Introduction **3**
Cluster: Sherlock **3**
Sherlock Setup 3
Using Sherlock 4
Sherlock Workflow 4
Sherlock Disk Usage 4
Sherlock SLURM Jobs 4
Cluster: SNAP **5**
Brando's Comments 5
Rylan's Recommend Setup 6
Brando's Recommended Setup 8
Always Put This At The Top of Your Scripts 9
Using vscode or pycharm (or an IDE) that directly connects to a server 10
Long Running Jobs on SNAP 10
The simplest way to use SNAP/Info labels (imo) 10
Secret Q&A doc from snap 11
Model Checkpoints on SNAP 12
Large Datasets on SNAP 12
Data crawling on SNAP 12
Installing Anaconda/Miniconda **12**
Conda init bricks? 14
SSHing onto SNAP and Passwordless Login **15**
IDE Use on Clusters **16**
PyCharm 16
VSCode 18
IDE Use on Clusters for Windows (VSCode) **19**
Lab Website **20**
Lab HuggingFace **21**
Lab GitHub **21**
Sang's Setup **21**
Everything Conda 21
Conda Alternatives 22
Conda for Finetune LLMs 22
LLMs everything 24
Conda for Deploying LLMs 25
Install Python 27

Install Jupyter Kernel 28
Install Torch 28
Everything Tmux 28
FAQ **29**
LLM APIs **29**

Introduction

Welcome! Hopefully, this tutorial will get you up and running on at least one of Stanford CS's many compute clusters.
Overview of Stanford Clusters

- There are several clusters you may use during your time here
- SNAP
    - Used by: Subset of labs in Stanford Computer Science
    - Managed by:
- Sherlock
    - Used by: All of Stanford
    - Managed by: Stanford
    - Properties:
- SC - previously called the Stanford AI Lab (SAIL) Cluster
    - Used:
    - Managed by:
    - Properties:
    - Onboarding https://docs.google.com/document/d/1WmdhdR_-uigbtHHclNz7sTB-vlFasTp8pQ0Y6L4eCm4/edit
- NLP Cluster
- Yamins' Cluster
    - Used by: Yamins, Goodman, Ganguli
    - Managed by: Yamins
    - Properties:

Cluster: Sherlock

- Sherlock is a Slurm cluster
- Slurm is a cluster management tool that makes running & queuing jobs easy, as well as managing equitable use and good resource sharing
- Slurm means that when you login, you're routed to a load balanced login node, which means a node in the cluster that can communicate quickly with the outside world with minimal compute
- You should not run code on the login node!
- Rather, use the head node to submit jobs to Slurm. Slurm

Sherlock Setup

- Email srcc-support@stanford.edu with your name and your SUNet ID (e.g. rschaef), cc'ing your supporting professor (e.g. Koyejo)
    - Instructions: https://www.sherlock.stanford.edu/docs/getting-started/
- Once approved, connect via SSH:
    - ssh <sunetid>@s

- [ ] For more details, see https://www.sherlock.stanford.edu/docs/getting-started/connecting/

Using Sherlock

Sherlock Workflow

- Because Sherlock is a SLURM cluster, the way to run code is to submit jobs to Slurm that specify what code you want to run and what resources each job will require (examples below). Slurm will then provision the necessary compute resources and run your jobs.

Sherlock Disk Usage

- When you login, Sherlock will show you our group's current disk usage and max disk usage

```
+-------------------------------------------------------------------------------+
| Disk usage for user rschaef (group: sanmi)                                    |
+-------------------------------------------------------------------------------+
|    Directory |  volume /   limit [          use%] | inodes /  limit (use%)|
+-------------------------------------------------------------------------------+
          HOME |    0.0B /  15.0GB [            0%] |      - /      - (  -%)
    GROUP_HOME |    0.0B /   1.0TB [            0%] |      - /      - (  -%)
       SCRATCH |   4.0KB / 100.0TB [            0%] |    1.0 /  20.0M (  0%)
 GROUP_SCRATCH |   4.0KB / 100.0TB [            0%] |    1.0 /  20.0M (  0%)
+-------------------------------------------------------------------------------+
 WARNING: files on SCRATCH and GROUP_SCRATCH are automatically purged
 90 days after their last content modification.
+-------------------------------------------------------------------------------+
```

-
- You have 15GB in your HOME directory (e.g. /home/users/rschaef)
  - To figure out what your HOME is, use: **echo $HOME**
- We collectively have 1 TB in our GROUP_HOME directory (i.e. /home/groups/sanmi)
  - To figure out what our GROUP_HOME is, use: **echo $GROUP_HOME**
- I recommend installing Anaconda under $HOME, and putting everything else (including virtual environments) in your personal directory under $GROUP_HOME because 15 GB is not enough

Sherlock SLURM Jobs

- In order to run code, you'll need to submit a job to SLURM specifying what code you want run and what resources you want provisioned for that run
- To do that, you'll need a SLURM bash script. Here is an example:

```
#!/bin/bash
#SBATCH -n 1                    # one node
```

```
#SBATCH --mem=32G                # RAM
#SBATCH --time=01:00:00          # total run time limit (D-HH:MM:SS)
#SBATCH --mail-type=FAIL

# Activate virtual environment.
source /home/groups/sanmi/rschaef/KoyejoLab-Rotation/emergence_venv/bin/activate

export PYTHONPATH=.

# write the executed command to the slurm output file for easy reproduction
#
https://stackoverflow.com/questions/5750450/how-can-i-print-each-command-before-executing
set -x

# -u is critical to ensuring results aren't buffered but are instead immediately written to stdout.
python -u notebooks/emergence/gpt3_addition_analyze.py
```

Cluster: SNAP

Brando's Comments

To gain access to the SNAP cluster, email action@cs.stanford.edu and cc Professor Koyejo. Provide your SUID ID number (e.g., 05756291) and your CSID (e.g., rschaef), *not* your SUNetID.

Before you read, I want to preface by giving the wiki page Snap/info lab give about their own cluster: https://ilwiki.stanford.edu/doku.php?id=start. That's the docs they made themselves and that I used to construct my own set up. You can also read the Introduction to SNAP slides.

The info labs/SNAP cluster is not set up in a standard way – e.g. with a HPC workload manager like slurm, condor, qsub or variants. Therefore the advice below might be strange for people with more experience. After too many hours trying to set up a nice workflow I recommend one of the following:

Read this: https://ilwiki.stanford.edu/doku.php?id=hints:storefiles. The tldr summary is DFS in snap is super slow so use LFS (means data & ckpts likely need to be duplicated across serves to gain speed or it's totally unusable). AFS is nice for sharing code across servers. DFS I don't know what it's useful for. I claim nothing even though they sent me this link: https://www.weka.io/learn/ai/what-is-network-file-system . If you find a use for DFS let everyone know please. I'm avoiding dfs like the plague.

Youtube show case of the recommended way on how to use snap: https://youtu.be/XEB79C1yfgE .

