



# *VSCoDe and Github Copilot for NCAR Systems*

CISL/CSG

*Brett Neuman*

October 24th, 2024

# Common NCAR Remote Connections

## Terminal

- *vi / emacs*
- *tmux*
- *No GUI*

```
...ata-access2/~ -- -bash ...    ~ -- -bash ...    ~ -- -bash ...    ~ -- -bash ...
(base) cisl-hotspings:~ bneuman$ derecho
Last login: Wed Oct  9 13:39:15 2024 from 67.176.37.208
*****
*      Welcome to Derecho - Tuesday, October 08 2024
*****
Today in the Daily Bulletin (https://arc.ucar.edu/articles):

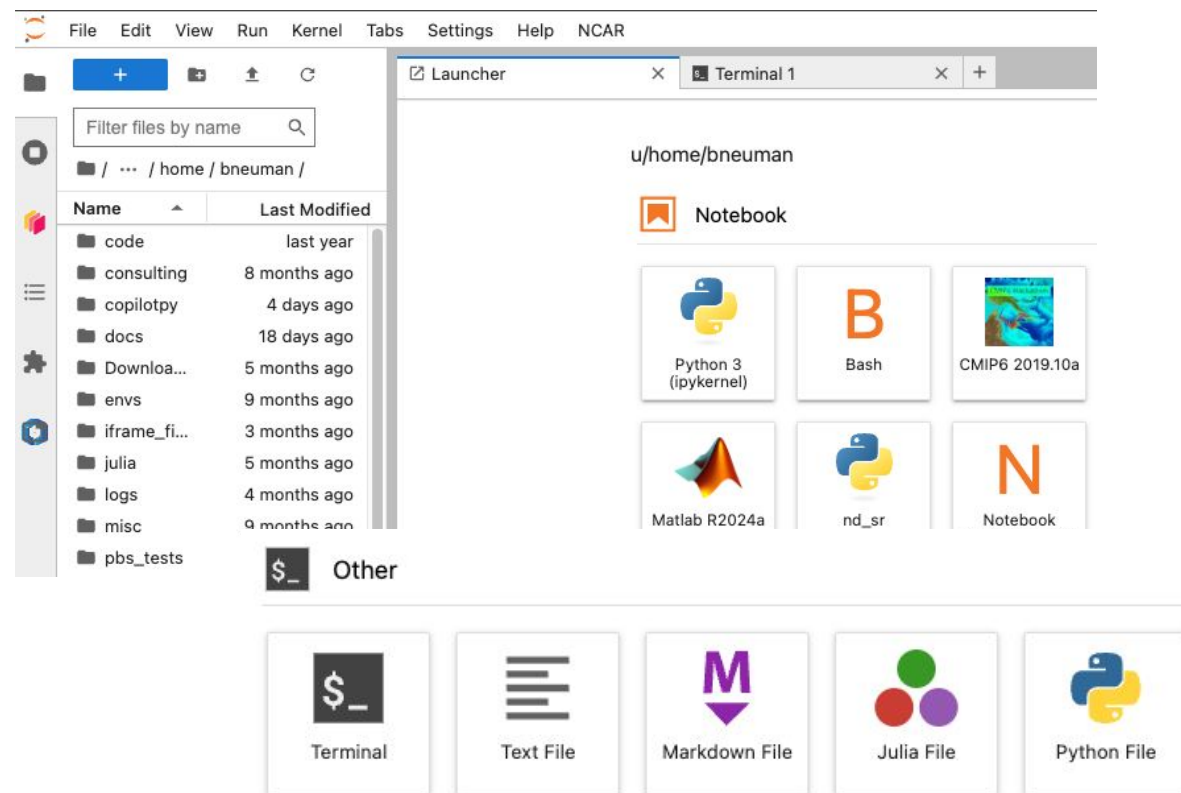
- HPC Systems Scheduled Maintenance - November 5, 2024
- Reminder: SEA Event on Developer Exchange Programs this Thursday
- Read the article: ICAS 2024 Symposium engages the international community
- Free hybrid seminar on convective
- Join the RSE movement! Second annual conference
- NSF NCAR Graduate Visitor Program

Documentation:      https://ncar-hpc.org
Get Engaged
  NCAR HPC Users Group: https://nhug.org
  Join us on Slack:     https://ncar.slack.com
Consulting Services:  https://rchelp.org
CISL Help:           cislhelp@ucar.edu

bneuman@derecho1:~> ls
access_report.txt  copilotpy  ei
```

## Jupyterhub

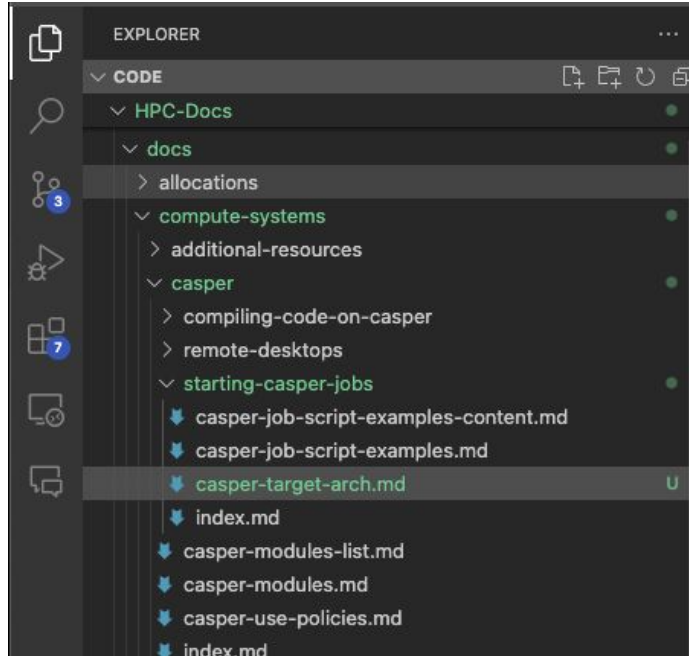
- *Jupyter notebooks*
- *Limited terminal integration*
- *GUI with file management*



# Why VSCode



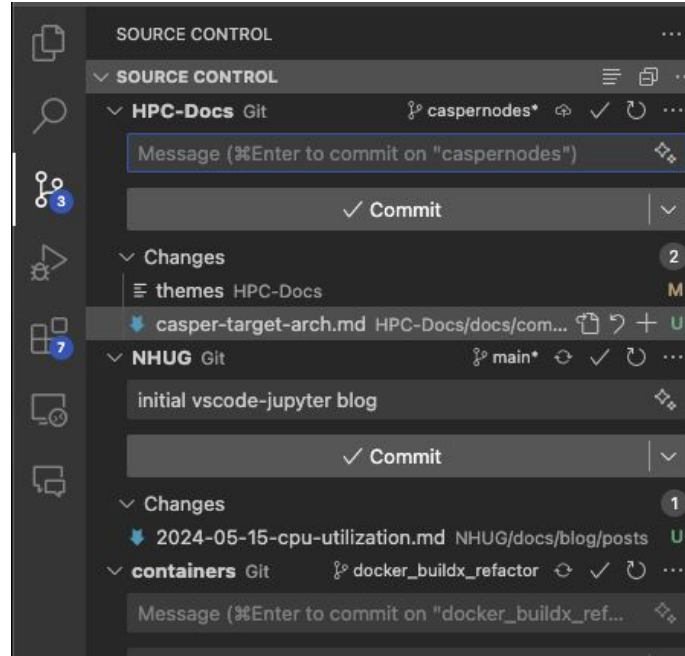
## File Explorer



- GUI
- Connect to GLADE
- Transfer files between local and remote machines!



