

Coding & Script Tips

史曜睿

AlphaLab, USTC

Oct 2025

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

□ 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

```
1 [ 1.3% Tasks: 34, 51 thr: 1 running
2 [ 0.7% Load average: 0.07 0.46 0.55
3 [ 1.3% Uptime: 02:44:47
4 [ 0.0%
Mem [ 750/3830MB
Swap [ 0/0MB

PID USER PRI NI VIRT RES SHR S CPU GPU MEM TIME+ Command
705 hisham 20 0 2344 840 692 S 2 0.0 0.0 0:00.01 /usr/bin/dbus-daemon --fork --print-pid 5 --print-address 7 --session
704 hisham 20 0 3772 596 432 S 2 0.0 0.0 0:00.00 dbus-launch --autolaunch=48133a1b5d2c363218a9104d4c04c7ec --binary-s
672 hisham 20 0 66268 9768 3816 S 3 0.0 0.2 0:00.13 urxvt -cr green -fn *-lode-* -fb *-lode-* -fi *-lode-* -fbi *-lode-*
673 hisham 20 0 9036 3864 2060 S 3 0.0 0.1 0:00.34 - zsh
577 hisham 20 0 13508 5688 2544 S 4 0.0 0.1 0:02.75 xterm -bg black -fg white -title XTerm -u8
578 hisham 20 0 8220 2868 1848 S 4 0.0 0.1 0:00.05 - zsh
14240 hisham 20 0 6864 2088 1240 R 3 0.7 0.1 0:15.66 htop -d 15
568 hisham 20 0 71590 56012 6372 S 1 0.0 1.4 0:01.39 urxvt -cr green -fn *-lode-* -fb *-lode-* -fi *-lode-* -fbi *-lode-*
569 hisham 20 0 3740 4600 2064 S 1 0.0 0.1 0:01.26 - zsh
516 root 20 0 2464 408 216 S 3 0.0 0.0 0:00.00 dhcpd eth0
410 hisham 20 0 6772 1376 1112 S 4 0.0 0.0 0:00.00 /bin/sh /System/Links/Executables/firefox
414 hisham 21 1 10411 5558 38448 S 2 0.7 14.5 54:37.11 /Programs/Firefox/27.0.1/bin/../firefox/firefox
405 root 20 0 4268 708 640 S 2 0.0 0.0 0:00.00 /System/Links/Executables/agetty tty1 9600
391 hisham 20 0 78840 27112 3896 S 1 0.0 0.7 0:02.37 urxvt -cr green -fn *-lode-* -fb *-lode-* -fi *-lode-* -fbi *-lode-*
392 hisham 20 0 9860 4668 2062 S 4 0.0 0.1 0:01.21 - zsh
327 root 20 0 4268 716 640 S 4 0.0 0.0 0:00.00 /System/Links/Executables/agetty tty6 9600
326 root 20 0 4268 712 640 S 1 0.0 0.0 0:00.00 /System/Links/Executables/agetty tty5 9600
325 root 20 0 4268 712 640 S 3 0.0 0.0 0:00.00 /System/Links/Executables/agetty tty4 9600
324 root 20 0 4268 712 640 S 4 0.0 0.0 0:00.00 /System/Links/Executables/agetty tty3 9600
323 root 20 0 2672 1076 876 S 1 0.0 0.0 0:00.00 /bin/login --
328 hisham 20 0 8500 2880 1868 S 1 0.0 0.1 0:00.10 - zsh
335 hisham 20 0 6904 1504 1220 S 1 0.0 0.0 0:00.01 sh /System/Links/Executables/startx
351 hisham 20 0 3552 700 608 S 4 0.0 0.0 0:00.00 xinit /Users/hisham/.xinitrc -- /usr/bin/X :0 -auth /Users/
363 hisham 20 0 9280 3296 1612 S 2 0.0 0.1 0:01.23 - x
375 hisham 20 0 0 0 0 Z 3 0.0 0.0 0:00.00 - 2monitors
374 hisham 20 0 6772 1412 1144 S 2 0.0 0.0 0:00.02 /bin/sh /Users/hisham/bin/flash_screensaver_daemon.sh
32481 hisham 20 0 5424 272 224 S 2 0.0 0.0 0:00.00 sleep 300
352 root 20 0 145H 80652 45584 S 1 0.0 2.1 2:05.81 /usr/bin/X :0 -auth /Users/hisham/.serverauth.335
360 root 20 0 145H 80652 45584 S 4 0.0 2.1 0:00.00 - x
353 root 20 0 145H 80652 45584 S 3 0.0 2.1 0:02.70 - x
310 messagebus 20 0 2944 888 720 S 2 0.0 0.0 0:00.00 /bin/dbus-daemon --system
208 root 20 0 2220 648 556 S 1 0.0 0.0 0:00.04 syslogd -n -m 0
207 root 20 0 3748 2212 544 S 3 0.0 0.1 0:00.05 klogd -n
143 root 20 0 5224 1420 912 S 1 0.0 0.0 0:01.36 /lib/udev/udev --daemon
```

