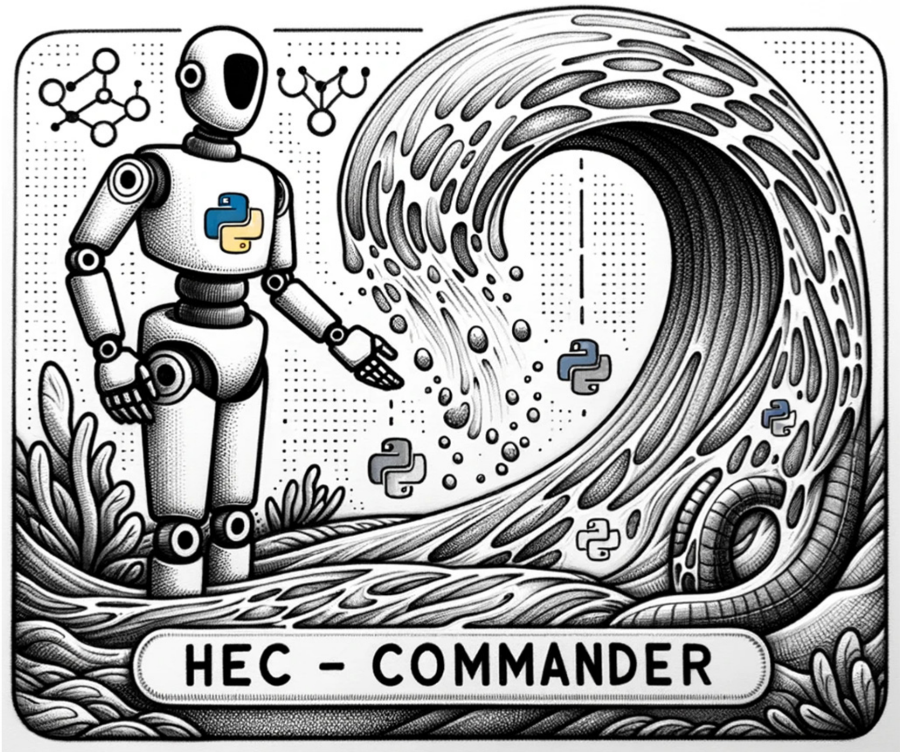
Leveraging AI-Assisted Scripting For HEC-RAS and HEC-HMS Automation

An Exploration of the Development of HEC-Commander Tools



William Katzenmeyer, P.E., C.F.M.
Senior Water Resources Technical Lead
C.H. Fenstermaker and Associates, LLC

ASFPM 2024 Conference
Thursday June 27, 2:00PM
Concurrent Session G9
10:30AM – 12:00PM

Outline

1. HEC-Commander Tools Intro → Where's the AI?
2. Why Parallelize → Benchmarking HEC-RAS Core Scaling
   - HEC-RAS 2D CPU Core Scaling
   - Platform Comparisons: Cloud, Laptop, Workstation and HPC
   - Parallelization in Practice
3. Best Practices for AI-Assisted Python Scripting
   - Code-Forward Approach
   - Notebook-style, Code Cell Level Modularity
   - AI Automated Environment and Dependency Setup
   - AI is Lowering Barriers to Adoption
4. AI-Assisted Coding: Lowering Barrier to Entry for Modeling Workflows
5. Prompting Examples and Strategies
6. Case Study: West Fork Calcasieu Model for Louisiana Watershed Initiative

William Katzenmeyer
Linkedin

HEC-Commander
Repository (GitHub)

AI-Coded Jupyter Notebook Supporting:

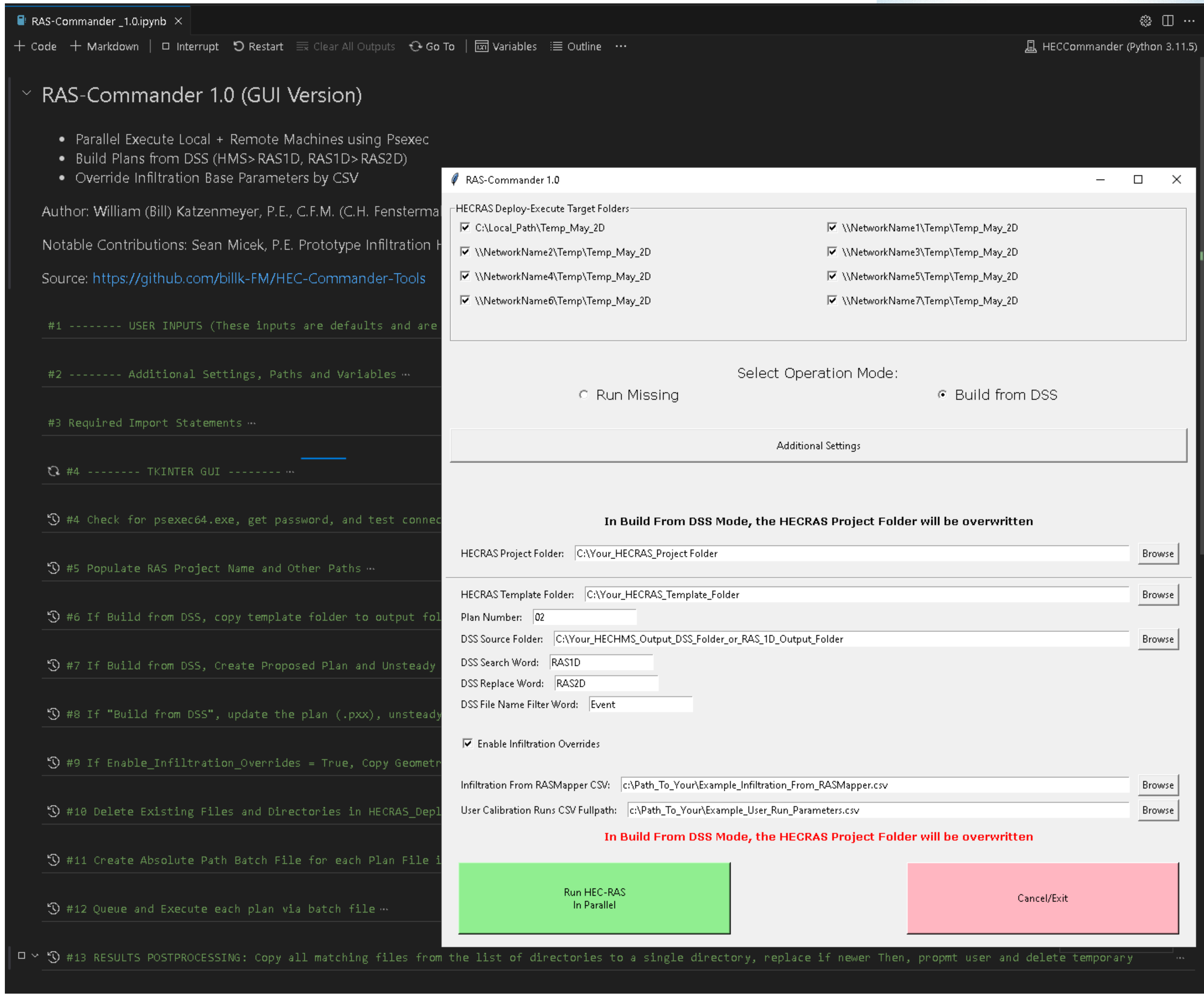
- Parallel HEC-RAS Execution
- Windows Native: Supports All Versions
- Leverage Multiple Workstations in Parallel
- Open Source, MIT License

RAS-Commander

Parallelizing HEC-RAS
In a Jupyter Notebook

Basic Components

- User Input and Settings
- New! Tkinter GUI
- File Deploy and Copy
- Batch File Creation
- Command Line Execution
- Results Collection



Flexible Operation:

- Bring Your Own Project
- Create Plans from HMS DSS Input Files
- Optional 2D Infiltration Overrides

RAS-Commander is ready for AI-Assisted editing to support your bespoke applications.

HMS-Commander

AI-Coded Jupyter Notebook Supporting

- Subbasin Parameter Editing
- DSS Output File Renaming
- Impervious Grid Scaling > 1.0
- Calibration Regions by shapefile
- CSV File Input
- Enables Linked HMS>RAS Calibration Workflows
- Modular Script Ready for AI-Editing for Bespoke Applications

Example CSV Input used for HMS-Commander and RAS-Commander 2D Infiltration Overrides:

user_run_number_from_csv	initial_deficit_scale	maximum_deficit_scale	percolation_rate_scale	impervious_area_scale	recession_factor	initial_flow_area_ratio	threshold_flow_to_peak_ratio	time_of_concentration_scale	storage_coefficient_scale
1	1.0	1	0.06	1	0.1	1	0.1	1	1
2	0.9	1	0.06	1	0.1	1	0.1	1	1

Where’s the AI?

A few clarifications on how we are using AI:

- AI is not operating the model
- AI isn’t making decisions or optimizing anything directly

Here is the secret sauce:

1. AI was used to write python code into notebooks, starting from plain language descriptions.
2. AI was also used to explain code segments it had written, to better teach me how to direct it with English commands to create the desired code output and functionality
3. As someone with very little prior python experience, I was able to generate useful code and innovative workflows that immediately unlocked innovations that were previously unreachable.

The most powerful capability of Large Language Model is the ability to speak multiple languages fluently. Especially deterministic languages like code.

Why Parallelize:

The Bitter Lesson by Richard Sutton emphasized:

Avoiding The Bitter Lesson in HEC-RAS Modeling

Breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning

By enabling parallel operation in support of wide parameter search operations, RAS-Commander represents an opposing approach for calibration workflows, based on efficiently scaling computation by search rather than relying on heuristic methods.

