Coding & Script Tips

史曜睿
AlphaLab, USTC
Oct 2025

Coding & Script Tips

- Basic Linux Commands
- Automation with Bash Script
- Monitor Your Run
- Learn from Official Tutorials

Basic Linux Commands

Built-in Linux Commands

Navigation & Directory Management:

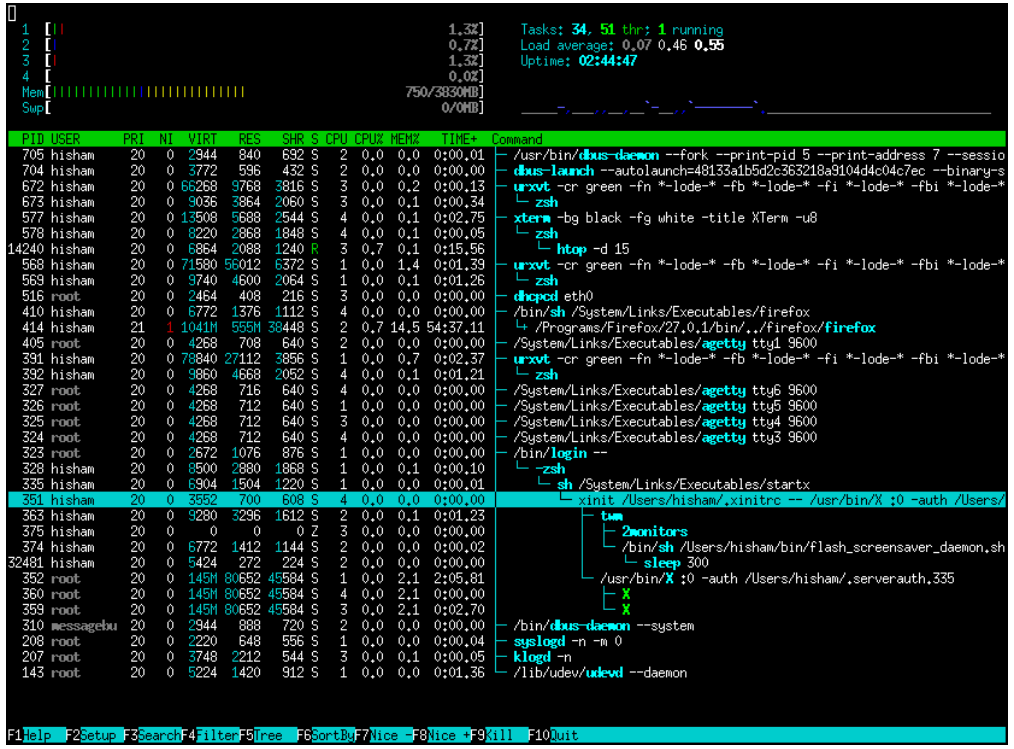
- `pwd` Prints the current working directory.
- `ls` Lists the contents of a directory.
- `cd [directory]` Changes the current directory to the specified one.
- `mkdir [dir_name]` Creates a new directory.
- `rmdir [dir_name]` Removes an empty directory.

File Management:

- `touch [file_name]` Creates an empty file or updates the timestamp of an existing one.
- `cat [file_name]` Displays the content of a file.
- `cp [source] [destination]` Copies files or directories.
- `mv [source] [destination]` Moves or renames files or directories.
- `rm [file_name]` Removes a file.
- `rm -r [dir_name]` Recursively removes a directory and its contents.

Basic Linux Commands

Useful Third-party Tools: Resource Monitoring

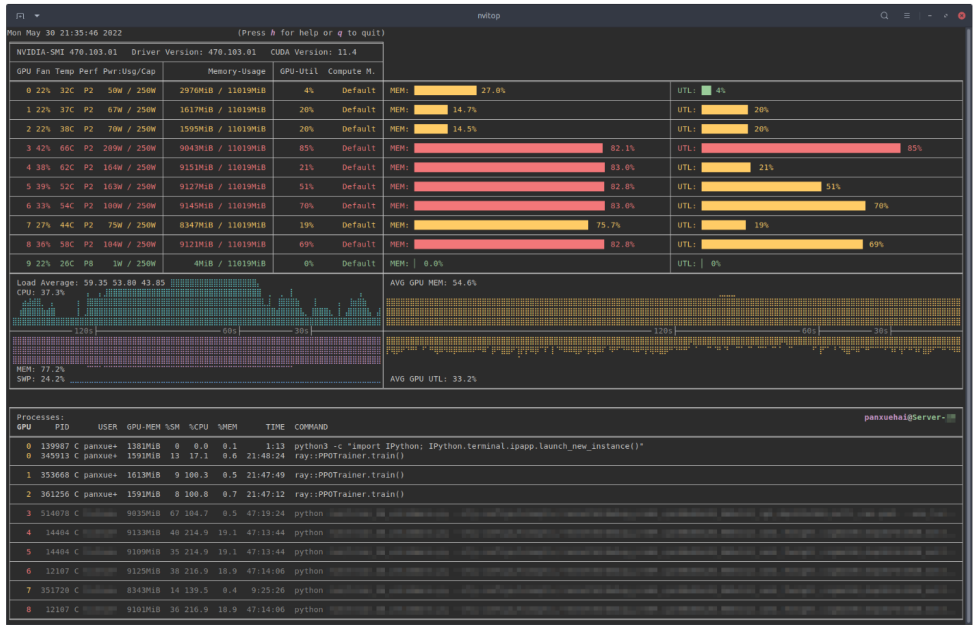


Htop: interactive process viewer

nvitop

Python 3.8+ | pypi v1.5.3 | conda v1.5.3 | docs passing | downloads 4M | Stars 6.2k | license Apache-2.0

An interactive NVIDIA-GPU process viewer and beyond, the one-stop solution for GPU process management. The full API references host at https://nvitop.readthedocs.io.

