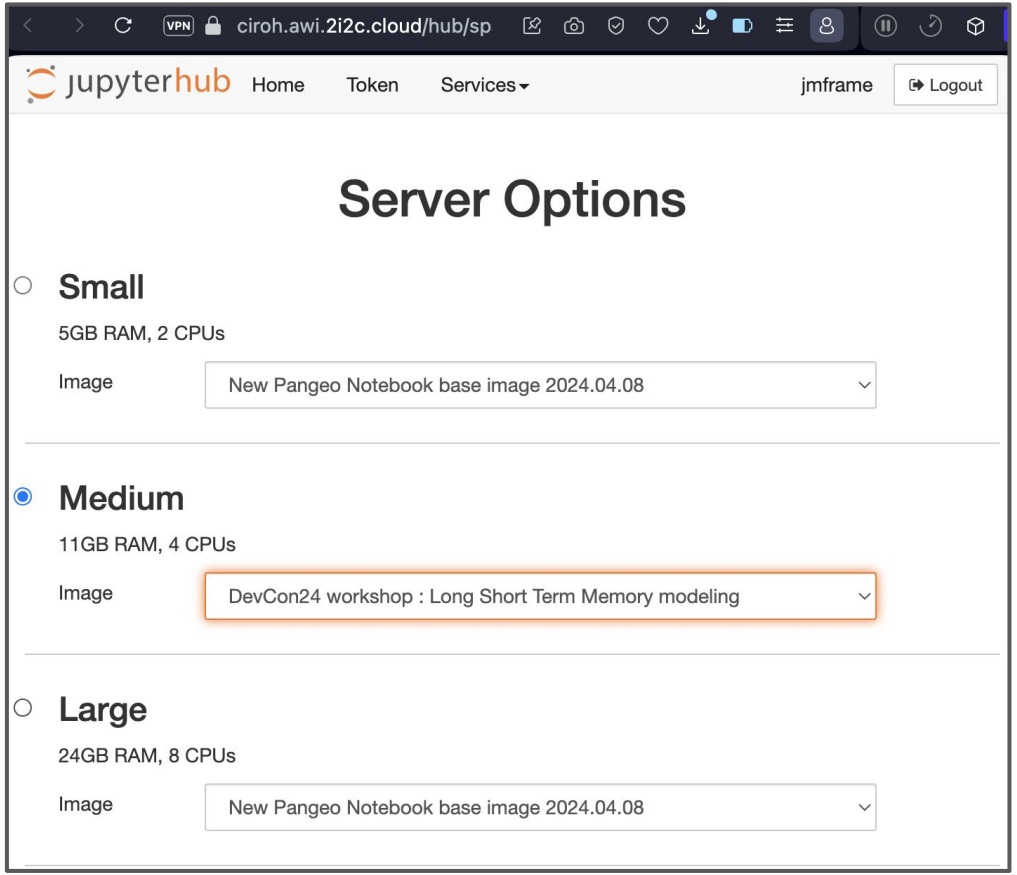
Deep Bucket Lab: basic information

- **Purpose of DeepBucketLab:** Interactive, hands-on tool for neural network-based modeling of hydrological processes
- **Educational Goals**
  - Importance of training models with large, diverse data
  - Introduce fundamentals of neural network modeling
- **Motivation**
  - Increasing importance of neural networks in hydrology
  - Growing demand for formal education in this area
- **Features of DeepBucketLab**
  - End-to-end platform for training neural network models
  - Transition from basics and synthetic data to practical, real world, examples
- **Impact**
  - Fill the gap in formal education on neural network-based hydrological modeling
  - Prepare students for advancements in hydrological prediction techniques

Deep Bucket Lab: Preliminaries



1. Go to **https://ciroh.awi.2i2c.cloud/**

2. Choose Server Option:

Medium.

3. On the Image dropdown, choose:

“DevCon24 Workshop: Long Short-Term Memory”

Deep Bucket Lab: Background on Neural Networks

History of Neural Networks

- **Mathematical Foundations**
  - **1763**: Bayes' Theorem by Thomas Bayes
  - **1943**: McCulloch and Pitts' binary threshold neuron model
  - **1957**: Perceptron by Frank Rosenblatt
  - **1950s-1960s**: Kolmogorov-Arnold representation theorem
- **Early Development**
  - **1974**: Backpropagation proposed by Paul Werbos
  - **1986**: Backpropagation popularized by Rumelhart, Hinton, and Williams
- **Challenges and Renewed Interest**
  - **1970s-1980s**: AI winter, reduced funding
  - **1980s**: Renewed interest in multi-layer perceptrons
- **Modern Neural Networks**
  - **1990s**: CNNs by Yann LeCun
  - **1991-1997**: LSTMs by Sepp Hochreiter
- **Attention is all you need**
  - **2017:** Vaswani et al. - Transformer model
  - **2018 - present**: Many variants combining transformer, CNN, gated recurrent networks like LSTM
- **2024: KAN and xLSTM**