Htop: interactive process viewer

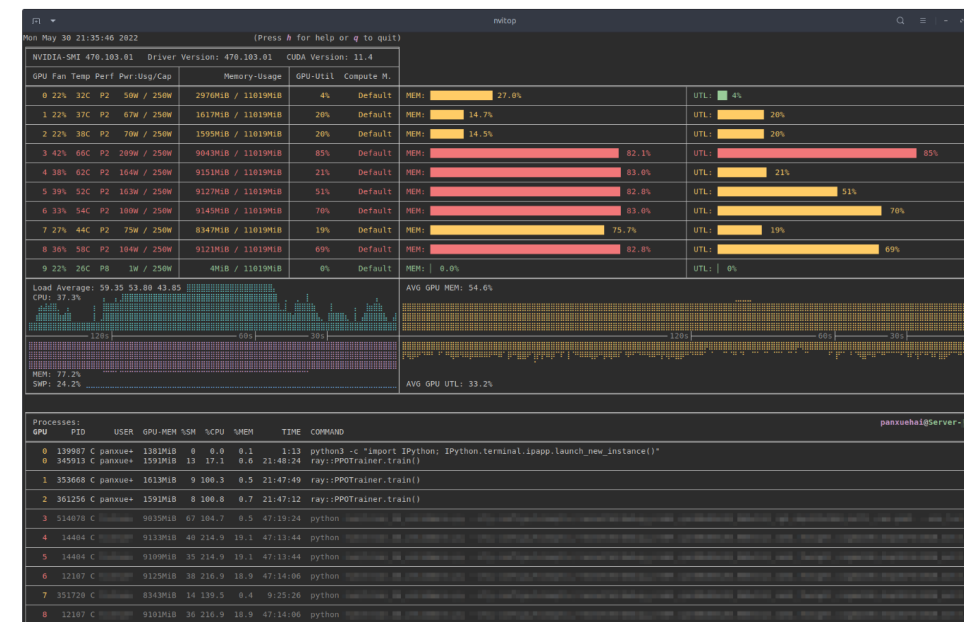
<https://htop.dev/>

<https://github.com/XuehaiPan/nvtop>

nvtop

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://nvtop.readthedocs.io>.



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

Basic Linux Commands

□ Useful Third-party Tools: Background Management

```
File Edit View Search Terminal Help
dfarrell@localhost:~/Projects
COMPUTEDATE ON;
end transaction;
DEBUG: Upload for hour 22 complete
DEBUG: Upload for hour 21 complete
DEBUG: Upload for hour 20 complete
DEBUG: Upload for hour 18 complete
DEBUG: Upload for hour 19 complete
DEBUG: Upload for hour 17 complete
DEBUG: Upload for hour 03 complete
DEBUG: Upload for hour 14 complete
DEBUG: Upload for hour 13 complete
DEBUG: Upload for hour 12 complete
DEBUG: Upload for hour 11 complete
DEBUG: Upload for hour 10 complete
DEBUG: Upload for hour 09 complete
DEBUG: Upload for hour 04 complete
DEBUG: Upload for hour 08 complete
DEBUG: Upload for hour 05 complete
DEBUG: Upload for hour 07 complete

DEBUG: Upload for hour 01 complete
DEBUG: Upload for hour 23 complete
DEBUG: Upload for hour 00 complete
DEBUG: Upload for hour 21 complete
DEBUG: Upload for hour 20 complete
DEBUG: Upload for hour 19 complete
DEBUG: Upload for hour 02 complete
DEBUG: Upload for hour 18 complete
DEBUG: Upload for hour 17 complete
DEBUG: Upload for hour 12 complete
DEBUG: Upload for hour 15 complete
DEBUG: Upload for hour 11 complete
DEBUG: Upload for hour 10 complete
DEBUG: Upload for hour 14 complete
DEBUG: Upload for hour 09 complete
DEBUG: Upload for hour 06 complete
DEBUG: Upload for hour 08 complete
DEBUG: Upload for hour 05 complete
DEBUG: Upload for hour 03 complete

from 's3://redshift.
ens_06.csv'
credentials 'aws_access_key_id=
s_key=
ACCEPTINVCCHARS
format as CSV
IGNOREBLANKLINES DELIMITER ','
MAXERROR 0
DATEFORMAT 'auto'
TIMEFORMAT 'auto'
TRUNCATECOLUMNS
COMPUTEDATE ON;
end transaction;
DEBUG: uploading file to S3: /mnt/tmp/load_table_redshift.0h_MFa/opens_0
8.csv => s3://redshift.
ns_08.csv
DEBUG: uploading file to S3: /mnt/tmp/load_table_redshift.CWh8RL/opens_0
7.csv => s3://redshift.
ns_07.csv

1 [||||100.0%] 9 [||||100.0%] 17 [||||100.0%] 25 [||||100.0%]
2 [||||100.0%] 10 [||||100.0%] 18 [||||100.0%] 26 [||||100.0%]
3 [||||100.0%] 11 [||||100.0%] 19 [||||100.0%] 27 [||||100.0%]
4 [||||100.0%] 12 [||||100.0%] 20 [||||100.0%] 28 [||||100.0%]
5 [||||100.0%] 13 [||||100.0%] 21 [||||100.0%] 29 [||||100.0%]
6 [||||100.0%] 14 [||||100.0%] 22 [||||100.0%] 30 [||||100.0%]
7 [||||100.0%] 15 [||||100.0%] 23 [||||100.0%] 31 [||||100.0%]
8 [||||100.0%] 16 [||||100.0%] 24 [||||100.0%] 32 [||||100.0%]
Mem[|||||56518/60140MB]
Swp[|||||0/0MB]
Tasks: 328, 17 thr; 209 running
Load average: 137.52 44.35 16.99
Uptime: 00:51:19

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
7854 dfarrell 20 0 110M 757M 5744 R 100.0 1.3 1:14.00 perl /var
7859 dfarrell 20 0 1102M 748M 5744 R 99.0 1.2 1:14.62 perl /var
7861 dfarrell 20 0 1093M 739M 5744 R 99.0 1.2 1:14.98 perl /var
7858 dfarrell 20 0 1102M 749M 5744 R 99.0 1.2 1:12.87 perl /var
7856 dfarrell 20 0 1121M 768M 5744 R 94.0 1.3 1:14.68 perl /var

[0] <s 1:dfarrell@host-2:~/Projects/perltricks 2:dfarrell@host-2:~/Projects- 3:dfarrell@host-2:~/Projects*> "dfarrell@batch2: ~" 17:39 11-Feb-16
```

tmux: terminal multiplexer

□ Useful Third-party Tools: Background Management

Essential tmux Commands:

