Manual

Get the code and data

1. clone the repo

git clone https://github.com/lancercat/Moose.git

2. Get data from

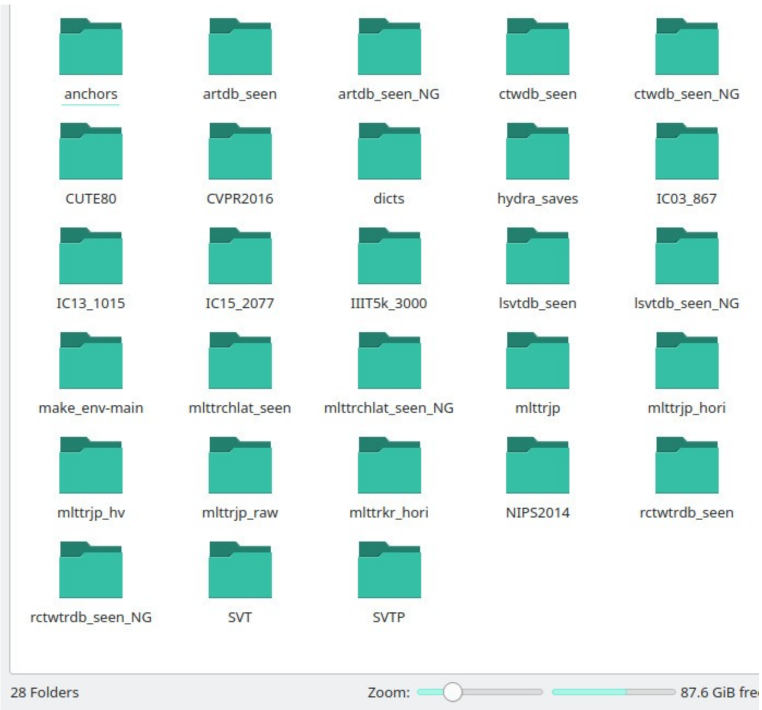
1. https://www.kaggle.com/datasets/vsdf2898kaggle/osocrtraining?rvi=1

and

2. https://www.kaggle.com/datasets/object300/mose-extra

Unzip 1 first, then override it with 2. you should get around 27 folders

If you have one or two missing or something fails to run, please email me



3. make a folder named anchors in the data dir, now you have 28

mkdir -p ~/ssddata/anchors/

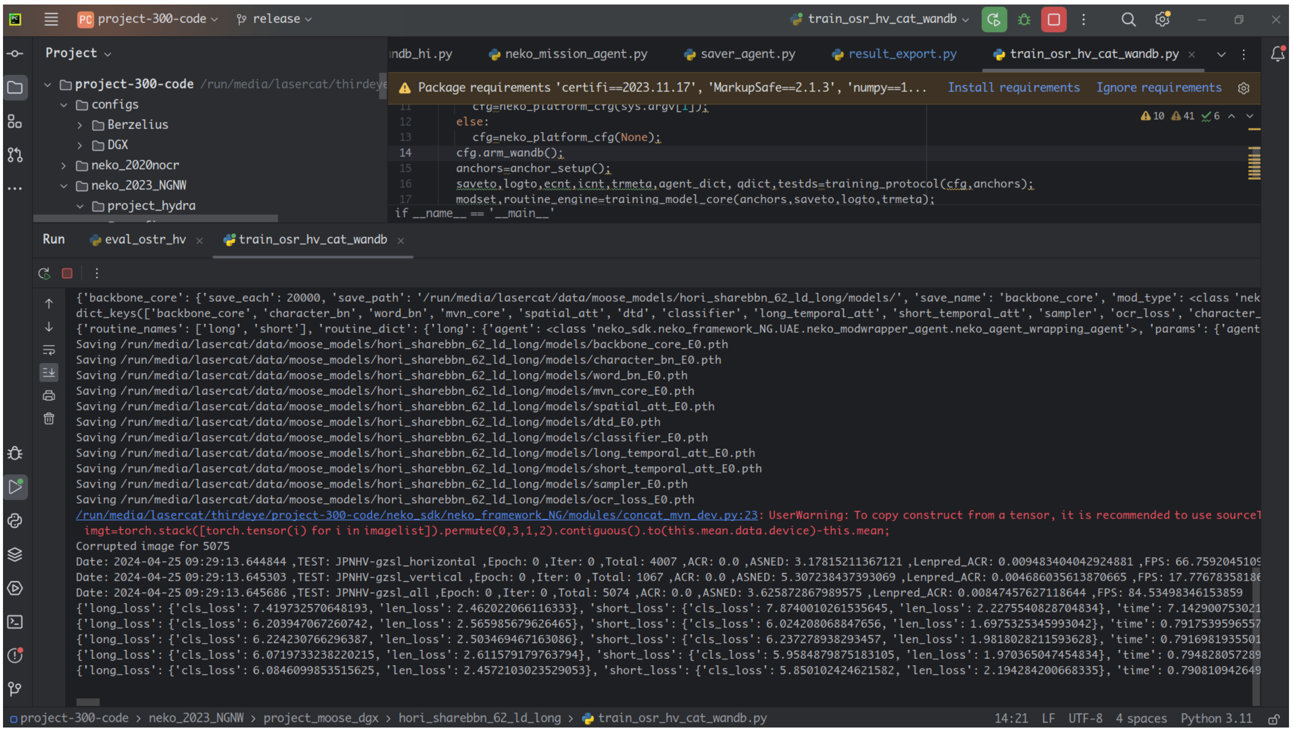
4. Put a “hydra_saves” folder in your home directory.

