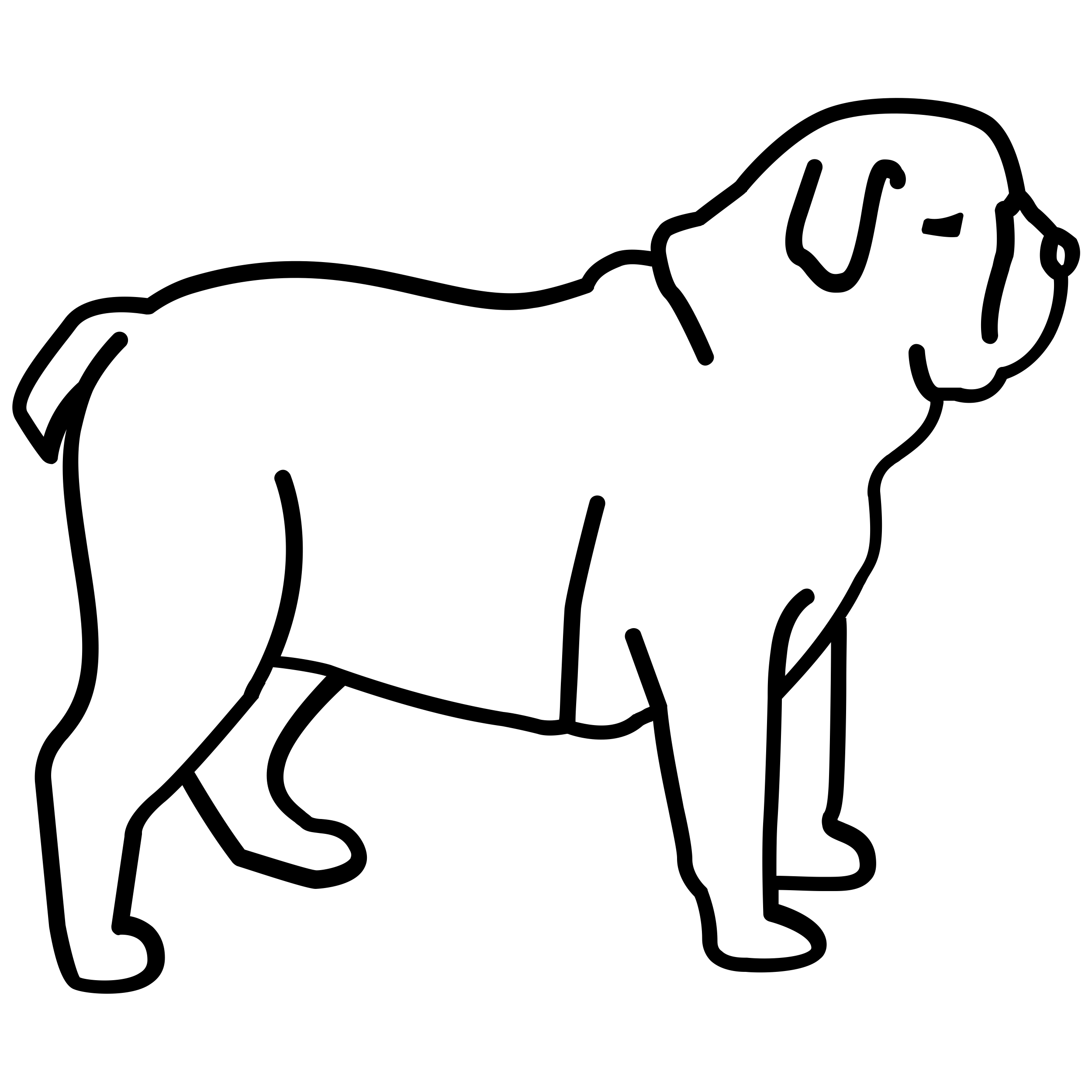
Ptolemy User’s Guide for Python Programmers