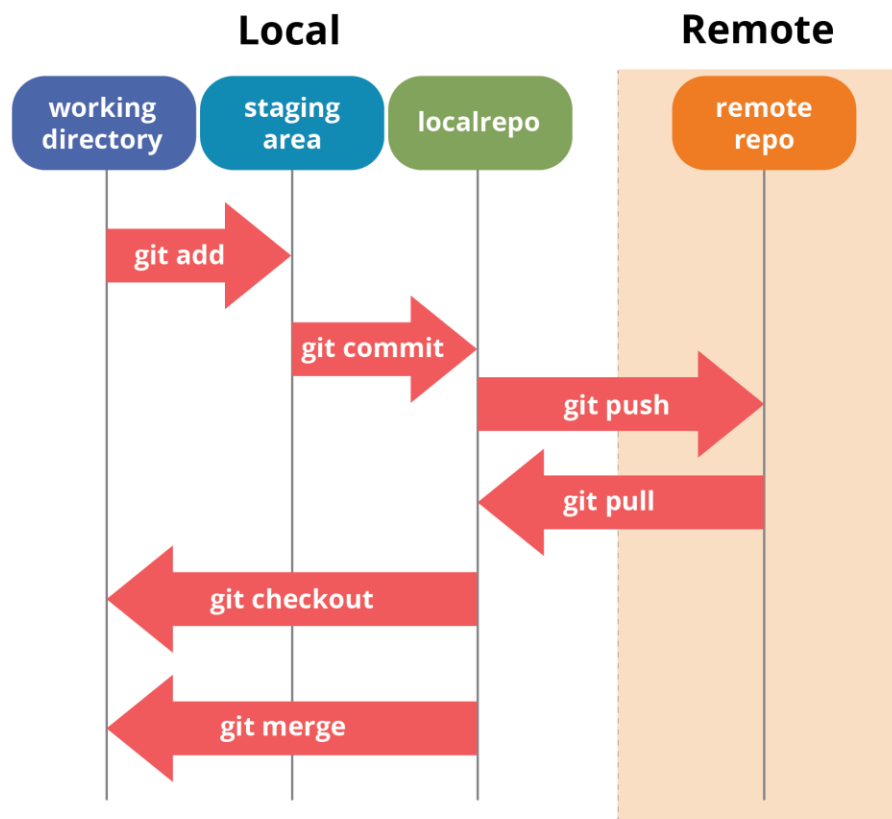
- Start a new session
- Re-attach to a running session
- kill/delete session mysession
- List sessions

```
tmux new -s my_session_name
tmux attach -t my_session_name
tmux kill-session -t mysession
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 horizontal line Press **Ctrl+b**, then **^**

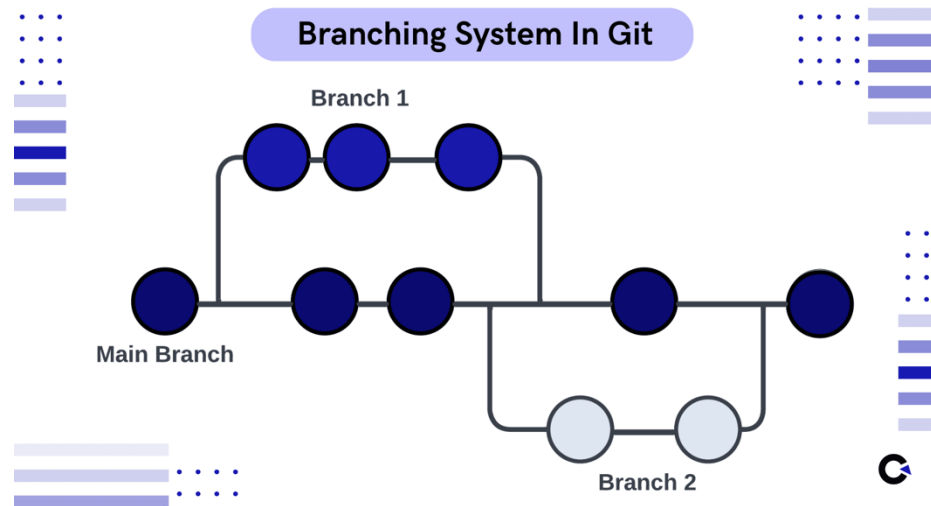
□ Version Management with Git



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

□ 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

- ✓ 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`

```
1  #!/bin/bash
2
3  name="John Doe"
4  echo "Hello, $name!"
5  number=42
6  echo "The number is $number"
7  |
```

```
1  #!/bin/bash
2
3  echo "Your PATH is $PATH"
4  export CUDA_VISIBLE_DEVICES=0
5  echo "Using GPU $CUDA_VISIBLE_DEVICES"
6  |
```

- 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

```
1  #!/bin/bash
2
3  echo "Please enter a number: "
4  read num
5
6  if [ $num -gt 0 ]; then
7      echo "$num is positive"
8  elif [ $num -lt 0 ]; then
9      echo "$num is negative"
10 else
11     echo "$num is zero"
12 fi
13
```

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.

```
1  #!/bin/bash
2
3  for i in {1..5}; do
4      echo "Iteration $i"
5  done
6
```

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

```
1  #!/bin/bash
2
3  count=1
4  while [ $count -le 5 ]; do
5      echo "Count is $count"
6      ((count++))
7  done
8
```

Automation with Bash Script

□ Use bash for parameter swipe

```
1  # train.py
2  import argparse
3  import time
4
5  # 1. Set up argument parser
6  parser = argparse.ArgumentParser(description='A simple training script.')
7  parser.add_argument('--lr', type=float, required=True, help='Learning rate')
8  parser.add_argument('--batch_size', type=int, required=True, help='Batch size')
9  parser.add_argument('--output_dir', type=str, required=True, help='Directory to save results')
10
11  args = parser.parse_args()
12
13  # 2. Print parameters to confirm they were received
14  print(f"--- Starting Training ---")
15  print(f"Learning Rate: {args.lr}")
16  print(f"Batch Size: {args.batch_size}")
17  print(f"Output Directory: {args.output_dir}")
18  print("Training for 5 seconds...")
19
20  # -----
21  # Do some training here
22  # -----
23
24  print("--- Training Finished ---")
25
```

Write a python script with argparse hyper-params

Automation with Bash Script

□ Use bash for parameter swipe

```
1 # train.py
2 import argparse
3 import time
4
5 # 1. Set up argument parser
6 parser = argparse.ArgumentParser(description='A simple training script.')
7 parser.add_argument('--lr', type=float, required=True, help='Learning rate')
8 parser.add_argument('--batch_size', type=int, required=True, help='Batch size')
9 parser.add_argument('--output_dir', type=str, required=True, help='Directory to save results')
10
11 args = parser.parse_args()
12
13 # 2. Print parameters to confirm they were received
14 print(f"--- Starting Training ---")
15 print(f"Learning Rate: {args.lr}")
16 print(f"Batch Size: {args.batch_size}")
17 print(f"Output Directory: {args.output_dir}")
18 print("Training for 5 seconds...")
19
20 # -----
21 # Do some training here
22 # -----
23
24 print("--- Training Finished ---")
25
```

Write a python script with argparse hyper-params

