Cheatsheet

1. Accessing the GPU Clusters

Step 1: Connect via University VPN

To access internal compute resources, **VPN access is mandatory**.

- Visit math.iisc.ac.in/sshinfo.html for detailed VPN setup instructions.
- Supported via **OpenVPN** / **Cisco AnyConnect**.
- Use your temporary team credentials to log in.
- Ensure VPN is active before proceeding to SSH.

Step 2: SSH into the Cluster

SSH (Secure Shell) is used to remotely access the terminal of the GPU nodes.

Basic SSH command:

```
ssh username@hostname
```

Example (within VPN):

```
ssh team01@1.2.3.4
```

- Replace `team01` with your assigned username, and `1.2.3.4` with the actual cluster's IP.
- You'll be prompted for your password.

Tips:

- If it's your first time connecting, type `yes` when asked to confirm the fingerprint.
- Use `screen` or `tmux` to prevent session loss during training.
- Do **not run jobs directly on the login node** — use SLURM to schedule jobs (explained in next slide).

2. Getting the Dataset

Option 1: Copy From Shared Directory (once you're on the cluster)

Each team has read-access to the shared dataset.

Command to copy dataset to your account:

```
cp -r /share/datasets/vehicle-detection ~/my-dataset
```

> -r stands for recursive, needed for folders.

Option 2: Download from Online Links

If you prefer working on your **local machine or online notebooks**, download the dataset using:

- **Google Drive**
- **Microsoft OneDrive**

These contain the **same dataset** as on the cluster.

OS-Specific Access Help

Linux / macOS

Open your Terminal and use:

```
ssh username@compute-node.math.iisc.ac.in
```

Use `scp` to copy files *to/from* your local machine:

```
scp username@compute-node.math.iisc.ac.in:~/my-dataset.zip .
```

Windows

Windows Terminal + WSL – recommended for full Linux-like experience

---

Linux Commands You Might Use

Command	Description
`ls`	List files in directory
`cd folder_name`	Change directory
`mkdir new_folder`	Make new folder
`cp source dest`	Copy file/folder
`mv old new`	Rename/move
`rm file.txt`	Delete a file
`rm -r folder`	Delete folder and its contents

3. Syncing Your Codebase

For Collaboration (Team of 4)

Each team has a **private Git repo** under our organization.

- Use Git to sync code between team members.
- We will look for:
  - Training & inference scripts
  - Code quality and structure
  - Version control & commit history
- We require you to provide a git commit hash when submitting your valset

Recommended Workflow:

```
git clone https://github.com/our-org/team01-vehicle-detection.git
git pull    # to fetch latest
git push    # to upload your changes
```

For Using the Cluster (Transfer to/from local)

Use `scp` to copy files (e.g., trained weights, data snippets, env files) between local and cluster.

Copy from local to cluster:

```
scp -r ./myfolder username@compute-node:~
```

Copy from cluster to local:

```
scp -r username@compute-node:~/outputs ./outputs
```

> Use -r for folders, and make sure you're connected to the VPN!

💡 Pro Tips

- Keep **your codebase clean** — modular functions, proper file structure.
- Push regularly — don't keep everything local!
- Avoid large binary files (like `.pt` models) in Git — use `scp` for those.

4. Online Notebooks (Colab, Kaggle)

Why Start Here?

Before jumping to cluster compute, **begin your development in a flexible, beginner-friendly environment**:

- Easier for debugging and experimenting
- No job scheduling or SSH hassle
- Pre-installed packages, GPU available
- Great for:
  - Dataset exploration & cleaning
  - Preprocessing & augmentations
  - Prototyping training pipelines
  - Training lightweight models

Recommended Platforms

Google Colab

- Free GPU (T4/P100)

- Integrates easily with Google Drive
- Mount your Git repo or upload datasets

Kaggle Notebooks

- 30h GPU quota/week (Tesla T4)
- Direct access to popular datasets
- Better for reproducibility & sharing

Suggested Workflow

1. Clone your **team repo** in the notebook:

   ```
   !git clone https://github.com/our-org/team01-vehicle-detection.git
   ```

2. Work on preprocessing / smaller model training.
3. Once confident, **move pipeline to the cluster** using `scp`.

> ⚠️ Avoid wasting SLURM hours on early trial-and-error stages.