mkdir -p ~/hydra_saves

5. Try training a moose from pycharm. If it works, it works.

Allow it some time for the first runs. It will build index files if it fails to load them from the disk.

It will also launch a testing dry run so we don’t fail midway.



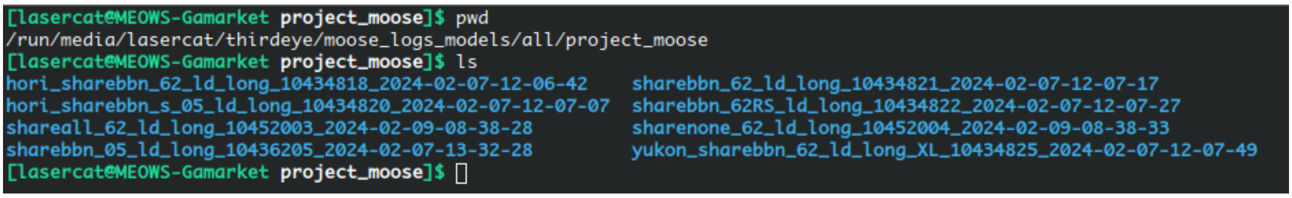
OK, that’s all. If it fails, open an issue or mail lasercat@gmx.us

Enjoy.

Reproducing Ablative Studies

1. Download the logs

https://www.kaggle.com/datasets/object300/log-moose-release



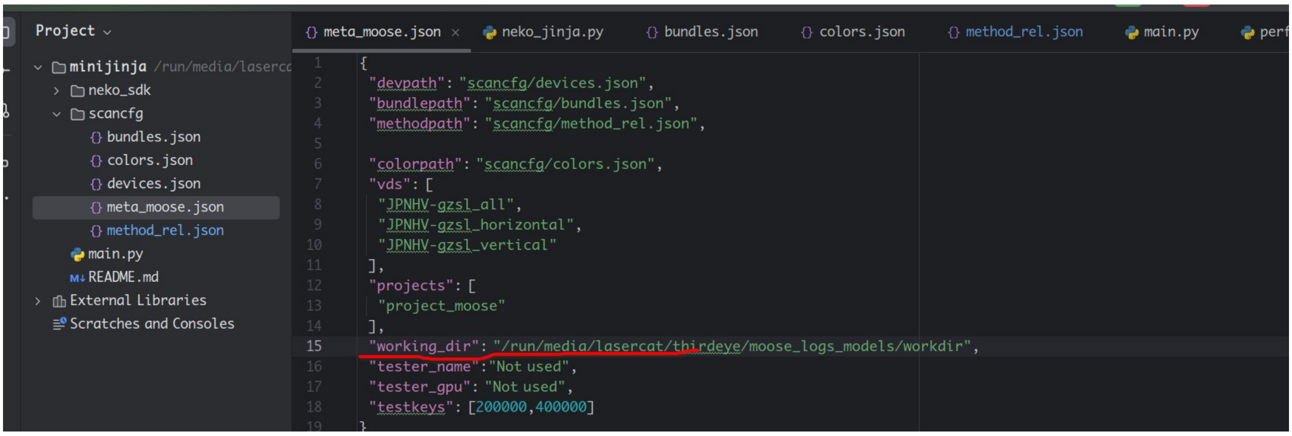
2 Download the visualizer:

git clone https://github.com/lancercat/minijinja.git

Set your working path in scancfg/metacfg.json

WARNING: I forgot if this thing will or will not drop your workingdir,

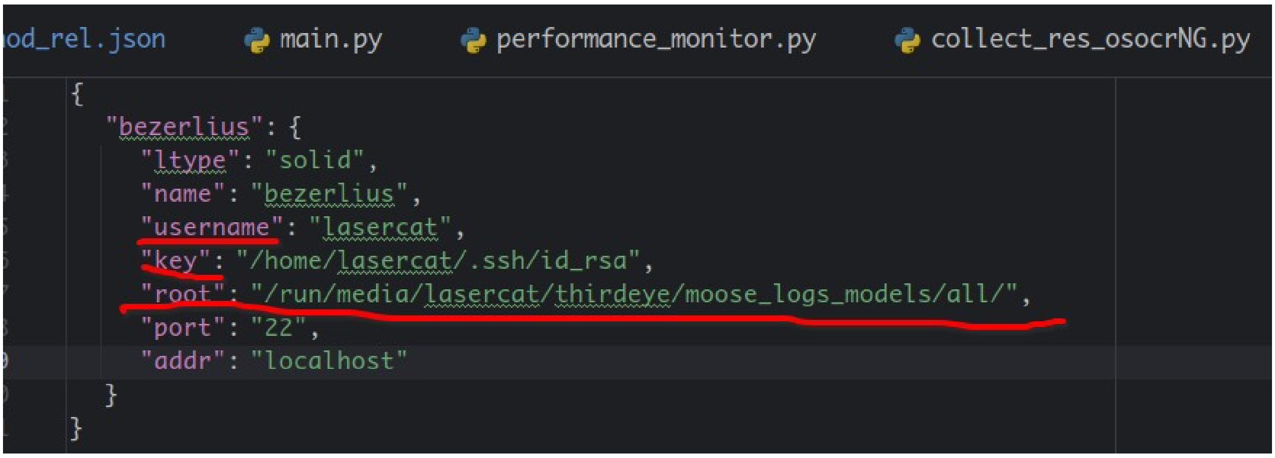
check yourself, put it in a sandbox, or make sure the workingdir is set right.



Also set your device info in devices.json

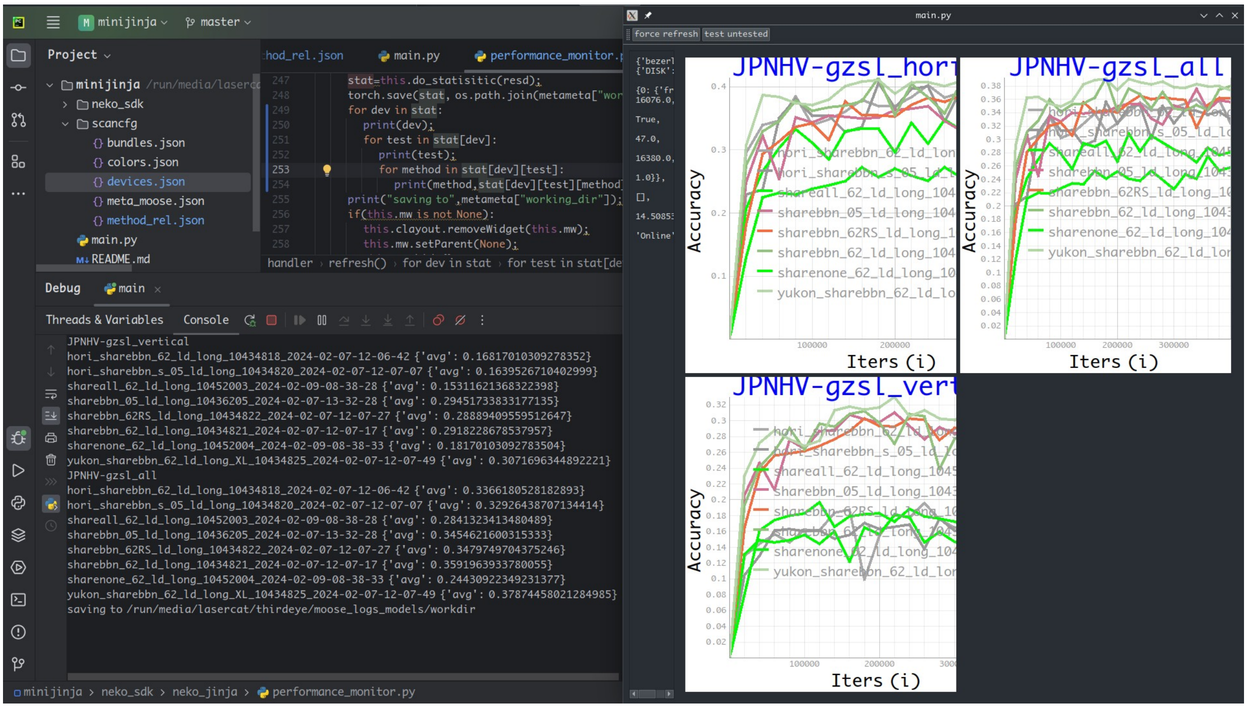
a) Specifically, set your username, sshkey (that can allow you access localhost, or wherever you store the downloaded logs)

b) Set root to where your downloaded logs are (pwd in step 1)



Trivia: This library is designed by me to periodically monitor, grab logs from, and issue commands to a fleet of servers located in different places, thus it uses ssh/rsync/paramiko.

