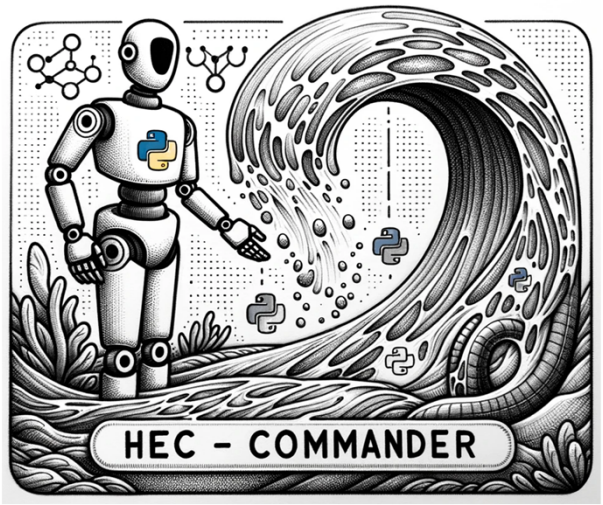
Leveraging AI-Assisted Scripting For HEC-RAS and HEC-HMS Automation

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

CAFM 2024 Conference

Wednesday November 13, 2024

2:15PM – 2:45PM

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
4. AI-Assisted Coding: Lowering Barrier to Entry for Modeling Workflows
5. Prompting Examples and Strategies
6. Coming Soon: RAS Commander Library and AI Assistant

William Katzenmeyer
Linkedin

HEC-Commander Repository (GitHub)

RAS-Commander

Parallelizing HEC-RAS In a Jupyter Notebook

AI-Coded Jupyter Notebook Supporting:

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

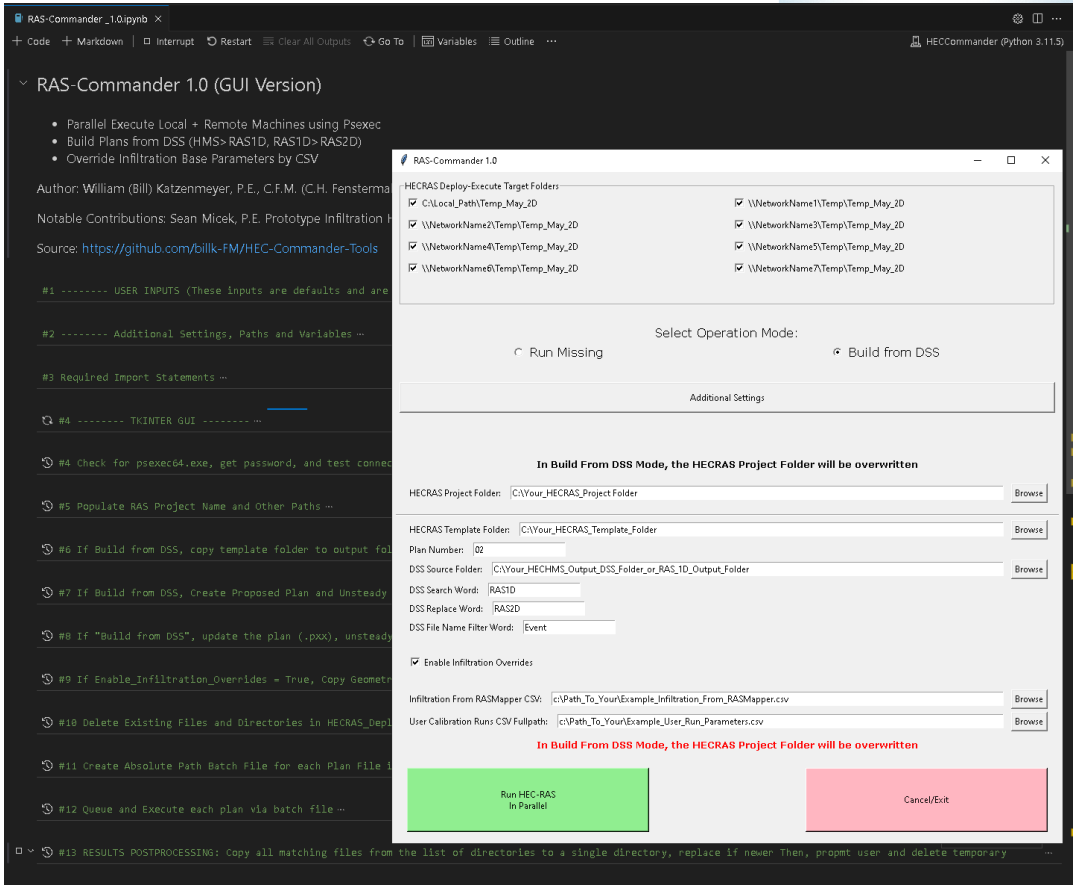
Basic Components

- 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.