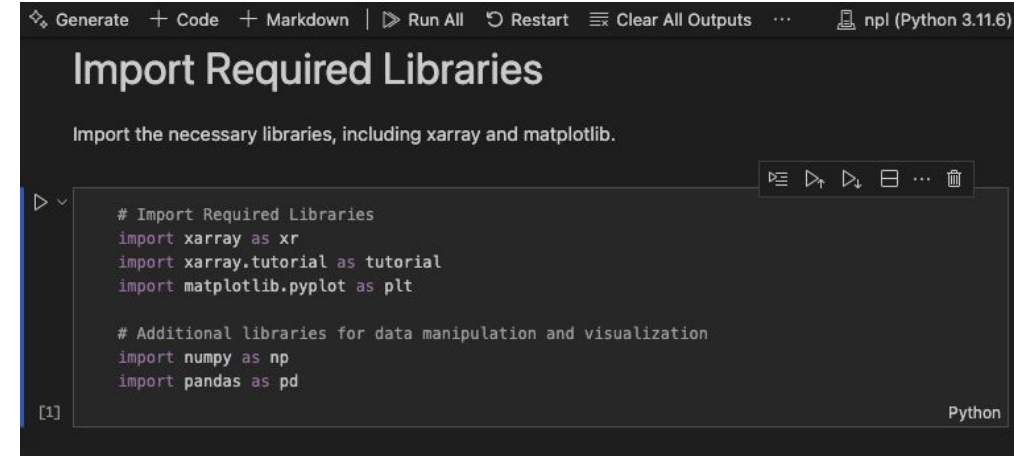
## Source Control



- Git visualizations
- Shortcut buttons for common operations



## Extensions



- Jupyter notebooks
  - With your custom NCAR or local kernels
- RemoteSSH
  - Connect to Derecho and Casper
- Code Autocomplete (Intellisense)
  - Separate extensions from Copilot
- Copilot

# Why VSCode

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a project structure with folders like 'HPC-Docs', 'NHUG', and 'posts'. The main editor area displays a file named 'vscode-hpcdocs.md' with the following content:

```
1 # Visual Studio Code
2
3 Visual Studio Code, also commonly referred to as VS Code, is a
4 source-code editor developed by Microsoft for Windows, Linux, macOS
5 and web browsers. Features include support for debugging, syntax
6 highlighting, intelligent code completion, snippets, code refactoring,
7 and embedded version control with Git.
8
9 The Visual Studio Code Remote - SSH extension allows you to open a
10 remote folder on any remote machine, virtual machine, or container
11 with a running SSH server and take full advantage of VS Code's feature
12 set. Once connected to a server, you can interact with files and
13 folders anywhere on the remote filesystem.
14
15 VSCode is available for NCAR issued laptops in the *Self Service*
16 application or can be downloaded directly from the Microsoft VS Code
17 website.
18
19 # Connecting to Derecho or Casper
20
21 The Visual Studio Code Remote SSH extension allows you to connect to a
22 NCAR system and to the GLADE filesystem virtual machine. Once connected
23 to a server, you can interact with files and folders anywhere on
24 GLADE. You can connect to the Derecho or Casper login nodes by
25 following these steps:
26
27 !!! example "Connecting to NCAR system login nodes"
28
29 == "Derecho"
30
31 1. Press F1 and run the *Remote-SSH: Open SSH Host* command.
32 2. Enter your NCAR CIT username in the following format in the
33 input box that appears and press enter: user@derecho.hpc.ucar.
34 edu
35 3. Enter your NCAR CIT password when prompted.
36 4. Accept the DUO Push request.
37
38 == "Casper"
```

The terminal at the bottom shows the following output:

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) cisl-hotsprings:Code bneuman$
```

## Visual Studio Code

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor developed by Microsoft for Windows, Linux, macOS and web browsers. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded version control with Git.

The Visual Studio Code Remote - SSH extension allows you to open a remote folder on any remote machine, virtual machine, or container with a running SSH server and take full advantage of VS Code's feature set. Once connected to a server, you can interact with files and folders anywhere on the remote filesystem.

VSCode is available for NCAR issued laptops in the *Self Service* application or can be downloaded directly from the Microsoft VS Code website.

## Connecting to Derecho or Casper

The Visual Studio Code Remote SSH extension allows you to connect to a NCAR system and to the GLADE filesystem virtual machine. Once connected to a server, you can interact with files and folders anywhere on GLADE. You can connect to the Derecho or Casper login nodes by following these steps:

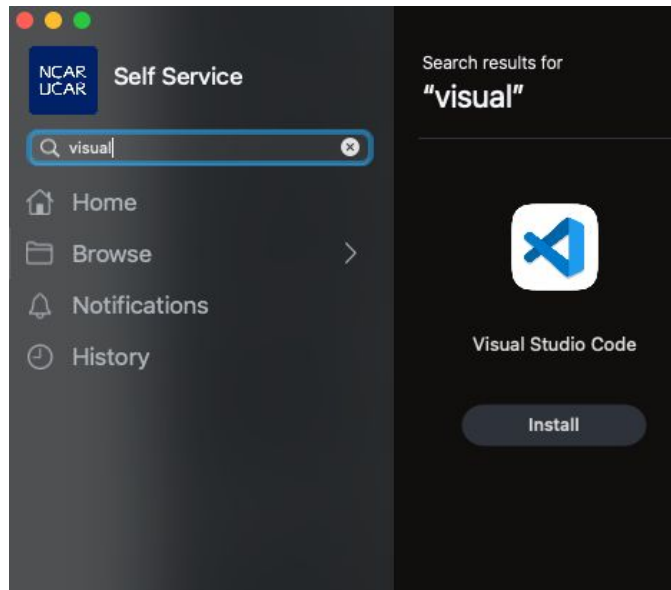
!!! example "Connecting to NCAR system login nodes" === "Derecho"

1. Press F1 and run the \*Remote-SSH: Open SSH Host\* command.
2. Enter your NCAR CIT username in the following format in the input box that appears and press enter:  
user@derecho.hpc.ucar.edu
3. Enter your NCAR CIT password when prompted.
4. Accept the DUO Push request.

# How to get VSCode

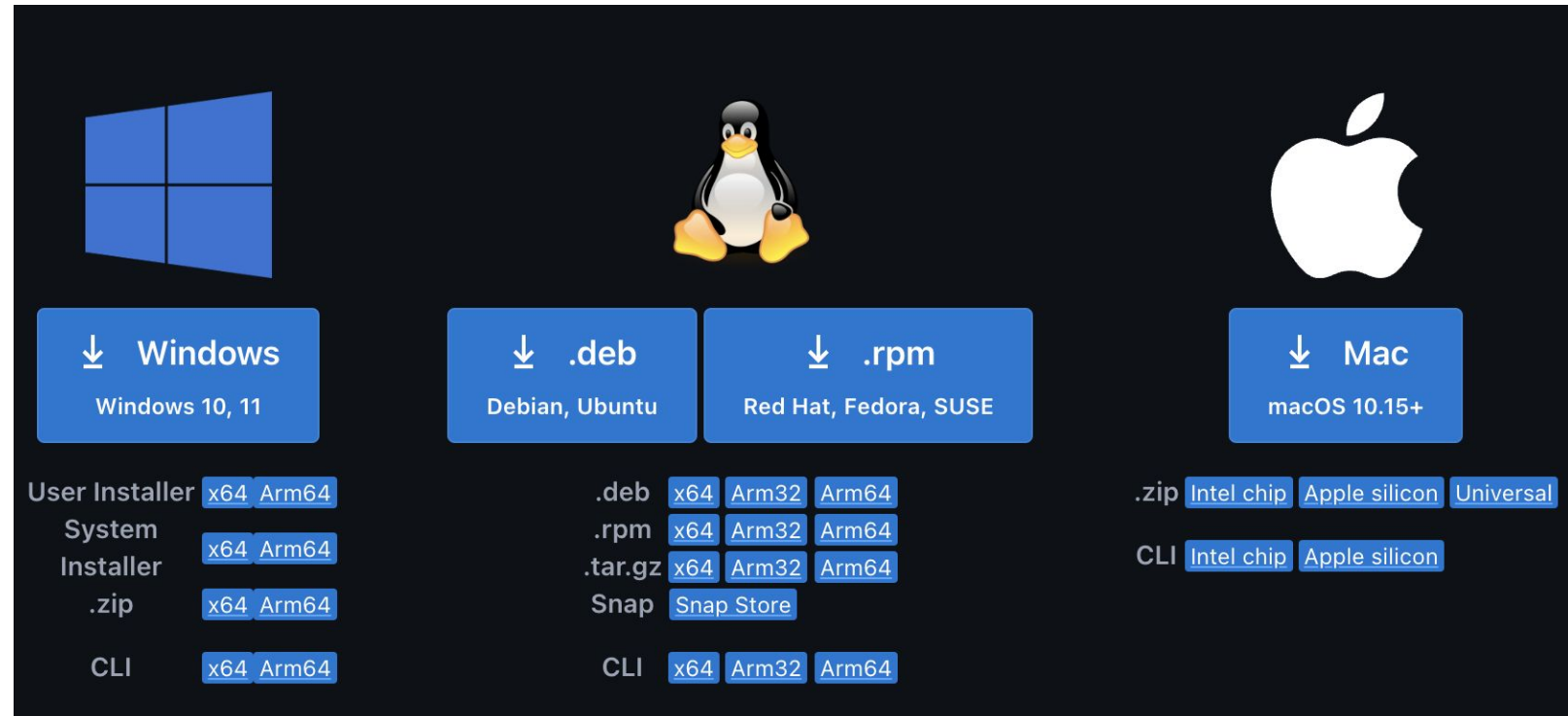
## NCAR device

- Self Service



## Otherwise

- <https://code.visualstudio.com/download>

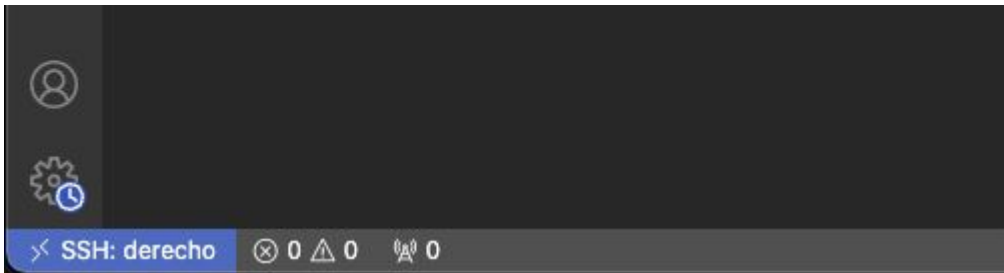


# Connecting to Derecho and Casper Login Nodes

The Visual Studio Code **RemoteSSH** extension allows you to connect to Derecho, Casper, and the GLADE filesystem. Once connected to a server, you can interact with files and folders anywhere on GLADE. You can connect to the Derecho or Casper login nodes by following these steps:

## **RemoteSSH**

- Press F1 and run the Remote-SSH: Open SSH Host command.
- Enter your NCAR CIT username in the following format in the input box that appears and press enter:  
*user@casper.hpc.ucar.edu*
- Enter your NCAR CIT password when prompted
- Accept the DUO Push request



The <> Button on the bottom left can also be used to quickly connect or reconnect to remote systems

Once you have configured a remote server for the first time, you can use the local terminal to launch a remote window:

```
code --remote ssh-remote+derecho.hpc.ucar.edu  
/glade/u/home/bneuman/myproject
```

# Terminal vs. Interactive Modes

## Terminal Mode

- Provides similar environment as the Mac Terminal over SSH
  - Ability to load modules, submit jobs, navigate filesystem, etc.
- Open files in the preview window using the command “`code <filename>`”
- Commands launched from the VSCode terminal will run on that node
  - e.g. On a compute node the terminal issued command will run on the compute node resources

## Interactive Mode

- VSCode visualization functionality
  - Includes operations with the File Explorer
  - Opening files with the ‘code’ command to generate a preview window
- Utilizes the ‘.vs-server’ folder and settings that are generated
  - Shows the running process as ‘node’
- Will run on the node that RemoteSSH connected to

**WARNING:** You should always assume your code is running on the login node unless you connect a new window to a compute node with an active interactive PBS job

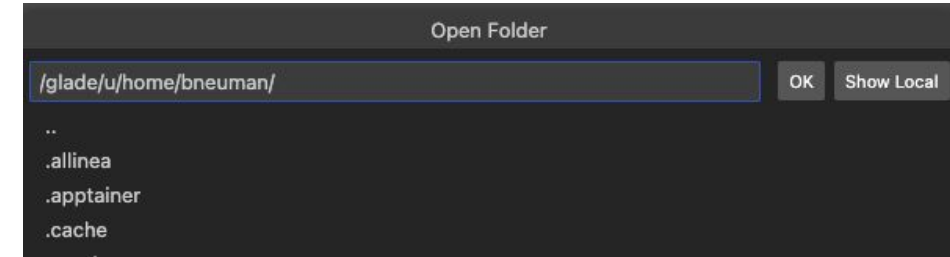
# GLADE and File Explorer

## Open Folder

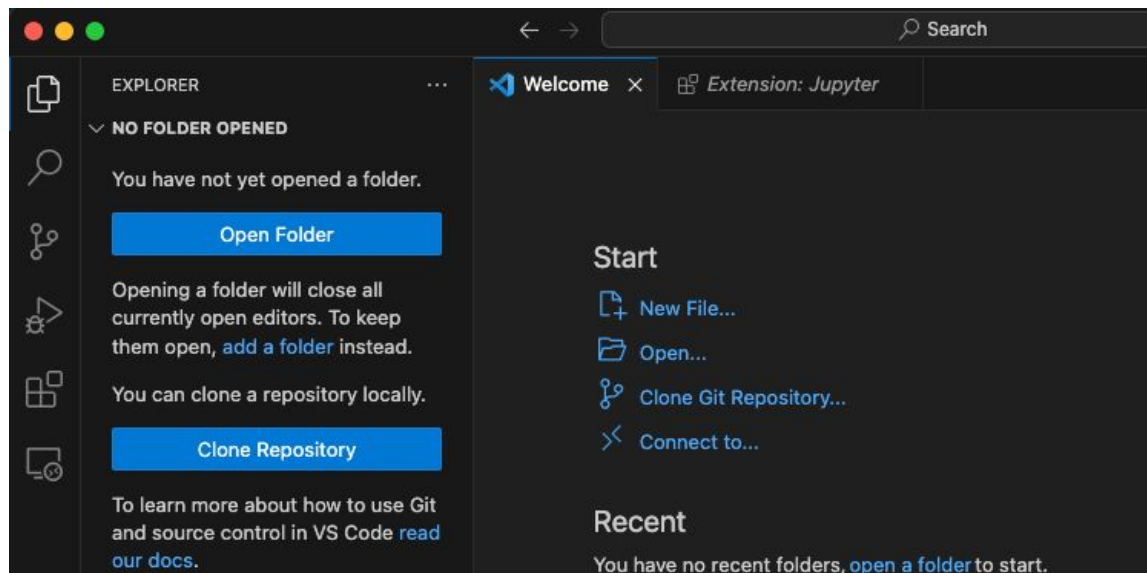
- Tries to open a new window

## Opening a folder generates a “Workspace”

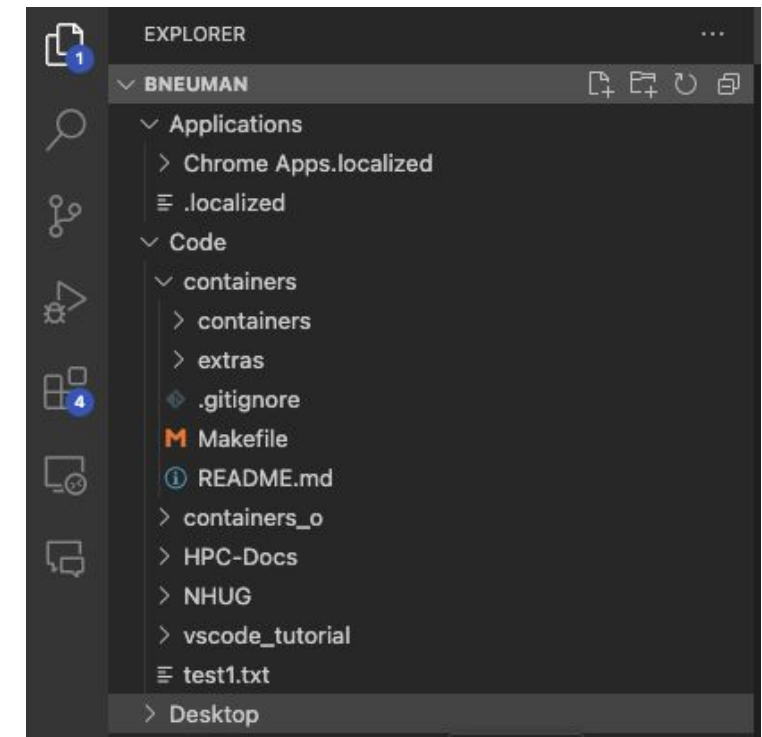
- VS Code scans this workspace for Intellisense and Copilot
- Be mindful of both CPU and memory utilization:
  - Intellisense will scan the entire workspace repeatedly
  - A larger workspace uses more memory on the login nodes



Prompt provides path to GLADE when connected to Derecho or Casper



Prompt to open a local or remote folder



Default File Explorer view



## Reduce memory and code scan delay

- Select a small folder to open
  - Limitations due to hidden folders
    - Usually there are repos
  - Control over what is visible in a workspace
  
- Modify the Settings file and exclude files matching patterns related to git, conda environments, and hidden files
  - *Code > Preferences > Settings*
  - Still scans the workspace