Deep Bucket Lab: LSTM for hydrology - brief history

Early Use

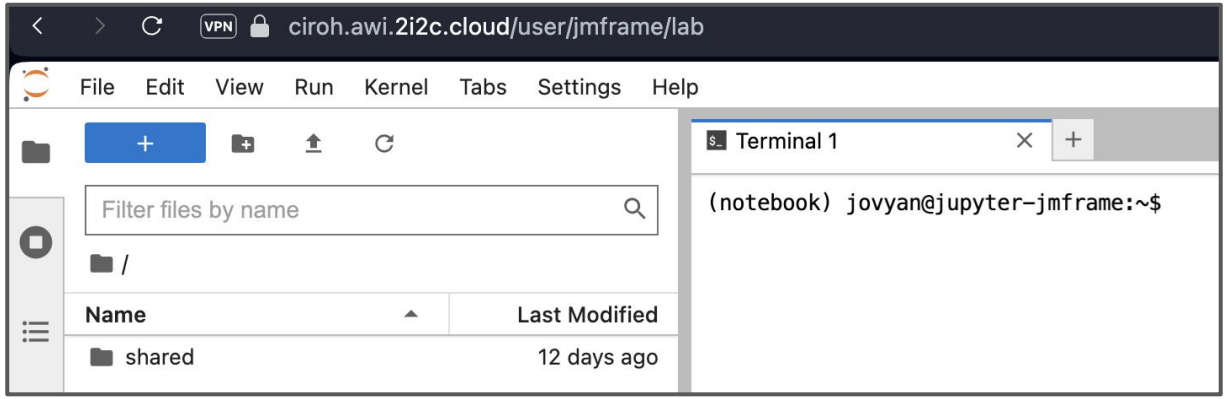
- 1995: Hsu et al. First DL use in hydrology (~30 years ago)
- 1999-2000: Consistent calls for expanded use of DL
- 2007: Krauße proposed for hydrology modeling
- 2015: Remesan and Mathew: "Hydrological Data Driven Modelling"
- 2016: Mhammedi et al. poor results due to overfitting
- **2017: Fang et al. (Shen group), for soil moisture (SMAP) modeling.**
- **2018: Kratzert et al. for Streamflow modeling. Exploiting lard data**
- 2019: **Ungauged basins** and general hydrologic laws
- 2020-2022: Surge in DL research due to LSTM's superiority in hydrology
- 2020's: Nearing et al. Potentially changing hydrology fundamentally
- 2020's: LSTM making better predictions across hydrology sub-disciplines
- 2020: Gauch et al. Importance of sufficient training data
- 2021: Gauch et al. multi-time step
- 2021-2023: Frame et al., physics-informed, extreme events, mass balance, distributed catchment modeling, mass conservation/mass balance, etc.
- 2022: Lees et al., probing LSTM for hydrologic states (snow, soil moisture, etc.)
- 2022: Feng et al. (Shen group), Differential parameter learning
- 2023: Giezendanner et al., conv-lstm for flood inundation modeling
- **2024: LSTM now tested successfully for just about every environmental variable**

LSTM was proposed as a hydrologic model by Krauße (2007): "[LSTM] is very promising because it provides an internal state which represents short and long term processes. Exactly these both classes of processes are necessary to represent both the state of the catchment and the dynamics of the rainfall-runoff process as a whole in fast reacting catchments"

Lynker

Deep Bucket Lab: Preliminaries

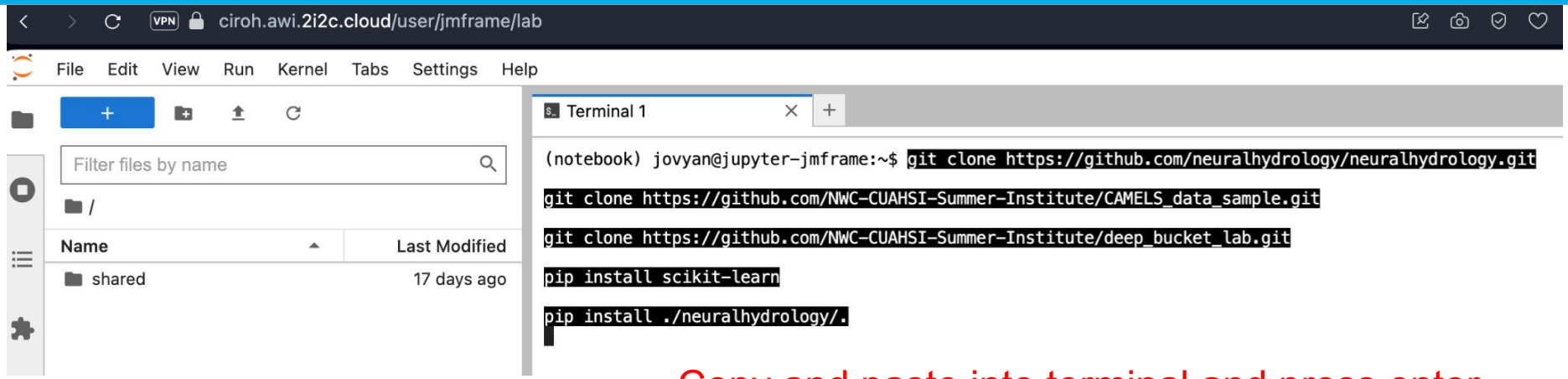
When the cloud compute platform loads, it should look like this:



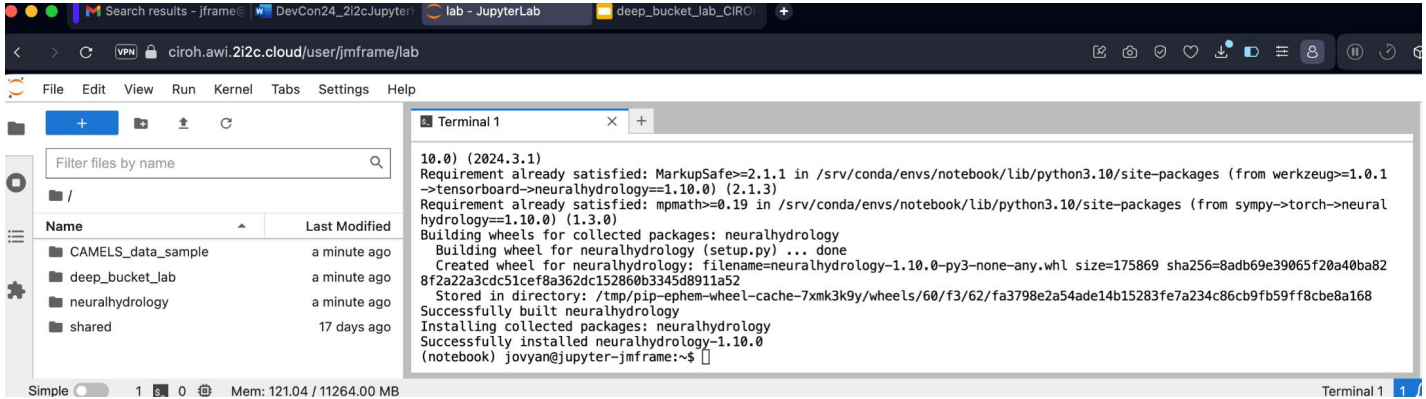
In the terminal, we need to clone some repositories and install a few things…

```
git clone https://github.com/neuralhydrology/neuralhydrology.git

git clone https://github.com/NWC-CUAHSI-Summer-Institute/CAMELS_data_sample.git

git clone https://github.com/NWC-CUAHSI-Summer-Institute/deep_bucket_lab.git

pip install scikit-learn

pip install ./neuralhydrology/.
```

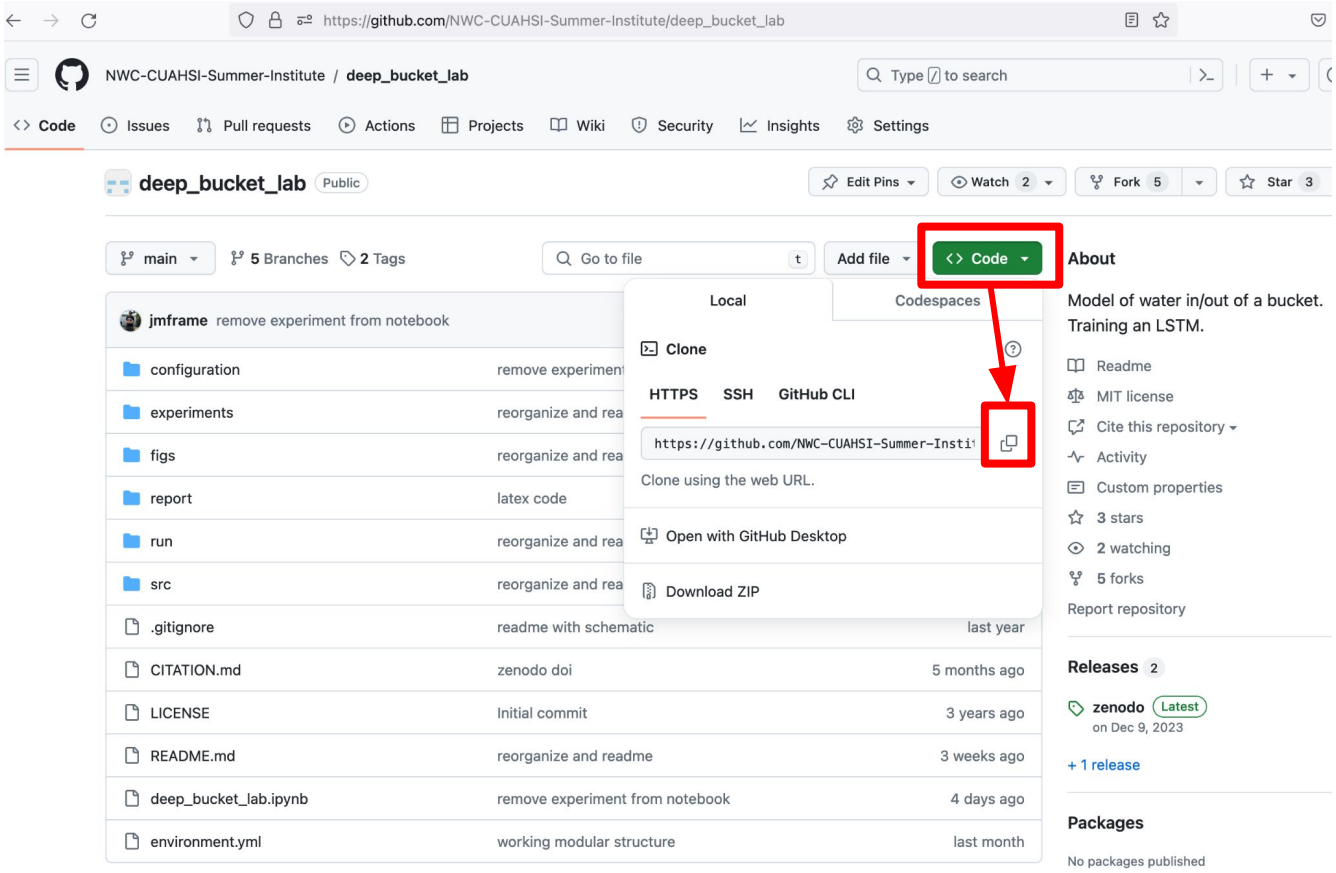
Literally copy and paste the text from the previous slide