Rylan's Recommend Setup

- Step 0: Join the SNAP Slack because everyone misses this step
  - the SNAP slack channel https://join.slack.com/t/snap-group/shared_invite/zt-1lokufgys-g6NOiK3gQi84NjlK_2dUMQ
- Step 1: Connect to a SNAP server
  - You need to SSH into a server directly (check here for passwordless login)
  - For setup, the first machine you choose doesn't much matter
  - Choose a server from this list and SSH into it
    - e.g., I choose turing1
    - Then I SSH into turing1 with ssh rschaef@turing1.stanford.edu
    - You may receive the following error: Unable to negotiate with 171.64.75.72 port 22: no matching host key type found. Their offer: ssh-rsa,ssh-dss
      - Following https://askubuntu.com/a/836064, you'll need to modify the SSH command
      - ssh -oHostKeyAlgorithms=+ssh-rsa rschaef@turing1.stanford.edu
- Step 2: Create your .bashrc files
  - SNAP doesn't use a .bashrc file per user, but rather .bashrc.user
  - Create your .bashrc.user file on /afs/
    - e.g. Rylan's is at /afs/cs.stanford.edu/u/rschaef/.bashrc.user
    - SNAP has three storage systems. AFS is the one that will always be available and loaded whenever you SSH into any SNAP machine
    - Following Brando's configuration, Rylan's .bashrc.user file has 1 line:
      - source /afs/cs.stanford.edu/u/rschaef/.bashrc.lfs
  - Create a .bashrc.lfs file on /afs/
    - e.g. Rylan's is at /afs/cs.stanford.edu/u/rschaef/.bashrc.lfs
    - Following Brando's configuration, Rylan's .bashrc.lfs file has a few lines
- Step 3: Install Anaconda/Miniconda
  - We recommend installing Anaconda/Miniconda on /lfs/, not /afs/
  - For detailed instruction, see Installing Anaconda/Miniconda below
  - I installed my Miniconda to /lfs/turing4/0/rschaef/miniconda3
    - Make sure to install Miniconda on LFS!!! Not AFS. If you install on AFS, you will run out of space.
  - Miniconda will ask: Do you wish the installer to initialize Miniconda3 by running conda init? [yes|no]
    - If you type yes, Miniconda installation will modify your /lfs/ .bashrc file and say something like:
      - modified /.bashrc
    - That file is irrelevant in the eyes of SNAP and will affect nothing

- You have two choices:
  - 1) Source that /lfs/turing4/0/rschaef/.bashrc file inside your .bashrc.user file
  - 2) Move the contents of that file into your .bashrc.user file (or your .bashrc.lfs file)
- If you do neither, conda will not work.

At the end of setup, Rylan has the following files:

File: /afs/cs.stanford.edu/u/rschaef/.bashrc.user

```
source /afs/cs.stanford.edu/u/rschaef/.bashrc.lfs
```

File: /afs/cs.stanford.edu/u/rschaef/.bashrc.lfs

```
# print current working directory with each prompt
export PS1="\u@\h \w$ "

# Use current machine as home.
export LOCAL_MACHINE_PWD=$(python3 -c "import
socket;hostname=socket.gethostname().split('.')[0];print('/lfs/'+str(hostname)+'/0/rschaef');")
mkdir -p $LOCAL_MACHINE_PWD
export WANDB_DIR=$LOCAL_MACHINE_PWD
export LFS_HOME=$LOCAL_MACHINE_PWD


cd $LFS_HOME
export TEMP=$LFS_HOME
export AFS_HOME=/afs/cs.stanford.edu/u/rschaef
export DFS_HOME=/dfs/scratch0/rschaef/

# Enable running with GPUs.
# - https://ilwiki.stanford.edu/doku.php?id=hints:gpu
# export PATH=/usr/local/cuda-11.7/bin:$PATH
# export LD_LIBRARY_PATH=/usr/local/cuda-11.7/lib64:$LD_LIBRARY_PATH
# source cuda11.7

# Initialize conda
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/lfs/turing4/0/rschaef/miniconda3/bin/conda' 'shell.bash' 'hook' 2>
/dev/null)"
if [ $? -eq 0 ]; then
        eval "$__conda_setup"
else
        if [ -f "/lfs/turing4/0/rschaef/miniconda3/etc/profile.d/conda.sh" ]; then
```

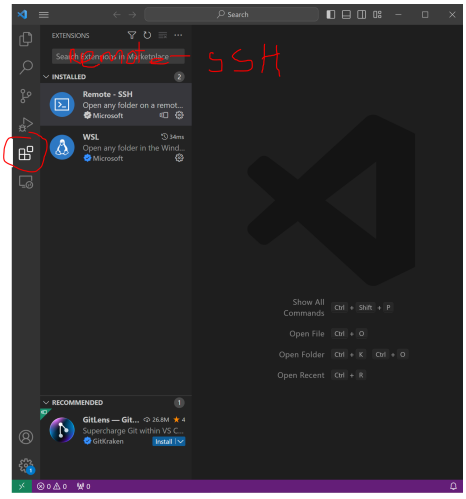
```
        . "/lfs/turing4/0/rschaef/miniconda3/etc/profile.d/conda.sh"
        else
        export PATH="/lfs/turing4/0/rschaef/miniconda3/bin:$PATH"
        fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

Because I want this to work from any machine, I override the conda init defaults to instead read:

```
# Initialize conda
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$("$LOCAL_MACHINE_PWD/miniconda3/bin/conda" 'shell.bash' 'hook' 2>
/dev/null)"
if [ $? -eq 0 ]; then
        eval "$__conda_setup"
else
        if [ -f "$LOCAL_MACHINE_PWD/miniconda3/etc/profile.d/conda.sh" ]; then
        . "$LOCAL_MACHINE_PWD/miniconda3/etc/profile.d/conda.sh"
        else
        export PATH="$LOCAL_MACHINE_PWD/miniconda3/bin:$PATH"
        fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

Brando's Recommended Setup

No matter what setup up for any cluster you use you need to solve the following problems



- Where $HOME points to? We recommend: $HOME -> lfs
- Where does your code live at? We recommend code -> afs
- Where you python env is at? We recommend: conda/python envs -> lfs (dfs super slow!)
- Where data is at? We recommend: data & logs/ckpts -> lfs (but might make sense to separate these)
- Where bashrc lives at: .bashrc.user & .bashrc.lfs -> afs
- Other local installs: other stuff e.g. other envs/local executables (e.g. opam, rugby, for my TP projs) -> lfs (due to speed in installation)
- how to code? way to sync server with local (pycharm deployment paths) -> make sure deployment works for each server you start using
- how to run experiments -> main.sh -> (snap kerberos) tmux

You always make a decision for the above -- even if it's not explicit, it happens implicitly by the workflow you use.
My decisions can be inferred/confirmed by reading the following:

- my .bashrc.lfs (for my current local file system/lfs setup) https://gist.github.com/brando90/729370f23345b1c46a063ab3ee540a3d
- my .bashrc.user sources my .bashrc.lfs or whatever setup I want to have at that moment: https://github.com/brando90/.dotfiles/blob/master/.bashrc.user
- and a past attempt with .bashrc.dfs: https://github.com/brando90/.dotfiles/blob/master/.bashrc.dfs
- For a concrete assignment style to use my setup see this: Brando Starter task Spring 2024
- And a github repo with details explanations of my setup https://github.com/brando90/snap-cluster-setup?tab=readme-ov-file#create-a-soft-link-for-your-cloned-github-project-lfs---afs

Continue for more details.