```
"files.watcherExclude": {
  "**/.git/objects/**": true,
  "**/.git/subtree-cache/**": true,
  "**/node_modules/**": true,
  "**/dist/**": true,
  "**/conda-envs/**": true
  "**/env/**": true,
},
"files.exclude": {
  "**/.git/objects/**": true,
  "**/node_modules/**": true,
},
"search.exclude": {
  "**/node_modules/**": true,
  "**/dist/**": true,
  "**/tmp/**": true,
  "**/.git/objects/**": true,
  "**/.git/subtree-cache/**": true,
  "**/.DS_Store": true,
  "**/.vscode": true,
  "**/__pycache__": true,
  "**/.pytest_cache": true,
},
```

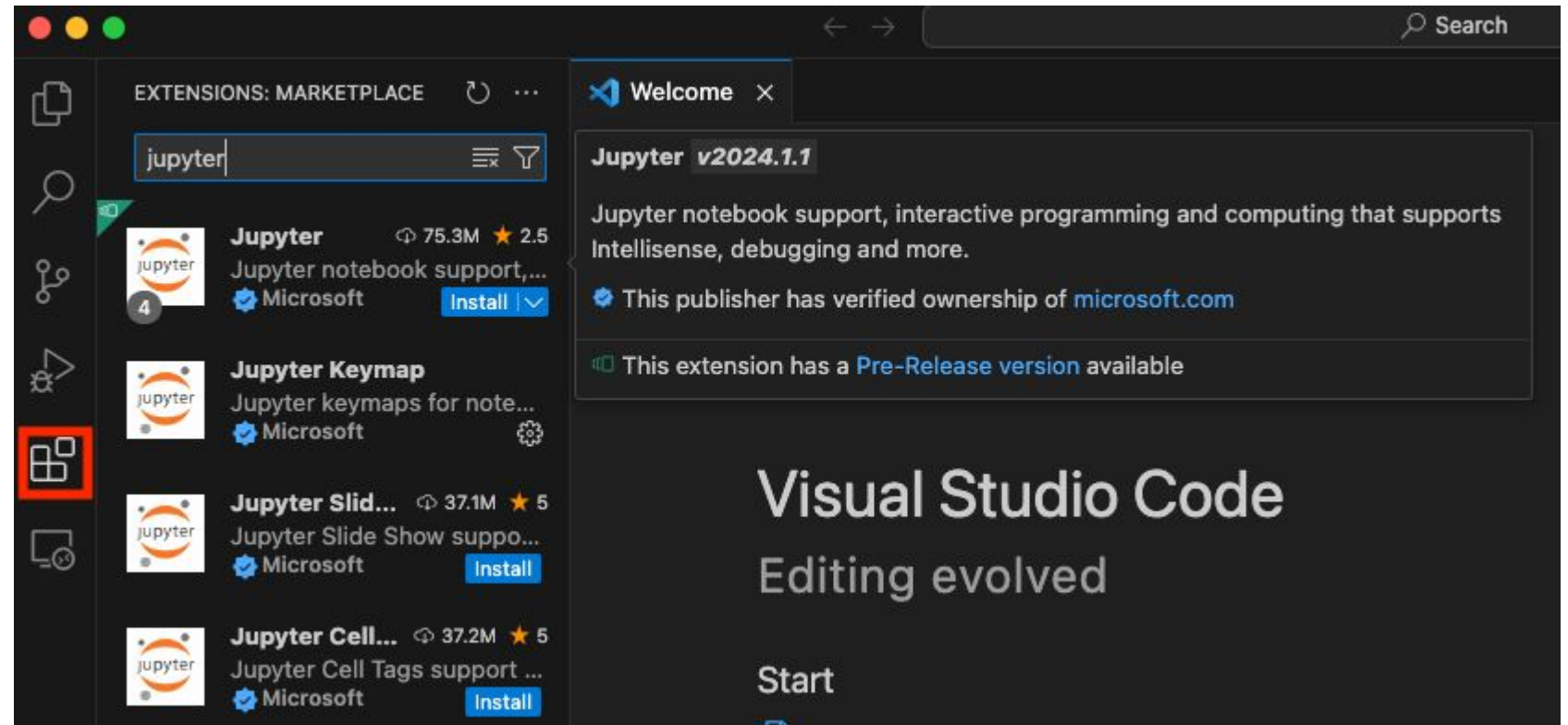
<https://ncar-hpc-docs.readthedocs.io/en/latest/environment-and-software/vscode/#file-explorer>

## Jupyter Extension

- Navigate to the Extensions tab
- Search for Jupyter
- To use custom conda environments to be available in VSCode, you must use the script:

```
create_conda_kernel
```

- This also makes the kernel available in Jupyterhub



# Jupyter Notebook Kernels

The screenshot displays the Jupyter Notebook interface. On the left, the Explorer panel shows a file tree for 'BNEUMAN [SSH: DERECHO.HPC.UCA]'. The main area shows a notebook cell with the following content:

```
Python GPU Session: CuPy and Legate  
By: Brett Neuman bneuman@ucar.edu, Consulting Services Group, CISL & NCAR  
Date: July 28th 2022  
Head to the NCAR JupyterHub portal and start a JupyterHub session on a Casper batch node (using 1 CPU, 1 GPU) and open the notebook in 12_CuPyAndLegate.ipynb. Be sure to clone (if needed) and update/pull the NCAR GPU_workshop directory.  
# Use the JupyterHub GitHub GUI on the left panel or the below shell commands  
git clone git@github.com:NCAR/GPU_workshop.git  
git pull
```

A 'Select a Python Environment' dialog box is open in the center, listing various environments. The 'npl (Python 3.10.12)' environment is selected and marked as 'Recommended'. A red box highlights the 'Select Kernel' button in the top right corner of the dialog.

## Kernel Select

- To select an environment, use the Python: Select Interpreter command from the Command Palette (`⇧⌘P`) or F1
- Kernel can also be selected with the 'Select Kernel' on an open notebook

# Connecting to Compute Nodes

## For large, interactive workloads you should connect to Casper compute nodes

- Steps to connect to a compute node interactive session:

1. Connect to a Casper login node using Remote-SSH
2. From the Terminal, launch an interactive job using `qsub -l`

```
qsub -I -A <project> -q casper -l select=1:ncpus=4:mpiprocs=4 -l  
walltime=01:00:00
```

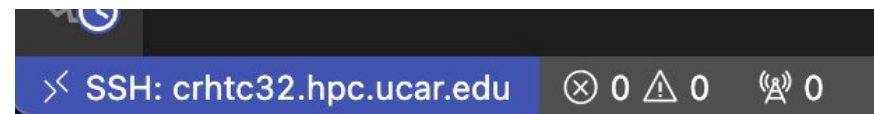
3. Identify the assigned compute node name:

```
$USER@crhtc32: ~>
```

```
echo $HOSTNAME
```

4. Launch a new window and connect directly to the compute node using Remote-SSH:

```
bneuman@crhtc32.hpc.ucar.edu
```



## Derecho will only run interactive workloads on login nodes

- Current security on Derecho prevents direct login to compute nodes from VSCode's RemoteSSH

# Copilot License

- **Join the NCAR GitHub Organization:** <https://github.com/NCAR>
- **Coordinate with your lab's budgeting for a project key (billing key) for charging CoPilot Charges (\$19/month).**
- **Submit a Help Request:**
  - **Create a request through the NCAR/UCAR Service Desk by emailing [help@ucar.edu](mailto:help@ucar.edu) or using the Service Desk portal.**
  - **Provide your GitHub ID and Project Key for charging CoPilot charges.**
- **Install the Github Copilot Extension in VSCode and login using your Github credentials**

**The Service Desk will process your request, confirm the billing information, and activate the license.**

**Note: NCAR only offers GitHub Copilot for Business, not the Enterprise tier.**



## Copilot Chat

- Menu pane -> Copilot
  - Opens default chat window
- Text based generation via prompts
- Scope is contained to the running chat
  - New chat can be started with the +
  - Move existing chat to a new window or editor:
- ... -> Open in New Window
  - New Window maintains scope and allows for new chat windows in parallel

The screenshot shows the VSCode Chat interface. The main chat window displays a conversation about the CESM (Community Earth System Model) directory structure. The chat content includes:

The `time/src` directory contains the main source code for the CESM (Community Earth System Model). Here are the main components typically found in this directory:

1. `drivers/`: Contains the driver code that coordinates the execution of different model components.
2. `externals/`: Contains external libraries and dependencies required by the CESM.
3. `share/`: Contains shared utilities and common code used across different components of the CESM.
4. `utils/`: Contains utility scripts and tools for managing and running the CESM.

Each of these components plays a crucial role in the functioning of the CESM:

- `drivers/`: Manages the interaction between different model components, ensuring they work together seamlessly.
- `externals/`: Provides necessary external dependencies and core CESM codebase.
- `share/`: Offers common functions and utilities used by various parts of the CESM.

What are the steps to execute the pbs...

Ask Copilot or type / for commands

A secondary chat window is open, showing a generated PBS script for launching an MPI job on NCAR's Casper cluster. The chat content includes:

Generating a PBS script for launching an MPI job on NCAR's Casper cluster


Explanation:

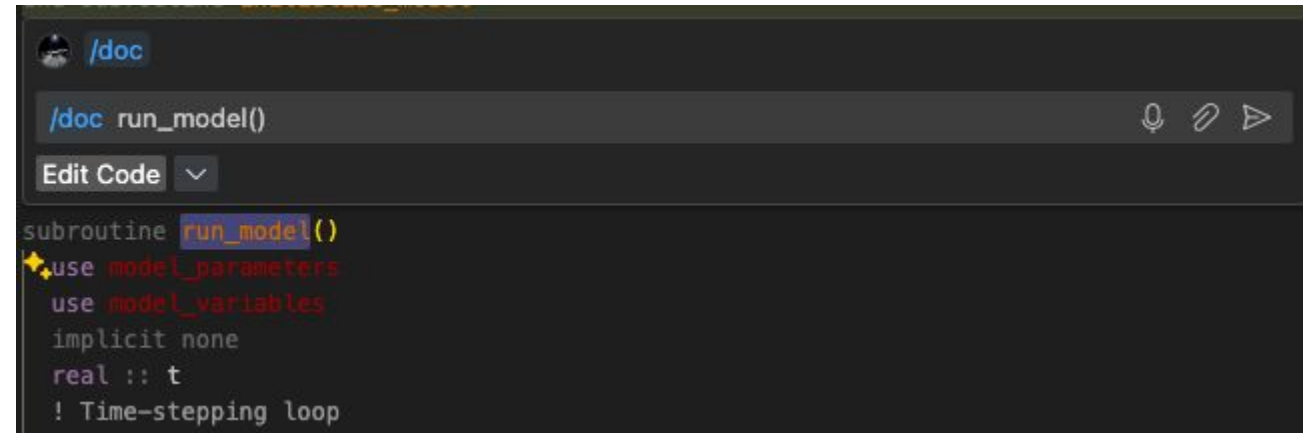
1. **PBS Directives:**
  - `#PBS -N my_mpi_job`: Sets the name of the job.
  - `#PBS -A your_project_code`: Specifies the project code for accounting.
  - `#PBS -l select=2:ncpus=36:mpiprocs=36:mem=109GB`: Requests 2 nodes, each with 36 CPUs and 109GB of memory.
  - `#PBS -l walltime=02:00:00`: Sets the wall time limit to 2 hours.
  - `#PBS -q regular`: Specifies the queue to submit the job to.
  - `#PBS -j oe`: Merges standard output and error files.
  - `#PBS -m abe`: Sends email notifications at the beginning, end, and in case of abortion.
  - `#PBS -M your_email@example.com`: Specifies the email address for notifications.
2. **Module Loading:**
  - Loads the necessary environment and MPI modules.
3. **Job Execution:**
  - Changes to the directory from which the job was submitted.





Ask Copilot or type / for commands

# Copilot Chat vs. Copilot In-context

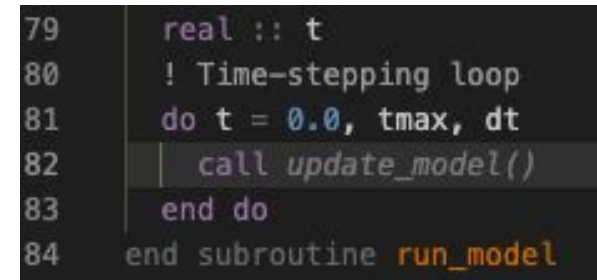
## Copilot prompt in your active window

- Open the Copilot prompt with  on your active preview window
  - Highlight sections of code to target or use Function names
- Copilot Autocomplete attempts to generate new code based on scope



```
 /doc  
 /doc run_model()     
 Edit Code   
 subroutine run_model()  
 *use model_parameters  
 use model_variables  
 implicit none  
 real :: t  
 ! Time-stepping loop
```

Copilot prompt opened in code editor