```
1 #!/bin/bash
2
3 echo "Starting hyperparameter sweep..."
4
5 # 1. Define the parameters you want to test
6 LEARNING_RATES=(0.01 0.001 0.0001)
7 BATCH_SIZES=(32 64 128)
8 BASE_OUTPUT_DIR="experiments"
9
10 # 2. Create the base directory for all experiments
11 mkdir -p $BASE_OUTPUT_DIR
12
13 # 3. Nested loops to iterate through all combinations
14 for LR in "${LEARNING_RATES[@]}"
15 do
16     for BS in "${BATCH_SIZES[@]}"
17     do
18         echo "---"
19         echo "Running with LR=$LR and Batch Size=$BS"
20
21         # Create a unique directory name for this experiment's results
22         OUTPUT_DIR="$BASE_OUTPUT_DIR/lr_${LR}_bs_${BS}"
23         mkdir -p $OUTPUT_DIR
24
25         # 4. Run the Python script with the current parameters
26         python train.py \
27             --lr $LR \
28             --batch_size $BS \
29             --output_dir $OUTPUT_DIR > "${OUTPUT_DIR}/training.log" &
30     done
31 done
32
33
```

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

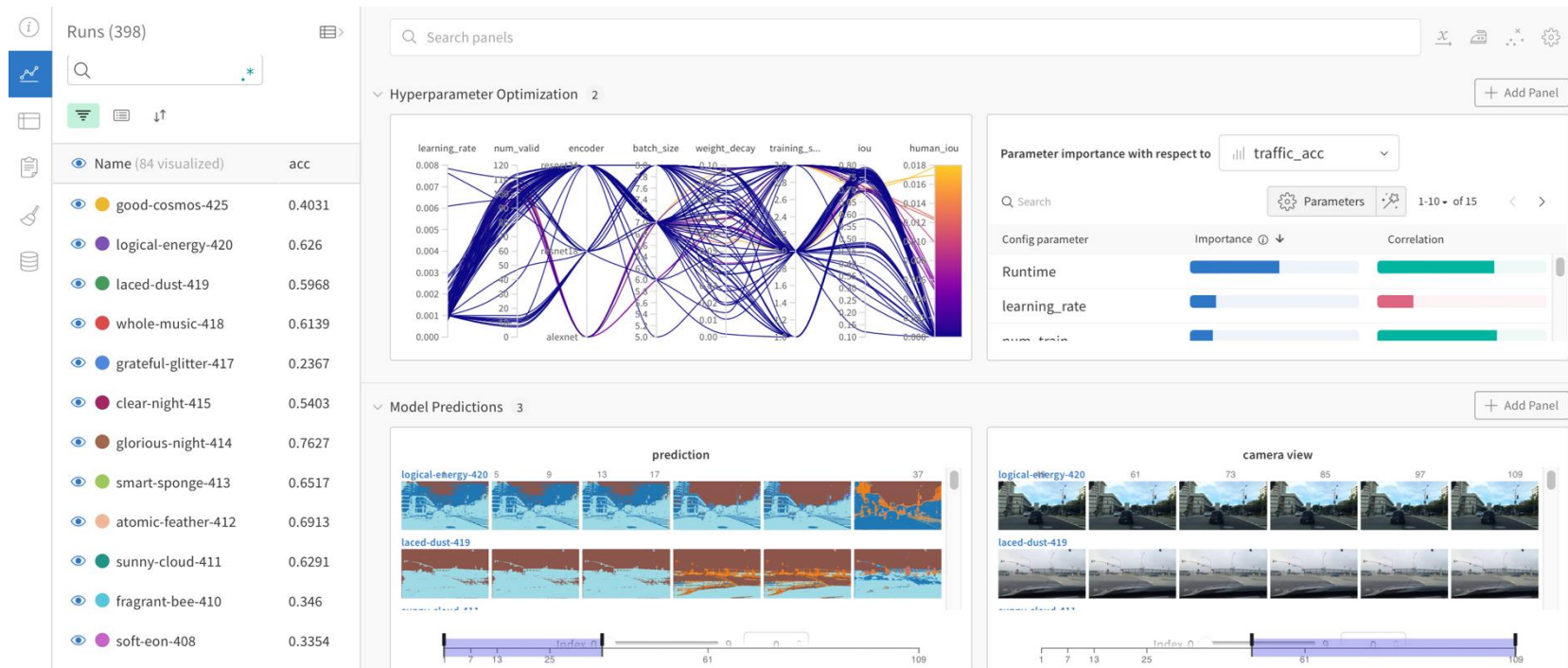
```
grep -r "<search_term>" .
```

Search inside files

- ✓ 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

□ 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

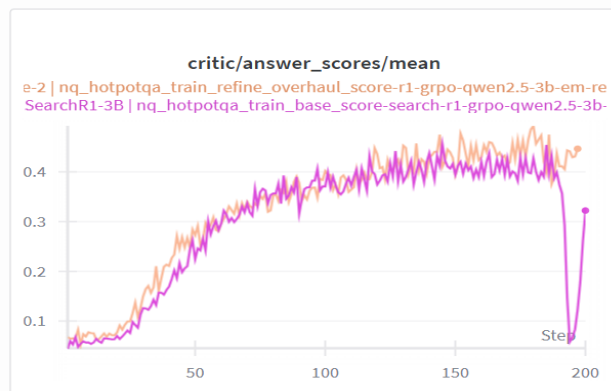
# 3. Log gradients and model parameters
run.watch(model)
for batch_idx, (data, target) in enumerate(train_loader):
    ...
    if batch_idx % args.log_interval == 0:
        # 4. Log metrics to visualize performance
        run.log({"loss": loss})
```



Quick start (PyTorch for example)