Always Put This At The Top of Your Scripts

```
# Rok asked us to include the following specifications in our code to prevent CPUs from spinning idly:
n_threads_str = "4"
os.environ["OMP_NUM_THREADS"] = n_threads_str
os.environ["OPENBLAS_NUM_THREADS"] = n_threads_str
os.environ["MKL_NUM_THREADS"] = n_threads_str
os.environ["VECLIB_MAXIMUM_THREADS"] = n_threads_str
os.environ["NUMEXPR_NUM_THREADS"] = n_threads_str
```

Using vscode or pycharm (or an IDE) that directly connects to a server

Since one can only run jobs by directly connecting to a specific server – using an IDE that directly runs, debugs and edits files remotely seem like a good option. Since I do not use this I cannot comment further but it seems a reasonable option – especially since slurm won't be in the way to run the debugger of the IDE directly with a GPU.

For long running jobs I strongly recommend to use their (very unorthodox) suggestion here: https://ilwiki.stanford.edu/doku.php?id=hints:long-jobs .

If you use Kerberos for passwordless logins, ensure GSSAPI is enabled in your SSH config; see here.

Long Running Jobs on SNAP

- SSH into your preferred machine
- Run: krbtmux
    - This will create a specialized tmux session
- Inside the tmux session, run: reauth
    - Then enter your password
    - This will prevent SNAP from kicking you off
- Navigate to wherever your code is & activate whatever environments you have in mind
- Then run your code!

The simplest way to use SNAP/Info labels (imo)

1. Read intro to get an account in Snap.
2. Connecting to SNAP: connect to the snap cluster using ssh, see: https://ilwiki.stanford.edu/doku.php?id=hints:remote-access but change it to the right name for the specific server you need. For server names see: https://ilwiki.stanford.edu/doku.php?id=snap-servers:snap-gpu-servers-stats . For stanford vpn see: https://ilwiki.stanford.edu/doku.php?id=hints:remote-access#stanford_vpn
3. Setting up bash: They named the usual .bashrc to the very unconventional .bashrc.user. For details see: https://ilwiki.stanford.edu/doku.php?id=hints:enviroment
4. Storage: For files and storage I personally recommend to choose a server or two that you like most and use that to have all your files & storage. It will be faster and it is simpler than trying to manage many paths to code, data, conda/python envs, .bashrc.files etc. i.e. use this: https://ilwiki.stanford.edu/doku.php?id=hints:storefiles#lfs_local_server_storage . You can also use dfs store (in that link) so that the code is available everywhere but dfs is slow and if you want to manage multiple paths do it at your own risk & time sink. To install conda see:

https://github.com/brando90/ultimate-utils/blob/master/sh_files_repo/download_and_install_conda.sh

5. Syncing code: personally I use pycharms ability to rsync on save when I work on servers https://www.jetbrains.com/help/pycharm/uploading-and-downloading-files.html . But you can use git for that if you want (though git is a versioning system not server local sync system). Or connect with your ide directly to the server.
6. Select a server: reminder based on 3 you should have select a server or two where most of your jobs will be done (based on the advice here).
7. Long lived jobs: due to very strange uncommon reasons, you have to run your jobs through the krbtmux and the reauth command. See details here: https://ilwiki.stanford.edu/doku.php?id=hints:long-jobs . I strongly recommend not skip this step.
8. GPUs: once you selected a server & know how to use krbtmux/krbscreen, use the GPUs as outlined here: https://ilwiki.stanford.edu/doku.php?id=hints:long-jobs . Likely by making a specific GPU available as you do experiments or debug.
9. Checking GPU availability: see https://ilwiki.stanford.edu/doku.php?id=snap-servers:snap-gpu-servers-stats and/or the SNAP slack channel https://join.slack.com/t/snap-group/shared_invite/zt-1lokufgys-g6NOiK3gQi84NjlK_2dUMQ .
10. Done! Now you can login to your fav server, sync your code with your fav method, have your files in the local sever, know how to set up GPUs, and run long-lived jobs with kbrtmux+reauth to run your experiments!

Comments.

- For help/support please do not message Brando directly. Instead, use the cluster-help or general channel in Snap where I will see it and will happily help (but everyone will benefit + others might help too).
- For other help message Rok https://profiles.stanford.edu/rok-sosic (in slack or email or whatever he prefers) or email il-action@cs.stanford.edu.

- To install conda see: https://github.com/brando90/ultimate-utils/blob/master/sh_files_repo/download_and_install_conda.sh

Secret Q&A doc from snap

https://docs.google.com/document/d/1pFBJFki9uJ69q0EcI4vcaB603lA_AXHdwaLMfqIovwE/edit#heading=h.efy1aim4jl4q read it for Q & As.

Model Checkpoints on SNAP

Large Datasets on SNAP

- `skampere1`
  - ImageNet: `/lfs/skampere1/0/rschaef/data/Imagenet2012`
  - LM Evaluation Harness: `/lfs/skampere1/0/rschaef/data/huggingface`
  - SBU Captions: `/lfs/skampere1/0/rschaef/data/sbucaptions`
  -
- `ampere1`
  - Anthropic HHH: `/lfs/ampere1/0/rschaef/data/huggingface/Anthropic___json`
  - FineWeb: `/lfs/ampere8/0/dhruvpai/KoyejoLab-EBLM/data/HuggingFaceFW/fineweb`
  - asdf
- `ampere8`
  - asdf
- `hyperturing2`
  - FineWeb: `/lfs/hyperturing2/0/rschaef/KoyejoLab-EBLM/data/HuggingFaceFW/fineweb`

Data crawling on SNAP

Currently, there are four machines on SNAP available for crawling: silk04-07 (https://ilwiki.stanford.edu/doku.php?id=other-servers:crawling-servers)

These machines have higher bandwidth than usual GPU machines and are suitable for data crawling. There is a strange thing: no /lfs on these machines. Instead, we have "/dfs/scratch0/" which operates somewhat like /lfs.