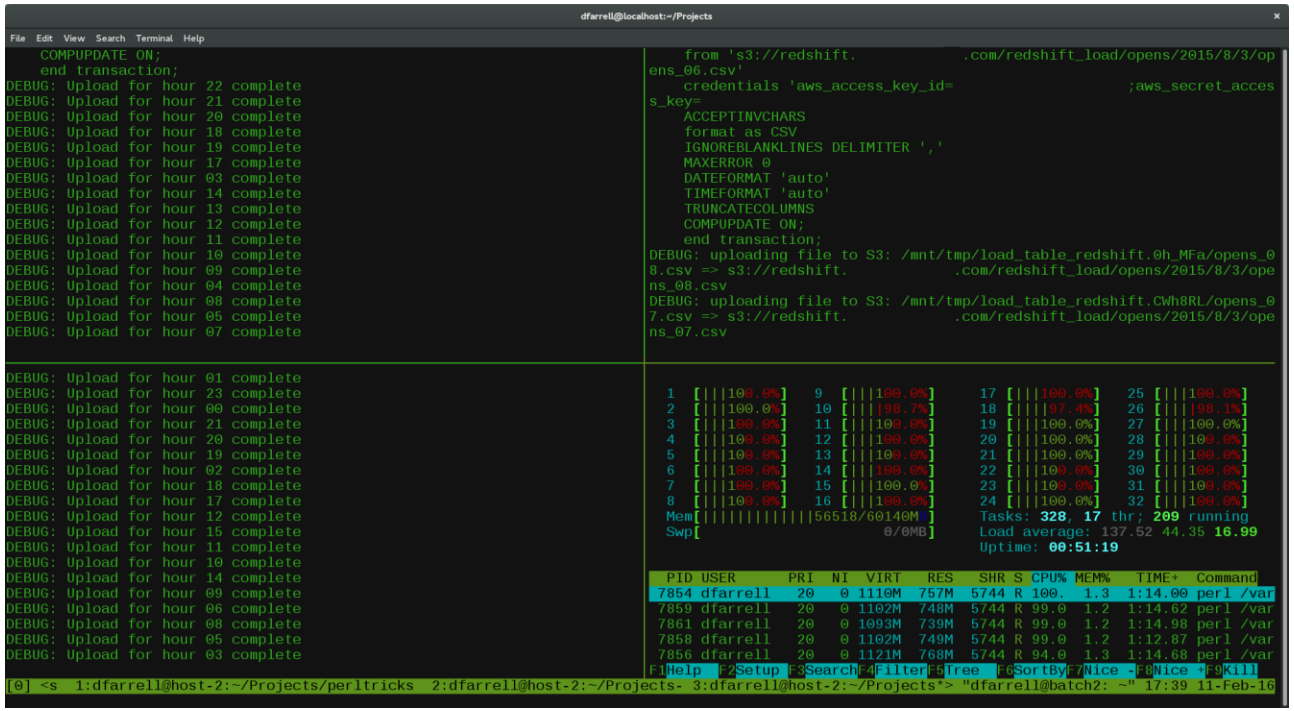


Nvitop/nvidia-smi/gpustat: NVIDIA-GPU process viewer

https://htop.dev/
https://github.com/XuehaiPan/nvitop

Basic Linux Commands

Useful Third-party Tools: Background Management



tmux: terminal multiplexer

https://github.com/tmux/tmux/wiki
https://tmuxcheatsheet.com/

Basic Linux Commands

Useful Third-party Tools: Background Management

Essential tmux Commands:

- Start a new session `tmux new -s my_session_name`
- Re-attach to a running session `tmux attach -t my_session_name`
- kill/delete session mysession `tmux kill-session -t mysession`
- List sessions `tmux ls`

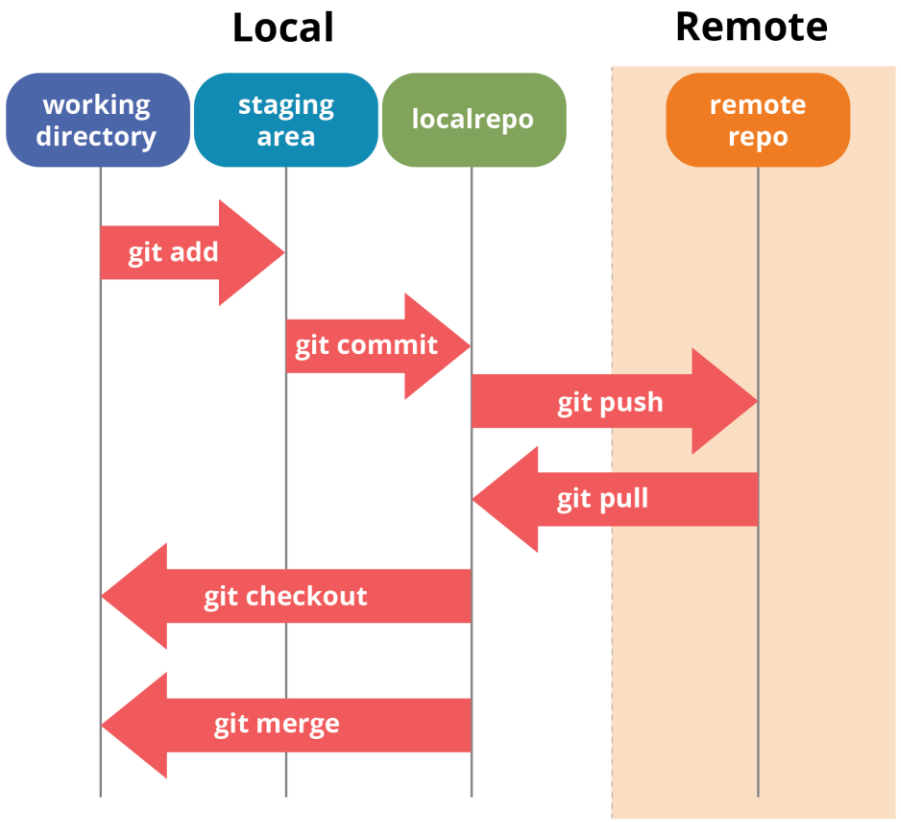
Shortcuts:

- Detach from the session Press `Ctrl+b`, then `d`
- Rename session Press `Ctrl+b`, then `$`
- Session and Window Preview Press `Ctrl+b`, then `w`
- Create window Press `Ctrl+b`, then `c`
- Switch between windows Press `Ctrl+b`, then `1~9`
- Split the current pane with a vertical line Press `Ctrl+b`, then `%`
- Split the current pane with a vertical line Press `Ctrl+b`, then `"`

https://github.com/tmux/tmux/wiki
https://tmuxcheatsheet.com/

Basic Linux Commands

Version Management with Git



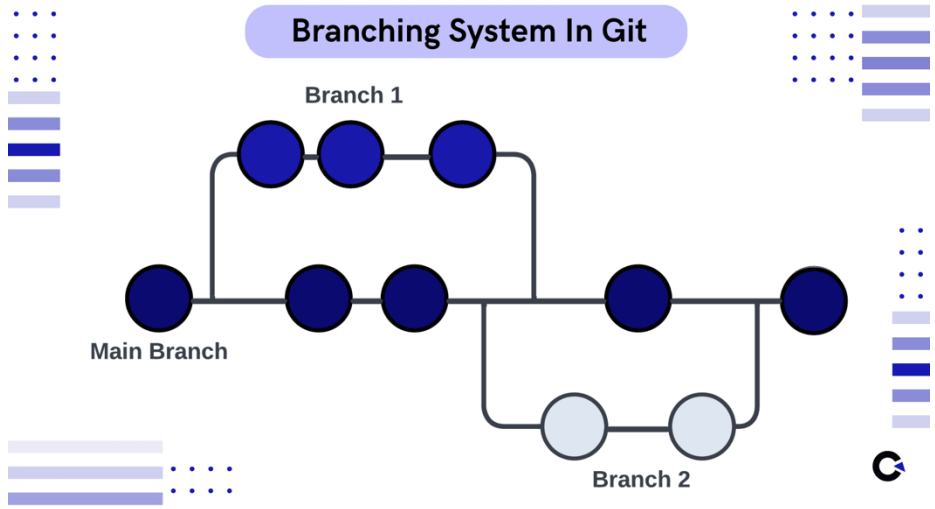
Repo Management:

- **`git add`**: Moves changes from the working directory to the staging area.
- **`git commit`**: Takes everything in the staging area and saves it as a permanent snapshot in local repository.
- **`git push`:** Uploads your committed changes from your local repository to the remote repository (like GitHub or GitLab).
- **`Git pull`**: the opposite of push.
- **`git checkout`**: update working directory to match a specific version from local repository
- **`git merge`**: combine changes from different branches in your local repository

https://liaoxuefeng.com/books/git/create-repo/index.html

Basic Linux Commands

Version Management with Git



Branch Management:

- **`git branch <branch-name>`**: lists all local branches.
- **`git switch/checkout`**: switch to an existing branch.
- **`git switch -c/checkout -b <branch-name>`:** create a new branch and immediately switch to it.
- **`git merge <branch-name>`**: combines the changes from another branch into your current branch.
- **`git branch -d <branch-name>:`** delete a branch.
- **`git reset –soft/hard <commit>`:** go back to <commit> while keeping/discarding all changes from your staging area

https://liaoxuefeng.com/books/git/create-repo/index.html

Coding & Script Tips

- ✓ Basic Linux Commands
- ☐ Automation with Bash Script
- ☐ Monitor Your Run
- ☐ Learn from Official Tutorials

Automation with Bash Script

Variables in Bash

- Vars can be assigned by **`var_name=value`**
- Vars can be accessed via **`$var_name`**

```bash
#!/bin/bash

name="John Doe"
echo "Hello, $name!"
number=42
echo "The number is $number"

```

```bash
#!/bin/bash

echo "Your PATH is $PATH"
export CUDA_VISIBLE_DEVICES=0
echo "Using GPU $CUDA_VISIBLE_DEVICES"

```

- Environment variables are special variables that affect the way processes run on your system.
- Env vars can be assigned via:
  - **`export var_name=value`**

Automation with Bash Script

Conditional statements

```bash
#!/bin/bash

echo "Please enter a number: "
read num

if [ $num -gt 0 ]; then
  echo "$num is positive"
elif [ $num -lt 0 ]; then
  echo "$num is negative"
else
  echo "$num is zero"
fi

```

Expressions that produce a boolean result, either true or false, are called conditions.

There are several ways to evaluate conditions, including **`if`**, **`if-else`**, **`if-elif-else`**, and nested conditionals.

Automation with Bash Script

Looping and Branching

The `for` keyword is followed by a variable name, a range of values, a `do` keyword marking the start of the loop block, and a `done` keyword marking the end of the loop block.

```
#!/bin/bash

for i in {1..5}; do
  echo "Iteration $i"
done

```

`While` loops execute a block of code as long as a specified condition is true.

```
#!/bin/bash

count=1
while [ $count -le 5 ]; do
  echo "Count is $count"
  ((count++))
done

```

Automation with Bash Script

Use bash for parameter swipe

```python
# train.py
import argparse
import time

# 1. Set up argument parser
parser = argparse.ArgumentParser(description='A simple training script.')
parser.add_argument('--lr', type=float, required=True, help='Learning rate')
parser.add_argument('--batch_size', type=int, required=True, help='Batch size')
parser.add_argument('--output_dir', type=str, required=True, help='Directory to save results')

args = parser.parse_args()

# 2. Print parameters to confirm they were received
print(f"--- Starting Training ---")
print(f"Learning Rate: {args.lr}")
print(f"Batch Size: {args.batch_size}")
print(f"Output Directory: {args.output_dir}")
print("Training for 5 seconds...")

# -------------
# Do some training here
# -------------

print("--- Training Finished ---")

```

Write a python script with argparse hyper-params

Automation with Bash Script

Use bash for parameter swipe

```python
# train.py
import argparse
import time

# 1. Set up argument parser
parser = argparse.ArgumentParser(description='A simple training script.')
parser.add_argument('--lr', type=float, required=True, help='Learning rate')
parser.add_argument('--batch_size', type=int, required=True, help='Batch size')
parser.add_argument('--output_dir', type=str, required=True, help='Directory to save results')

args = parser.parse_args()

# 2. Print parameters to confirm they were received
print(f"--- Starting Training ---")
print(f"Learning Rate: {args.lr}")
print(f"Batch Size: {args.batch_size}")
print(f"Output Directory: {args.output_dir}")
print("Training for 5 seconds...")

# -------------
# Do some training here
# -------------

print("--- Training Finished ---")