2D HEC-RAS Performance Scaling

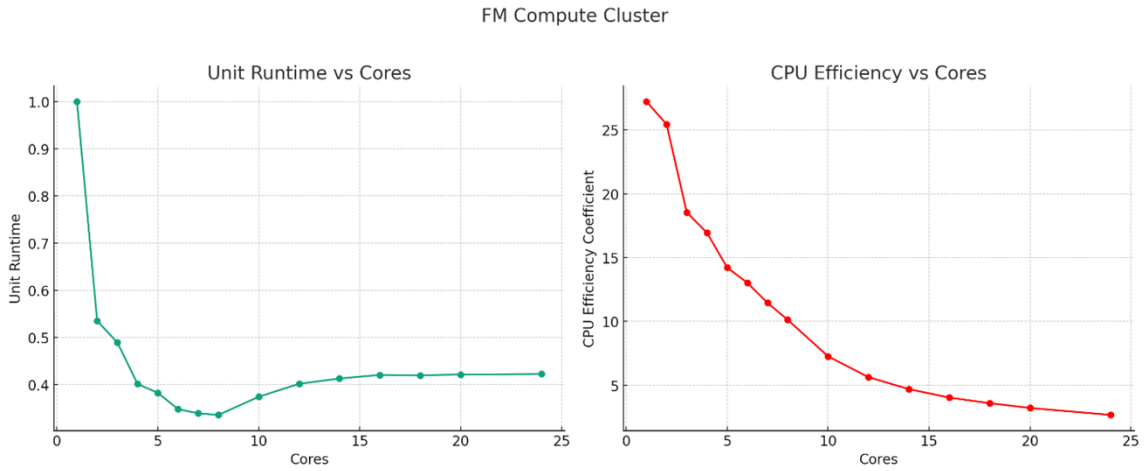
Fenstermaker Local Compute Cluster: Benchmarking Insights

Unit Runtime

To simplify comparisons

Best Value =

0.31 @ 8 cores



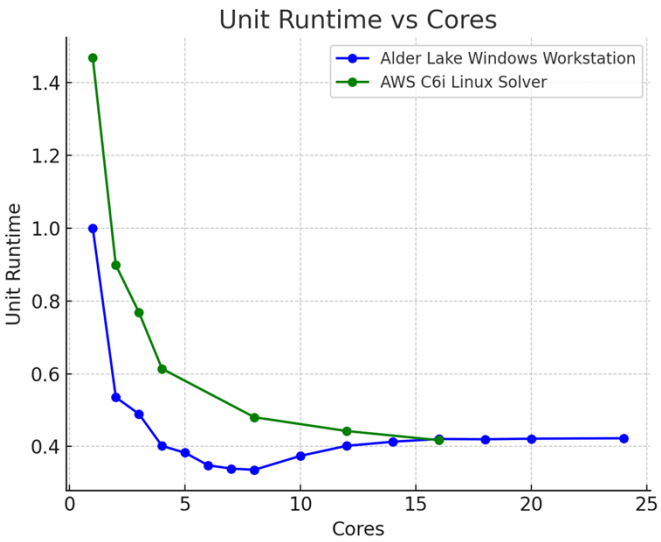
Scaling is linear with clock speed, but efficiency drops significantly beyond 2 cores