Installing Anaconda/Miniconda

- On any of the above clusters, you may want to install anaconda
- You can run Brando's script https://github.com/brando90/ultimate-utils/blob/master/sh_files_repo/download_and_install_conda.sh
- Or you can manually step through the following sequence of commands
  - Get Miniconda:

    - wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
- Make sure you can execute the script:
    - chmod +x Miniconda3-latest-Linux-x86_64.sh
- Execute the script:
    - ./Miniconda3-latest-Linux-x86_64.sh
- Read through the user agreement and type “yes”
- You will be prompted where you want miniconda to be located. Feel free to use the default location or choose your own preferred location
    - **IF YOU ARE ON SNAP, PUT MINICONDA on “/lfs/” NOT “/afs/”**
    - This is because afs has some size limit (like 1 GB or maybe 5 GB) and if you put your conda installation there, you will not have sufficient space to install packages
    - Consequently, if you want conda on multiple machines, you’ll need to install it on each machine
    - I put mine at /lfs/<machine name e.g. turing1>/0/rschaef/miniconda3

```
Miniconda3 will now be installed into this location:
/home/users/rschaef/miniconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/users/rschaef/miniconda3] >>>
```

- Go make tea for ~3 minutes
- When prompted whether you want the installer to initialize Miniconda, type “n”

```
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
[no] >>> yes
no change     /home/users/rschaef/miniconda3/condabin/conda
no change     /home/users/rschaef/miniconda3/bin/conda
no change     /home/users/rschaef/miniconda3/bin/conda-env
no change     /home/users/rschaef/miniconda3/bin/activate
no change     /home/users/rschaef/miniconda3/bin/deactivate
no change     /home/users/rschaef/miniconda3/etc/profile.d/conda.sh
no change     /home/users/rschaef/miniconda3/etc/fish/conf.d/conda.fish
no change     /home/users/rschaef/miniconda3/shell/condabin/Conda.psm1
no change     /home/users/rschaef/miniconda3/shell/condabin/conda-hook.ps1
no change     /home/users/rschaef/miniconda3/lib/python3.10/site-packages/xontrib/conda.xsh
no change     /home/users/rschaef/miniconda3/etc/profile.d/conda.csh
modified      /home/users/rschaef/.bashrc
```

- Log out and log back in (or run: source .bashrc)docke
- Make sure conda is up to date:

    - conda install conda-build
    - conda update -n base -c defaults conda
    - conda update conda
    - conda update --all
- Feel free to create a conda environment:
    - conda create cn rylan_env python=3.9
    - conda activate rylan_env
- Feel free to check pip and ensure pip is up to date
    - # - make sure pip is up to date
    - which python
    - pip install --upgrade pip
    - which pip
    - which pip3

Conda init bricks?

You followed all of the steps above. Conda installs just fine. But then you type conda init, and you get some error message like this:

```
[sudo] password for schundi:
# >>>>>>>>>>>>>>>>>>>> ERROR REPORT <<<<<<<<<<<<<<<<<<<<

    Traceback (most recent call last):
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/site-packages/conda/exceptions.py", line 1132, in __call__
        return func(*args, **kwargs)
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/site-packages/conda/cli/main.py", line 69, in main_subshell
        exit_code = do_call(args, p)
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/site-packages/conda/cli/conda_argparse.py", line 122, in do_call
        return getattr(module, func_name)(args, parser)
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/site-packages/conda/cli/main_init.py", line 33, in execute
        return initialize(context.conda_prefix, selected_shells, for_user, args.system,
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/site-packages/conda/core/initialize.py", line 113, in initialize
        run_plan_elevated(plan2)
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/site-packages/conda/core/initialize.py", line 709, in run_plan_elevated
        result = subprocess_call(
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/site-packages/conda/gateways/subprocess.py", line 98, in subprocess_call
        stdout, stderr = process.communicate(input=stdin)
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/subprocess.py", line 1154, in communicate
        stdout, stderr = self._communicate(input, endtime, timeout)
      File "/afs/cs.stanford.edu/u/schundi/miniconda/lib/python3.10/subprocess.py", line 1985, in _communicate
        input_view = memoryview(self._input)
    TypeError: memoryview: a bytes-like object is required, not 'str'

`$ /afs/cs.stanford.edu/u/schundi/miniconda/bin/conda init`
```

My guess is conda is trying to write to the .bashrc file, which requires sudo permissions, instead of .bashrc.user

One thing you can try is writing the conda init statement yourself in .bashrc.user. It will be this:

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/afs/cs.stanford.edu/u/schundi/miniconda/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/afs/cs.stanford.edu/u/schundi/miniconda/etc/profile.d/conda.sh" ]; then
        . "/afs/cs.stanford.edu/u/schundi/miniconda/etc/profile.d/conda.sh"
    else
        export PATH="/afs/cs.stanford.edu/u/schundi/miniconda/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

Make sure to change the paths so they reflect the path to conda on your machine! After modifying the file use source ~/.bashrc.user or restart your connection. Hopefully you should see a (base) next to your name & path, indicating conda is working.
Copyable text:

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/afs/cs.stanford.edu/u/YOUR USERNAME HERE/miniconda/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/afs/cs.stanford.edu/u/YOUR USERNAME HERE/miniconda/etc/profile.d/conda.sh" ]; then
        . "/afs/cs.stanford.edu/u/YOUR USERNAME HERE/miniconda/etc/profile.d/conda.sh"
    else
        export PATH="/afs/cs.stanford.edu/u/YOUR USERNAME HERE/miniconda/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

When using SNAP, conda init might raise an error and ask you for a report. Just skip the conda init step and proceed.

Or just skip the conda init block.

SSHing onto SNAP and Passwordless Login

WARNING: This has been known to break and cause a “Heisenbug”, in Brano’s words. Use at your own risk.

SNAP uses Kerberos to authenticate new users. To log in without typing your password each time, you need to obtain a Kerberos “ticket” on your laptop and let SSH use it. What this means:

- **macOS**: open `Ticket Viewer.app` → “+” → log in as `<CSID>@CS.STANFORD.EDU` (note the UPPERCASE).
- **Linux**: `kinit <CSID>@CS.STANFORD.EDU` (then optionally `aklog` for AFS tokens).

Add this to `~/.ssh/config` to enable Kerberos-based SSH:

```
Host *.stanford.edu
  GSSAPIAuthentication yes
  GSSAPIDelegateCredentials yes
  PreferredAuthentications gssapi-with-mic,publickey,password

# example host
Host skampere1
  HostName skampere1.stanford.edu
  User <your_csid>
```