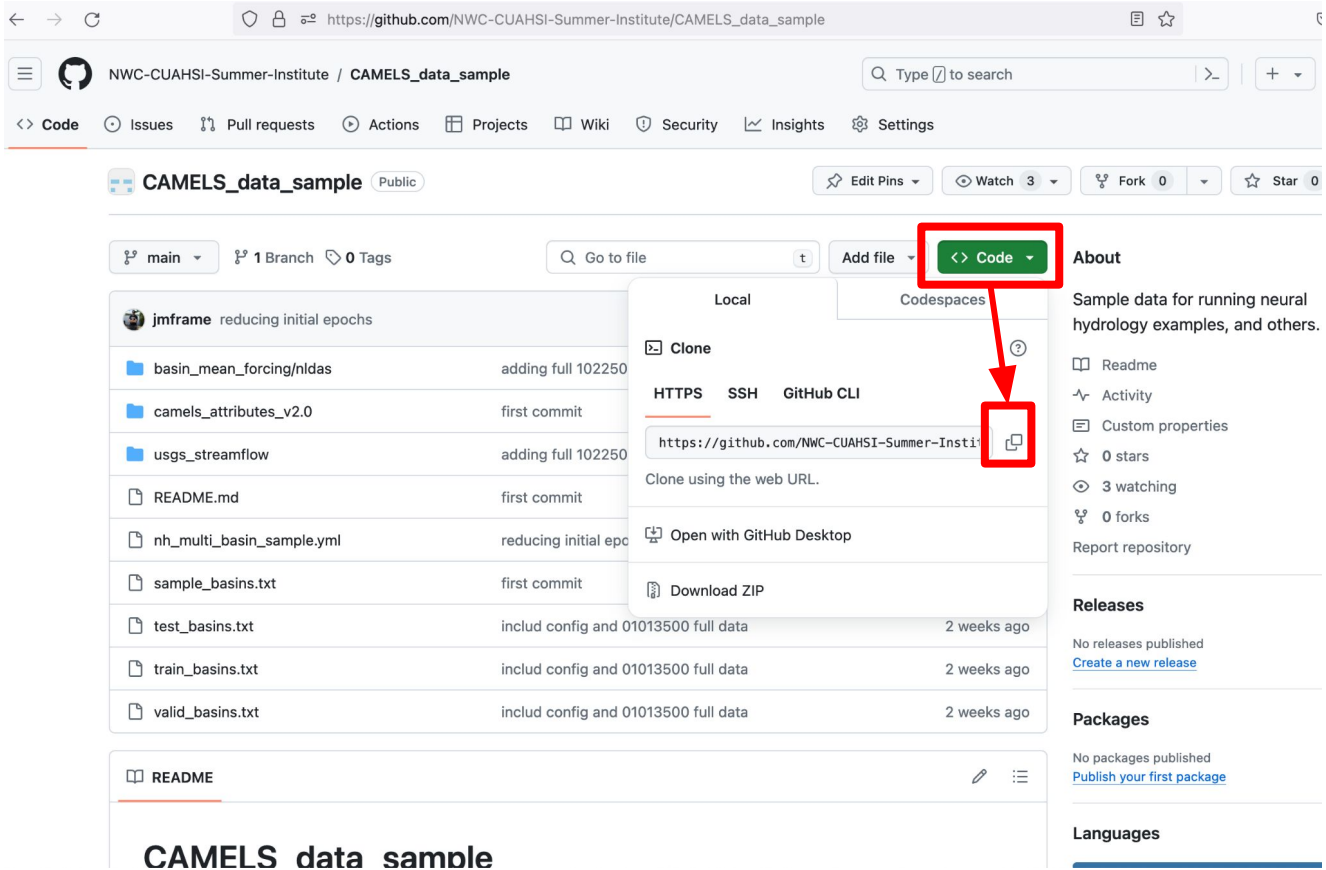
Copy and paste into terminal and press enter