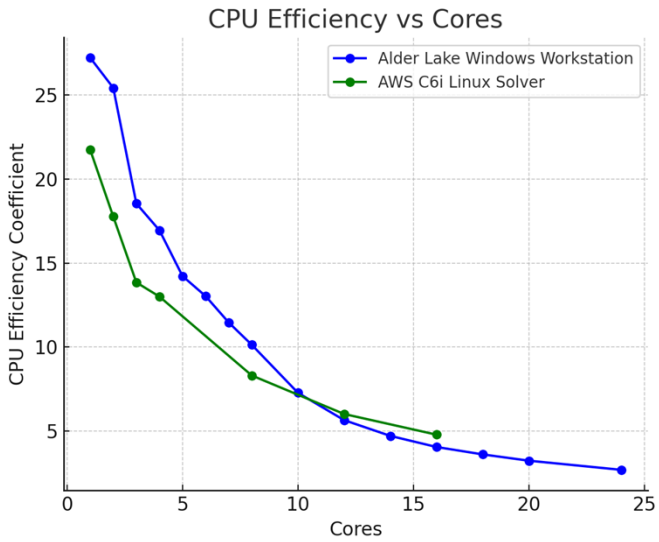
HEC-RAS 2025 will eventually feature GPU acceleration and explicit solution schemes that scale more linearly with core count*

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.



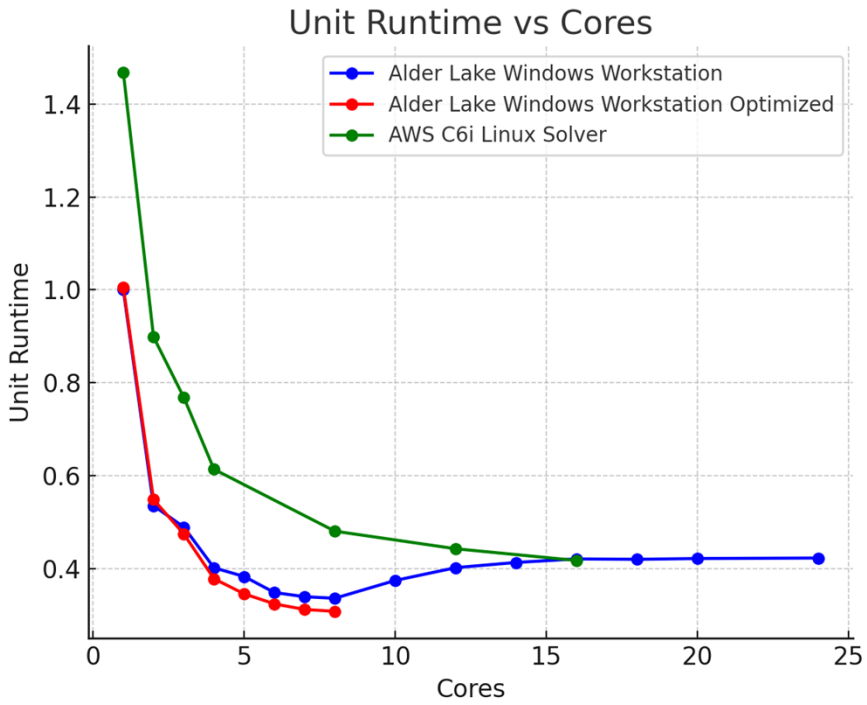


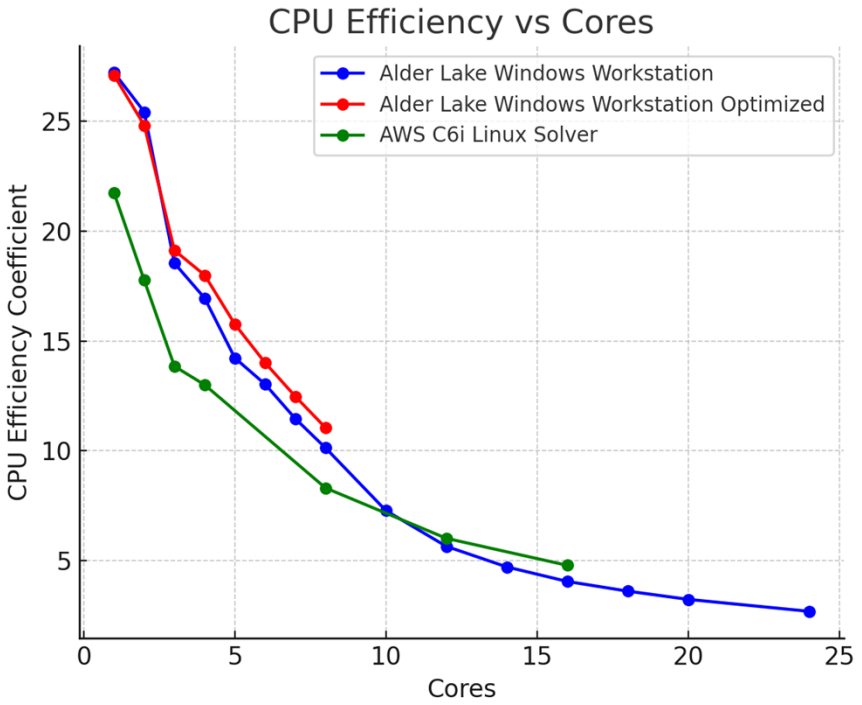
Cloud-scale architectures do successfully scale beyond 8 cores, but have similar efficiency characteristics