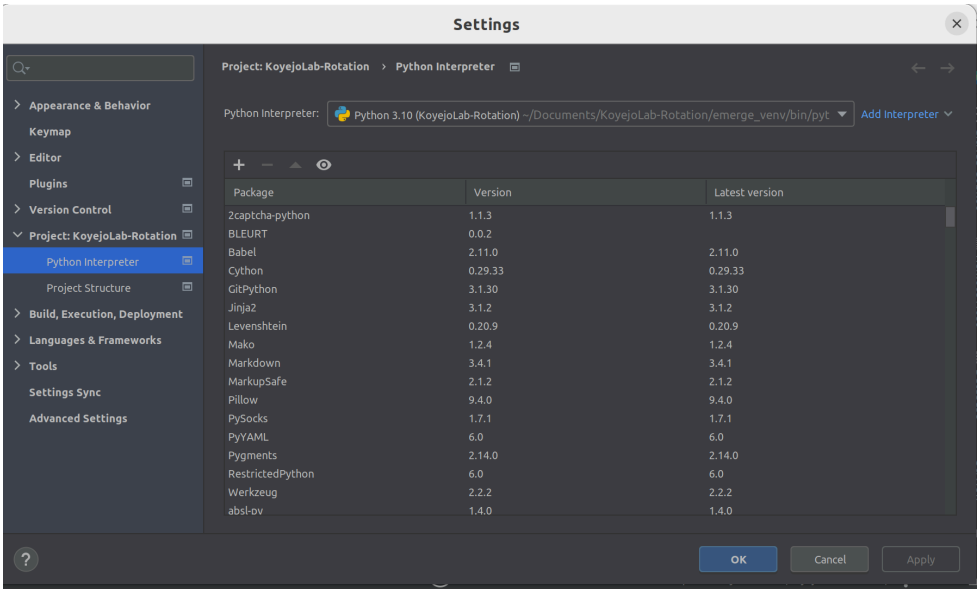
OR

```
Host skampere1 skampere2
   Hostname %h.stanford.edu
   User <csid>
   GSSAPIAuthentication yes
   GSSAPIDelegateCredentials yes
```

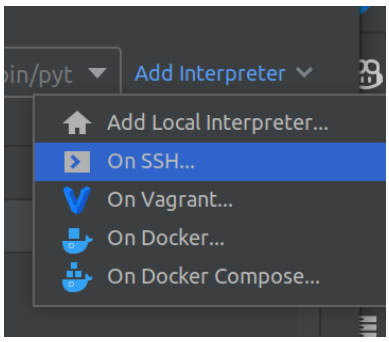
IDE Use on Clusters

PyCharm

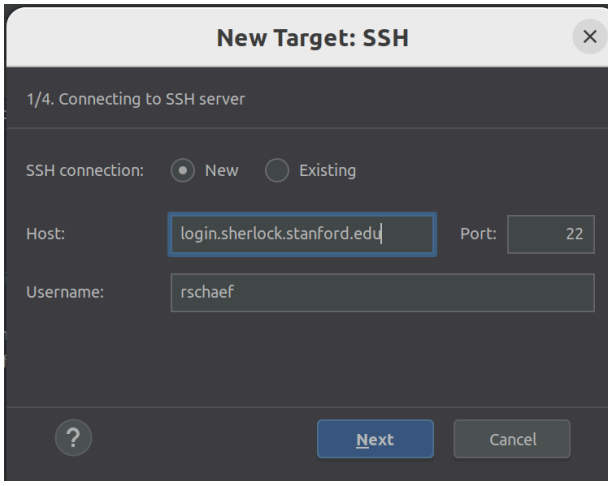
- PyCharm uses the term "**Deployment**" to describe automatically synchronizing code between machines
- The below instructions use Sherlock for demonstrative purposes, but the instructions should be the same for other clusters, e.g., SNAP
- How to create a deployment (Option 1):
  - On your cluster, create a virtual environment e.g. via python3 -m venv <name_of_my_virtual_env>
  - Open settings:



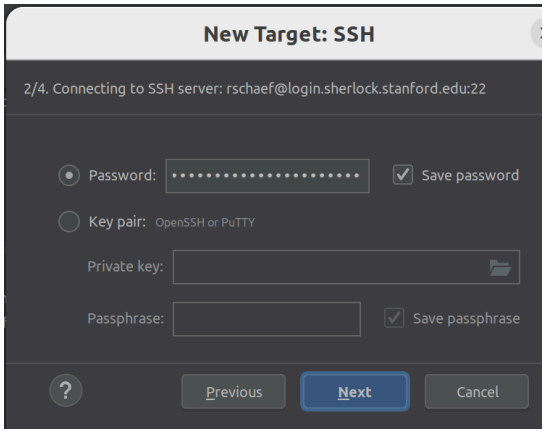
  - Select "Add Interpreter", then "On SSH"



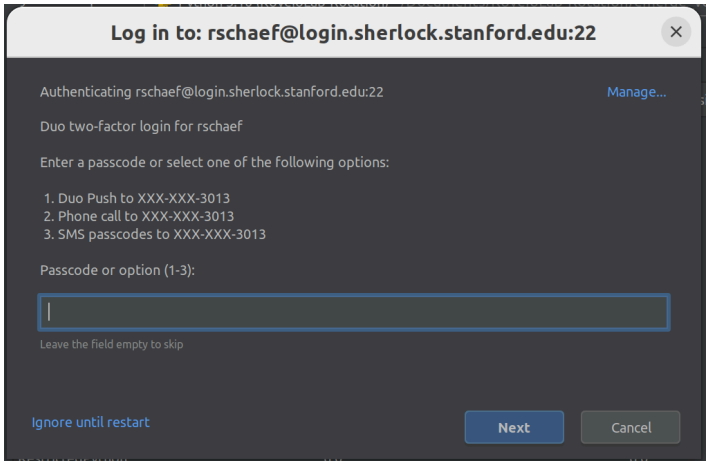
- Specify the SSH connection. If this is a new cluster, you'll want to select "New"



- Enter your password:



- 2-Factor Authenticate if necessary



- Note: After successfully connecting, PyCharm will immediately try to rsync to check that files can be transferred successfully. However, some clusters (e.g. Sherlock) will require 2 Factor Authentication again, and PyCharm will just hang

- If this happens, follow the instructions here: https://www.sherlock.stanford.edu/docs/advanced-topics/connection/#avoiding-multiple-duo-prompts
- Specifically, on your local machine, create a file ~/.ssh/config, and enable multiplexing:

```
Host login.sherlock.stanford.edu
  ControlMaster auto
  ControlPath ~/.ssh/%l%r@%h:%p
```

- This means that once you authenticate, so long as you keep that initial connection open, you won't need to reauthenticate