Benchmarking HEC-RAS Core Scaling

Cloud Systems

Midrange Workstations

Midrange Laptops

Local HPC Systems

Benchmarking was collected for all major CPU platforms (circa 2023) to explore core scaling behavior and identify chipsets and platforms offering best overall performance

2D HEC-RAS Performance Scaling

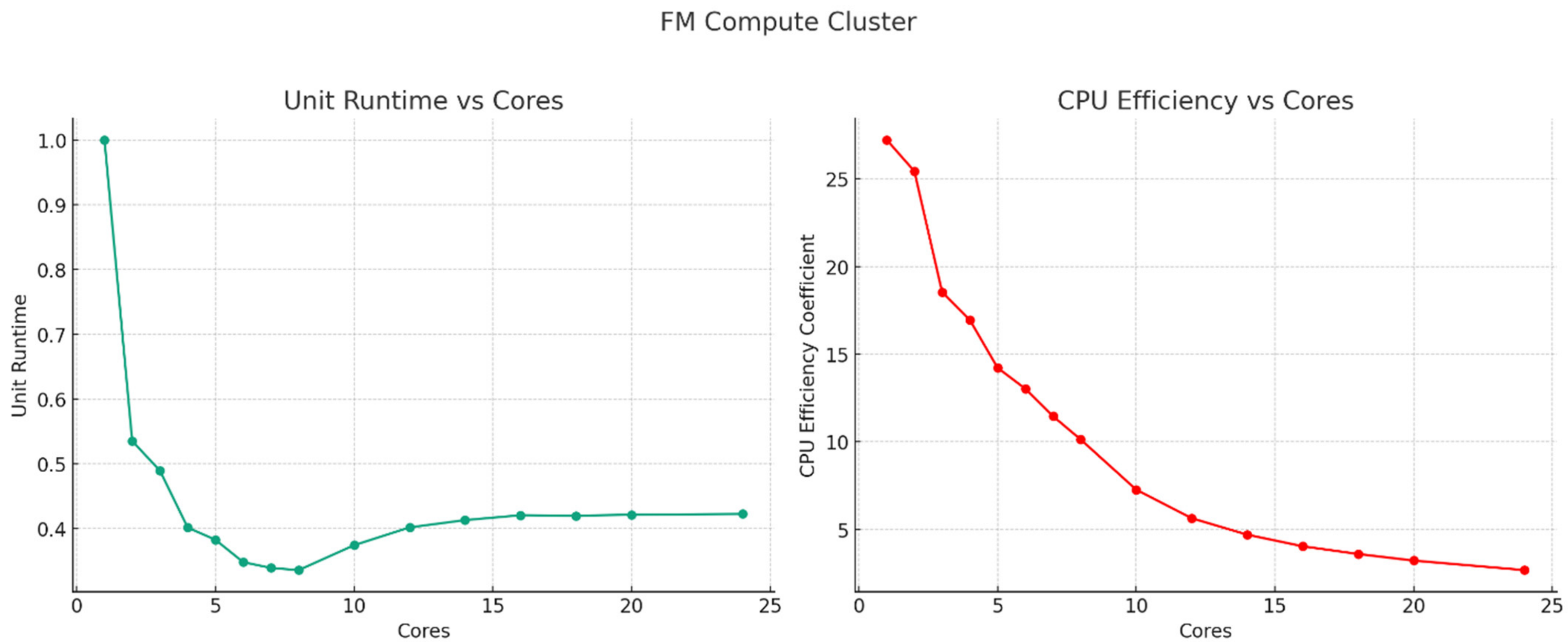
Fenstermaker Local Compute Cluster: Benchmarking Insights

1 Unit Runtime = 1 Day

To simplify comparisons

Best Value =

0.31 @ 8 cores



Scaling is linear with clock speed, but efficiency drops significantly beyond 2 cores