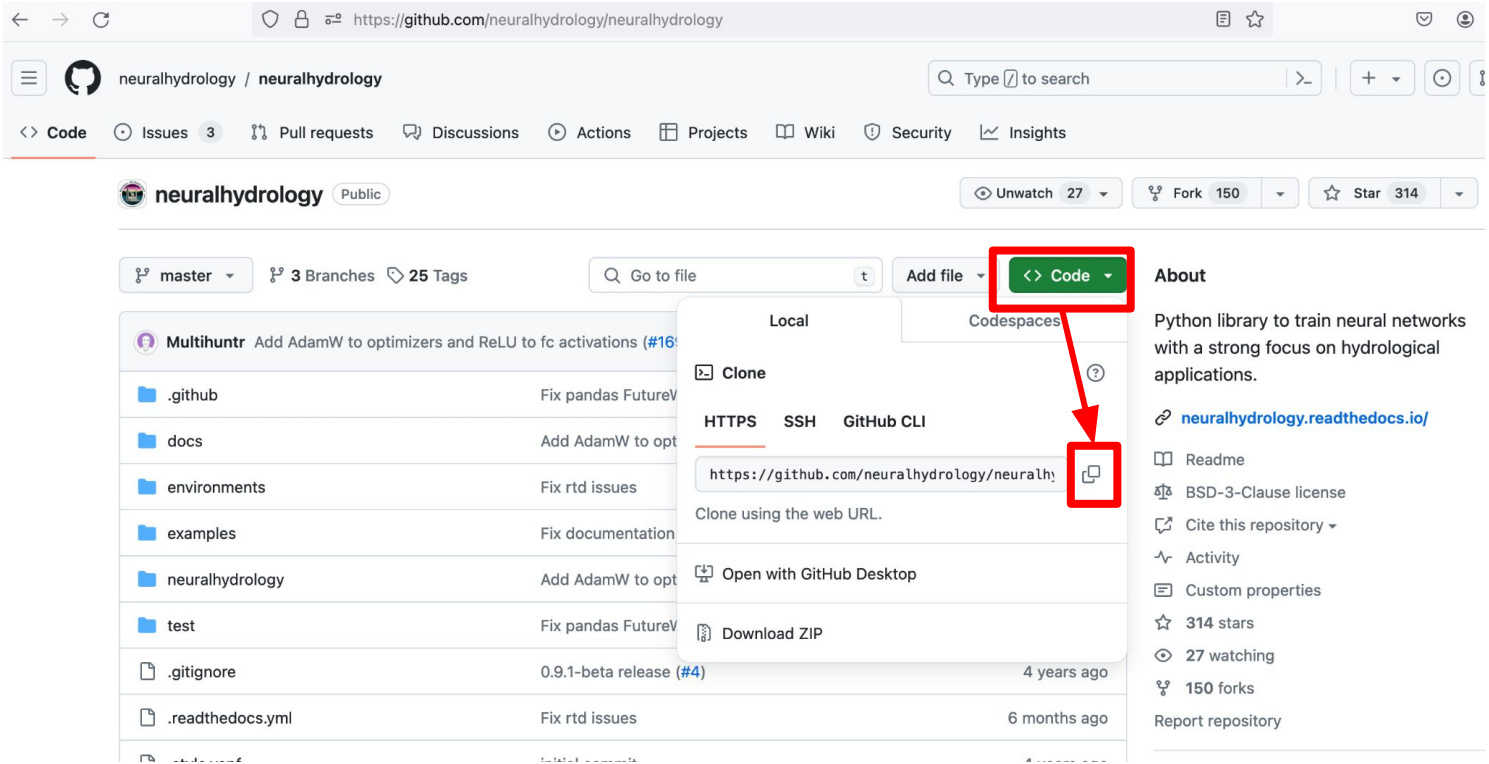
https://github.com/NWC-CUAHSI-Summer-Institute/deep_bucket_lab.git