- Sometimes clusters will requi

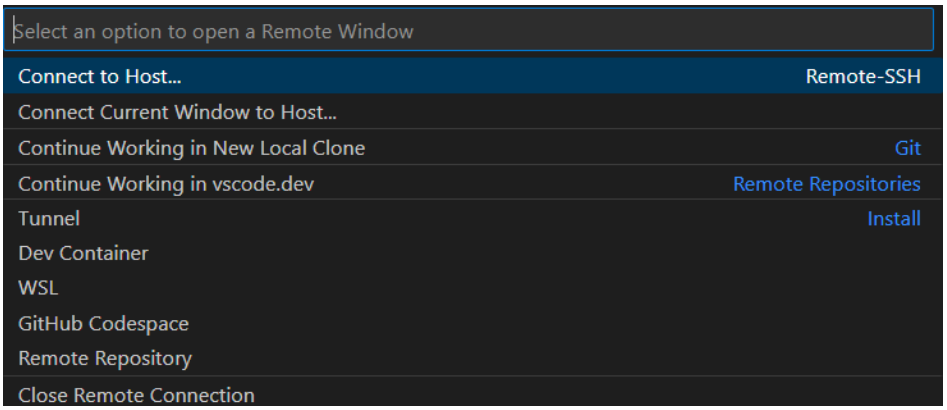
VSCode

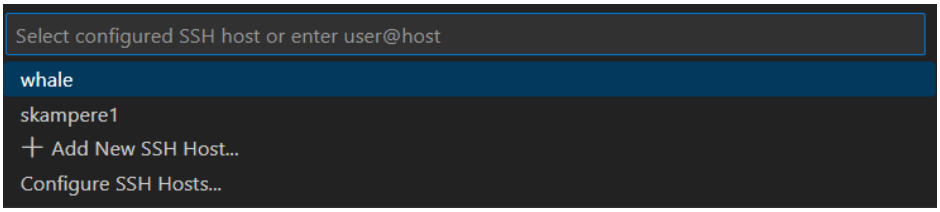
1. Configure SSH Config File:
   a. touch ~/.ssh/config (if nonexistent)
   b. nano ~/.ssh/config
   c. Example Below (replace 'shinyw' with your user and the corresponding machine skampere1):

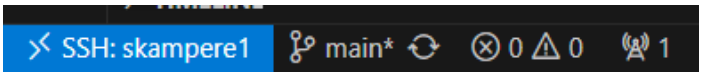
```
Host whale
    HostName whale.stanford.edu
    User shinyw

Host skampere1
    HostName skampere1.stanford.edu
    User shinyw
    ProxyJump whale
```

2. Launch VS Code:
   a. Click bottom left corner (b) to connect to host, which opens window (c):
   b. 

   c. 

d. 

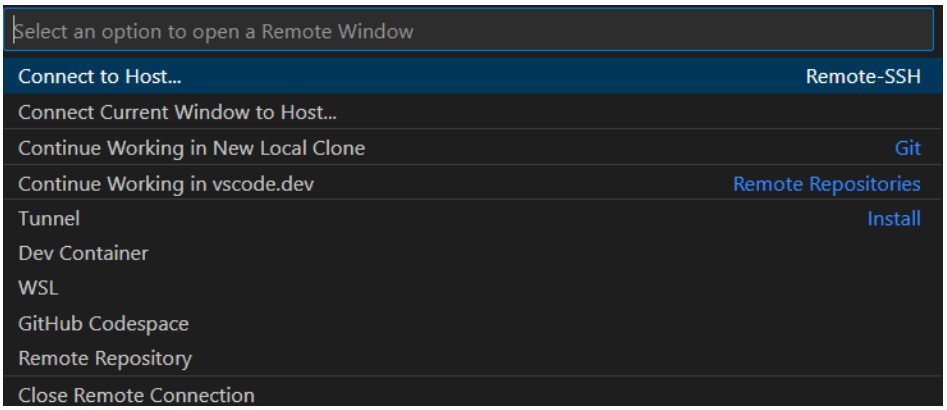
e. Remote-SSH: Connect to Host and select skampere1', or whichever machine you are on. 'skampere1' is shown in above screenshot (d) for example:
   i. Enter password/phrase as needed, follow prompted steps.
   ii. VS Code will establish the SSH connection through whale.stanford.edu to skampere1.stanford.edu.
   iii. 

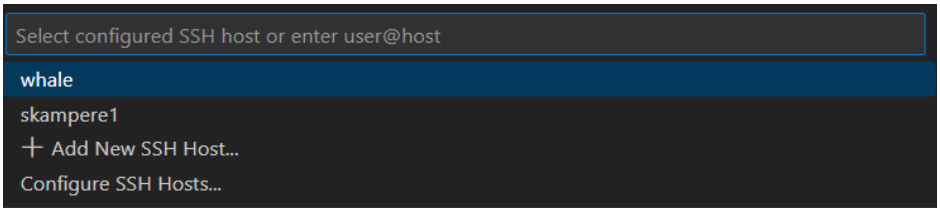
3. Other:
   a. [Snap Servers] Shiny's Access Steps
   b. Ask alylee15@stanford.edu
   c. See the Cluster Q&A (this does not work on Koyejo/Mercury servers)

IDE Use on Clusters for Windows (VSCode)

Since the above tutorial only applies linux based systems the one described here is the same process for windows based systems.

1. Launch VS Code:
   a. Click bottom left corner (b) to connect to host, which opens window (c), click on *Configure New Hosts...* (d), which opens window (e) and then click on the first file path that is seen:
   b. 

   c. 

d. 

e. 

   i. Select the first option

f. Simply paste the text in the box below in the file that opens(replace 'shinyw' with your user and the corresponding machine skampere1):

```
Host whale
    HostName whale.stanford.edu
    User shinyw

Host skampere1
    HostName skampere1.stanford.edu
    User shinyw
    ProxyJump whale
```

g. Then follow the same steps as (b), (c) , and click on *skampere1*, it will ask for your CSID password twice, once for the whale ssh and once for jumping to the *skampere1* cluster.

Lab Website

The lab website is at https://stair.cs.stanford.edu/, and the GitHub is https://github.com/stair-lab/stair. If you don't have access, ping Sang sttruong@stanford.edu . Sang built the website in 2022 but has no longer had the bandwidth to maintain it. *Each quarter, we should assign one person responsible for keeping the website up-to-date for that quarter (e.g. updating the list of rotators, new papers, new awards, etc). The idea is that this responsibility will be distributed.* The website folder is under /afs. To publish changes to the live website, you need permission. Email action@cs.stanford.edu and request to be added to the koyejolab ldap group.

Essentially, since you can't build the website on SNAP, I typically build it on my local machine (Mac) and sync it to SNAP via GitHub.

To build locally:
bundle exec jekyll build

Commit your change to GitHub

Connect to server: /afs/cs/group/koyejolab/stair

Do git pull

```
cp -r /afs/cs/group/koyejolab/stair/_site /afs/cs/group/koyejolab/www
```

Lab HuggingFace

https://huggingface.co/stair-lab

Lab GitHub

Sang's Setup

Under the resource constraint on SNAP, I recommend using the DFS file server (instead of AFS or LFS) since it would make *machine hopping* painless. Unless you do something heavily I/O related, it doesn't make much difference in performance. The setup instructions above are mainly for AFS or LFS, and weird errors happen when you try to apply that for DFS, hence the existence of this note.