3. Go run the script main.py You are golden if you see the curves we presented in SSBA.



Note for sharebbn_62RS_ld_long_10434822_2024-02-07-12-07-27 the horizontal/vertical stats DO NOT make sense, because the vertical images with aspect ratio >2 are rotated without the logger knowing, so the numbers come from a different split of all test images. That's why we only report overall accuracy in Table 3.

4. Now let's match the statistical data to the logs in the tables in the paper.

Table2, Row 1

Horizontal	Dropped	Horizontal	32.93	16.4	37.33

```
hori_sharebbn_s_05_ld_long_10434820_2024-02-07-12-07-07 {'avg': 0.32926438707134414}
hori_sharebbn_s_05_ld_long_10434820_2024-02-07-12-07-07 {'avg': 0.1639526710402999}
hori_sharebbn_s_05_ld_long_10434820_2024-02-07-12-07-07 {'avg': 0.3732842525580234}
```

Table 2, Row 2

Single-Horizontal	Keep as is	Horizontal, Vertical	34.55	**29.45**	35.9

```
sharebbn_05_ld_long_10436205_2024-02-07-13-32-28 {'avg': 0.3454621600315333}
sharebbn_05_ld_long_10436205_2024-02-07-13-32-28 {'avg': 0.29451733833177135}
sharebbn_05_ld_long_10436205_2024-02-07-13-32-28 {'avg': 0.3590279510856015}
```

Table 2, Row 3

Horizontal-MoE	Dropped	Short, Long	33.66	16.82	**38.15**

```
hori_sharebbn_62_ld_long_10434818_2024-02-07-12-06-42 {'avg': 0.3366180528182893}
hori_sharebbn_62_ld_long_10434818_2024-02-07-12-06-42 {'avg': 0.16817010309278352}
hori_sharebbn_62_ld_long_10434818_2024-02-07-12-06-42 {'avg': 0.381473047167457}
```

Table 2, Row 4

MOoSE	Keep as is	Short, Long, Vertical	**35.92**	29.18	37.71

```
sharebbn_62_ld_long_10434821_2024-02-07-12-07-17 {'avg': 0.3591963933780055}
sharebbn_62_ld_long_10434821_2024-02-07-12-07-17 {'avg': 0.2918228678537957}
sharebbn_62_ld_long_10434821_2024-02-07-12-07-17 {'avg': 0.3771368854504617}
```

Table 3, Row 1

Horizontal-MoE	Dropped	Short, Long	33.66

```
hori_sharebbn_62_ld_long_10434818_2024-02-07-12-06-42 {'avg': 0.3366180528182893}
```

Table 3, Row 2

Rotated	Rotate to Horizontal	Short, Long	34.8

```
sharebbn_62RS_ld_long_10434822_2024-02-07-12-07-27 {'avg': 0.3479749704375246}
```

Table 3, Row 3

Single-Horizontal	Keep as is	Horizontal, Vertical	34.55

```
sharebbn_05_ld_long_10436205_2024-02-07-13-32-28 {'avg': 0.3454621600315333}
```

Table 3, Row 4

MOoSE	Keep as is	Short, Long, Vertical	**35.92**

```
sharebbn_62_ld_long_10434821_2024-02-07-12-07-17 {'avg': 0.3591963933780055}
```

Table 4, Row 1

Share None			25.45	18.17	26.1

```
sharenone_62_ld_long_10452004_2024-02-09-08-38-33 {'avg': 0.24430922349231377}
sharenone_62_ld_long_10452004_2024-02-09-08-38-33 {'avg': 0.18170103092783504}
sharenone_62_ld_long_10452004_2024-02-09-08-38-33 {'avg': 0.2609807836286498 5}
```

Table 4, Row 2

Share All	✓	✓	28.41	15.31	31.9

```
shareall_62_ld_long_10452003_2024-02-09-08-38-28 {'avg': 0.31901984027951086}
shareall_62_ld_long_10452003_2024-02-09-08-38-28 {'avg': 0.2841323413480489}
shareall_62_ld_long_10452003_2024-02-09-08-38-28 {'avg': 0.15311621368322398}
```

Table 4, Row 3

MOoSE		✓	**35.92**	**29.18**	**37.71**

```
sharebbn_62_ld_long_10434821_2024-02-07-12-07-17 {'avg': 0.3591963933780055}
sharebbn_62_ld_long_10434821_2024-02-07-12-07-17 {'avg': 0.2918228678537957}
sharebbn_62_ld_long_10434821_2024-02-07-12-07-17 {'avg': 0.3771368854504617}
```

Done.