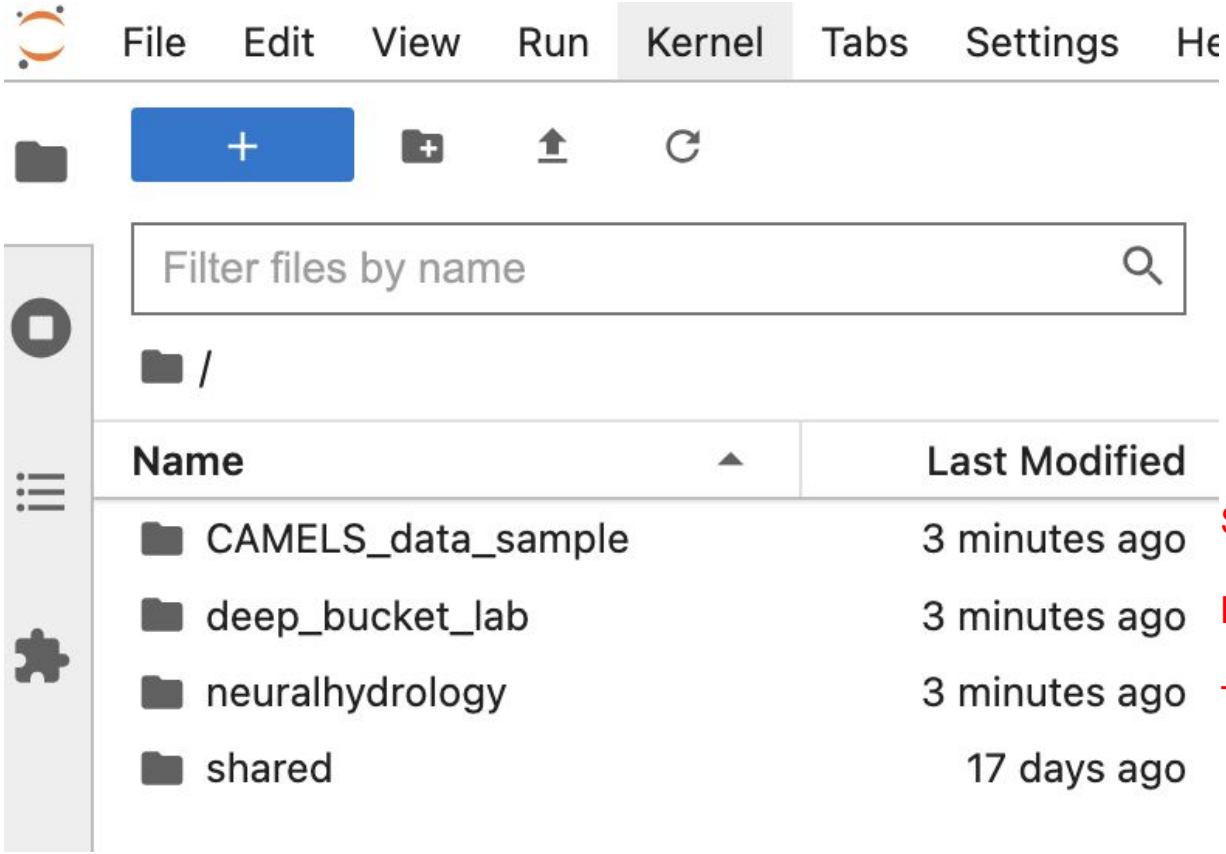
https://github.com/NWC-CUAHSI-Summer-Institute/CAMELS_data_sample.git