Monitor Your Run

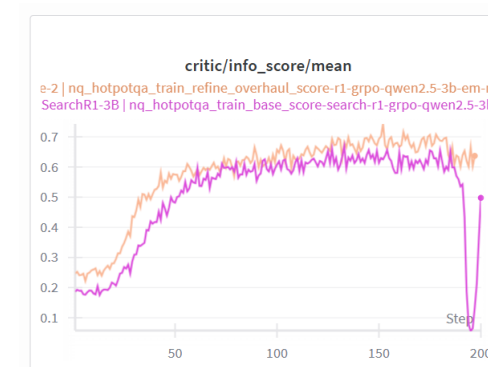
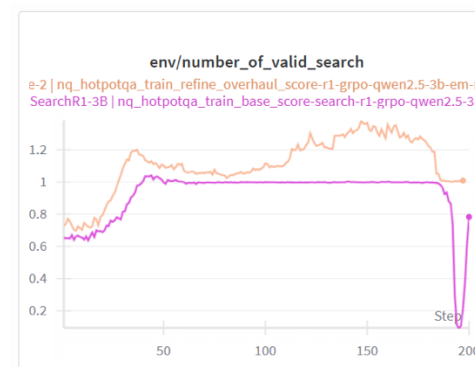
□ It's never too much to log



Comparison between **AutoRefine** and **Search-R1**

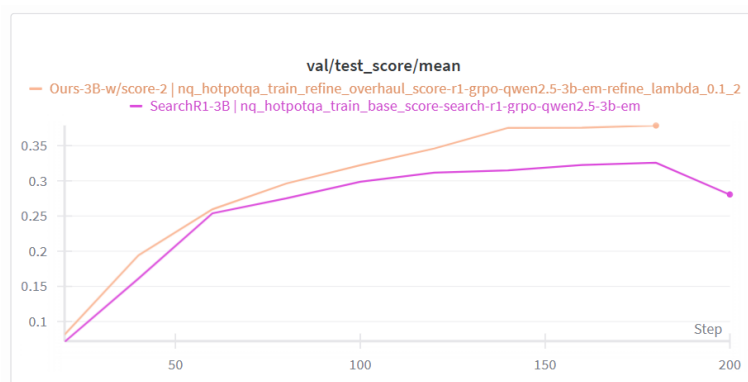
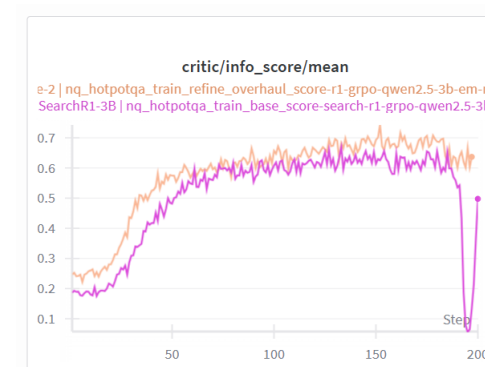
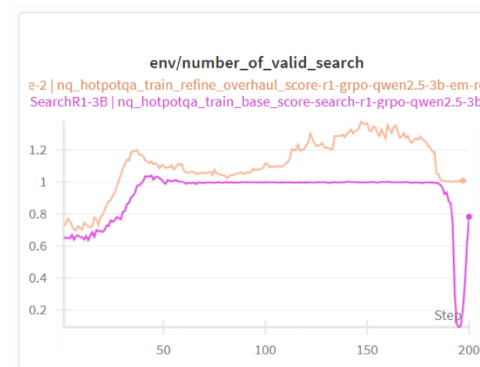
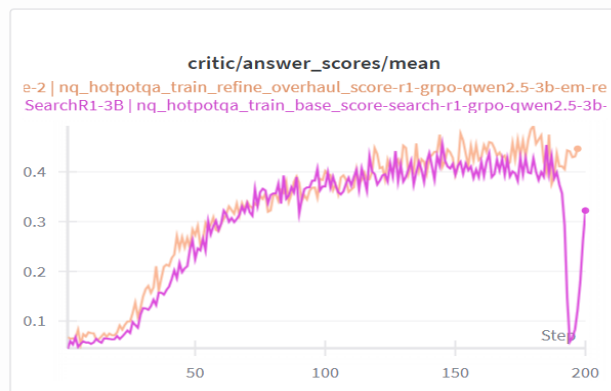
The models have similar **Answer Correctness**