For those who **believe** that performances from different iterations subject to Normal Distribution, you can meddle the code to enable the output of pair-wise t-tests.

Reproducing the Benchmarks

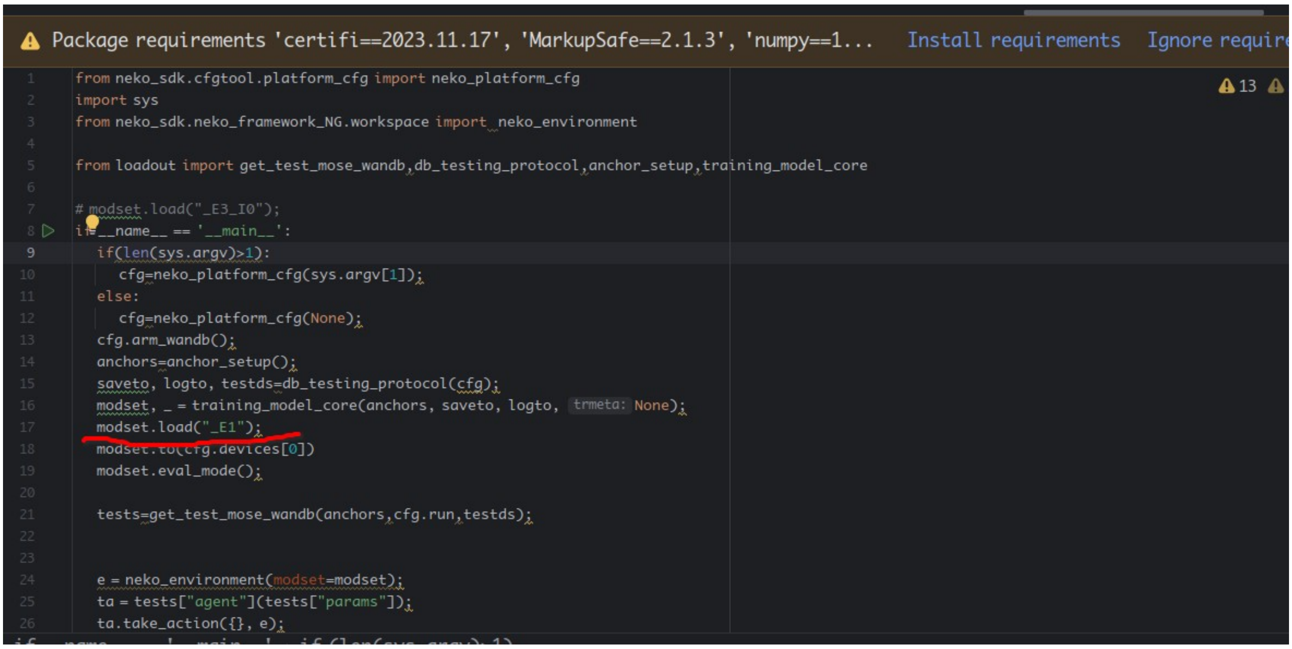
Great, now we want to check the remaining tables, and if our released models match the logs.

1. Download the models (40+Gib):

https://www.kaggle.com/datasets/object300/model-moose-release

2. The general approach:

a. Edit the test file (*/eval_*.py) to choose the iteration you want to run from



```
Package requirements 'certifi==2023.11.17', 'MarkupSafe==2.1.3', 'numpy==1...   Install requirements   Ignore require
from neko_sdk.cfgtool.platform_cfg import neko_platform_cfg
import sys
from neko_sdk.neko_framework_NG.workspace import neko_environment

from loadout import get_test_mose_wandb,db_testing_protocol,anchor_setup,training_model_core

# modset.load("_E3_I0");
if __name__ == '__main__':
    if(len(sys.argv)>1):
        cfg=neko_platform_cfg(sys.argv[1]);
    else:
        cfg=neko_platform_cfg(None);
    cfg.arm_wandb();
    anchors=anchor_setup();
    saveto, logto, testds=db_testing_protocol(cfg);
    modset, _ = training_model_core(anchors, saveto, logto, trmeta: None);
    modset.load("_E1");
    modset.to(cfg.devices[0])
    modset.eval_mode();

    tests=get_test_mose_wandb(anchors,cfg.run,testds);


    e = neko_environment(modset=modset);
    ta = tests["agent"](tests["params"]);
    ta.take_action({}, e);
```

b. Run them.