```

Write a python script with argparse hyper-params

```bash
#!/bin/bash

echo "Starting hyperparameter sweep..."

# 1. Define the parameters you want to test
LEARNING_RATES=(0.01 0.001 0.0001)
BATCH_SIZES=(32 64 128)
BASE_OUTPUT_DIR="experiments"

# 2. Create the base directory for all experiments
mkdir -p $BASE_OUTPUT_DIR

# 3. Nested loops to iterate through all combinations
for LR in "${LEARNING_RATES[@]}"
do
  for BS in "${BATCH_SIZES[@]}"
  do
    echo "---"
    echo "Running with LR=$LR and Batch Size=$BS"

    # Create a unique directory name for this experiment's results
    OUTPUT_DIR="${BASE_OUTPUT_DIR}/lr_${LR}_bs_${BS}"
    mkdir -p $OUTPUT_DIR

    # 4. Run the Python script with the current parameters
    python train.py \
      --lr $LR \
      --batch_size $BS \
      --output_dir $OUTPUT_DIR > "${OUTPUT_DIR}/training.log" &

  done
done

```

Use For loop in bash script to iterate over different parameter sets

Automation with Bash Script

Other useful bash commands

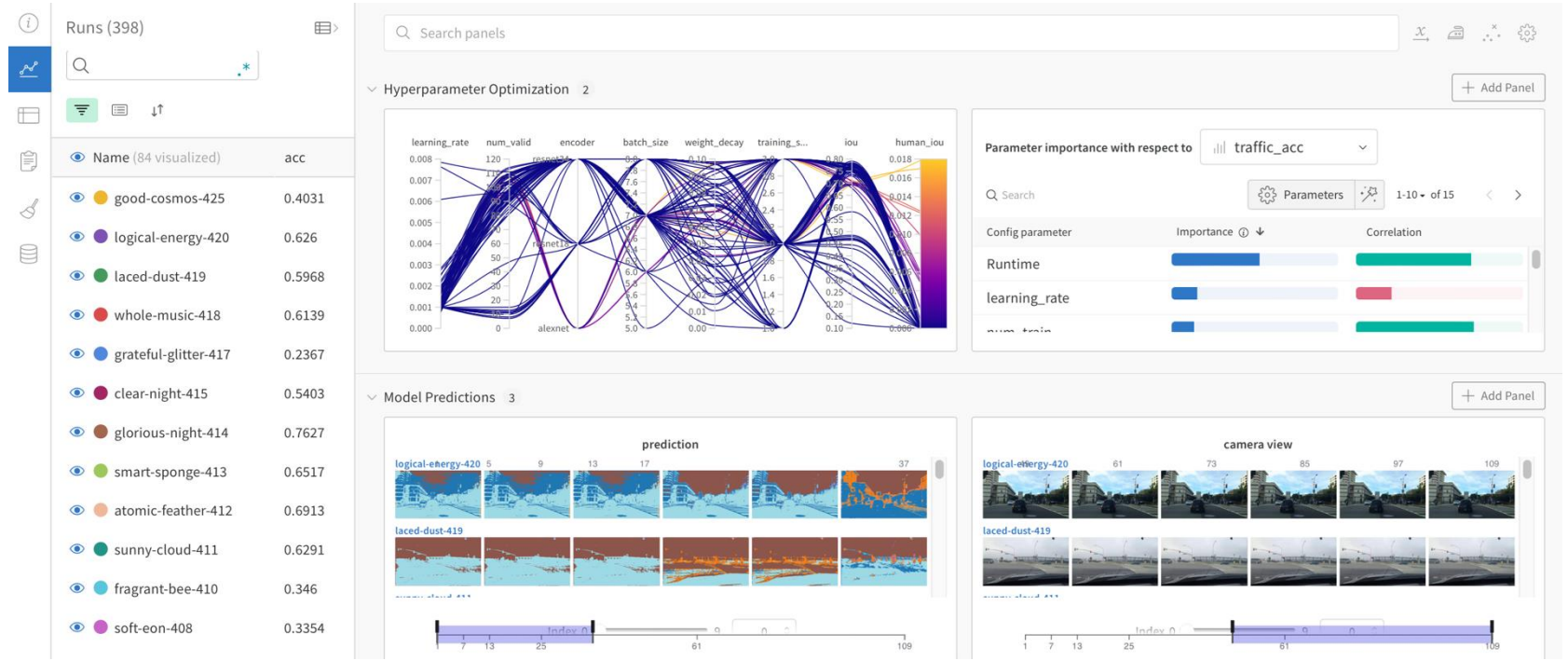
`ln -s /path/to/file /path/to/symlink`	Create soft links
`rsync -avzP ./file remote:~/my_project/`	Send files with Breakpoint Continuing
`<command> > log_file.out 2> log_file.err`	Redirects stdout and std err
`exec > log_file.out`	Redirect stdout to file
`exec 2> log_file.err`	Redirect stderr to file
`alias <shortcut>="<full command>"`	Make alias of an existing command
`df –h`	See free space on devices
`du -sh *`	See the size of each folder
`grep -r "<search_term>" .`	Search inside files

Coding & Script Tips

- ✓ Basic Linux Commands
- ✓ Automation with Bash Script
- ☐ Monitor Your Run
- ☐ Learn from Official Tutorials

Monitor Your Run

Logging with WandB



WandB (weights and biases) is a powerful tool for live visuals of data

https://github.com/wandb/wandb

Monitor Your Run

Logging with WandB

Call `.watch` and pass in your PyTorch model to automatically log gradients and store the network topology. Next, use `.log` to track other metrics. The following example demonstrates an example of how to do this:

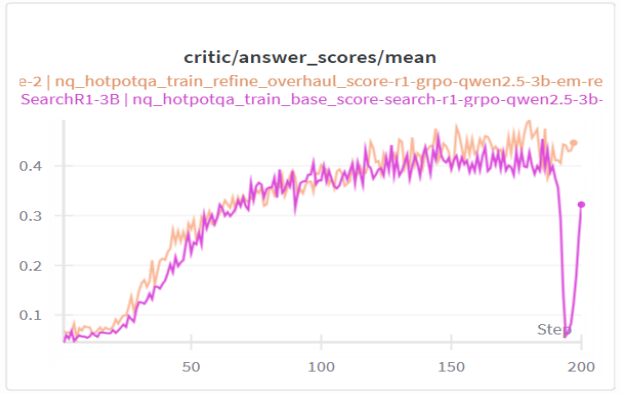
```
import wandb

# 1. Start a new run
run = wandb.init(project="gpt4")

# 2. Save model inputs and hyperparameters
config = run.config
config.dropout = 0.01

# 3. Log gradients and model parameters
run.watch(model)
for batch_idx, (data, target) in enumerate(train_loader):
    ...
    if batch_idx % args.log_interval == 0:
        # 4. Log metrics to visualize performance
        run.log({"loss": loss})
```