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

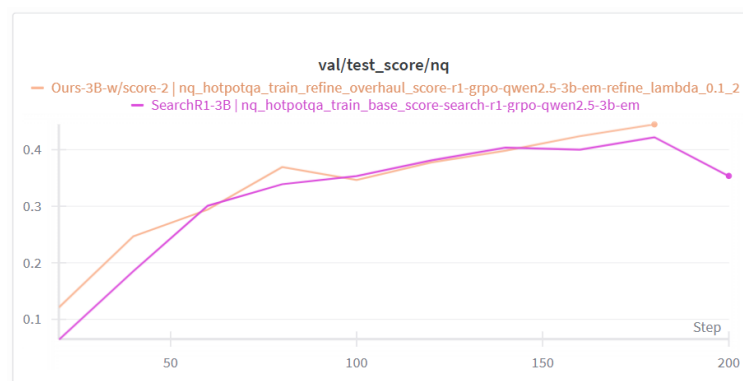


Monitor Your Run

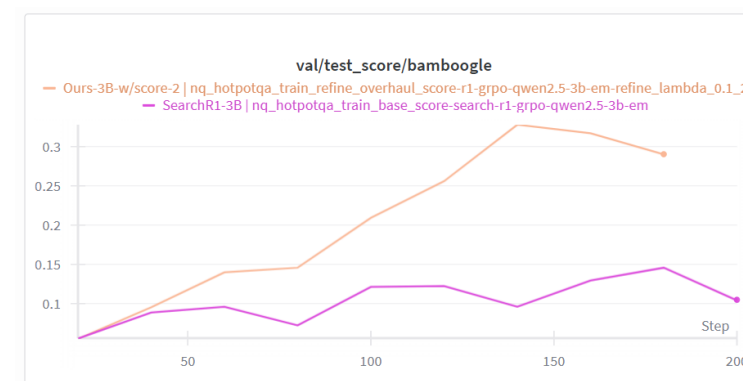
□ It's never too much to log



Mean Validation Acc



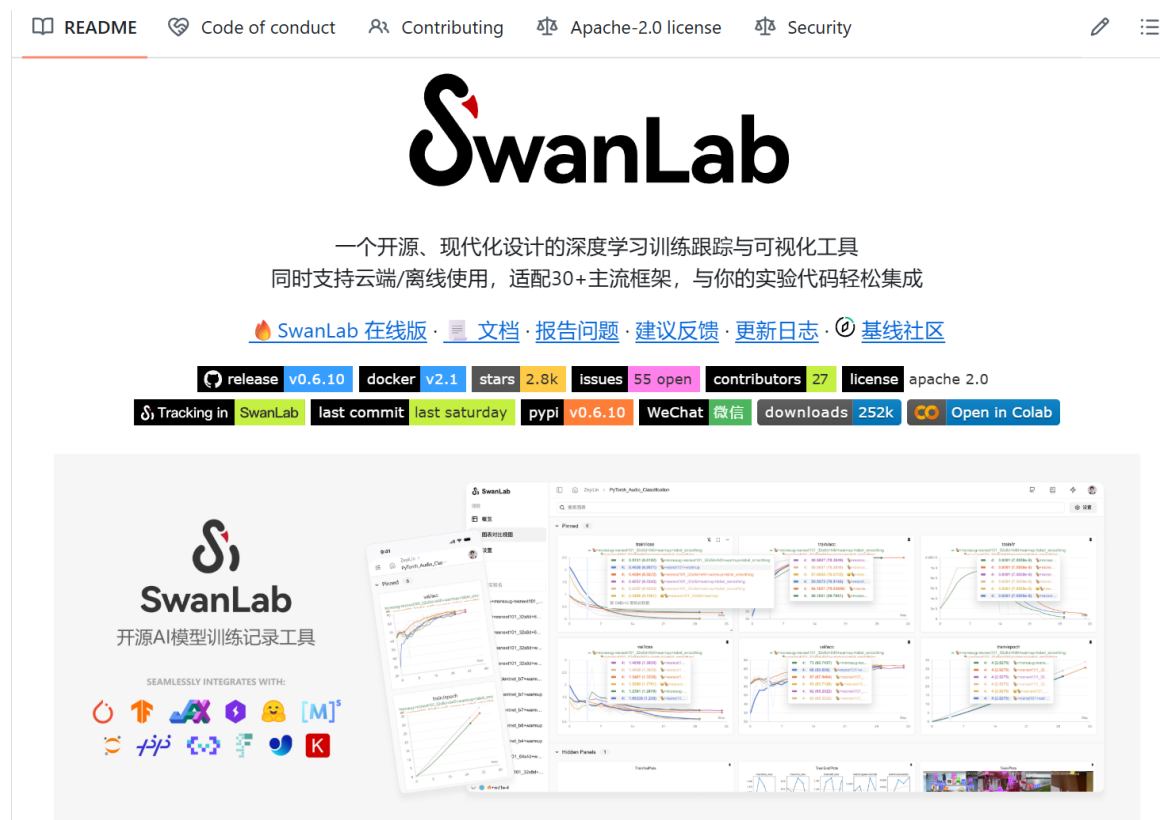
Val Acc on **Single-Hop** Dataset



Val Acc on **Multi-Hop** Dataset

Monitor Your Run

□ An Alternative for Chinese Users: SwanLab

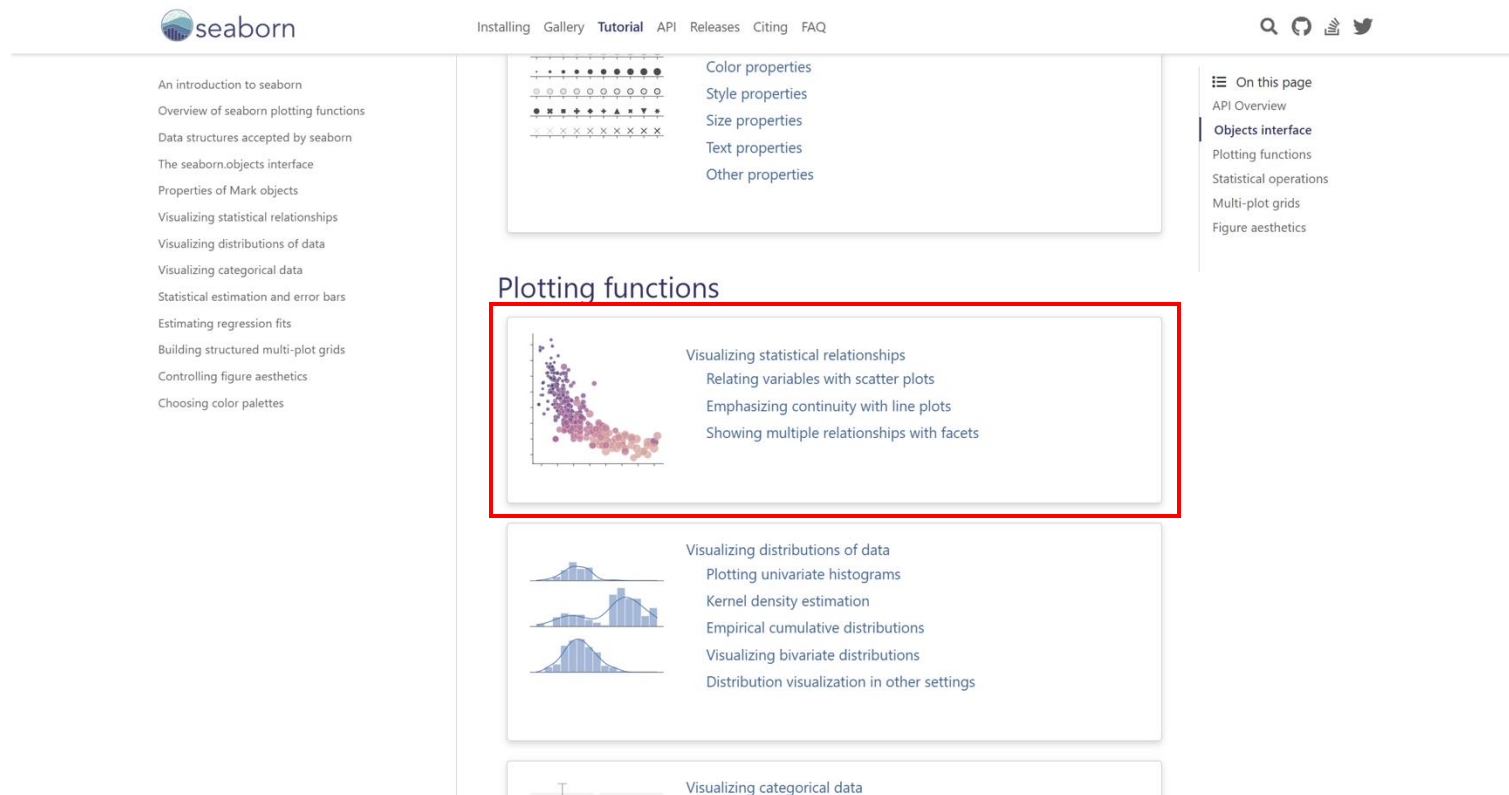


The screenshot displays the GitHub repository for SwanLab. The repository name "SwanLab" is prominently featured at the top. Below it, a description in Chinese states: "一个开源、现代化设计的深度学习训练跟踪与可视化工具" (An open-source, modern design deep learning training tracking and visualization tool) and "同时支持云端/离线使用, 适配30+主流框架, 与你的实验代码轻松集成" (Also supports cloud/offline use, compatible with 30+ mainstream frameworks, integrates easily with your experimental code). Navigation links include "README", "Code of conduct", "Contributing", "Apache-2.0 license", and "Security". A row of statistics shows: release v0.6.10, docker v2.1, stars 2.8k, issues 55 open, contributors 27, and license apache 2.0. Below this, a row of badges indicates: Tracking in SwanLab, last commit last Saturday, pypi v0.6.10, WeChat 微信, downloads 252k, and Open in Colab. The main content area shows a preview of the SwanLab application, which includes a sidebar with the SwanLab logo and the text "开源AI模型训练记录工具" (Open-source AI model training record tool). The main panel displays various training metrics and graphs, such as "train loss", "val loss", "train acc", "val acc", and "train f1", along with a "Hidden States" section.