3. Now let's verify the tables:

Table 5 (Note this will take a while as it will render results and dump them to your disk)

For the regular model, run sharebbn_62_ld_long/eval_ostr_hv.py

For the large model, run yukon_sharebbn_62_ld_long_XL/eval_ostr_hv.py

GZSL	Latin, Shared Kanji		MOoSE	34.27	-	-	-

```
JPNHV-JPNHV-GZSL_all ,Epoch: 0 ,Iter: 0 ,Total: 5074 ,ACR: 0.34272763106030746 ,ASNED: 0.42176494282960925 ,Lenpred_ACR: 0.824398896334253 ,FPS: 24.212209338404254
```

	Unique Kanji, Kana		MOoSE-XL	**37.39**	-	-	-

```
JPNHV-JPNHV-GZSL_all ,Epoch: 0 ,Iter: 0 ,Total: 5074 ,ACR: 0.3738667717776902 ,ASNED: 0.37571522656115813 ,Lenpred_ACR: 0.8149389042175799 ,FPS: 28.545341704352165
```

OSR	Latin, Shared Kanji	Unique Kanji	MOoSE	69.08	71.56	94.05	81.28

```
'JPNHV-JPNHV-OSR_all', 'Epoch': 0, 'Iter': 0, 'Total': 5074.0, 'KACR': 0.6908249807247494, 'R': 0.715647339158062, 'P': 0.9405010438413361, 'F': 0.812810103743798}
```

		Kana	MooSE-XL	**75.56**	**74.05**	**95.79**	**83.53**

```
'TEST': 'JPNHV-JPNHV-OSR_all', 'Epoch': 0, 'Iter': 0, 'Total': 5074.0, 'KACR': 0.7555898226676947, 'R': 0.7405348159915277, 'P': 0.9578767123287671, 'F': 0.8352
```

GOSR	Latin, Shared Kanji	Kana	MOoSE	56.73	62.27	76.98	68.85

```
'JPNHV-JPNHV-GOSR_all', 'Epoch': 0, 'Iter': 0, 'Total': 5074.0, 'KACR': 0.5673382820784729, 'R': 0.6227171492204899, 'P': 0.7698237885462555, 'F': 0.68850036936715
```

	Unique Kanji		MooSE-XL	**59.81**	**63.83**	**81.89**	**71.74**

```
'JPNHV-JPNHV-GOSR_all', 'Epoch': 0, 'Iter': 0, 'Total': 5074.0, 'KACR': 0.5980911983032874, 'R': 0.6383073496659243, 'P': 0.8188571428571428, 'F': 0.717396
```

OSTR	Shared Kanji	Latin	MOoSE	59.96	**80.42**	84.91	82.61

```
'JPNHV-JPNHV-OSTR_all', 'Epoch': 0, 'Iter': 0, 'Total': 5074.0, 'KACR': 0.5996751928542428, 'R': 0.8042895442359249, 'P': 0.8491710473109584, 'F': 0.8261211644374508}
```

	Unique Kanji	Kana	MooSE-XL	**61.75**	80.01	**88.18**	**83.9**

```
'JPNHV-JPNHV-OSTR_all', 'Epoch': 0, 'Iter': 0, 'Total': 5074.0, 'KACR': 0.6175395858708892, 'R': 0.8000765990042129, 'P': 0.8818066694807936, 'F': 0.838955
```

And if the fonts are too small try zooming it up. That's why PDFs are always better than paperbacks

Trivia: Yukon is the largest subspecies of the moose family.

Now, match these lines with our log

```
JPNHV-JPNHV-GZSL_all ,Epoch: 0 ,Iter: 0 ,Total: 5074 ,ACR: 0.34272763106030746 ,ASNED: 0.42176494282960925 ,Lenpred_ACR: 0.824398896334253 ,FPS: 24.212209338404254
```

```
Date: 2024-02-08 11:57:51.648944 ,TEST: JPNHV-gzsl_all ,Epoch: 1 ,Iter: 0 ,Total: 5074
,ACR: 0.34272763106030746 ,ASNED: 0.4219409099448747 ,Lenpred_ACR: 0.824398896334253
,FPS: 82.59956780138607
epoch done
```

```
JPNHV-JPNHV-GZSL_all ,Epoch: 0 ,Iter: 0 ,Total: 5074 ,ACR: 0.3738667717776902 ,ASNED: 0.37571522656115813 ,Lenpred_ACR: 0.8149389042175799 ,FPS: 28.545341704352165
```

```
20994 Date: 2024-02-08 21:38:27.550164 ,TEST: JPNHV-gzsl_all ,Epoch: 1 ,Iter: 0 ,Total: 5074
      ,ACR: 0.3738667717776902 ,ASNED: 0.37571522656115813 ,Lenpred_ACR: 0.8149389042175799
      ,FPS: 81.3376144798369
20995 epoch done
```

You can now test other models to verify if they match the logs we uploaded.