Quick start (PyTorch for example)

https://github.com/wandb/wandb

Monitor Your Run

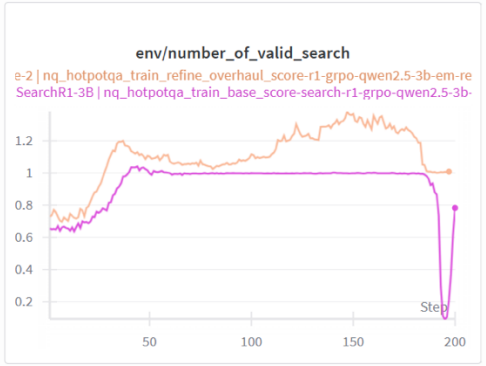
It's never too much to log

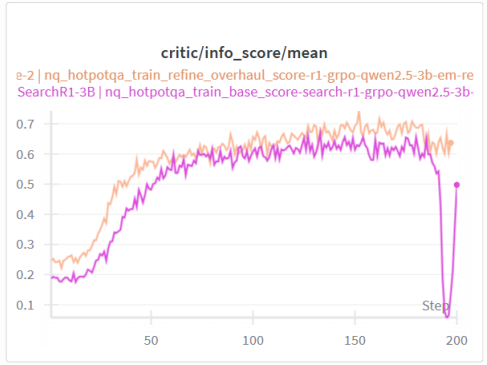


Comparison between AutoRefine and Search-R1

The models have similar **Answer Correctness**

… But the **Number of Search Calls** and **Search Success Rates** are different.