5. Setting Up Your Python Environment

Why You Need This

Clusters **don't come with Python environments pre-installed.**

To manage dependencies cleanly and avoid version conflicts, we recommend using:

Miniconda

- Lightweight Python environment manager
- Ideal for isolated project setups
- Easy to use even for beginners

Installing Miniconda (one-time setup)

Step 1: Download Miniconda on the cluster

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

Step 2: Run the installer

```
bash Miniconda3-latest-Linux-x86_64.sh
```

> Follow the prompts — choose default options.

Step 3: Activate Conda

```
source ~/miniconda3/bin/activate
```

Creating and Using an Environment

Create your project environment:

```
conda create -n vehicle-env python=3.10
```

Activate it:

```
conda activate model-env
```

Install dependencies:

```
pip install -r requirements.txt
# or use conda install if preferred
```

Pro Tips

- Save your packages with:

  ```
  pip freeze > requirements.txt
  ```

- Conda environments are **persistent**, even after logout
- Avoid installing system-wide Python packages — always use your environment

6. Running Jobs on the Cluster (SLURM)

SLURM = Job Scheduler

Our GPU cluster uses **SLURM** to manage access to limited compute resources.

Resource & Scheduling Rules

- **1 job at a time per team** — No parallel jobs allowed
- 20 teams sharing **14 NVIDIA A6000 GPUs (48GB VRAM each)**
- Each team gets (once they move from being queued to execution):
  - **6 CPU cores**
  - **48GB RAM**
  - **1 GPU card**
- **Max runtime per job: 4 hours**
  - Jobs auto-requeued if incomplete
  - Use **checkpointing** to save progress
- **Storage limit: 200GB**

- Datasets take ~30–40GB
- Avoid large log files or unnecessary model dumps

Sample SLURM Job Script: `run_job.sh`

```
#!/bin/bash
#SBATCH --job-name=vehicle_train
#SBATCH --gres=gpu:1
#SBATCH --cpus-per-task=6
#SBATCH --mem=48G
#SBATCH --time=04:00:00
#SBATCH --output=logs/%x-%j.out

source ~/miniconda3/bin/activate vehicle-env
python train.py --config config.yaml
```

Submit the job:

```
sbatch run_job.sh
```

Check job status:

```
squeue -u your_username
```

Best Practices

- Use `logs/` **folder** to store SLURM output
- Save checkpoints every 30 mins
- Test scripts with **small epochs** on Colab/Kaggle before full runs
- Clean up unused models or temp files regularly
- Compress logs or old results if hitting storage limit

7. What is Checkpointing?

> Checkpointing = Saving your model's progress during training.

Why It Matters

- SLURM jobs **auto-terminate after 4 hours** and will be **requeued** (i.e. will re-run the job)
- Without checkpoints, you'll **lose all training progress**

How It Works

- Save model weights periodically:

```
torch.save(model.state_dict(), "checkpoint_epoch10.pt")
```

- On restart, resume training:

```
model.load_state_dict(torch.load("checkpoint_epoch10.pt"))
```

> Most training frameworks (PyTorch, TensorFlow, Keras) support it natively.

Pro Tip:

Save a checkpoint every **N epochs or M minutes**

→ Helps you recover from **interruptions or crashes**

7. Docker Submissions – How We Evaluate Your Model

Why Docker?

- Ensures **consistency**: same environment across all teams
- Guarantees **reproducibility**: same code, same output
- Allows **automated, isolated evaluation** at scale

> Your model will be tested inside your Docker image, on our test set, without manual intervention.

Submission Requirements

You'll submit:

```
docker_username/image_name:tag
```

Your Docker image **must be public** on Docker Hub

It must:

- Accept these CLI arguments:

```
--input_dir /path/to/testset --output_dir /path/to/results
```

- Write predictions to the `output_dir`
- Exit cleanly (no crashes, no unhandled exceptions)

Important Constraints

- Image size must be **< 50 GB**
  - No training datasets or unnecessary files inside
- No interactive scripts (e.g., `input()` calls, notebooks)

- Include **only what's needed for inference**

How We'll Run It:

We'll do something like:

```
docker run \
  -v /testdata:/input_dir \
  -v /results:/output_dir \
  docker_username/image_name:tag \
  --input_dir /input_dir --output_dir /output_dir
```

If your container crashes → **Submission is invalid**