Everything Conda

Step-by-step Instructions:
source /dfs/user/sttruong/miniconda3/bin/activate
conda activate **eval_llm**

```
export
LD_LIBRARY_PATH=/dfs/user/sttruong/miniconda3/envs/eval_llm/lib/python3.10/site-packages/torch/lib:/dfs/user/sttruong/miniconda3/envs/eval_llm/lib:$LD_LIBRARY_PATH
export CUDA_VISIBLE_DEVICES=0
# Test if we can send a tensor to cuda if you have recently hopped across machines.
import torch
print(torch.cuda.is_available())
a=torch.rand(5)
a.cuda()

# This is my attempt to install a unified environment for all SNAP machines, from turing to ampere
# After reinstalling CUDA, it will work for a few days, and fail for unknown reason, sadly.
# Please ignore if you are short in time.
conda install nvidia/label/cuda-11.8.0::cuda-toolkit -y
conda install -c pytorch -c nvidia pytorch==2.1.0 pytorch-cuda=11.8 -y
# To ensure cuda installation is successful, run
nvcc --version
```

To clone a conda environment:
conda create --name <new_env> --clone train_llm

Conda Alternatives

```
python -m pip install virtualenv
python -m virtualenv -p /dfs/user/sttruong/env/python3.10/bin/python3.10 bosd
deactivate
```

Conda for Finetune LLMs

Motivation: If you try to finetune LLMs with the vanilla Hugging Face pipeline, you will likely run into OOM errors unless you tune a tiny model on a relatively large machine. Using Llama Factory will save you from that. The environment set up for this can be irritating, hence the existence of this note. Ping `sttruong@stanford.edu` if you have feedback.

Step-by-step Instruction:
📌 Step 1: Clone Llama Factory
```
git clone https://github.com/hiyouga/LLaMA-Factory
cd LLaMA-Factory
```

📌 Step 2: Create an environment (assuming access to conda)
```
source /dfs/user/sttruong/miniconda3/bin/activate
conda create -n train_llm python=3.10 -y
conda activate train_llm
```

📌 Step 3: Install two packages: deepspeed and unsloth
```
pip install deepspeed==0.12.6 flash-attn==2.4.1
pip install -r requirements.txt
```

pip install "unsloth[cu118_ampere] @ git+https://github.com/unslothai/unsloth.git"

📌 Step 4: Export the required paths
export LIBRARY_PATH=/dfs/user/sttruong/miniconda3/envs/test_env/lib/python3.10/site-packages/torch/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=/dfs/user/sttruong/miniconda3/envs/test_env/lib/python3.10/site-packages/torch/lib:$LD_LIBRARY_PATH
export HF_HOME="/dfs/local/0/sttruong/env/.huggingface"
export TRANSFORMERS_CACHE="/lfs/local/0/sttruong/env/.huggingface"
export HF_DATASETS_CACHE="/lfs/local/0/sttruong/env/.huggingface/datasets"

📌 Step 5: Config GPUs and nodes for training/finetuning
accelerate config
- This machine
- multi-GPU
- node: [number_of_nodes]
- Machine rank: [index_of_each_machine_start_from_0]
- IP address of the machine that will host the main process: [IP...]
- Port: 5000
- check errors: NO
- torch dynamo: NO
- DeepSpeed: yes
- DeepSpeed file: NO
- ZeRO: 2 # In case not enough memory, please use ZeRO 3
- Offload optimizer: cpu
- Offload parameters: cpu
- Gradient accumulation: [based_on_command]
- Gradient clipping: [based_on_command]
- Clipping value: [based_on_command]
- Save 16-bit: yes
- deepspeed.zero.Init: yes # If training llama/mistral, please set to NO
- Number of GPUs: [Number_GPUs] - Number of GPUs should be equal for all nodes
- Dtype: BF16

📌 Step 6: Run finetuning/training code. Below is an example of pretraining:
accelerate launch src/train_bash.py \
    --stage pt \
    --do_train True \
    --model_name_or_path <path_to_model> \
    --use_fast_tokenizer True \
    --finetuning_type freeze \
    --template <template> \
    --flash_attn True \
    --dataset_dir data \
    --dataset <dataset_name>\
    --preprocessing_num_workers 32 \
    --cutoff_len <model_max_length>\
    --num_train_epochs <num_epochs> \
    --bf16 True \

```
--tf32 False \
--per_device_train_batch_size 2 \
--gradient_accumulation_steps 64 \
--learning_rate 1e-4 \
--lr_scheduler_type cosine \
--max_grad_norm 1.0 \
--weight_decay 0.001 \
--logging_steps 1 \
--warmup_ratio 0.03 \
--save_steps 2 \
--neftune_noise_alpha 0 \
--num_layer_trainable 32 \
--name_module_trainable self_attn.q_proj,self_attn.k_proj,self_attn.v_proj,self_attn.o_proj \
--output_dir <output_dir>\
--save_total_limit 3 \
--plot_loss True \
--report_to neptune
```

LLMs everything

State of software packages for LLMs

Framework	Training/FT	Deployment	Multi-GPU	Multimodal	Universal LLM support
LLaMa Factory	Yes	-	Yes	-	Yes
Text Generation Inference	-	Yes	Yes	-	Yes
Ollama	-	Yes	Maybe/Not tested	Yes	Limited available - Can add LLM manually
LLaVa	Yes	Yes	Yes	Yes	Limited available - Can add LLM manually
llama.cpp	-	Yes	Maybe/Not tested	Yes	Limited available - Can add LLM manually
LLM Foundry	Yes	-		-	Limited available for MPT and

					DBRX
llm.c	Yes	-	Maybe/Not tested	-	Limited available for GPT2

Conda for Deploying LLMs

Motivations: You want to deploy large LLMs (e.g. Llama2 70B) across different GPUs for evaluation.

Step-by-step Instruction: Steps 1-4 are only required the first time.

📌 Step 1: Starting the conda environment:

```
source /dfs/user/sttruong/miniconda3/bin/activate
conda create -n eval_llm python=3.10 -y
conda activate eval_llm
```

📌 Step 2: Install Rust, Cargo, and OpenSSL. Rust is a highly efficient, memory-safe language, making it ideal for the heavy computations.

```
export CARGO_HOME="/dfs/user/sttruong/.cargo"
export RUSTUP_HOME="/dfs/user/sttruong/.rustup"
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
# Select modify PATH variable: NO
wget openssl-3.0.13.tar.gz && tar xvf openssl-3.0.13.tar.gz
./config --prefix=/dfs/user/sttruong/miniconda3/envs/openssl
--openssldir=/dfs/user/sttruong/miniconda3/envs/openssl
cd openssl-3.0.13 && make install -j32
source "/dfs/user/sttruong/.cargo/env"
export PATH=/dfs/user/sttruong/.cargo/bin:/dfs/user/sttruong/miniconda3/envs/openssl/bin:$PATH
export
LD_LIBRARY_PATH=/dfs/user/sttruong/miniconda3/envs/eval_llm/lib:/dfs/user/sttruong/miniconda3/
envs/openssl/lib64:$LD_LIBRARY_PATH
export OPENSSL_DIR=/dfs/user/sttruong/miniconda3/envs/openssl
```