1.0



**Table of Contents**

[1. Prerequisites 3](#_Toc161759245)

[2. Gaining Access 3](#_Toc161759246)

[3. Steps for Signing Up 3](#_Toc161759247)

[4. Using Ptolemy 4](#_Toc161759248)

[4.1. Logging into Ptolemy Staging Server with SSH 5](#_Toc161759249)

[4.2. File Structure 6](#_Toc161759250)

[4.3. Modules on Ptolemy 6](#_Toc161759251)

[4.4. Creating a Python Environment 6](#_Toc161759252)

[4.5. Scheduling a Job 7](#_Toc161759253)

[4.6. Running the Job 7](#_Toc161759254)

[5. Appendix 1: slurm.sh 9](#_Toc161759255)

[6. Appendix 2: hello.py 10](#_Toc161759256)

[7. Appendix 3: Frequently Asked Questions 11](#_Toc161759257)

[7.1. How do I move files to Ptolemy? 11](#_Toc161759258)

[7.2. What if I need help? 11](#_Toc161759259)

[7.3. Who wrote this document? 11](#_Toc161759260)

# Prerequisites

To use Ptolemy you must have:

1. A Mississippi State University Net ID and password
2. A Smartphone Running the DuoMobile app. Logging into the GPU system via SSH or SFTP will require receiving and consenting to a push notification or typing in a code.
3. Either a wired connection to the MSU network or a VPN connection. See this site to get a VPN client set up on your machine: <https://servicedesk.msstate.edu/TDClient/45/Portal/KB/?CategoryID=80>

# Gaining Access

To gain access to Ptolemy, email Mike Navicky ([navicky@hpc.msstate.edu](mailto:navicky@hpc.msstate.edu)).

# Steps for Signing Up

When you receive an invitation email, you’re ready to start this section. To go from an initial account to a Ptolemy account, follow these steps:

1. Click on the link in the email to fill out request and set up your temporary password.
2. Wait to hear that your account has been created.
3. Visit <https://taps.hpc.msstate.edu> and login with your Net ID and temporary password. The screen will look like this:  
     
   A screenshot of a computer

   Description automatically generated
4. You will notice that the Training Status for “IT Security Training” and “Insider Threat Training” says “Incomplete” in red for both. Click “Start Training” on each of these courses and you will complete a training course.
5. Once your training is complete, the portal will look like this:  
     
   A screenshot of a computer

   Description automatically generated
6. You are now ready to use the system.

# Using Ptolemy

Using Ptolemy requires logging into a Ptolemy staging server, transferring your data and code, then scheduling your job to run on the GPU cluster using SLURM, a popular HPC job scheduler.

## Logging into Ptolemy Staging Server with SSH

Follow these steps:

Using the command line or an SSH tool, SSH to ptolemy-login-1.arc.msstate.edu. You will log in with your username and password and then complete a DUO push. The session looks like this.

A screenshot of a computer error

Description automatically generated

## File Structure

When first logging into ptolemy-login-1.arc.msstate.edu you will be in the /home/NETID directory for your account.

Your code should be placed in /scratch/ptolemy/users/NETID/

It is a great idea to create a subdirectory within your scratch directory for each project / job you will be running.

## Modules on Ptolemy

The server ptolemy-login-1.arc.msstate.edu uses a piece of software called “ml” – module loader – to load modules that will enable you to run analysis jobs on the machine.

For a python programmer, the key modules you will load are Python itself, CUDA (to enable the use of GPUs), and Miniconda (to enable loading additional Python packages). You can do this by hand in an interactive shell (e.g., “ml python/3.9.15” will load Python) but you will eventually want to capture all the modules you will be loading in a SLURM shell script. Feel free to load modules and play around.

You can see all available modules by typing “ml avail”.