```
79     real :: t  
80     ! Time-stepping loop  
81     do t = 0.0, tmax, dt  
82         call update_model()  
83     end do  
84 end subroutine run_model
```

Copilot Autocomplete

## 2022 Paper “Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model” (Luccioni, Viguiet, Ligozat, <https://arxiv.org/pdf/2211.02001>)

- BLOOM:
  - 176 Billion parameters
  - Trained on 1.6 Petabytes of data

<b>Total training time</b>	118 days, 5 hours, 41 min
<b>Total number of GPU hours</b>	1,082,990 hours
<b>Total energy used</b>	433,196 kWh
<b>GPU models used</b>	Nvidia A100 80GB
<b>Carbon intensity of the energy grid</b>	57 gCO <sub>2</sub> eq/kWh

Table 1: Key statistics about BLOOM model training – for more details about our methodology, see Section 4.2.

Table from <https://arxiv.org/pdf/2211.02001>



# Inferring Energy Usage for Training

## Projected power and CO<sub>2</sub> emissions based on BLOOM

- Factors in three phases
  - Total power consumption for *Training*
  - CO<sub>2</sub> Emissions of *Dynamic Energy* during Training
  - CO<sub>2</sub> Emissions of *Dynamic Energy* and *Passive Datacenter Energy* during Training

<b>Model name</b>	<b>Number of parameters</b>	<b>Datacenter PUE</b>	<b>Carbon intensity of grid used</b>	<b>Power consumption</b>	<b>CO<sub>2</sub>eq emissions</b>	<b>CO<sub>2</sub>eq emissions × PUE</b>
GPT-3	175B	1.1	429 gCO <sub>2</sub> eq/kWh	1,287 MWh	<i>502 tonnes</i>	552 tonnes
Gopher	280B	1.08	330 gCO <sub>2</sub> eq/kWh	<i>1,066 MWh</i>	<i>352 tonnes</i>	380 tonnes
OPT	175B	<i>1.09</i> <sup>2</sup>	<i>231 gCO<sub>2</sub>eq/kWh</i>	<i>324 MWh</i>	70 tonnes	<i>76.3 tonnes</i> <sup>3</sup>
BLOOM	176B	1.2	57 gCO <sub>2</sub> eq/kWh	433 MWh	25 tonnes	30 tonnes

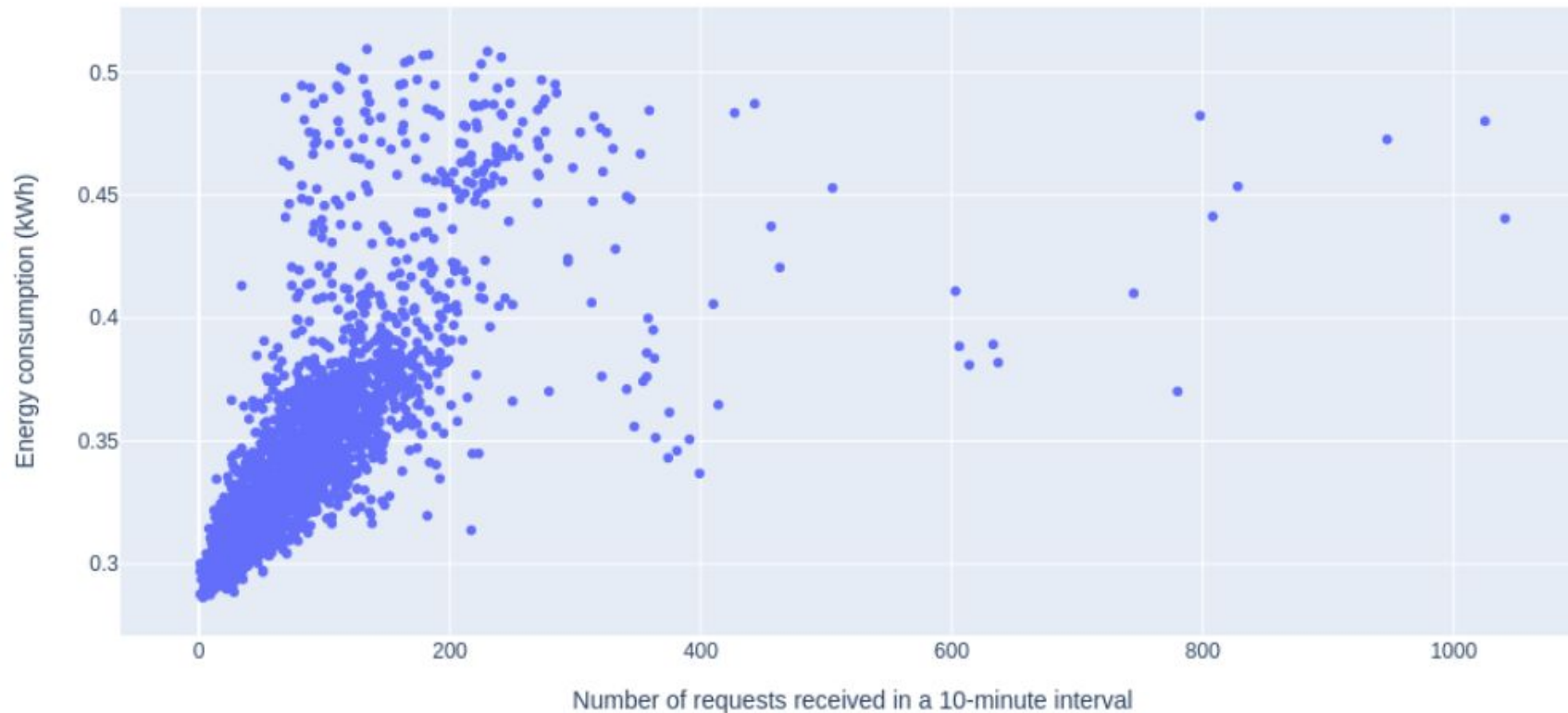
Table 4: Comparison of carbon emissions between BLOOM and similar LLMs. Numbers in *italics* have been inferred based on data provided in the papers describing the models.

Table from <https://arxiv.org/pdf/2211.02001>

# Energy Usage - Post Training

## Energy used per prompt estimations

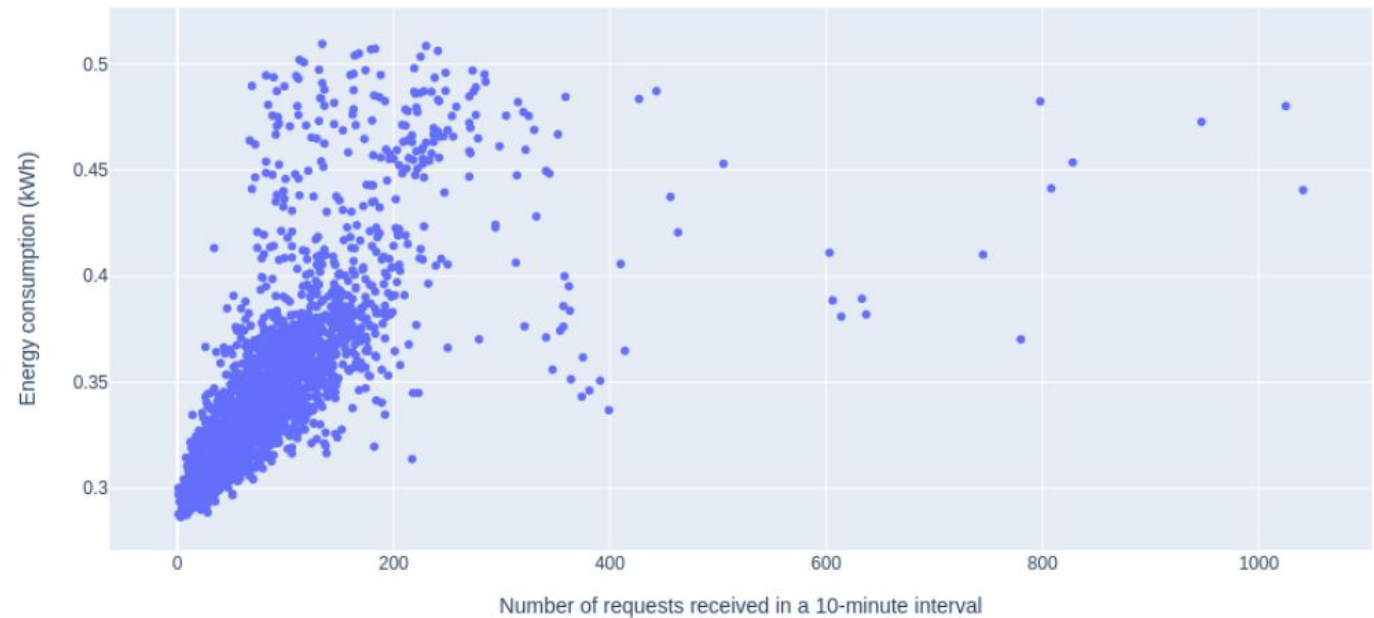
- ~1 request in a 10 minute window uses 0.30kWh of energy
  - Near idle amount of energy for running A100 GPU (idle measured at ~0.28kWh)
- Consumption energy mostly scales with requests
  - Occasional spikes in energy consumption



16x A100 GPUs running BLOOM,  
176B Parameter LLM

# Energy Usage - Post Training

Prompts	kWh	Light bulbs (LED 18Wh/bulb)
0	0.28	15.55
1	0.30	16.67
100	0.35	19.44
200	0.45	25.00
Prompts	kWh	Liters of water (0.21MJ/liter)
0	0.28	4.80
1	0.30	5.15
100	0.35	6.00
200	0.45	7.72



Gleick, P.H. and Cooley, H.S.  
“Energy implications of bottled  
water.” *Environmental Research  
Letters* 4 (2009) 014009

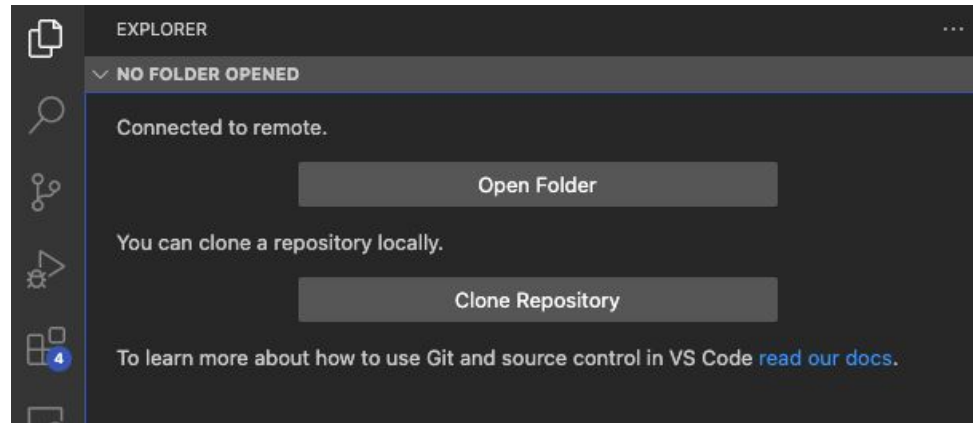
# Copilot Agents

## Prompts can be directed to produce better results:

- Using `@workspace` before you prompt gives more control on how the AI will process your prompt
- Available agents:
  - **@github**
    - Get answers grounded in web search and code search
    - Takes less time to produce results compared to `@workspace`
  - **@terminal**
    - Ask how to do something in the terminal
  - **@workspace**
    - Ask about your workspace
    - Can take significantly longer to produce results
      - More energy due to additional inference
      - Still sends your prompt to model with additional inputs from workspace
  - **@vscode**
    - Ask questions about VS Code

## Workspaces

- Copilot defaults to *only* using the open files in the preview window



No path open in File Explorer