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

Learn from Official Tutorials (Example 1)

□ Seaborn: Visualizing statistical relationships



The screenshot shows the Seaborn tutorial website. The left sidebar contains a list of topics, with 'Visualizing statistical relationships' highlighted. The main content area is titled 'Plotting functions' and is highlighted with a red box. It contains three sections: 'Visualizing statistical relationships' (with a scatter plot), 'Visualizing distributions of data' (with histograms and density plots), and 'Visualizing categorical data' (with a bar plot). The 'Visualizing statistical relationships' section lists: 'Relating variables with scatter plots', 'Emphasizing continuity with line plots', and 'Showing multiple relationships with facets'.

seaborn

Installing Gallery **Tutorial** API Releases Citing FAQ

On this page
API Overview
Objects interface
Plotting functions
Statistical operations
Multi-plot grids
Figure aesthetics

Visualizing statistical relationships

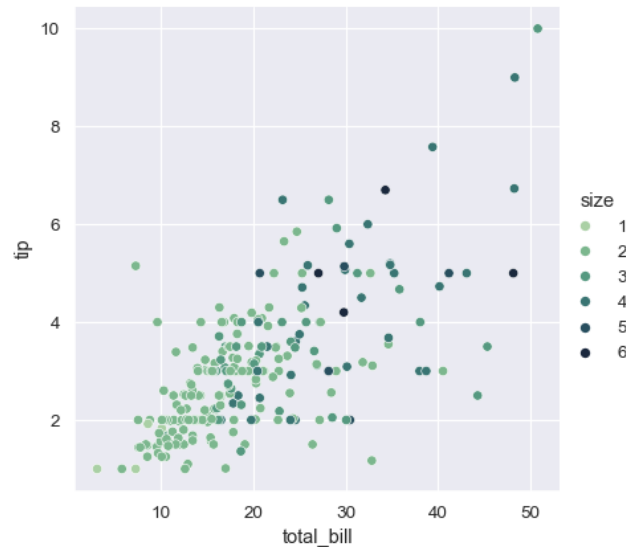
Relating variables with scatter plots
Emphasizing continuity with line plots
Showing multiple relationships with facets

Visualizing distributions of data
Plotting univariate histograms
Kernel density estimation
Empirical cumulative distributions
Visualizing bivariate distributions
Distribution visualization in other settings

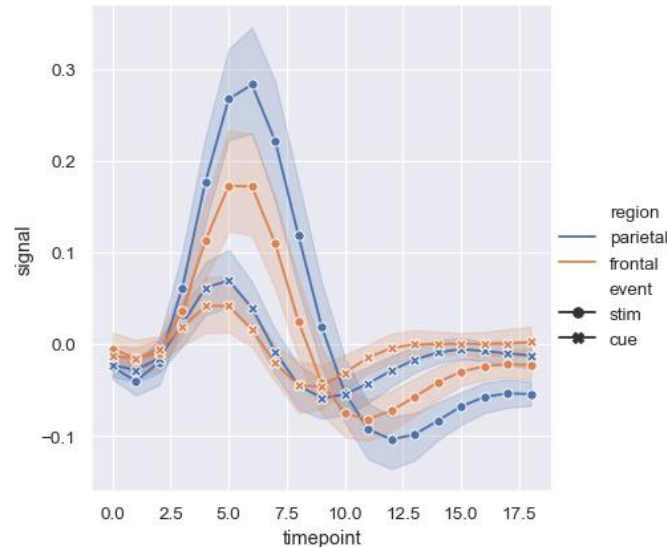
Visualizing categorical data

Learn from Official Tutorials (Example 1)

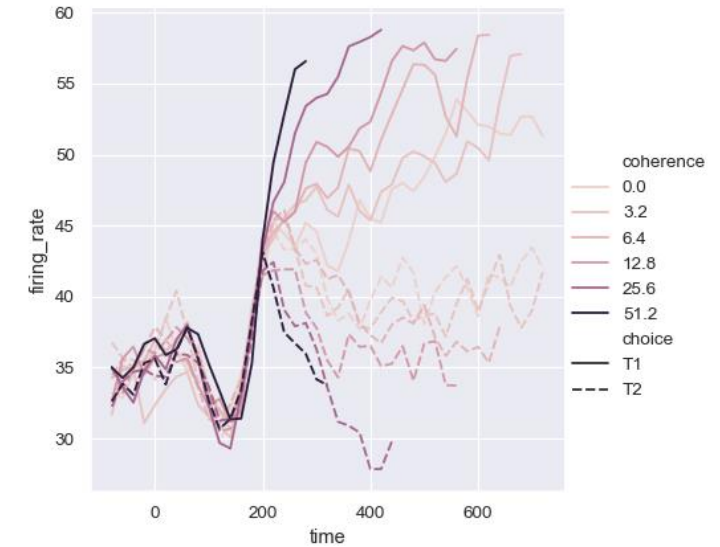
Seaborn: Visualizing statistical relationships