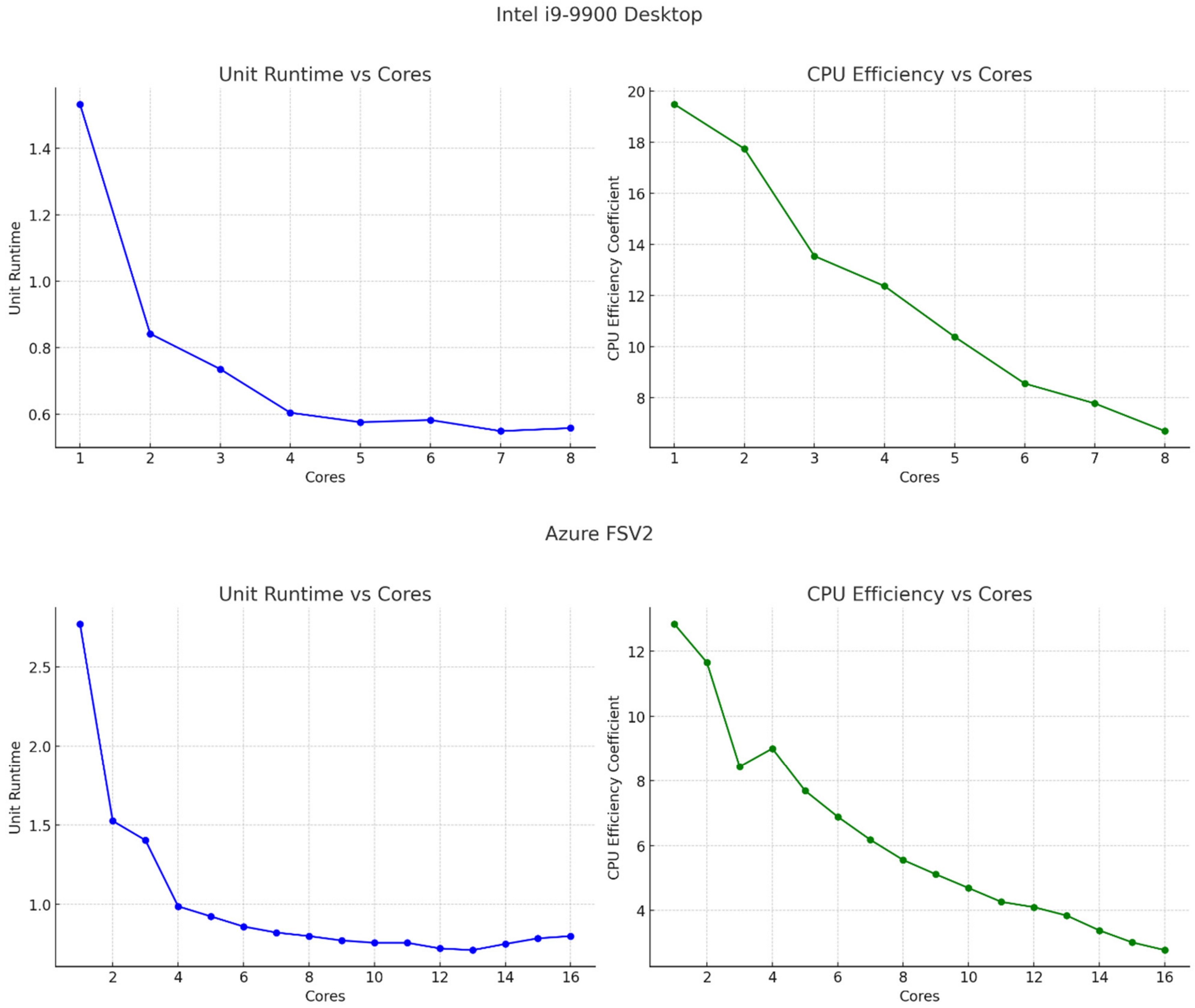
Midrange Desktop vs Public Cloud

- Midrange desktops (i9) outperform typical public cloud platforms on Windows

- Massively inefficient core scaling on the cloud

- Optimizing efficiency can help manage cloud costs

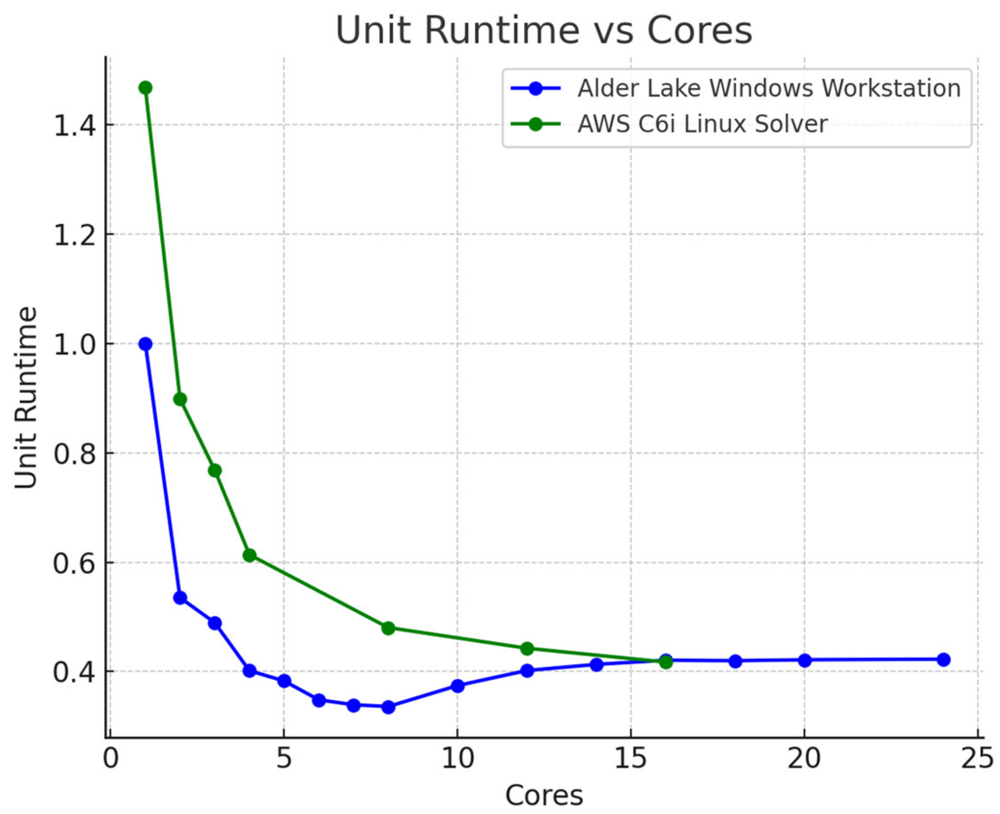
Costs are still higher in the cloud

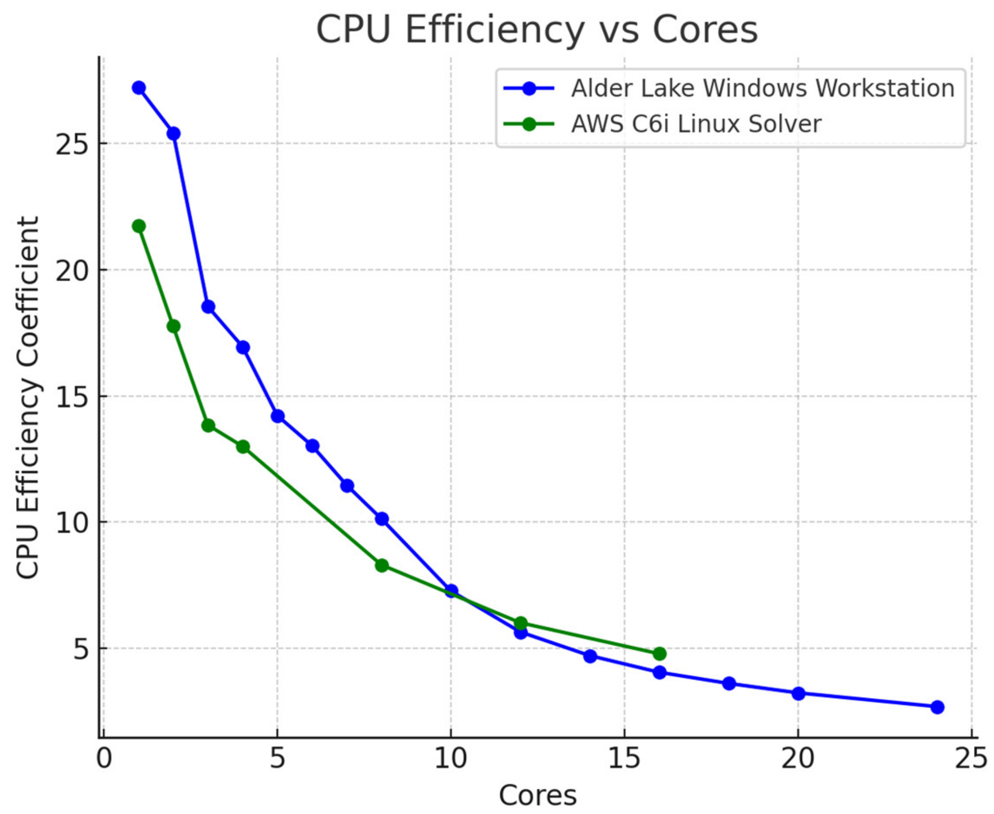


Comparison: Local Compute vs Best Public Cloud

<u>Chipset/Generation:</u>
Intel Alder Lake

Server-class architectures do not have "performance" and "efficiency" cores and can scale up to 16 cores on AWS C6i. Larger instances hurt performance.



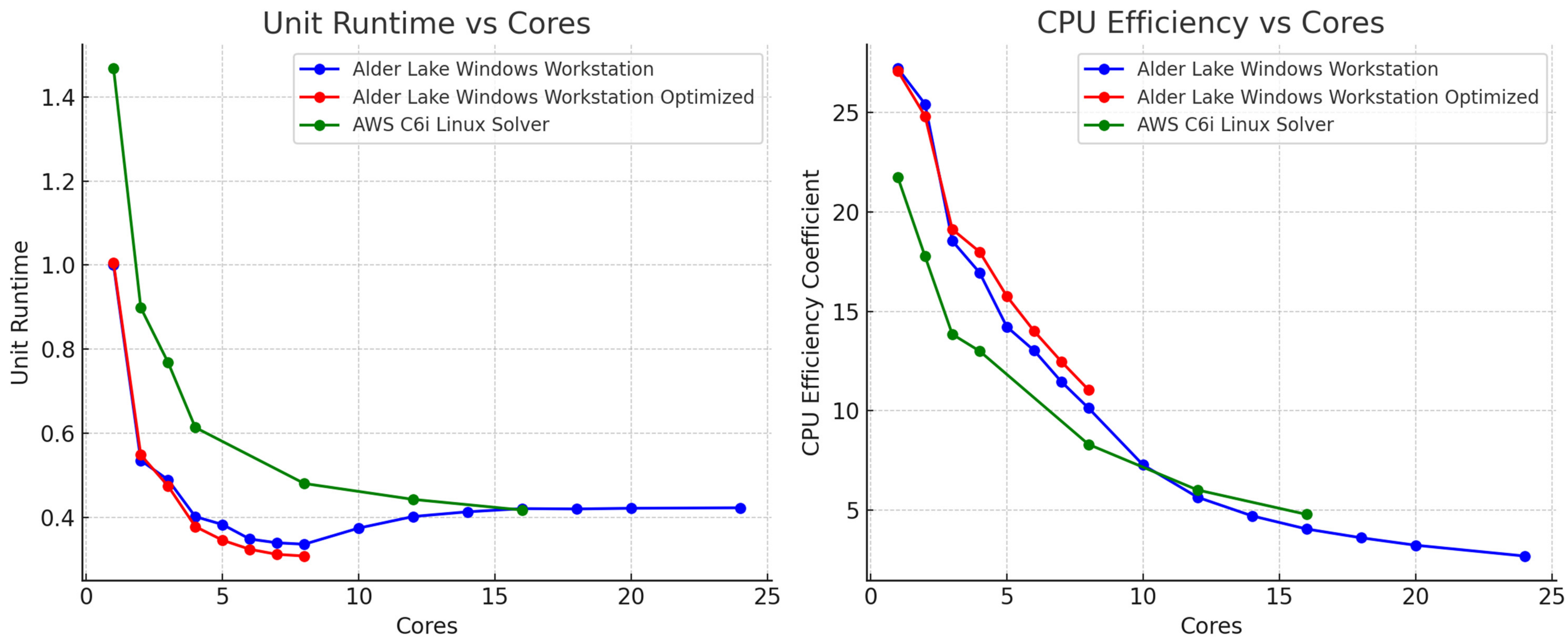


Cloud-scale architectures do successfully scale beyond 8 cores, but have similar efficiency characteristics

Cloud instances are relatively cheap, but utilizing it effectively is needlessly complex. Local

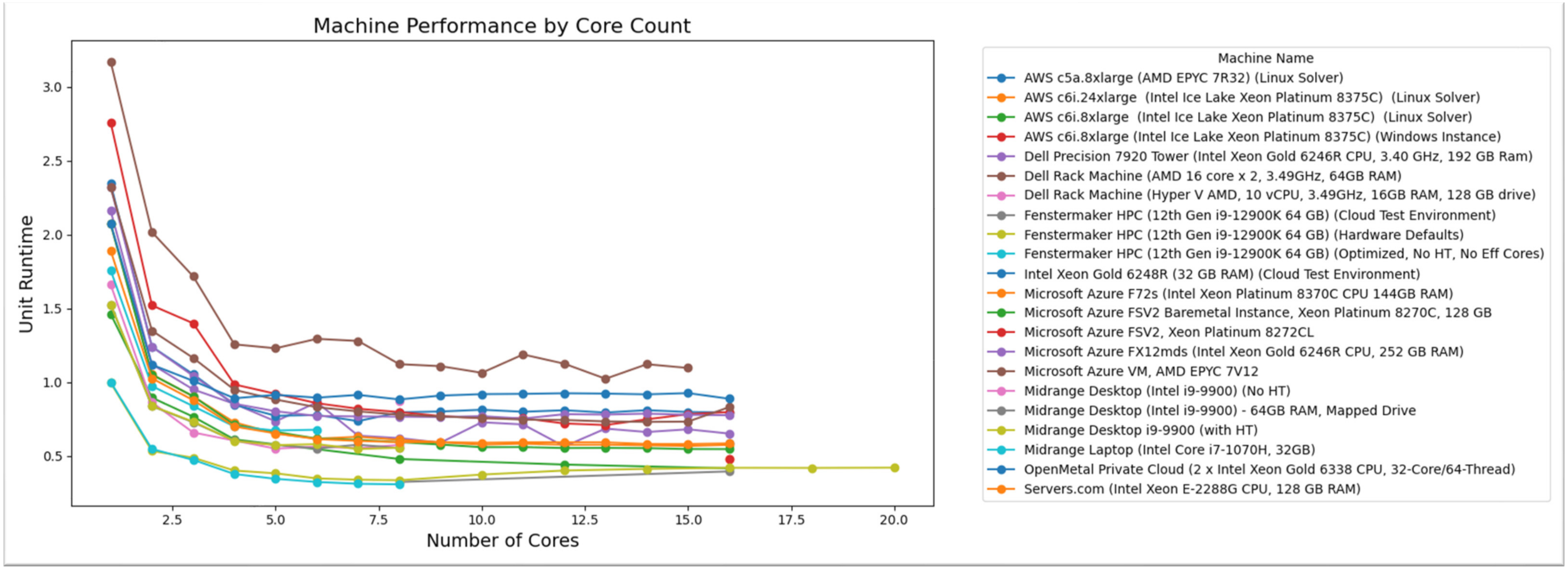
Optimize CPU Settings

Disable Hyperthreading *Disable "Efficiency" cores* *Install Intel XTU Tuning*



Take the free 10%

Composite Benchmarking Results



All Results are Recorded on the HEC-Commander GitHub Repo

- Benchmarking results as CSV
- Markdown files containing datasets and plots
- Includes AI-generated python code

Source Blog Post:
Benchmarking is All You Need

Drag-Drop this file into your favorite LLM, tell it your CPU and ask it for upgrade options