https://github.com/neuralhydrology/neuralhydrology.git



Now, what is in your Jupyter environment



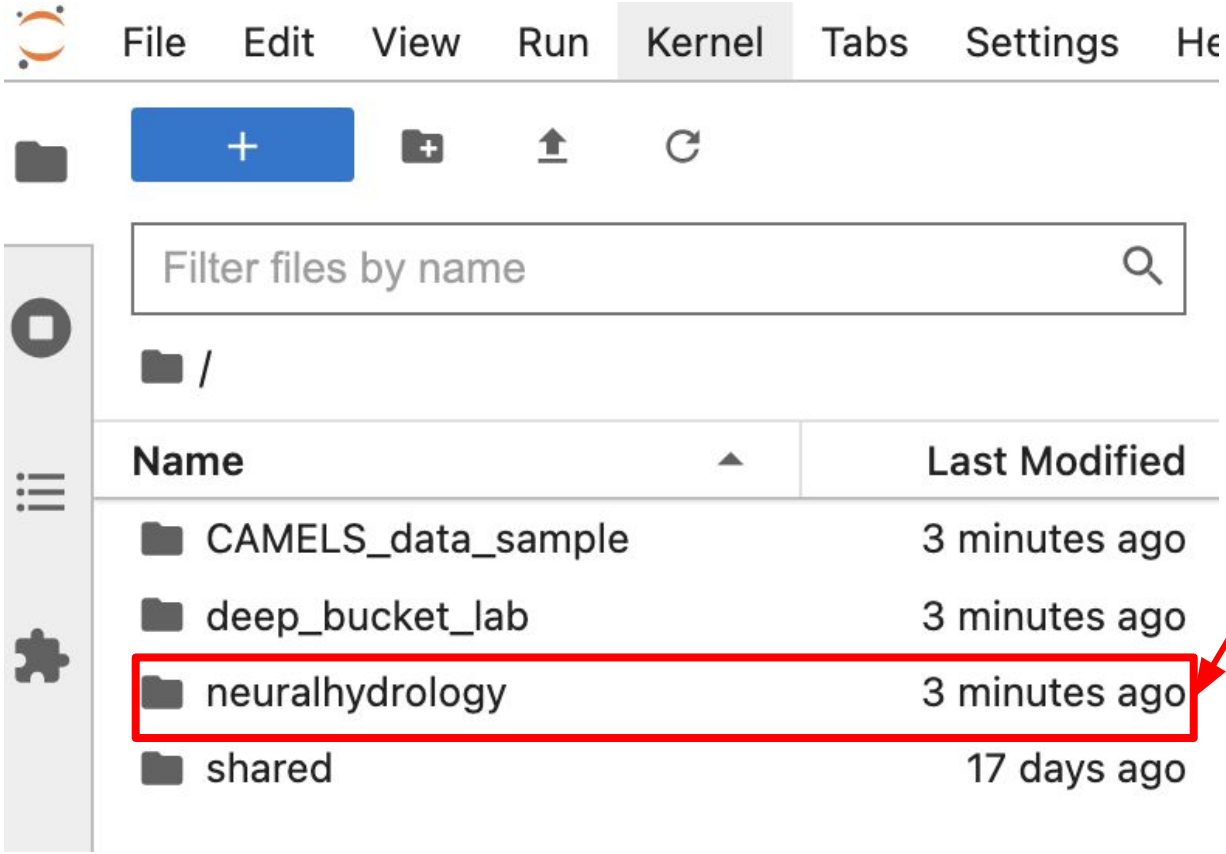
Subset of CAMELS data

Learn LSTM with synthetic data

Train a REAL LSTM model

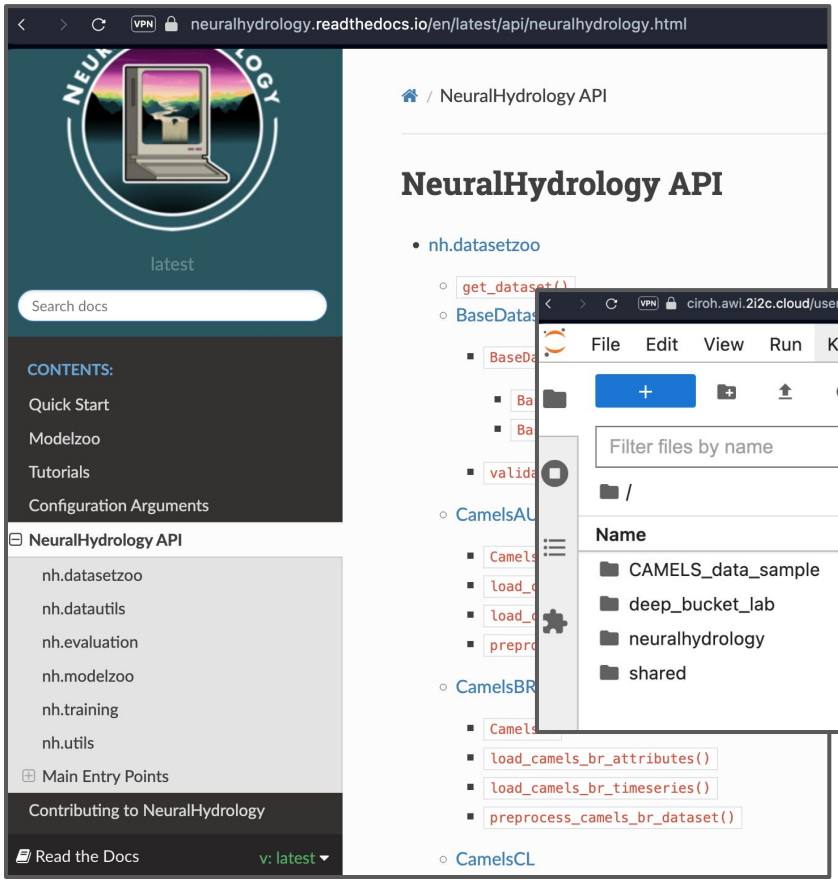
Lynker

Now, what is in your Jupyter environment



Let's start by running NeuralHydrology

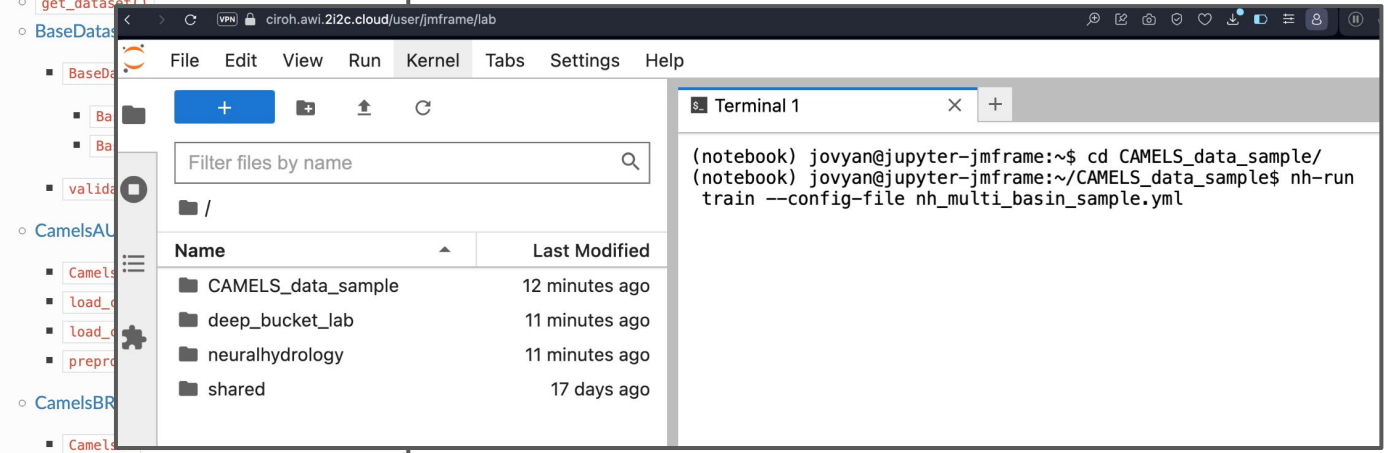
Learn how to train and run a NeuralHydrology LSTM