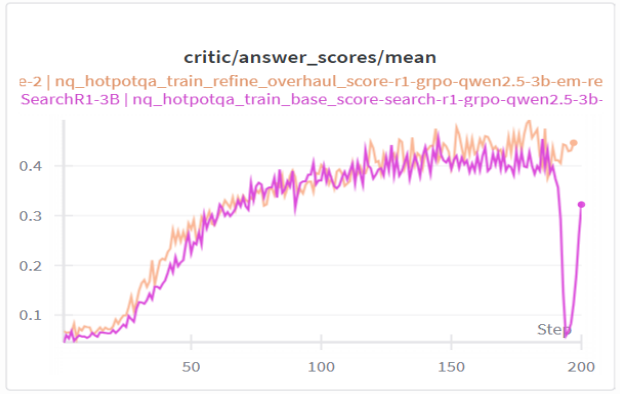


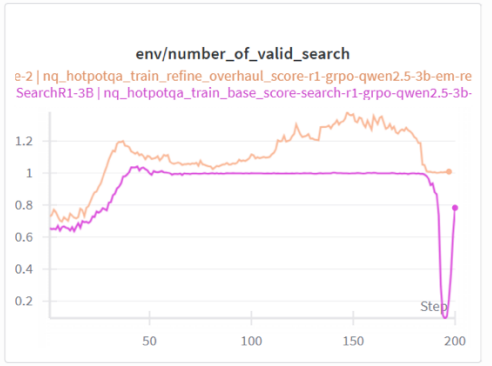


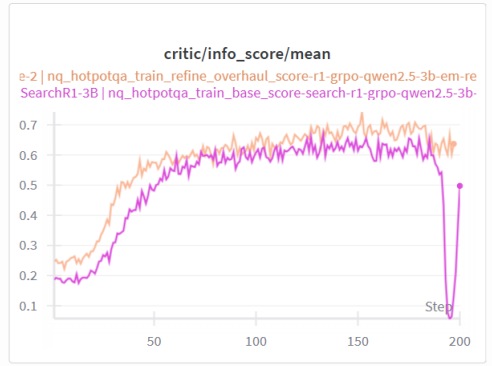
https://github.com/wandb/wandb

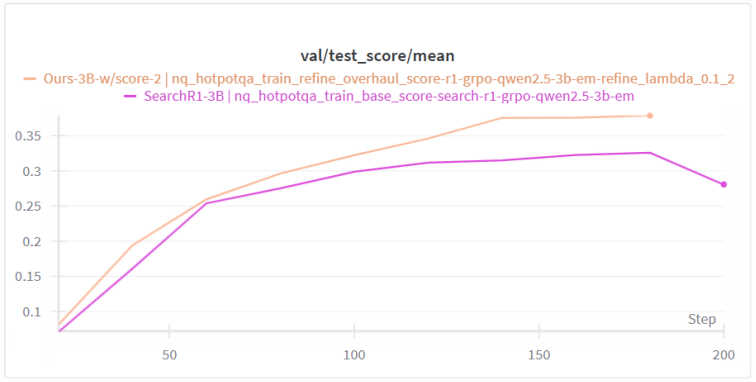
Monitor Your Run

- It's never too much to log



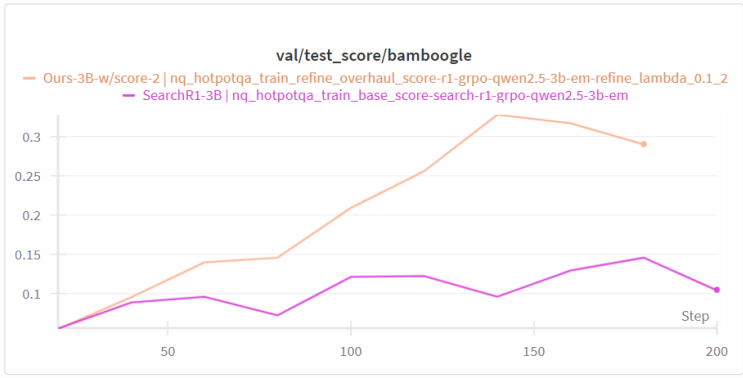






Mean Validation Acc

Val Acc on **Single-Hop** Dataset



Val Acc on **Multi-Hop** Dataset

https://github.com/wandb/wandb

Monitor Your Run

An Alternative for Chinese Users: SwanLab

README | Code of conduct | Contributing | Apache-2.0 license | Security

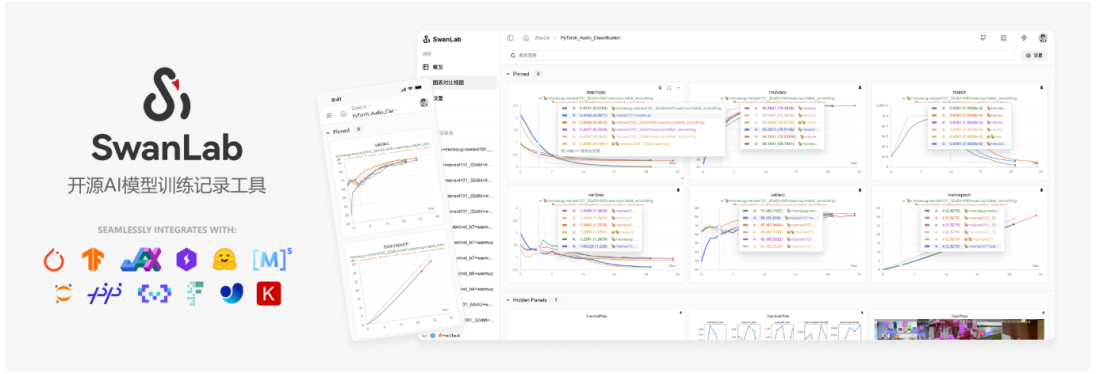
SwanLab

一个开源、现代化设计的深度学习训练跟踪与可视化工具
同时支持云端/离线使用，适配30+主流框架，与你的实验代码轻松集成

SwanLab 在线版 · 文档 · 报告问题 · 建议反馈 · 更新日志 · 基线社区

release v0.6.10 | docker v2.1 | stars 2.8k | issues 55 open | contributors 27 | license apache 2.0

Tracking in SwanLab | last commit last saturday | pypi v0.6.10 | WeChat 微信 | downloads 252k | Open in Colab



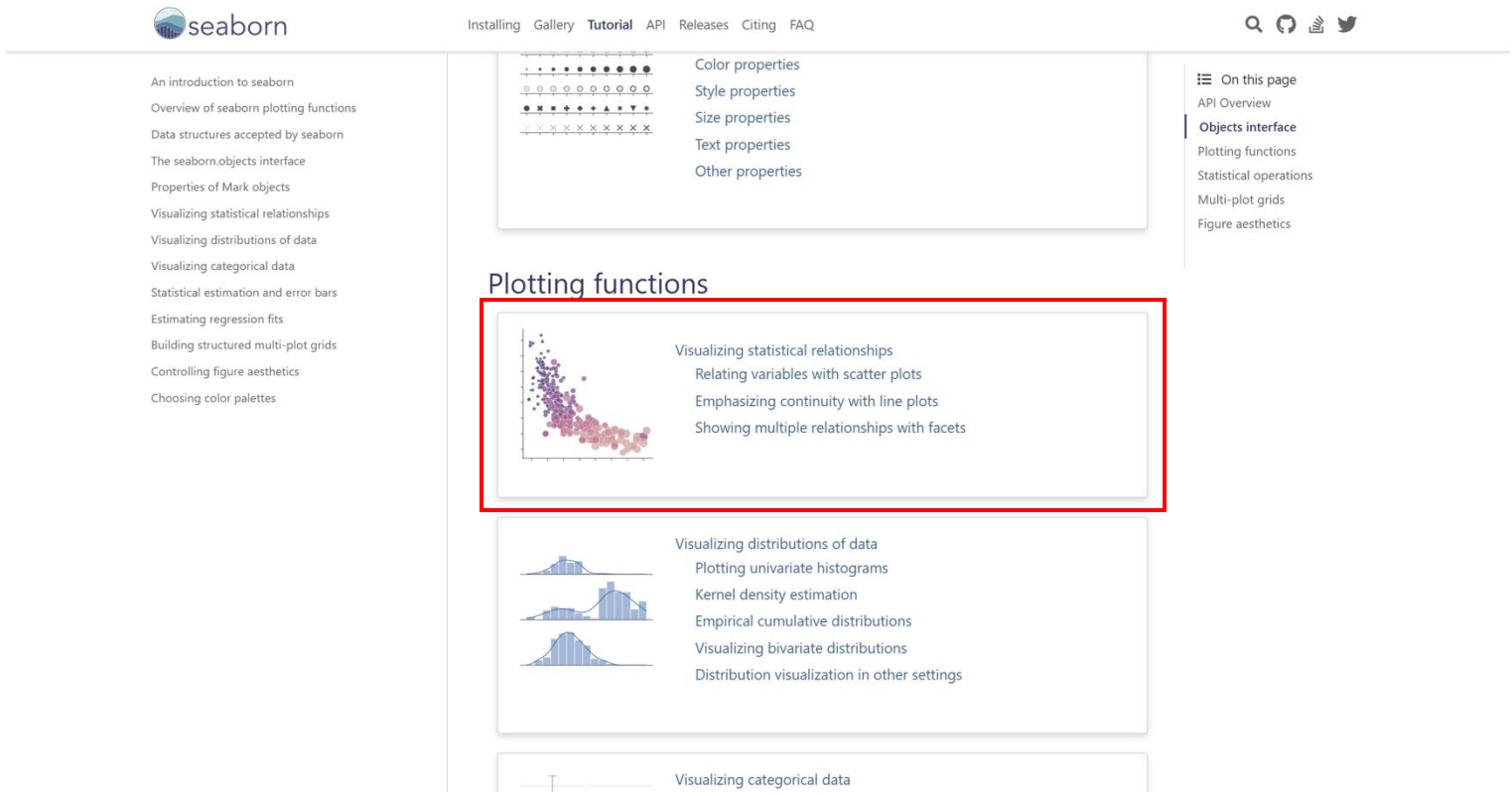
https://github.com/SwanHubX/SwanLab

Coding & Script Tips

- ✓ Basic Linux Commands
- ✓ Automation with bash
- ✓ Monitor Your Run
- ☐ Learn from Official Tutorials (Examples)

Learn from Official Tutorials (Example 1)