```
glade > u > home > bneuman > copilotpy > $ pbs_modelrun.sh
1  #!/bin/bash
2  #PBS -N my_mpi_job
3  #PBS -A your_project_code
4  #PBS -l select=2:ncpus=128:mpiprocs=128:mem=512GB
5  #PBS -l walltime=02:00:00
```

Files open in preview window

@workspace what is my current workspace

### Workspace

Used 2 references

```
$ pbs_modelrun.sh:1-110 ~/copilotpy
≡ forecastmodel.f90:154-190 /glade/u/home/bneuman
```

Your current workspace includes a Fortran-based numerical model for simulating fluid dynamics using the Navier-Stokes equations. The workspace contains the following key components:

#### 1. Fortran Source Code:

- File: `forecastmodel.f90`
- Modules:
  - `model_parameters`: Defines model parameters such as grid dimensions, time step, maximum time, and viscosity.
  - `model_variables`: Defines model variables such as velocity components (`u`, `v`, `w`), their updated values (`u_new`, `v_new`, `w_new`), and pressure.
- Subroutines:
  - `initialize_model`: Initializes the model variables.
  - `run_model`: Runs the model by iterating over time steps and calling `update_model`.
  - `update_model`: Updates the model variables using the Navier-Stokes equations.
- Main Program:
  - `forecast_model`: Orchestrates the initialization and execution of the model.

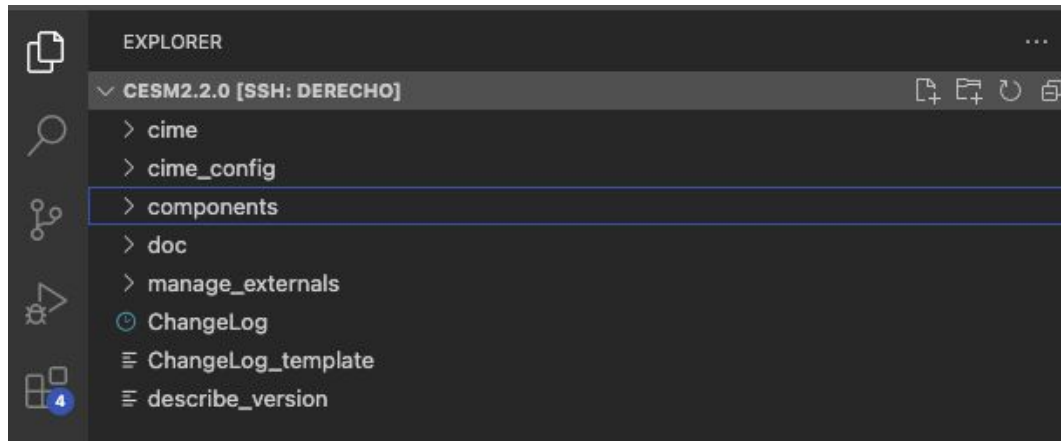
#### 2. PBS Job Script:

- File: `pbs_modelrun.sh`
- Content: Configures and submits an MPI job using PBS directives. It includes loading necessary modules, changing to the working directory, and running the MPI job with `mpiexec`.

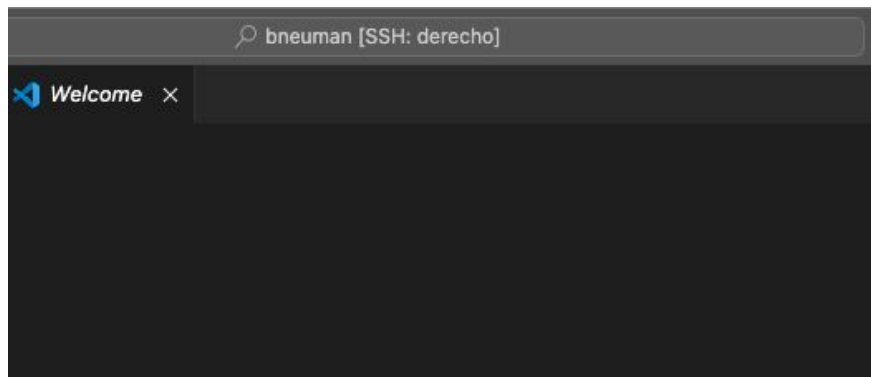
# Workspaces with File Explorer

## Workspaces with a path in File Explorer

- Searches entire workspace
- Prompt generation can take significantly longer



CESM path on GLADE open in File Explorer



No files open in preview window

@workspace what is my current workspace

```
</> Workspace
  Used 12 references
  include_user_mods:1 cime_config/testmods_dirs/allactive/cism/test_coupling
  shell_commands:1-2 cime_config/testmods_dirs/allactive/cism/test_coupling
  include_user_mods:1 cime_config/testmods_dirs/allactive/defaultio
  user_nl_cice:1-73 cime_config/testmods_dirs/allactive/defaultio
  user_nl_clm:1-4 cime_config/testmods_dirs/allactive/defaultio
  include_user_mods:1 cime_config/testmods_dirs/allactive/defaultiomi
```

Your current workspace appears to be a project related to CESM (Community Earth System Model) with a structure that includes various directories and files for configuration, documentation, and source code. The main directories and files include:

- `.github/`: Contains GitHub-related templates and configurations.
- `cime/`: Contains the core infrastructure for the model, including scripts, source code, and documentation.
- `cime_config/`: Contains configuration files and test modifications.
- `components/`: Likely contains different components of the CESM.
- `doc/`: Contains documentation files.
- `manage_externals/`: Contains tools and scripts for managing external dependencies.
- Various README and LICENSE files.

The workspace includes configuration files for Sphinx documentation, test modifications for different components, and scripts for managing external dependencies.

# Methods to Specify Prompt Inputs

## Additional methods to drill down on paths, files, or highlighted code

- **#editor**
  - The visible source code in the active editor
- **#file**
  - Specify a file
    - Explain `#file:src/main.cpp`
    - Still bound by the workspace
      - Won't open / analyze files outside of it
- **#selection**
  - Highlighted portion of an open file in the active window
    - Explain `#selection`
- **#terminalLastCommand**
- **#terminalSelection**

*Not all options are available for in-context code*

```
Explain the functions of the script
#file:/glade/u/home/bneuman/scripts/pbs_scripts/pbs_cross.sh