Parallelization In Practice

Giving 70% to Gain 70%

For an assumed 1 day runtime at 1 core

2 Cores = 0.55 days

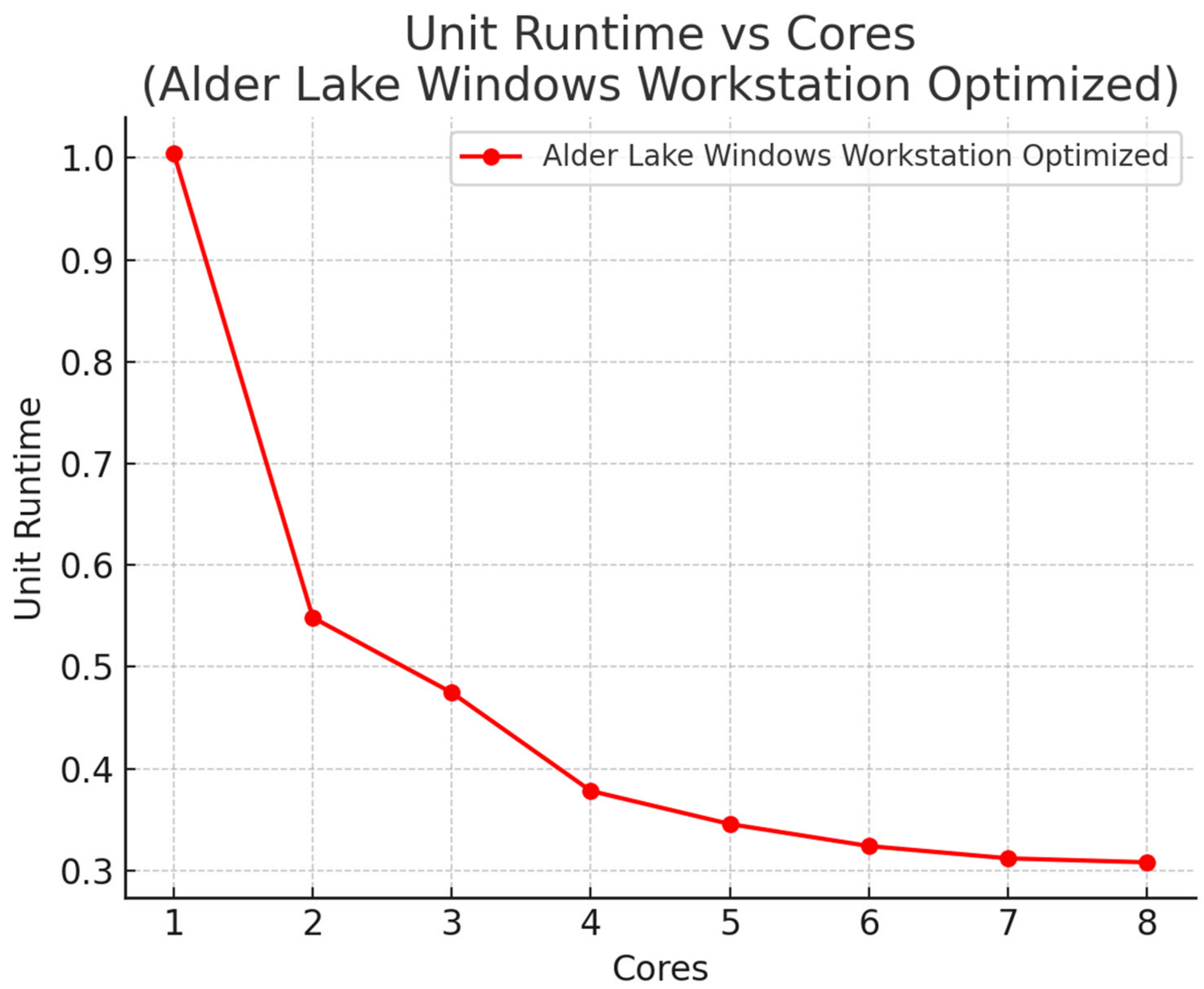
8 cores = 0.31 days

Without parallelization, running at 2 cores is around 77% slower

With Parallelization utilizing all 8 cores

3 run batches @ 2 cores = 0.55 Days

3 runs at 8 cores = 0.93 days



With parallelization, batched runs sets are 72% faster w/same CPU by maximizing efficiency

But How Do We Parallelize

HECRASController

- Lack of Documentation
- Limited to COM32 Interface
- No RASMapper Automation
- No Parallel Execution

Market Solutions

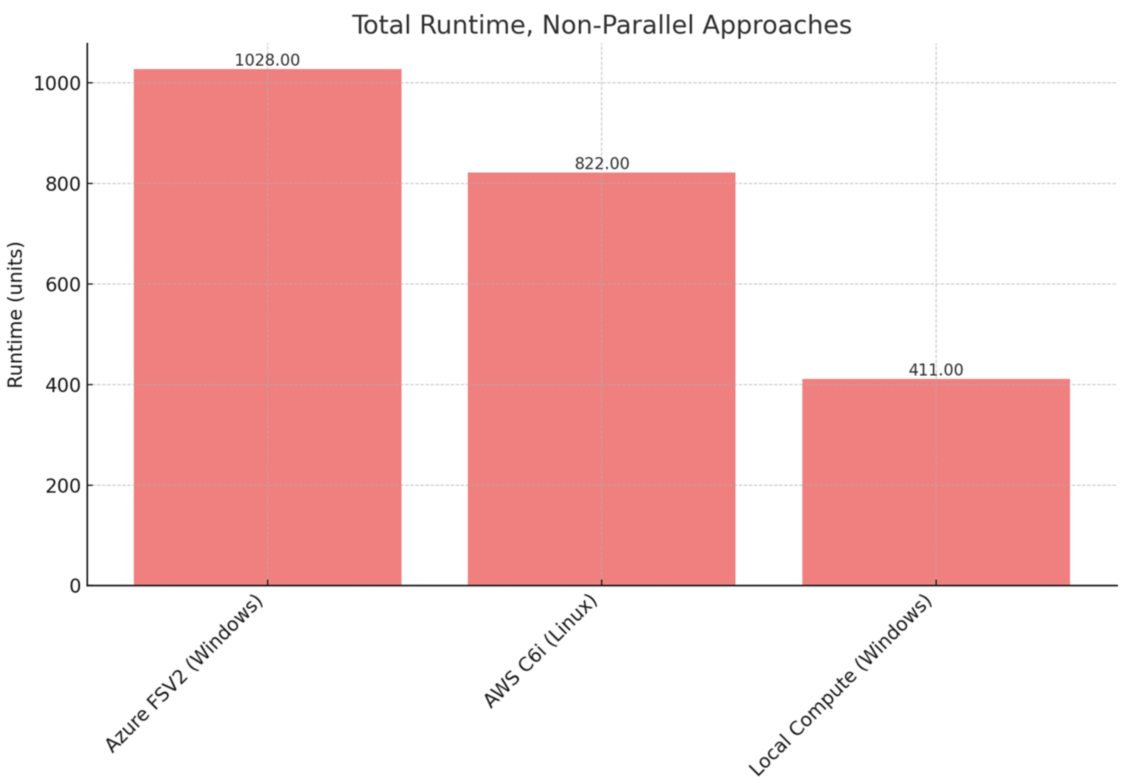
- All Built on Linux/Cloud
- No access to latest versions
- Proprietary
- *Not Free*
- Data Transfer Bottlenecks

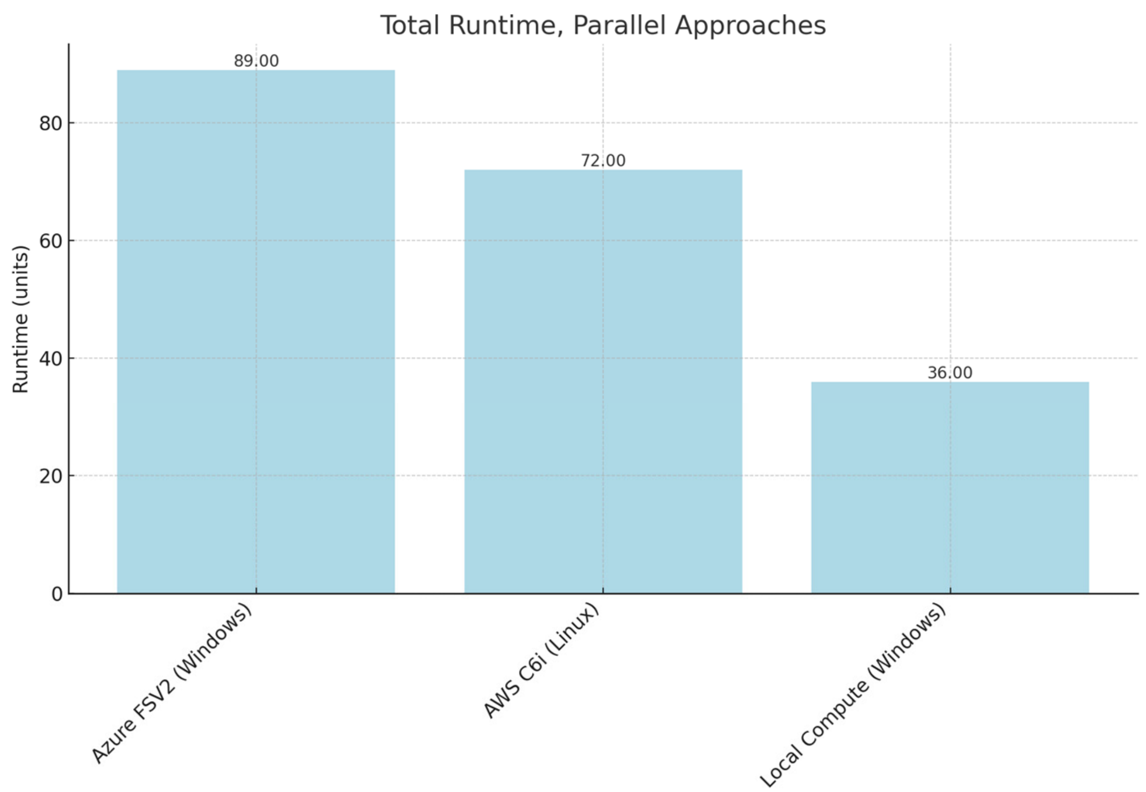
A Better Solution was Needed

So I coded it myself with AI

Case Study: West Fork Calcasieu Model
Louisiana Watershed Initiative

AI Authored Scripts gave us an over 10x boost in effective throughput:





10x Engineering: By The Numbers

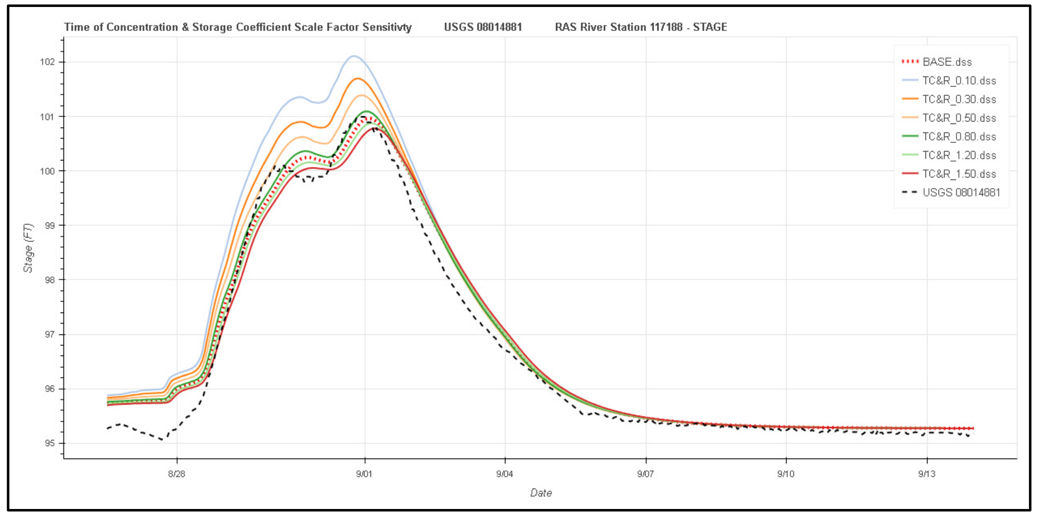
A focus on hardware selection and effective utilization of idle compute capacity for low-cost horizontal scaling yielded very good results throughout Region 4's efforts

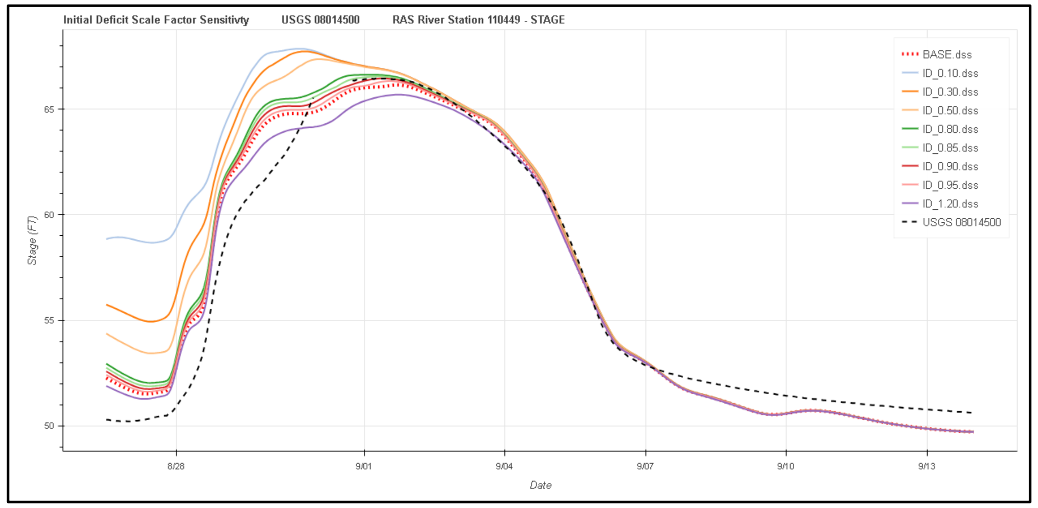
Case Study: West Fork Calcasieu Model

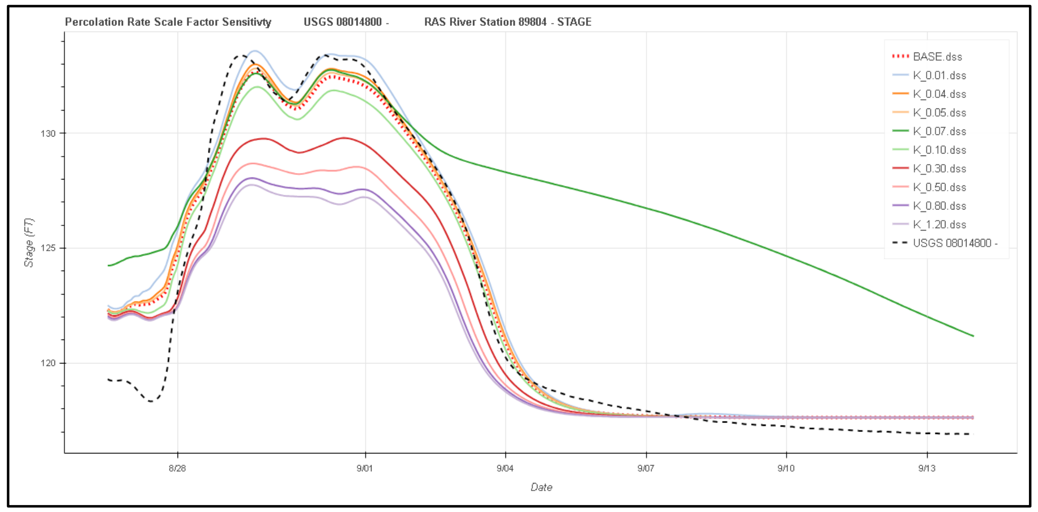
Region 4, Louisiana Watershed Initiative

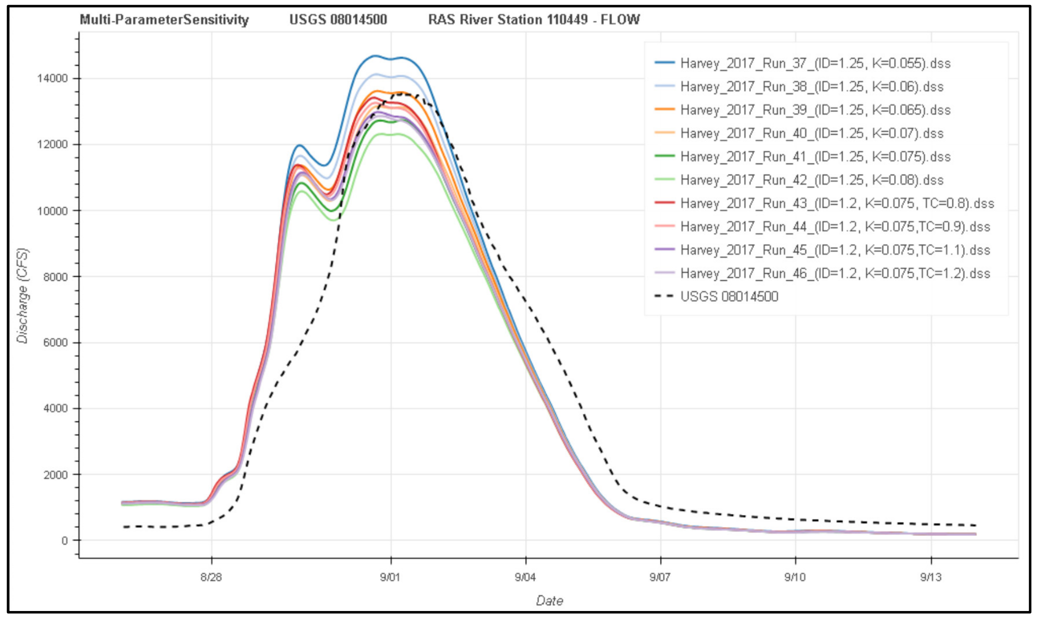
LOUISIANA
WATERSHED
INITIATIVE
working together for sustainability and resilience

Leveraging and order of magnitude more compute and data allowed innovative calibration and validation approaches with more deterministic results.









Revisiting The Bitter Lesson:

"Breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning"

R.H. Sutton

The future is parallelization! You can implement your own innovative tools today with lower barriers to entry than ever before by leveraging LLM's.

Stacking Gains

From

to

Blog Post:
From 10x to 0.25x
By the Numbers

Avoiding Public Cloud: 35% to 240% gains

- Depending on instance, provider
- Linux instances consistently performed ~15% faster than Windows

Maximizing Single Core Performance for Model Development

- Disabling Hyperthreading: +10%
- Disabling Efficiency Cores: Avoiding 5-25% Performance Penalty

Maximizing Efficiency w/Batched Calibration/Validation, Parallelization

- Compute batches 70% faster on single machine
- Additional Linear Scaling through Remote Execution (**+100% Per Machine**)
- Roughness: ~50 runs, HMS Sensitivity/Calibration 96 Runs per set
- Tested with 12 total compute nodes, 48 parallel runs

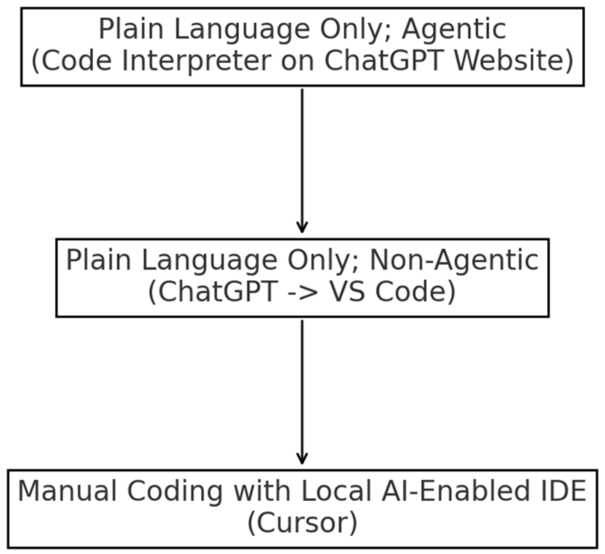
We will come back to this again in the Case Study at the end of the presentation.

Back to AI:
What does the Future Look Like?

- Innovative new tools developed even closer to the technical experts
- Drastically lowered barriers to writing and executing code
- More automated data analysis methods
- Less technical drudgery
- More focus on higher-level planning and thinking
- Ability to quickly innovate around commercial software limitations

AI-Assisted Python Scripting:

3 Basic Levels of AI Interaction with Python Code:



Each level of interaction drastically shortens the learning curve into the next.

ChatGPT's Code Interpreter is a wrapper for a Jupyter server with a sandboxed Python Environment. This generates a plethora of training data, making Python the most efficient language that GPT is most proficient at utilizing. By observing the python code generated in code interpreter, the user becomes familiar with the libraries and methods utilized by code interpreter, and becomes better able to direct the AI to create the desired output.