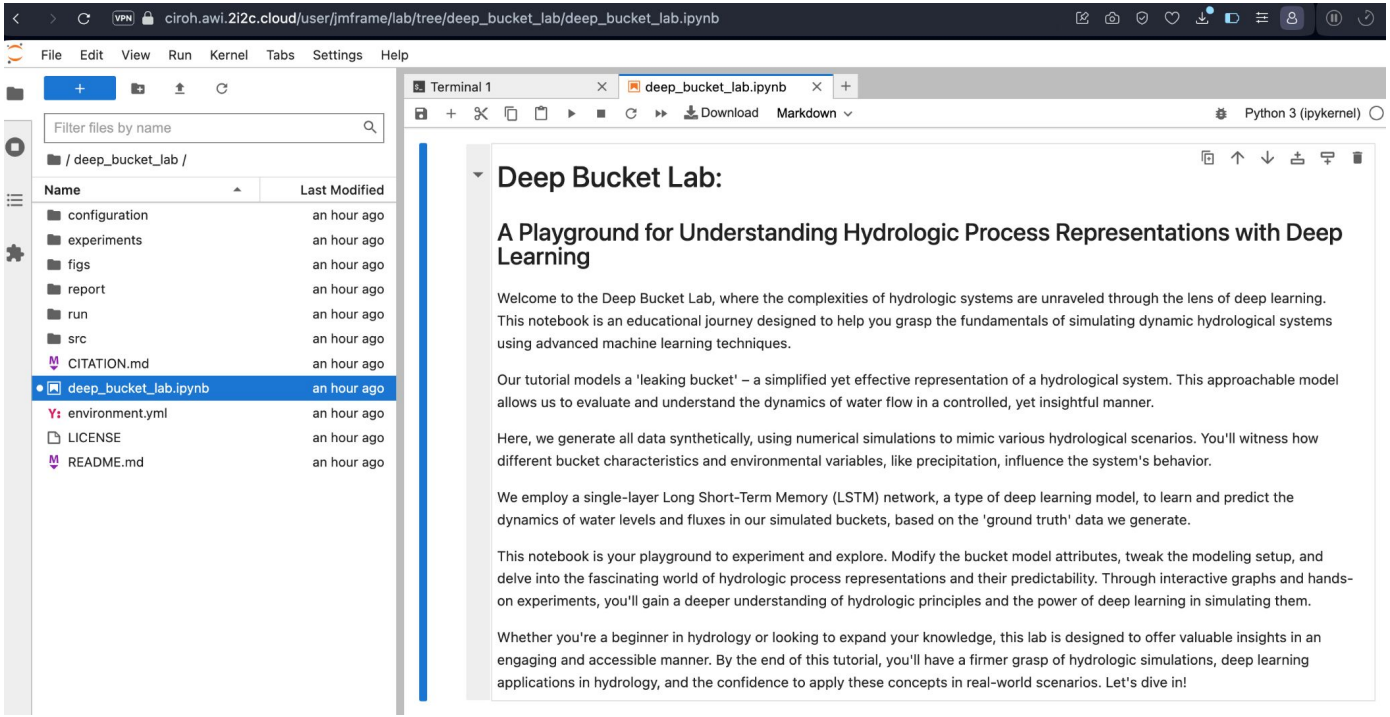
Keystrokes to train NeuralHydrology with sample data:

cd CAMELS_data_sample/

nh-run train --config-file nh_multi_basin_sample.yml



Now let's see what is actually going on with LSTM models



https://github.com/NOAA-OWP/lstm

https://github.com/NOAA-OWP/lstm/tree/master

README License Security

Basic Model Interface (BMI) for streamflow prediction using Long Short-Term Memory (LSTM) networks

This Long Short-Term Memory (LSTM) network was developed for use in the Next Generation Water Resources Modeling Framework (NextGen). LSTMs are able to provide relatively accurate streamflow predictions when compared to other model types. This module is available through a Basic Model Interface (BMI).

- Adaption from NeuralHydrology
- Sample Data
- Configurations
- Trained LSTM Model
- Dependencies
- Running BMI LSTM
- Weights and Biases
- Trained LSTM Model
- Unit Test

Adaption from NeuralHydrology

This module is dependent on a trained deep learning model. The forward pass of this LSTM model `nextgen_cuda_lstm.py` is heavily based on NeuralHydrology's `CudaLSTM`. Other model classes can be applied but `bmi_lstm.py` would need to load it in. More information about the python package NeuralHydrology can be found here.

./lstm/bmi_lstm.py

```
class bmi_LSTM(Bmi):
        def initialize( self, bmi_cfg_file=None ):
                …
        def update(self):
                …
        def get_training_configurations(self):
                …
        def get_scaler_values(self):
                …
        def create_scaled_input_tensor(self):
                …
        def scale_output(self):
                …
        def read_initial_states(self):
                …
        def set_static_attributes(self):
                …
        def initialize_forcings(self):
                …

        …etc.
```