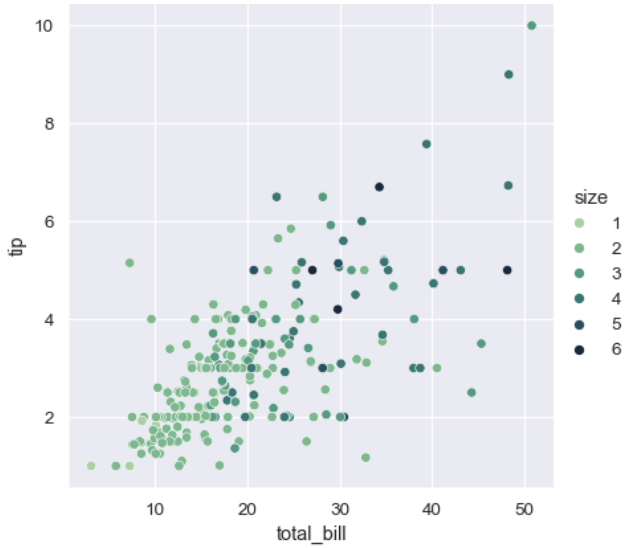
Seaborn: Visualizing statistical relationships



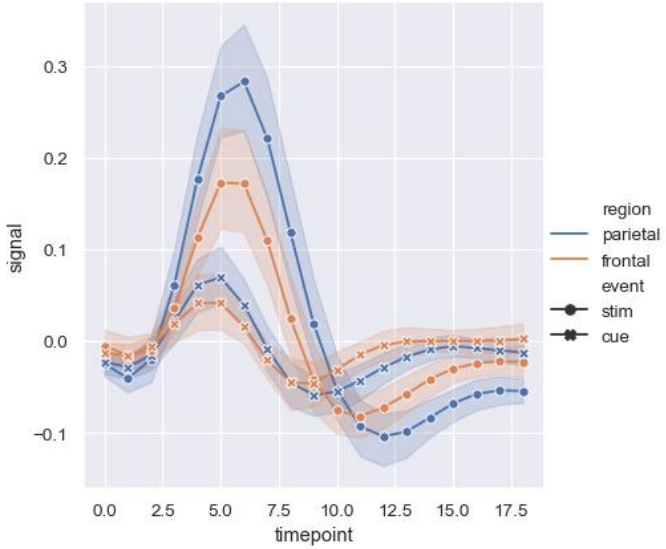
https://seaborn.pydata.org/tutorial.html

Learn from Official Tutorials (Example 1)

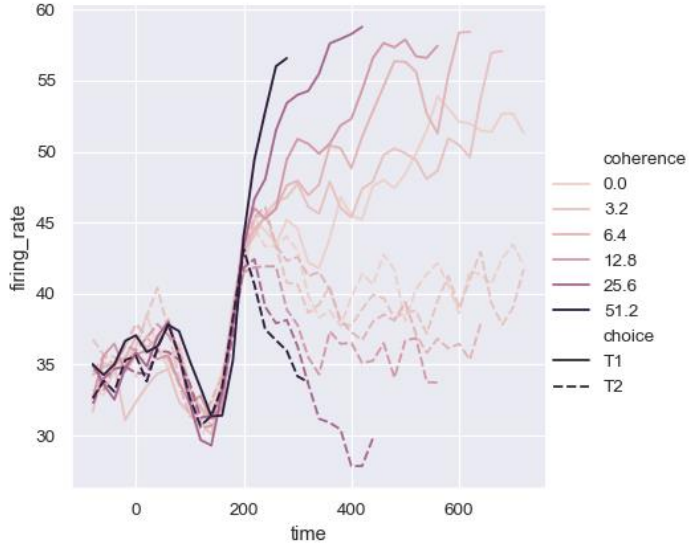
Seaborn: Visualizing statistical relationships



Relating variables with scatter plots



Aggregation and representing uncertainty

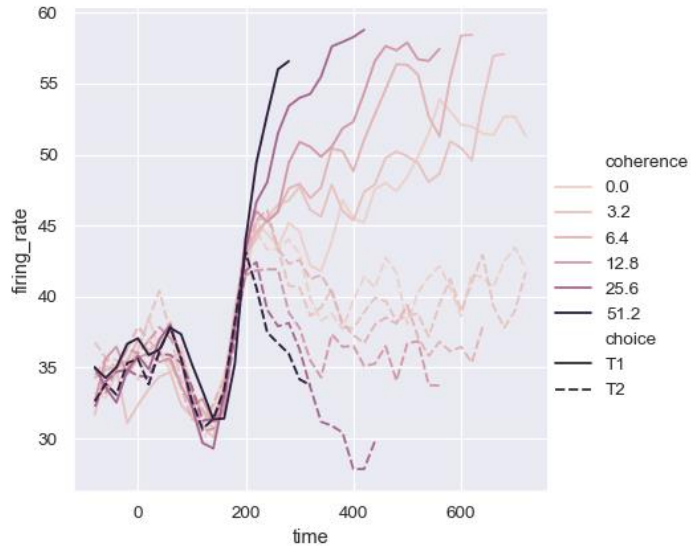


Plotting subsets of data with semantic mappings

https://seaborn.pydata.org/tutorial/relational.html

Learn from Official Tutorials (Example 1)

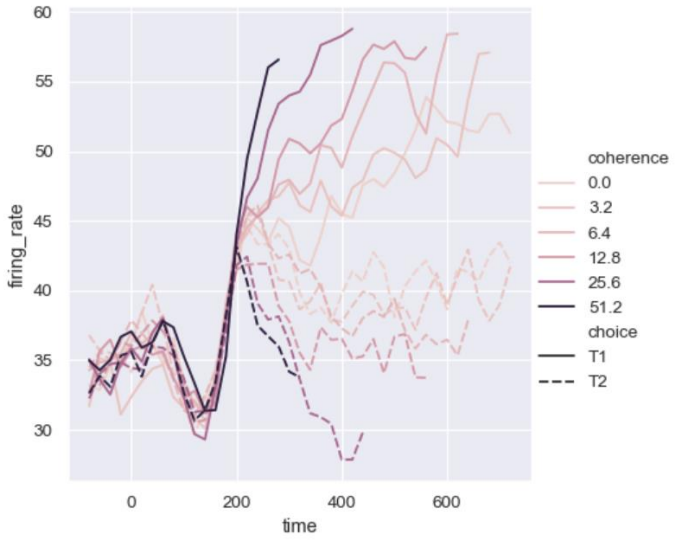
Seaborn: Visualizing statistical relationships