AI-Assisted Python Scripting:
Lowering Barriers to Increase Adoption

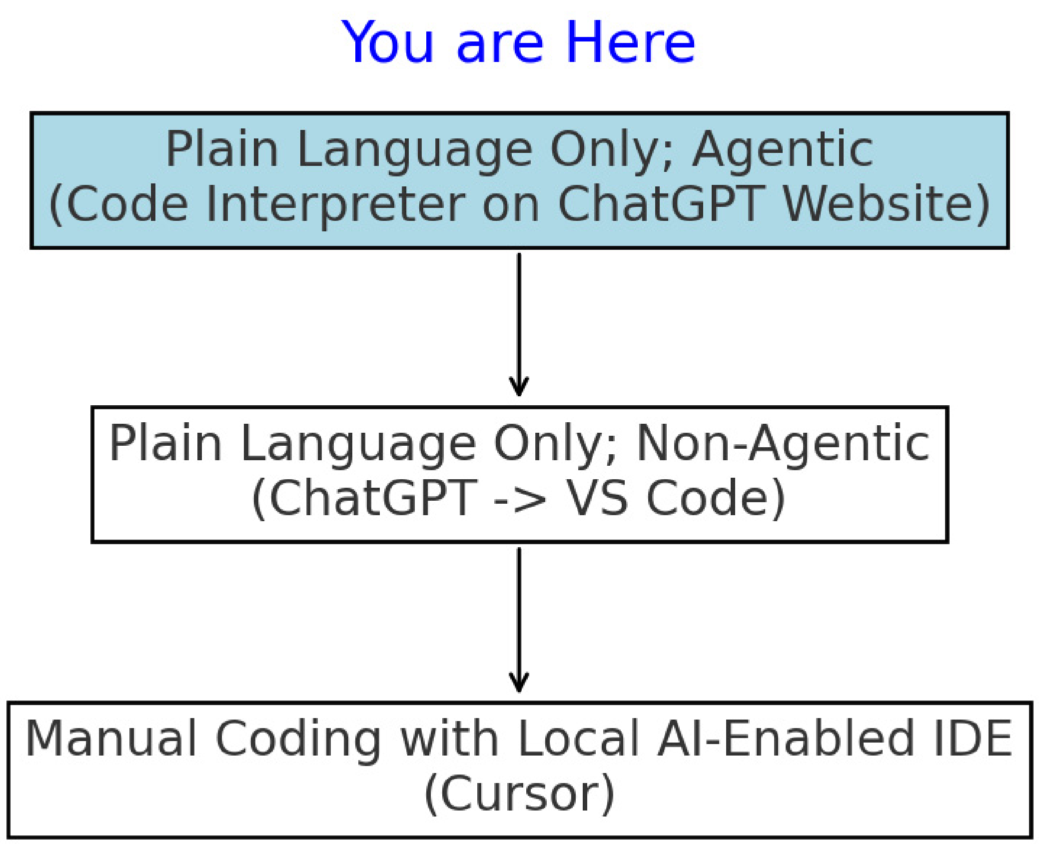
AI can assist beginner users with:

- Utilizing code autonomously with Code Interpreter
- Explaining Code Segments
- Step by step instructions on Development Environment Setup
- Assembling bespoke workflows from natural language
- Utilizing a vast array of open source python packages

This drastically lowers barriers to adoption and utilization python code, and opens a new frontier of development of innovative software tools.

Prompting Examples and Strategies

Let's explore what you can do at the first level of interaction:



- Intelligent Voice Notes and Dictations
- Office Application Assistant
  "Help me with VBA Scripting in Excel"
- Expert Software Assistants
  "Write a QGIS script to do this"
- Powerful Agentic Calculations
  "Fit a log-normal distribution to the data and calculate return periods"

Code Interpreter can handle:

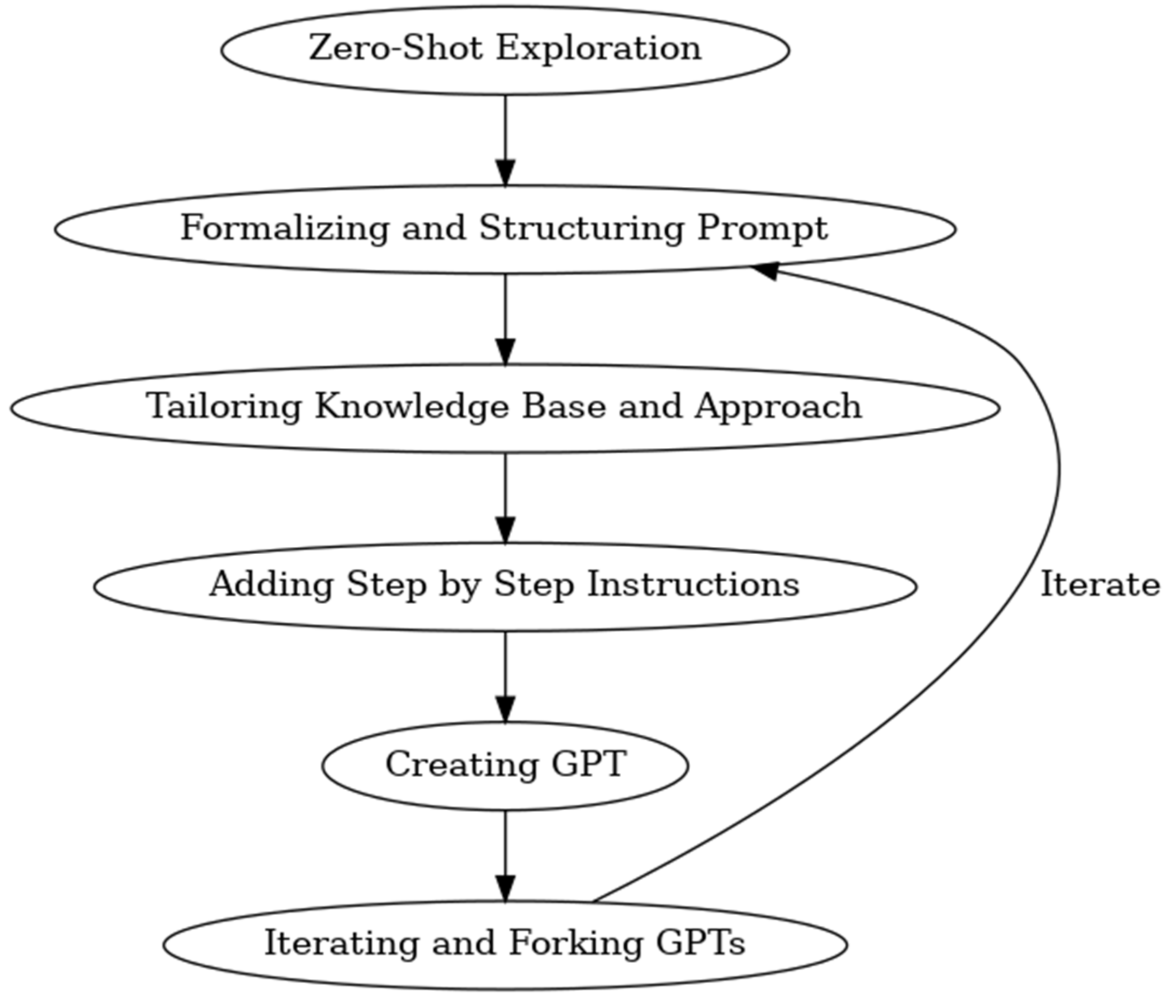
- Basic GIS Operations
- Sort/Filter/Display Datasets.
- Create Charts and Graphs
- Assemble simple code operations to solve problems Excel can't solve.
- Input/Output files up to ~1GB

Prompt Improvement Pipeline

As tasks are repeated, a prompt improvement pipeline approach can be adopted

Each task is an opportunity to improve the prompt and add or remove parameters.

Automating small tasks, then larger tasks, and eventually outgrowing the capabilities of the web interface is the point.



Process Diagram Source

“Prompt Engineering”

The most impactful tips and tricks for improving your prompts generally revolve around providing the AI custom instructions and context:

When prompting an LLM, focus on:

- Providing clear, well-structured directions
- Use Delineators to Separate Instructions from Context
- Understand the Limitations:
  - Limited Context Windows
  - Limited Retrieval from Large Documents
  - Probabilistic Operation, not Deterministic
  - File size and library limitations in Code Interpreter
  - No internet access (blame the AI safety patrol)

Be ready to Iterate, Iterate, Iterate!

Basic “prompt engineering” is typically:

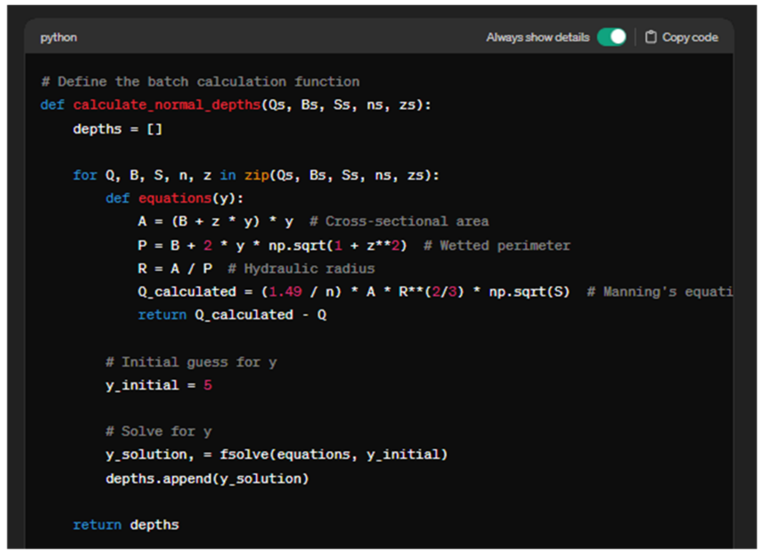
- Role (Persona)
- Constraints
- Contextual Data
- Instructions
- Desired Output
- Examples

Prompts can also be structured as code

Your AI Assistant doesn't know what it's doing here, unless you tell it

<u>AI-Assisted Python Scripting:</u>
Notebook Based, Code-Forward Approach

GPT can just as easily write a script for you to execute locally. Since the backend for Code interpreter is a Jupyter Notebook, the format suddenly became very useful for small to medium complexity scripts due to the ability to have robust AI assistance.