Relating variables
with scatter plots



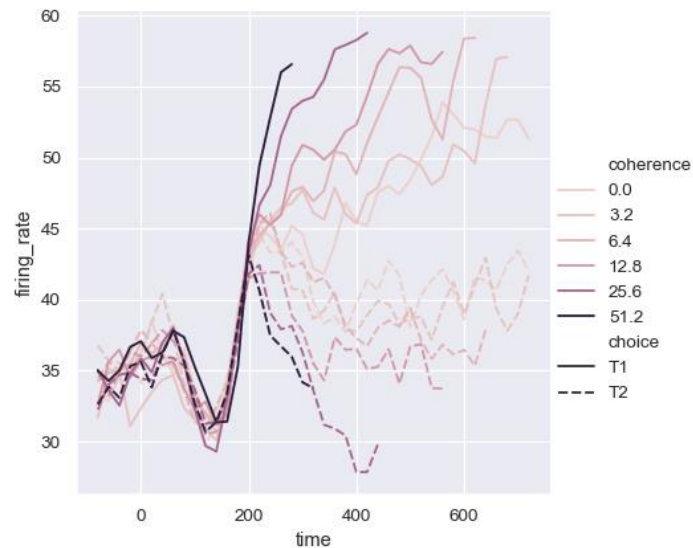
Aggregation and
representing uncertainty



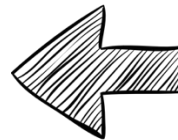
Plotting subsets of data
with semantic mappings

Learn from Official Tutorials (Example 1)

Seaborn: Visualizing statistical relationships

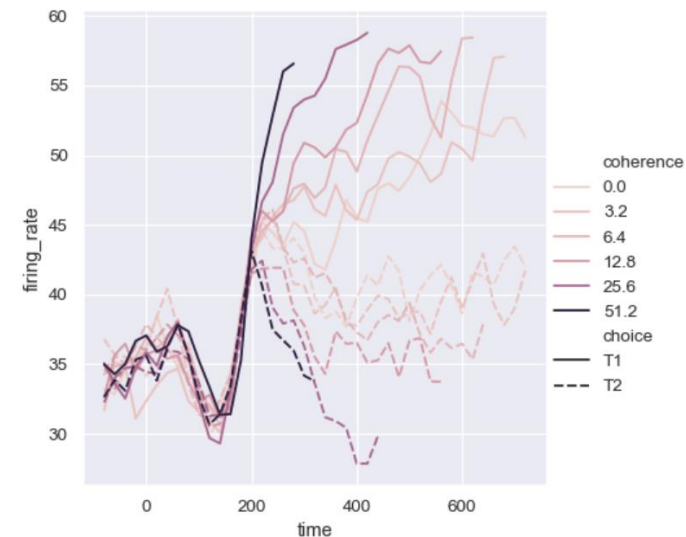


Plotting subsets of data
with semantic mappings



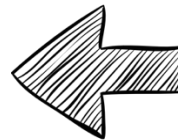
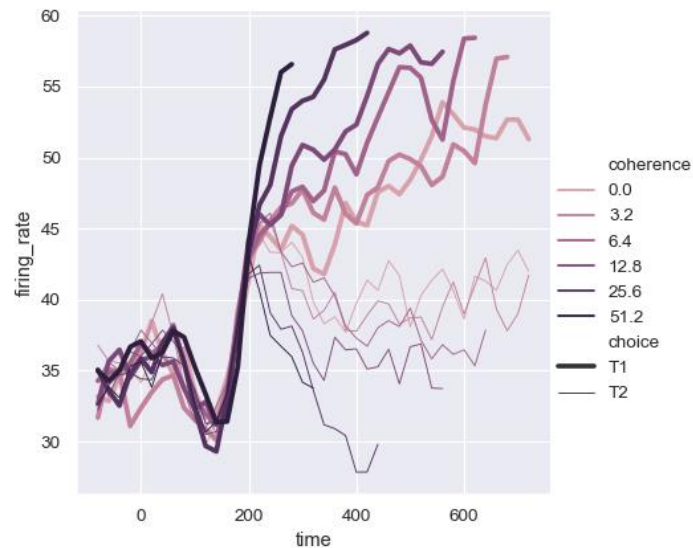
The default colormap and handling of the legend in `lineplot()` also depends on whether the hue semantic is categorical or numeric:

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



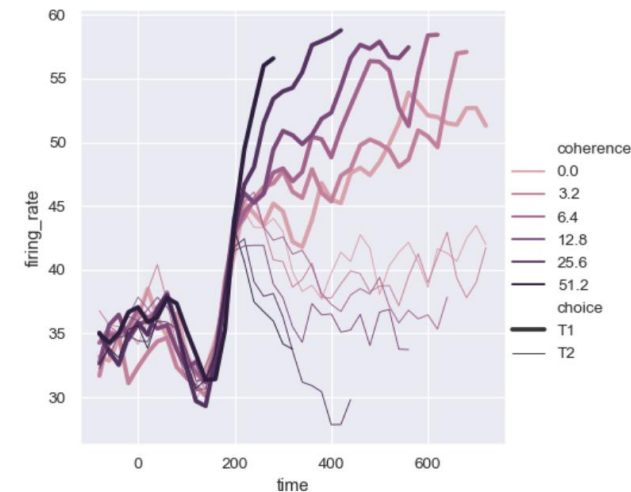
Learn from Official Tutorials (Example 1)

Seaborn: Visualizing statistical relationships



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,  
)
```



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_bd45ccel55181c382bae2fb47104400e19f5f98b50d90083e537cle7
Pending:
(no pending 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



Thanks for Listening!