Plotting subsets of data with semantic mappings

The default colormap and handling of the legend in `lineplot()` also depends on whether the hue semantic is categorical or numeric:

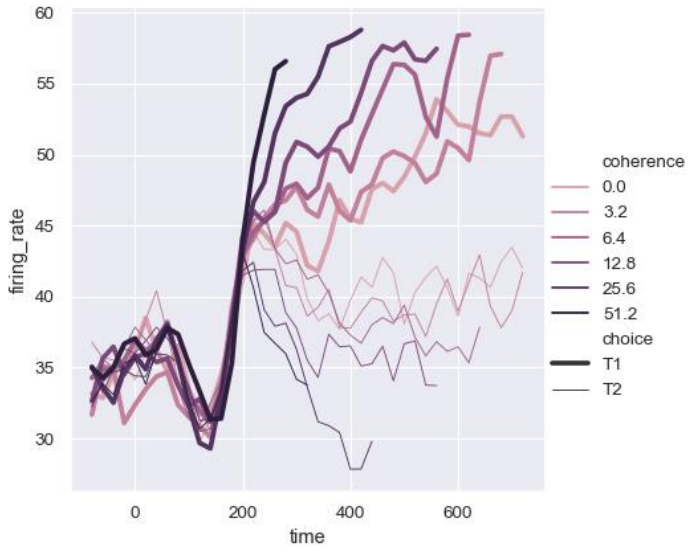
```
dots = sns.load_dataset("dots").query("align == 'dots'")
sns.relplot(
    data=dots, kind="line",
    x="time", y="firing_rate",
    hue="coherence", style="choice",
)
```



https://seaborn.pydata.org/tutorial/relational.html

Learn from Official Tutorials (Example 1)

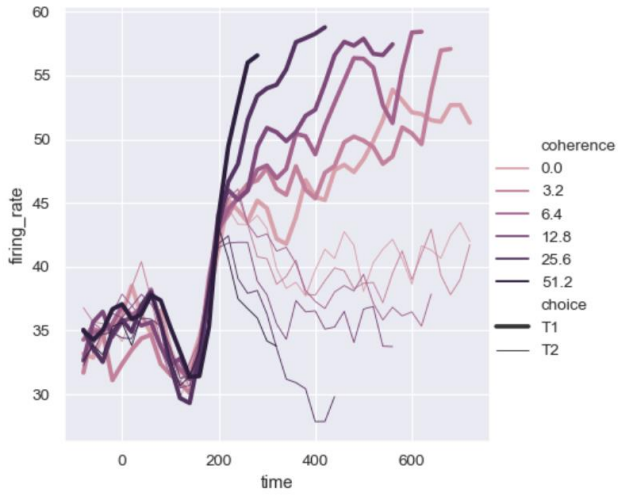
Seaborn: Visualizing statistical relationships



Plotting subsets of data with semantic mappings

While the `size` variable will typically be numeric, it's also possible to map a categorical variable with the width of the lines. Be cautious when doing so, because it will be difficult to distinguish much more than "thick" vs "thin" lines. However, dashes can be hard to perceive when lines have high-frequency variability, so using different widths may be more effective in that case:

```
sns.relplot(
    data=dots, kind="line",
    x="time", y="firing_rate",
    hue="coherence", size="choice", palette=palette,
)
```



https://seaborn.pydata.org/tutorial/relational.html

Learn from Official Tutorials (Example 2)

Verl: Do Multinode Training with Ray

Option 1: Launch Manually

Set up multinode ray cluster

1. Start head node with `ray start --head --dashboard-host=0.0.0.0`, there're 2 address you should care about:

- GCS address: `ray start --address=<address>`, where worker node should connect to.
- Dashboard address: `<address>:8265`, where you should submit job to the cluster.

```
(Trial 42705364 worker_0) open_verl [main] $ ray start --head --dashboard-host=0.0.0.0
Usage stats collection is disabled.

Local node IP: 10.124.46.192

--------------------
Ray runtime started.
--------------------

Next steps
  To add another node to this Ray cluster, run
    ray start --address='10.124.46.192:6379'   gcs address

  To connect to this Ray cluster:
    import ray
    ray.init()

  To submit a Ray job using the Ray Jobs CLI:
    RAY_ADDRESS='http://10.124.46.192:8265' ray job submit --working-dir . -- python my_script.py

  See https://docs.ray.io/en/latest/cluster/running-applications/job-submission/index.html
  for more information on submitting Ray jobs to the Ray cluster.

  To terminate the Ray runtime, run
    ray stop

  To view the status of the cluster, use
    ray status

  To monitor and debug Ray, view the dashboard at
    10.124.46.192:8265   dashboard address

  If connection to the dashboard fails, check your firewall settings and network configuration.
```

Step 1: Start head node

2. Start worker node with `ray start --address=<address>` you get above.

```
(Trial 42705364 worker_1) verl [main] $ ray start --address='10.124.46.192:6379'
Local node IP: 10.124.46.255
[2025-03-13 20:07:05,718 W 22335 22335] global_state_accessor.cc:429: Retrying to get node

--------------------
Ray runtime started.
--------------------

To terminate the Ray runtime, run
  ray stop
```

Step 2: Start worker nodes

3. Now you should see the cluster have 2 nodes with `ray status`.

```
(Trial 42705364 worker_0) verl [main] $ ray status
======== Autoscaler status: 2025-03-13 20:07:28.068210 ========
Node status
---------------------------------------------------------------
Active:
 1 node_a4af50030171229cc93b559050f5f190333641b0b539d6e68a518753
 1 node_bd45cce15518lc382bae2fb47104400e19f5f98b50d90083e537c1e7
Pending:
 (no pending nodes)                                  2 nodes
Recent failures:
 (no failures)

Resources
---------------------------------------------------------------
Usage:
 0.0/384.0 CPU
 0.0/16.0 GPU
 0B/3.47TiB memory
 0B/372.53GiB object_store_memory

Demands:
 (no resource demands)
```

Step 3: Now you can see all nodes

https://verl.readthedocs.io/en/latest/start/multinode.html

Alpha Lab

Thanks for Listening!