📌 Step 3: Install protoc, text-generation-inference

```
conda install -c "nvidia/label/cuda-11.8.0" cuda-toolkit -y
conda install -c pytorch -c nvidia pytorch==2.1.2 pytorch-cuda=11.8 -y
PROTOC_ZIP=protoc-21.12-linux-x86_64.zip
curl -OL https://github.com/protocolbuffers/protobuf/releases/download/v21.12/$PROTOC_ZIP
unzip -o $PROTOC_ZIP -d /dfs/user/sttruong/miniconda3/envs/eval_llm/ bin/protoc
unzip -o $PROTOC_ZIP -d /dfs/user/sttruong/miniconda3/envs/eval_llm/ 'include/*'
rm -f $PROTOC_ZIP
git clone https://github.com/huggingface/text-generation-inference
cd text-generation-inference
git checkout 0d72af5ab01a5b1dabd5beda953403d63b1886e0
cargo clean
BUILD_EXTENSIONS=True make install
```

In case the compilation (d.) has errors related to C++ version (usually turing/hyperturing), install C++11:
`conda install -c conda-forge gxx=11.4.0 libstdcxx-ng=11.4.0`
then rerun (d.)

📌 Step 4: Compile and install some necessary libraries for optimal deployment
```
export MAX_JOBS=32
cd server
make install-vllm-cuda
make install-awq # for quantization, not work for below A100
make install-eetq # for quantization
make install-flash-attention # For GPU below A100
make install-flash-attention-v2-cuda # For GPU >= A100
cd flash-attention-v2/csrc
cd ft_attention && python setup.py install && cd .. &
cd fused_dense_lib && python setup.py install && cd ..&
cd fused_softmax && python setup.py install && cd .. &
cd layer_norm && python setup.py install && cd .. &
cd rotary && python setup.py install && cd .. &
cd xentropy && python setup.py install && cd .. &
cd ../../../
pip install mamba_ssm megablocks causal_conv1d
```

📌 Step 5: Now we can deploy LLMs. Note that some packages (e.g. flask attention) will require newer CUDA; hence, the code will fail on turing machine. Indeed, it will only work on ampere1, ampere2, mercury 1-4, skampere 1, and hyperturing.
```
source /dfs/user/sttruong/miniconda3/bin/activate
conda activate eval_llm
source "/dfs/user/sttruong/.cargo/env"
export CARGO_HOME="/dfs/user/sttruong/.cargo"
export RUSTUP_HOME="/dfs/user/sttruong/.rustup"
export PATH=/dfs/user/sttruong/.cargo/bin:/dfs/user/sttruong/miniconda3/envs/openssl/bin:$PATH
export
LD_LIBRARY_PATH=/dfs/user/sttruong/miniconda3/envs/eval_llm/lib/python3.10/site-packages/torch
/lib:/dfs/user/sttruong/miniconda3/envs/eval_llm/lib:/dfs/user/sttruong/miniconda3/envs/openssl/lib
64:$LD_LIBRARY_PATH
export OPENSSL_DIR=/dfs/user/sttruong/miniconda3/envs/openssl
# For the llama family, the number of GPUs has to be a multiple of 4, and don't allocate machine that are almost full.
export CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
```

📌 Step 6: Launch text generation inference server
```
text-generation-launcher \
   --model-id meta-llama/Llama-2-7b-chat-hf \
   --port 8889 \
   --max-input-length 4096 \
   --max-total-tokens 8192 \
   --max-batch-prefill-tokens 4096
```

📌 Step 7: Calling the server from python or a terminal.

```
# from a terminal
curl --location 'https://localhost:8889/generate' \
--header 'Content-Type: application/json' \
--data '{
    "inputs": "[INST] Question: Who is Albert Einstein?\nAnswer: [/INST] ",
    "parameters": {
        "temperature": 1.0,
        "top_p": 1.0,
        "top_k": 50,
        "repetition_penalty": 1.0,
        "max_new_tokens":1024
    }
}'

# from Python
import requests
import json
url = "https://localhost:8080/generate"
payload = json.dumps({
  "inputs": "[INST] Question: Who is Albert Einstein?\nAnswer: [/INST] ",
  "parameters": {
    "temperature": 1,
    "top_p": 1,
    "top_k": 50,
    "repetition_penalty": 1,
    "max_new_tokens": 1024
  }
})
headers = {
  'Content-Type': 'application/json'
}
response = requests.request("POST", url, headers=headers, data=payload)
print(response.text)
```

Install Python

Motivations: Some packages require a newer Python version to run.
Step-by-step Instructions:

```
# go to a designated folder that stores your environment package
cd /dfs/user/sttruong/env
wget https://www.python.org/ftp/python/3.10.11/Python-3.10.11.tar.xz
tar -xvf Python-3.10.11.tar.xz
cd Python-3.10.11/
./configure --prefix=/dfs/user/sttruong/env/python3.10
make install -j 64
```

To change the default to python3.10, edit your ~/.bashrc.user file: export PATH=/dfs/user/sttruong/env/texlive/2023/bin/x86_64-linux:/dfs/user/sttruong/env/python3.10/bin:/dfs/user/sttruong/env/openbabel/bin:$PATH
alias python="python3.10"

Install Jupyter Kernel

Go to the virtual env or conda environment
pip install ipykernel
python -m ipykernel install --user --name=pyro-playground
To delete a kernel:
jupyter kernelspec uninstall geometric_ampere
To start Jupyter Notebook:
jupyter-notebook --no-browser --ip 0.0.0.0 --port 10000

Install Torch

Motivations: There are two families of machines in SNAP: the one with old CUDA (e.g., turing machines) and the one with new CUDA (ampere, hyperturing, mercury, etc). You will need a separate CUDA environment for each family.

CUDA version on Ampere is 11.7, meaning the latest PyTorch version we can use is 2.0.1
pip install torch==2.0.1
CUDA version on Turing is 11.3, meaning the latest PyTorch version we can use is 1.12.1
pip install torch==1.12.1

Everything Tmux

tmux new -s session_name
Login server
run krbtmux: /afs/cs/software/bin/krbtmux
run reauth: /afs/cs/software/bin/reauth
run jupyter: jupyter lab --no-browser
Control+B, then press D
tmux list-sessions
tmux a -t [screen_id]

Everything ngrok

Go to ngrok.com and register an account.

Install ngrok
wget https://bin.equinox.io/c/bNyj1mQVY4c/ngrok-v3-stable-linux-amd64.tgz
tar -xvzf https://bin.equinox.io/c/bNyj1mQVY4c/ngrok-v3-stable-linux-amd64.tgz
./ngrok config add-authtoken <your_token_on_dashboard_ngrok_com>

Get ngrok domain
On ngrok dashboard > Domains > New domain

Start ngrok
Start jupyter or any services you want. Remember the deployment port
./ngrok http --domain=<your_domain> <port>
You can now access your machine from the domain <your_domain>

Can you put the instruction to use W&B sweep here?

FAQ

Cluster Q&A

LLM APIs

See here