python   Always show details   Copy code

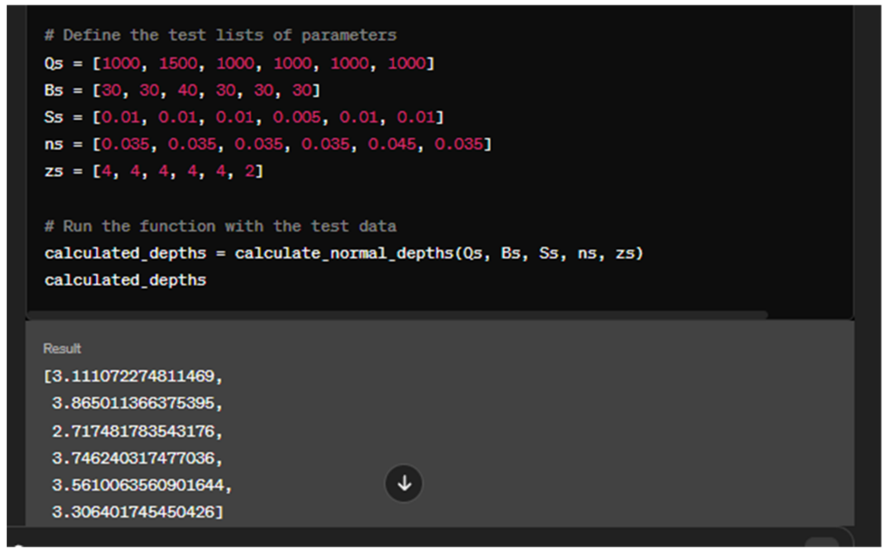
```python
# Define the batch calculation function
def calculate_normal_depths(Qs, Bs, Ss, ns, zs):
    depths = []

    for Q, B, S, n, z in zip(Qs, Bs, Ss, ns, zs):
        def equations(y):
            A = (B + z * y) * y  # Cross-sectional area
            P = B + 2 * y * np.sqrt(1 + z**2)  # Wetted perimeter
            R = A / P  # Hydraulic radius
            Q_calculated = (1.49 / n) * A * R**(2/3) * np.sqrt(S)  # Manning's equati
            return Q_calculated - Q

        # Initial guess for y
        y_initial = 5

        # Solve for y
        y_solution, = fsolve(equations, y_initial)
        depths.append(y_solution)

    return depths
```



```python
# Define the test lists of parameters
Qs = [1000, 1500, 1000, 1000, 1000, 1000]
Bs = [30, 30, 40, 30, 30, 30]
Ss = [0.01, 0.01, 0.01, 0.005, 0.01, 0.01]
ns = [0.035, 0.035, 0.035, 0.035, 0.045, 0.035]
zs = [4, 4, 4, 4, 4, 2]

# Run the function with the test data
calculated_depths = calculate_normal_depths(Qs, Bs, Ss, ns, zs)
calculated_depths
```

Result

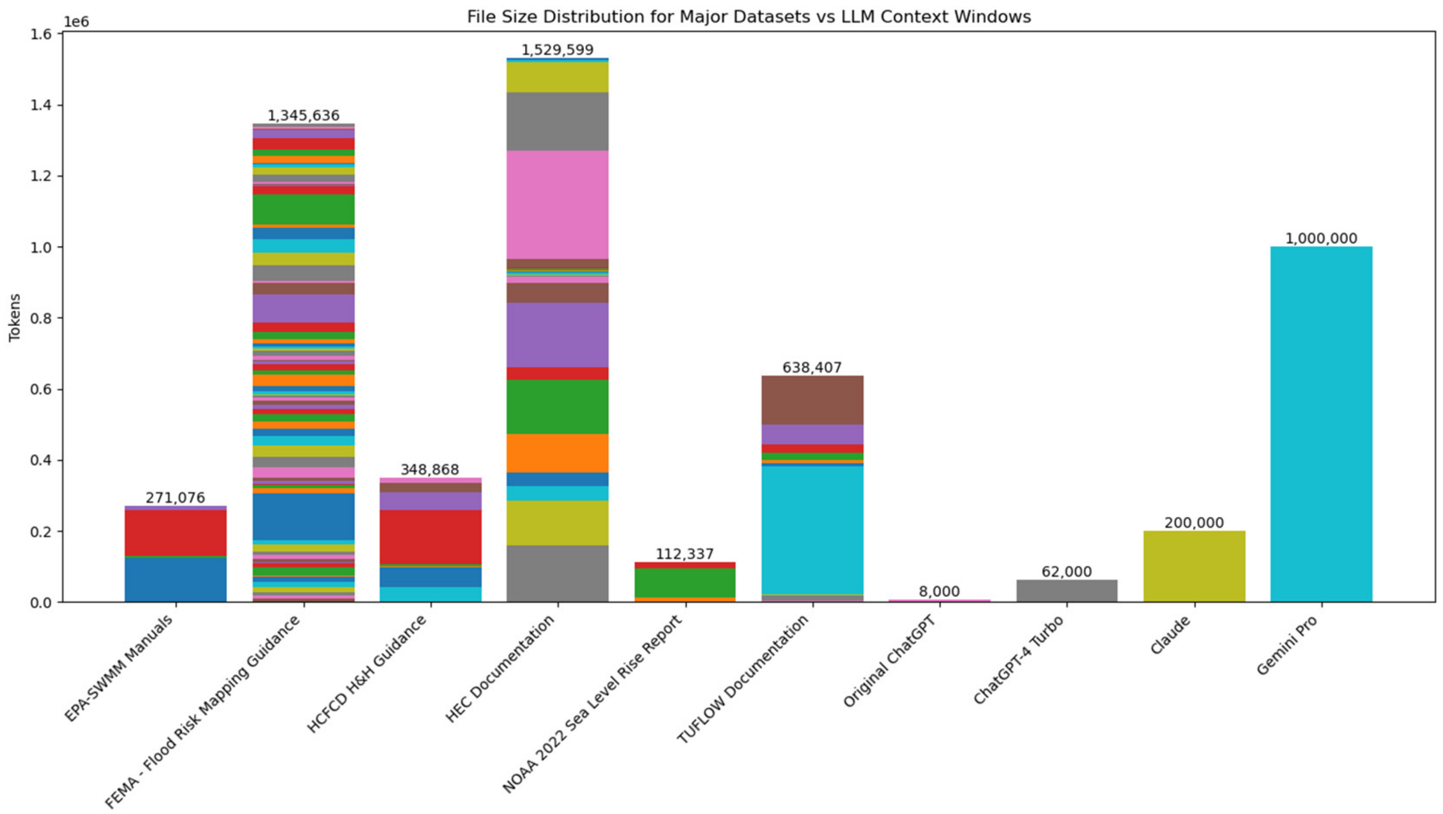
```
[3.111072274811469,
 3.865011366375395,
 2.717481783543176,
 3.746240317477036,
 3.5610063560901644,
 3.306401745450426]
```

By starting with small, useful operations that execute flawlessly within the code interpreter environment, non-coding users can begin chaining simple functionalities together within a local notebook, then iterating with GPT to achieve their workflow automation.

AI-Assisted Python Scripting: Context Window Driven Modularity

Context window driven modularity emerges, as code cells are naturally limited to the size of the context window before manual curation of context is required for each code request. Ideally, code cells should be around ½ the size of a typical LLM context window. For ChatGPT, this is somewhere around 30-62k tokens currently (depending on subscription level).

Hiding code away in libraries is ideal for software development but introduces friction for beginner users, as this context must be manually included to create coherent code. Code can then be later combined and refactored for efficiency, standardization or to prepare it to be modularized further into a library or class.



*For non-coders who are relying on LLM's to do the heavy lifting of their coding effort, including **all** custom code within the notebook where possible is ideal.*

Open-source libraries should already be in the LLM's training, which maximizes coverage and prevents hallucinations.

AI-Assisted Python Scripting:
Automated Python Environment Setup

Python environment setup is the #1 friction point in trying to create portable scripts

Utilizing Jupyter's ability to run commands in the terminal, we can detect missing packages and install them using any combination of package managers (pip, conda, git). This is ideal when building for a single platform (typically Windows).

The user provides import statements, and the GPT provides:

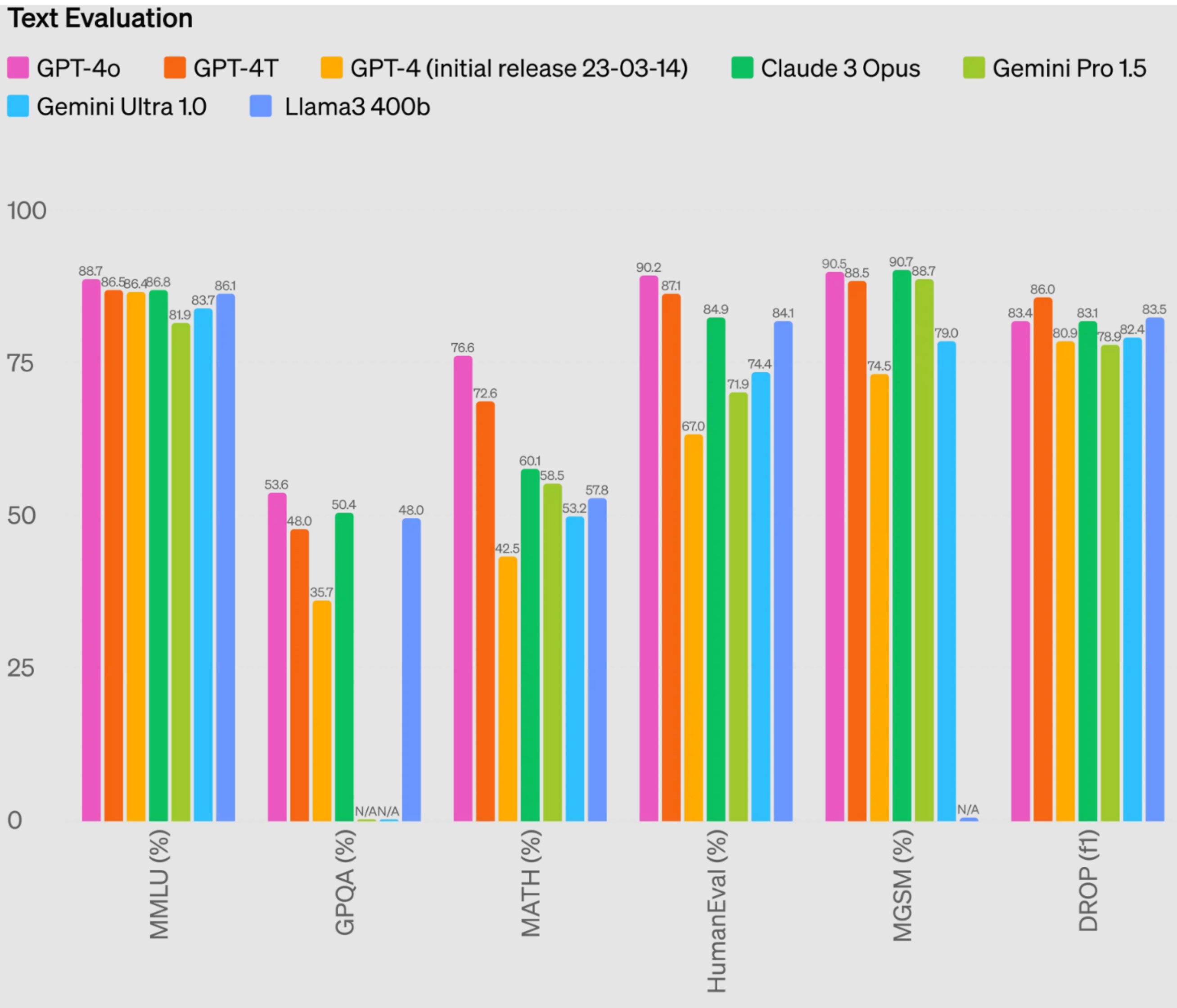
- Import detection
- Automatic package installation
- User can specify conda, pip, or special instructions