GitHub Copilot

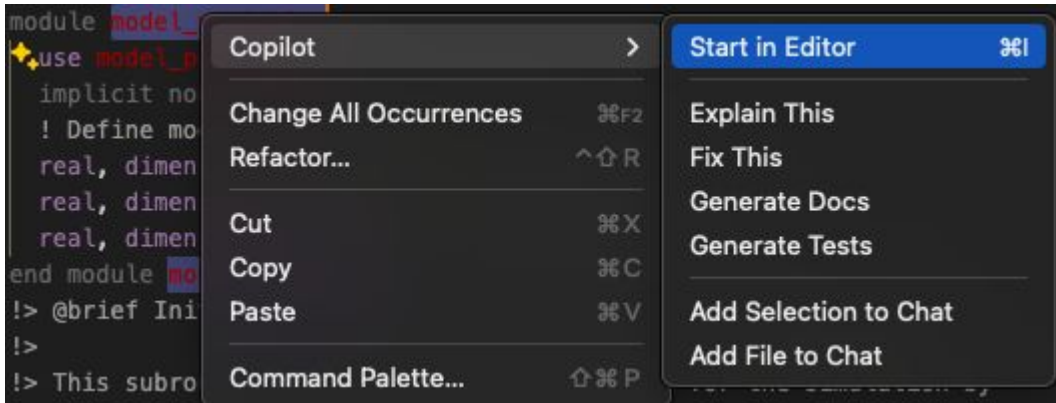
To explain the functions of the script located at
/glade/u/home/bneuman/scripts/pbs_scripts/pbs_cross.sh, I would need to
review the content of the script. Please open the script in your editor or provide the
content of the script so I can analyze and explain its functions.
```

Limitation when selecting a file outside of your workspace

```
glade > u > home > bneuman > forecastmodel.f90
1  module model_parameters
2  * implicit none
3  * ! Define model parameters here
4  * integer, parameter :: nx = 100, ny = 100, nz = 50
5  * real, parameter :: dt = 0.01, tmax = 10.0
6  * real, parameter :: viscosity = 0.001
7  end module model_parameters
8
9  module model_variables
10 use model_parameters
11 implicit none
```

Highlight a portion of the code to use in `#selection`

# Shortcut Behaviors



These Copilot shortcuts are available in-context by right clicking and selecting from the list

## Common tasks have shortcuts available

- **/explain**
  - (@workspace) Explain how the code in your active editor works
    - defaults to @workspace scope
  - /explain #file:src/main.cpp
- **/fix**
  - (@workspace) Propose a fix for the problems in the selected code
    - Highlight sections to fix
- **/doc**
  - Add documentation for a symbol
    - Works on highlighted text
  - /doc run\_model()
  - Specify documentation standards
    - /doc using doxygen comment standard
- **/tests**
  - Generates unit tests for highlighted code



# Combining Agents and Copilot Functions + Custom Agents

## Combining the three selections for a Prompt

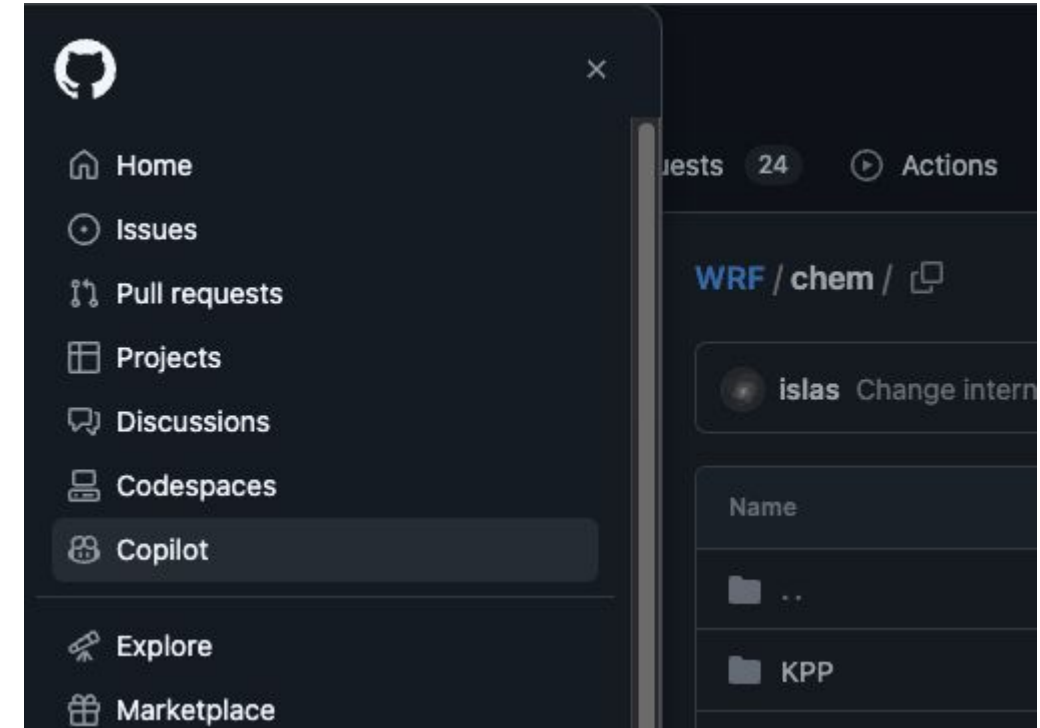
- Efficient way to prompt
- Provides more control over scope of data
  - `@workspace /explain #file:src/main.cpp`
  - `@workspace /fix #file:fibonacci.f90`
  - `@github /explain`

## Copilot directly through Github repository

- Menu -> Copilot -> Search for a repository
  - Limited unless repository is indexed
    - Limit to number of indexed repositories

## Custom Agents

- It is possible to create custom agents and



Menu from <https://www.github.com>  
Select Copilot to search for a repository  
Generate a chat window for the selected repository

# Discussion on Prompt Output

## Running quite a few example prompts

- Based on *my experience with Copilot*
  - .. and a few similar reports of generating code

## Things Copilot seems to be pretty good at:

- Scripting
  - bash, Python, etc.
- Code documentation
- Generating plots from data in scope
- Generalized code and functions
  - Webapps, Database operations, Fibonacci Sequence
- Estimating memory, MPI ranks, grid sizes
  - With user provided constraints

To determine the maximum domain sizes that can be used with 470GB of memory, we need to calculate the memory usage of the arrays in the `forecastmodel.f90` code and then find the maximum dimensions that fit within the available memory.

### Memory Calculation

#### 1. Array Dimensions:

- `nx`, `ny`, `nz`: Domain sizes to be determined.

#### 2. Number of Arrays:

- There are 7 arrays: `u`, `v`, `w`, `u_new`, `v_new`, `w_new`, `pressure`.

#### 3. Memory per Element:

- Assuming `real` is 4 bytes (single precision).

#### 4. Total Memory Usage:

- Memory per array: `nx * ny * nz * 4 bytes`
- Total memory for all arrays: `7 * nx * ny * nz * 4 bytes`

#### 5. Available Memory:

- 470GB =  $470 * 1024^3$  bytes = 504658657280 bytes

### Summary

With 470GB of memory, you can use domain sizes such as:

- `nx = ny = nz ≈ 2610`
- `nx = 1000, ny = 1000, nz ≈ 18023`

These are just examples, and you can adjust the dimensions as needed, ensuring that the product of `nx * ny * nz` does not exceed 18023594800.

“What domain sizes can be used if I only have 470GB of memory?”

# Discussion on Prompt Output - Pt 2

```
Could you generate a PBS script to launch a MPI job on NCAR's Casper cluster?
```

GitHub Copilot

```
Sure, here's a PBS script to launch an MPI job on NCAR's Casper cluster:
```

```
#!/bin/bash
#PBS -N my_mpi_job
#PBS -A your_project_code
#PBS -l select=2:ncpus=36:mpiprocs=36:mem=109GB
#PBS -l walltime=02:00:00
#PBS -q regular
#PBS -j oe
#PBS -m abe
#PBS -M your_email@example.com

# Load necessary modules
module load ncarenv
module load intel
module load mpt

# Change to the directory from which the job was submitted
cd $PBS_0_WORKDIR

# Run the MPI job
mpiexec -n 72 ./your_mpi_executable
```

This pulls information from creating a PBS script for Cheyenne

## Running quite a few example prompts

- Based on *my experience with Copilot*
  - .. and a few similar reports of generating code

## Things Copilot seems it could work on improving:

- Dealing with large code bases
  - Will generate code in a function but never calls it, requires additional guidance
- Dealing with complex scientific HW/SW solutions
  - Requires many additional input constraints to provide non-trivial solutions
- Generating NCAR specific outputs
  - nco, PBS job scripts, etc.
  - @workspace of documentation does help with this