Table 6:

Run yukon_sharebbn_62_ld_long_XL/eval_ostr_g1.py

			MOoSE-XL*	39.56	-	-	-

```
TEST: JPN-JPN-GZSL_all ,Epoch: 0 ,Iter: 0 ,Total: 4009 ,ACR: 0.3956098777750062 ,ASNED: 0.3425975596464096 ,Lenpred_ACR: 0.8116737340982789 ,FPS: 144
```

			MOoSE-XL*	64.80	**80.49**	89.12	**84.59**

```
'TEST': 'JPN-JPN-OSTR_all', 'Epoch': 0, 'Iter': 0, 'Total': 4009.0, 'KACR': 0.6480125523012552, 'R': 0.804959465903672, 'P': 0.8912354804646251, 'F': 0.84590328238
```

You may have noticed that single batch FPS reaches 144 now, fast, but if you want to meddle with the code you may reach faster results at multibatch.

Because it no longer renders individual results or writes to my cheap wearing-SSD.

HW: 4060Ti 16G on Thunderbolt3(TH3P4G3), 12400, 4*8GiB DDR5 @4000, Lenovo P360 Ultra.

Done. All numbers reported.

Testing Single images

Okay, now you are curious about our web pictures in Figure 1.

1. Grab the files from

https://www.kaggle.com/datasets/object300/mose-extra?select=athenaNG

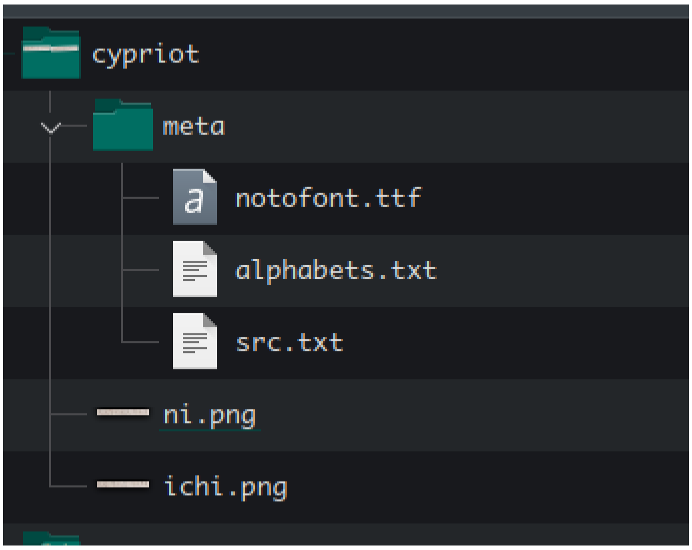
2. Set paths in yukon_sharebbn_62_ld_long_XL/eval_ostr_athena.py

```
SRC = "/home/lasercat/ssddata/athenaNG";
DST = "/home/lasercat/ssddata/athenaNG_results";
```

2. Run yukon_sharebbn_62_ld_long_XL/eval_ostr_athena.py and collect results from DST

3. To add a language:

a. You need a folder like this

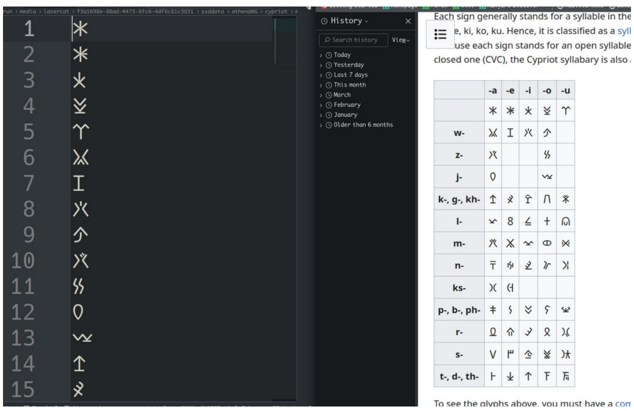


You first make a folder for the data and then you make a meta folder in it. For the meta folder

1) You download the corresponding noto font from google, and rename it to notofont.ttf

E.g. https://fonts.google.com/noto/fonts?noto.query=Cypriots

2) you scrape wikipedia to get the characterset:

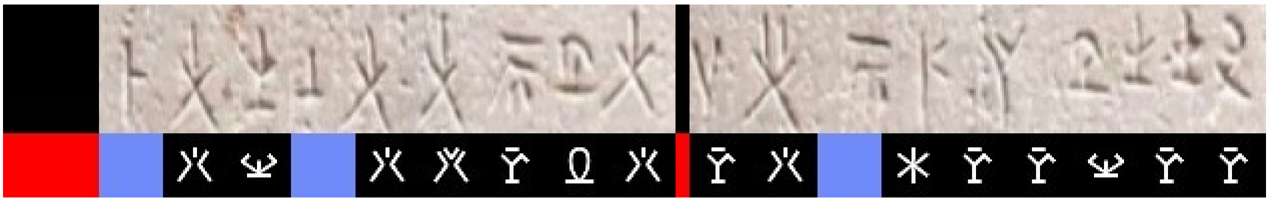


3) Then register it in your script: yukon_sharebbn_62_ld_long_XL/eval_ostr_athena.py

```
langs = ["russian", "english", "korean","cypriots"];
```

4) Finally, run the script and collect the results

(It will automatically generate the pt file if it's not there so don't worry)



Hit and miss, due to the carving style is not covered by the training set.