## Creating a Python Environment

We recommend interactively creating a conda environment that you can later load in your SLURM script to run your Python code. Follow these steps:

1. First, load the Python and Miniconda modules like this:  
     
   ml python/3.9.15  
   ml miniconda3/4.10.3
2. Because of disk quotas, it is important that your conda packages get installed in the scratch drive. Configure conda to use a dir in your scratch drive this way:  
     
   conda config --add pkgs\_dirs /scratch/ptolemy/users/netid/packages
3. Then, create a Conda environment file and save it as environment.yml. For example:

|  |
| --- |
| channels:  - anaconda  dependencies:  - numpy~=1.19.2  - pandas==1.4.1  - tensorflow-gpu |

This environment file would load numpy, pandas, and tensorflow to support a Python script capable of building a neural network. Unfortunately, this collection of packages is not the latest Tensorflow, but it will get you started.

1. You can now see if you have any miniconda environments already created by typing:  
     
   conda env list
2. Then, to create the environment that you described in your environment.yml file, run the following. Note that you can change “myname” to a name of your choice.  
     
   conda env create --name myname -f environment.yml --force
3. When conda is finished creating your environment, run:  
     
   conda env list  
     
   to see the successfully created environment now available.  
     
   Note that any modules you load only last as long as your session. Thus, if you want to interrupt your work, no problem, just go back to step 2, load Python and Miniconda, and your environments will still be there waiting for you (conda env list) because they are persisted in your directories.
4. Sidenote: You can manually remove an environment (for example, if you run out of space) by executing:  
     
   conda env remove --name myname

## Scheduling a Job

Now that you have created a conda environment, it’s time to schedule the running of your job. The job will be triggered by a scheduler named “SLURM”, and you will control it with a script.

For purposes of illustration, we will schedule the running of a Python program (hello.py) that will train a neural network to recognize handwritten digits in the mnist data set.

Our script, saved as slurm.sh, is in Appendix 1 below. Lines you will want to customize are highlighted in light grey. Our python program, saved as hello.py is in Appendix 2 below. The program depends upon data in the form of a file, mnist.npz. You can download all of these files necessary to try out Ptolemy here:

<https://github.com/docbarlow/ptolemy_demo>

## Running the Job

Now that your slurm.sh script is created, you can ask SLURM to run your job. Using the following command:

sbatch ./slurm.sh

You can monitor the execution of your jobs by using:

squeue -u jhb11

And, armed with the job id, you can cancel a job using:

scancel <job\_id>

# Appendix 1: slurm.sh

#!/bin/bash

## This line requests that the job be allocated one node

#SBATCH -N 1

## This line requests one task for the job

#SBATCH -n 1

## This specifies the amount of memory required for the job.

## Here, it's set to 10 Gigabytes. This memory is allocated per node.

#SBATCH --mem=10G

## This sets the partition (or queue) to gpu-a100

## Partitions are defined by the system administrators and

## determine the set of nodes the job can run on.

#SBATCH -p gpu-a100

## This specifies the account for job charging or accounting.

## Here, it's set to an account named 'test'. Use this for now.

#SBATCH -A test

## This sets the time limit for the job.

## Here, the job can run for up to 1 hour.

## If the job exceeds this limit, SLURM will terminate it.

#SBATCH -t 1:00:00

## This requests a specific generic resource (GRES), in this case,

## a GPU. Specifically, it requests one Nvidia A100 GPU with 10 GB of memory.

#SBATCH --gres=gpu:a100\_1g.10gb:1

## This sets the name of the job to "jhb11-GPU-mnist".

## This name will appear in the queue and in various SLURM reports,

## helping you to identify your job.

#SBATCH --job-name "jhb11-GPU-mnist"

## This specifies that all the output from the job (stdout) will

## be written to the file output.out. This file will be created in the

## directory from which the SLURM script is submitted.

#SBATCH --output=output.out

## each of these lines uses the "module load" command

## to load the modules our python script will need to use

ml cuda

ml python/3.9.15

ml miniconda3/4.10.3

## change directories to your user scratch location where your

## script is

cd /scratch/ptolemy/users/jhb11/hello\_world

## now create your conda environment - doesn't work through slurm??

## conda env create --file ./environment.yml --name jhb11-GPU-mnist --force

## activate conda environment

source activate jhb11-GPU-mnist

## Set a path you can use inside your python script to find your data files

export DATA\_FILE\_PATH="/scratch/ptolemy/users/jhb11/hello\_world"