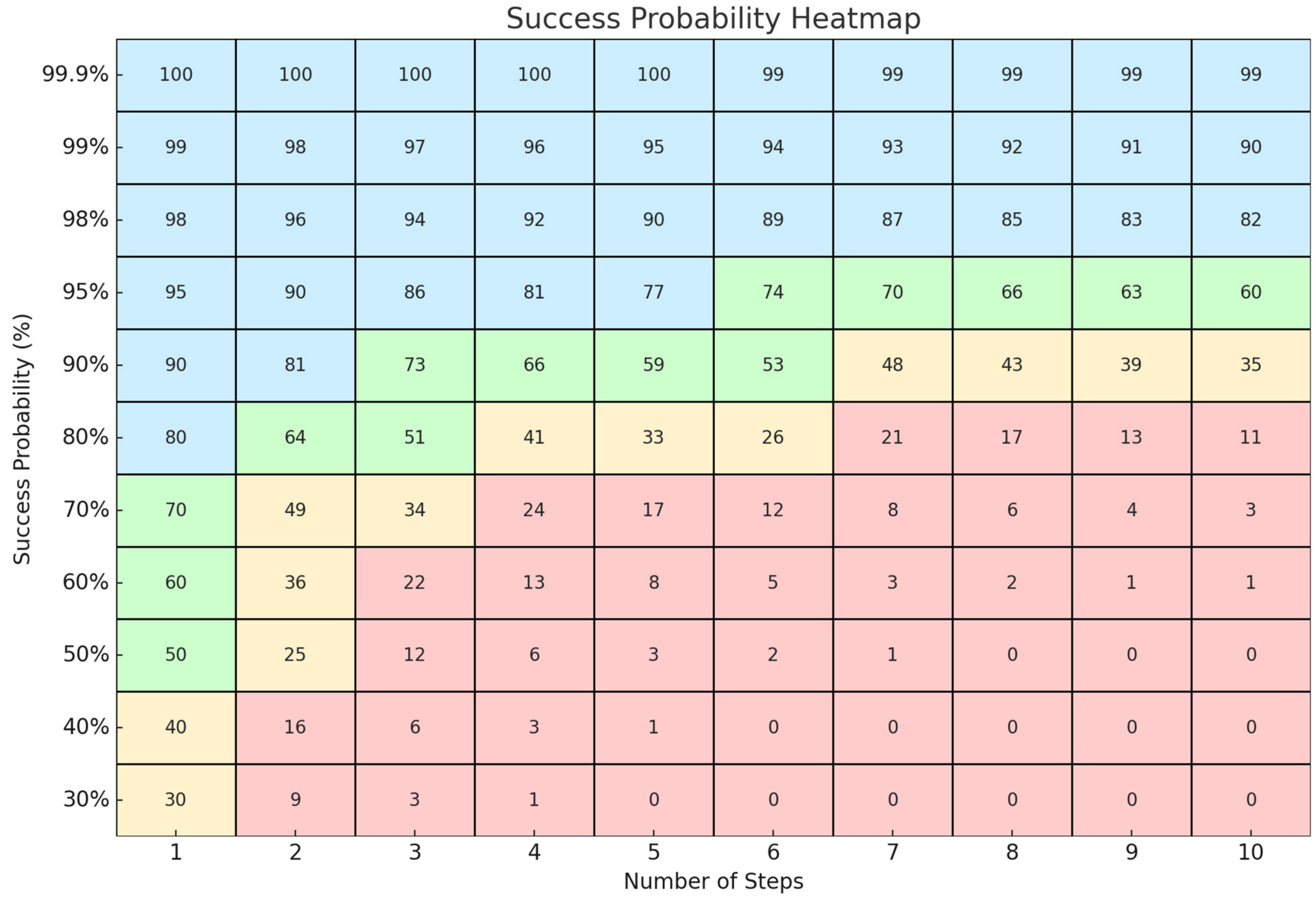
Jupyter Notebook Portability Enhancer GPT (ChatGPT)

Output:

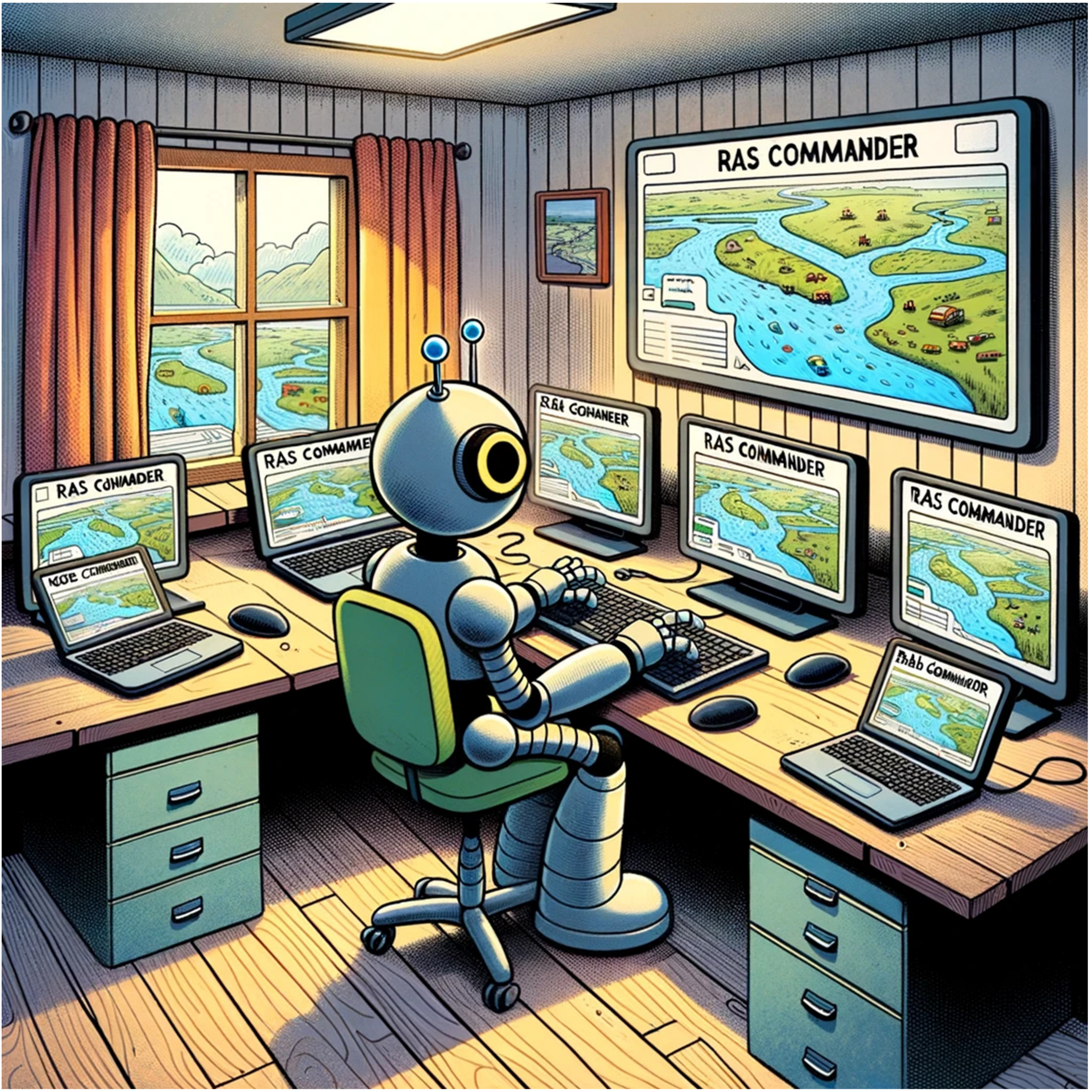
Complete, ready-to-run code cell to automate environment setup

Have Reasonable Expectations





At even a 88% accuracy rate, chained operations will still exhibit high probability of errors and hallucinations.
The "regenerate" button is still your friend! ***Iterating is an integral part of using LLM's.***



But Wait

There's More

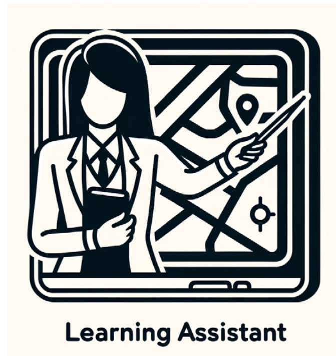
HEC-Commander Repository

Open Source Notebooks:

HEC-Commander Repository (GitHub)

Blogs

ChatGPT Examples and GPT's



Miscellaneous H&H Tools related to LWI Region 4 Efforts