on the GPU.

### 🔧 **How Is Batching Enabled in VLLM?**

**Batching is automatically handled by VLLM.**  
You **don’t need to manually enable it**. It works by collecting incoming requests and grouping them if possible, based on similar token lengths and timing.

### 🎯 **What Does Batching Achieve?**

| **Benefit** | **Description** |
| --- | --- |
| 🧠 **Better GPU utilization** | Multiple requests can share the GPU’s compute power more efficiently. |
| ⚡ **Higher throughput** | More tokens per second → faster performance under load. |
| 🪄 **Lower latency at scale** | When multiple users hit the API, batching reduces total compute time. |
| 📉 **Reduced memory fragmentation** | Helps keep memory usage efficient and reduces overhead. |

### 🔍 Related Settings in VLLM

* --max-num-seqs: Upper limit for batching (how many can be handled in a single batch).
* --gpu-memory-utilization: Determines how much memory can be used across batches.
* --max-batch-size (optional): You can explicitly cap the batch size if needed.

### 🔄 Example Scenario

If 5 users send requests around the same time:

* Without batching: Each is handled separately → slower and inefficient.
* With batching: All 5 are grouped into a batch → processed in parallel → faster response.