## change directories to your user scratch location where your

## script is

cd /scratch/ptolemy/users/jhb11/hello\_world

## run the script

python hello\_world.py

# Appendix 2: hello.py

import numpy as np

import pandas as pd

import random

import tensorflow as tf

import os

from tensorflow.keras.models import Sequential, model\_from\_json

from tensorflow.keras.layers import Flatten, Conv2D, Dense, MaxPooling2D

from tensorflow.keras.optimizers import SGD

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.datasets import mnist

# see if we have multiple gpus

print("Num GPUs Available: ", len(tf.config.experimental.list\_physical\_devices('GPU')))

# this is convenient – your job will fail if the tensor flow version

# you’re using isn’t built to use gpu / cuda

if not tf.test.is\_built\_with\_cuda():

print("Not built with cuda… exiting")

exit(1)

# set path to mnist file

# path comes from environment variable set in our slurm script

file\_path = os.getenv('DATA\_FILE\_PATH') + "/mnist.npz"

# load the mnist data

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data(file\_path)

# reshape to set up grayscale color channel for each 28x28 image

x\_train = x\_train.reshape((x\_train.shape[0], 28, 28, 1))

# normalize color between 0 and 1

x\_train = x\_train.astype ('float32') / 255.0

# create label for it

y\_train = to\_categorical (y\_train)

# setup the model

model = Sequential([

# Input shape is 28x28x1 like our images

# There will be 32 convolutions (little parts of the image) that are 3x3

# The activation function will be RELU; we talked about it in class

Conv2D(32, (3, 3), activation='relu', strides=1, input\_shape=(28, 28,1)),

# The next layer should pool together all the stuff we learned in the previous

# layer - (TODO: explain pooling layers and why they are helpful)

MaxPooling2D((2, 2)),

# Get a flat tensor - a continuous list of values

Flatten(),

# TODO: explain this one

Dense(100, activation='relu'),

# This is our last or "output layer". Notice it has 10 values like our labels (0-9).

# Softmax gives us a probability distribution so we can see which digit is most likely

#

Dense(10, activation='softmax')

])

optimizer = SGD(learning\_rate=0.01, momentum=0.9)

# compile the model

model.compile(

optimizer=optimizer,

loss='categorical\_crossentropy',

metrics=[ 'accuracy' ]

)

# fit the model

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=32)

# serialize model to JSON

model\_json = model.to\_json()

with open("model1.json", "w") as json\_file:

json\_file.write(model\_json)

# serialize weights to HDF5

model.save\_weights("model1.h5")

print("Saved model to disk")

# Reshape all the test data

x\_test = x\_test.reshape((x\_test.shape[0], 28, 28, 1))

x\_test = x\_test.astype ('float32') / 255.0

y\_test = to\_categorical(y\_test)

# Now let's test the model

score = model.evaluate(x\_test, y\_test, verbose=1 )

print("%s: %.2f%%" % (model.metrics\_names[1], score[1]\*100))

# Appendix 3: Frequently Asked Questions

## How do I move files to Ptolemy?

Use sftp. You can either use this from the command line:

sftp netid@ ptolemy-login-1.arc.msstate.edu

or you can download a good SFTP client for Mac or Windows.

## What if I need help?

If you need technical support, please email Mike Navicky: [navicky@hpc.msstate.edu](mailto:navicky@hpc.msstate.edu)

## Who wrote this document?

The initial draft was written by Jonathan Barlow: [barlow@datascience.msstate.edu](mailto:barlow@datascience.msstate.edu) to help MSU users begin using the new cluster. Feel free to send any corrections or suggestions his way.