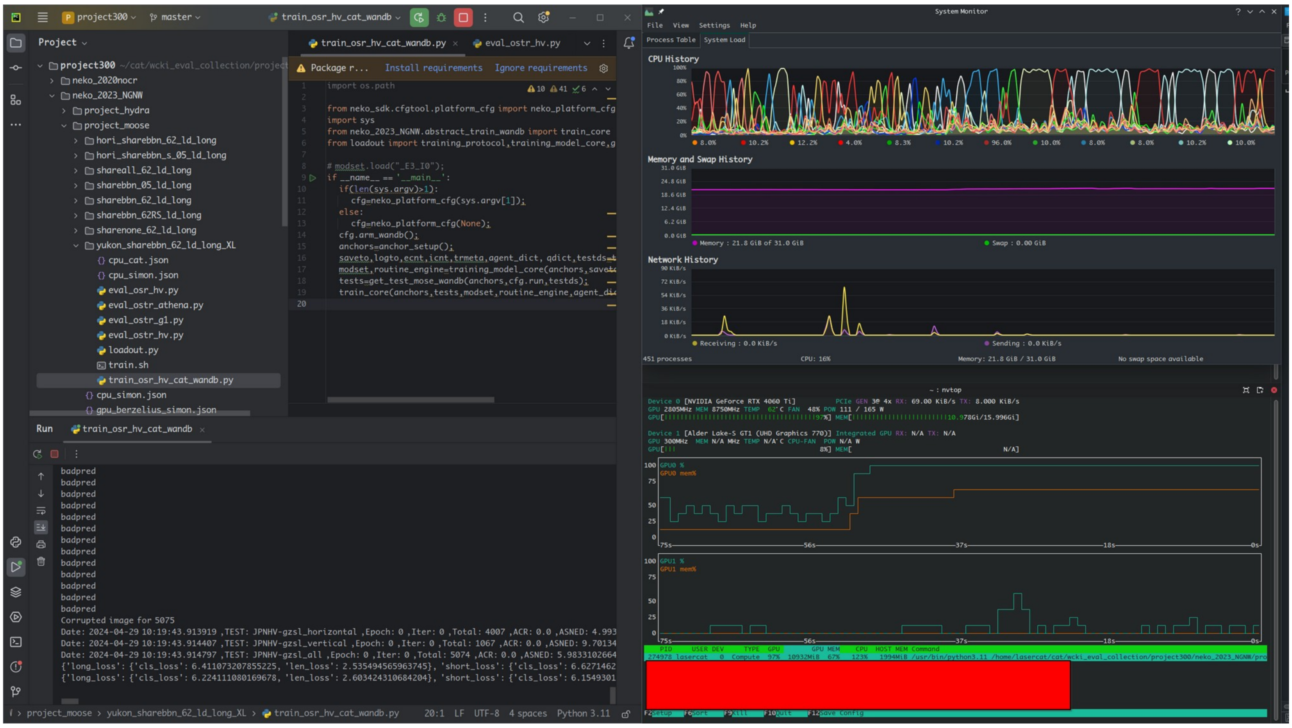
We will work on this in the future

Why we are not using this to benchmark the model? I don't read or type Cypriots, nor do have enough data to benchmark the model with. Never bench a model with 10 samples.

It's not like I pick Japanese simply out of my personal preference. It's more or less a data availability issue.

Training your own model

Get your data from kaggle, and run yukon_sharebbn_62_ld_long_XL/train_osr_hv_cat_wandb.py



If you see the “long_loss” lines, you are good.

The program takes 12 GB of FREE main memory, thus we recommend having 32GiB or more

You will roughly need 10-11 GB VRAM for training the large model.

We recommend Tesla P40s (server line) / 4060Ti (desktop line) or later models, though they are currently more expensive than they should be.

For extremely low-budget options, you may use M40-12GB, or if lucky it may run on a P102-10G GPU, which should be dirt cheap now.

WE DO NOT RECOMMEND THEM, but if you have to, you have to.

If anyone has P102s please confirm if that thing works or not.

If anyone has any luck on AMD GPUs, please let us know...

That’s all for now.

This work only sets the stage, and we will be back with something far more interesting.