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

Optimize CPU Settings

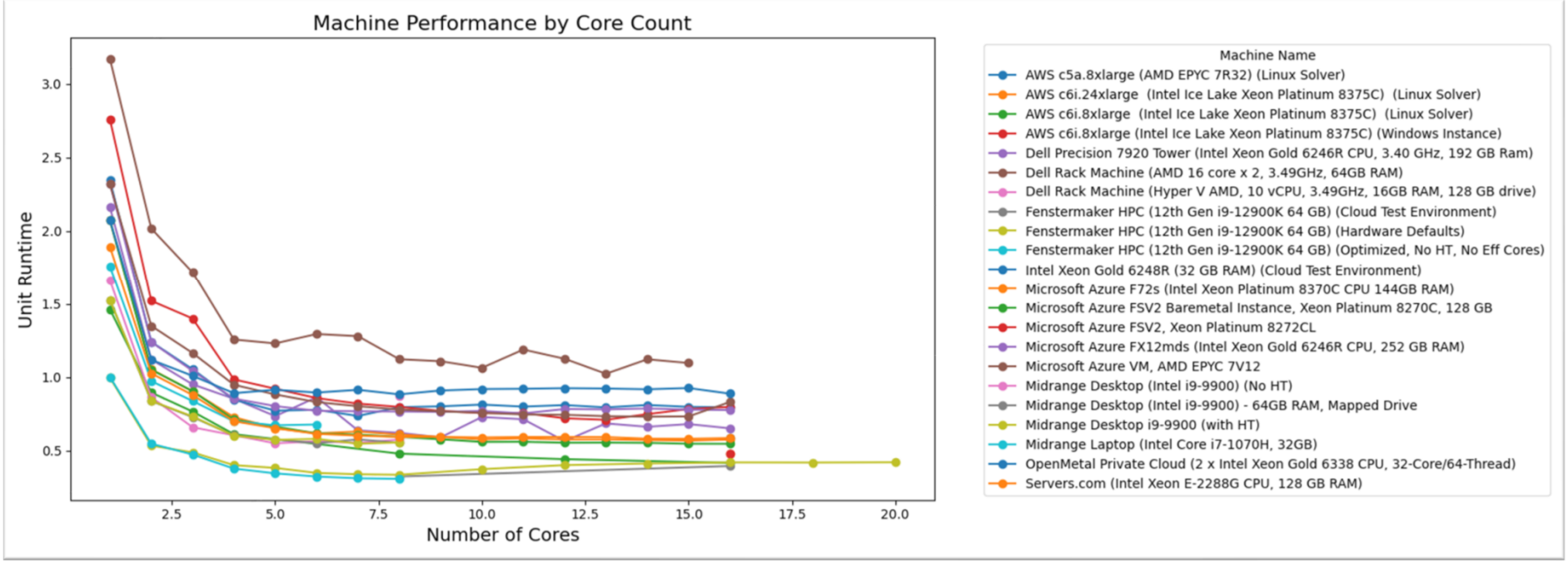
Disable Hyperthreading *Disable “Efficiency” cores* *Install Intel XTU Tuning*





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

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

Source Blog Post:
Benchmarking is All You Need

Parallelization In Practice

Giving 70% to Gain 70%

For an assumed 1 day runtime at 1 core

2 Cores = 0.55 days

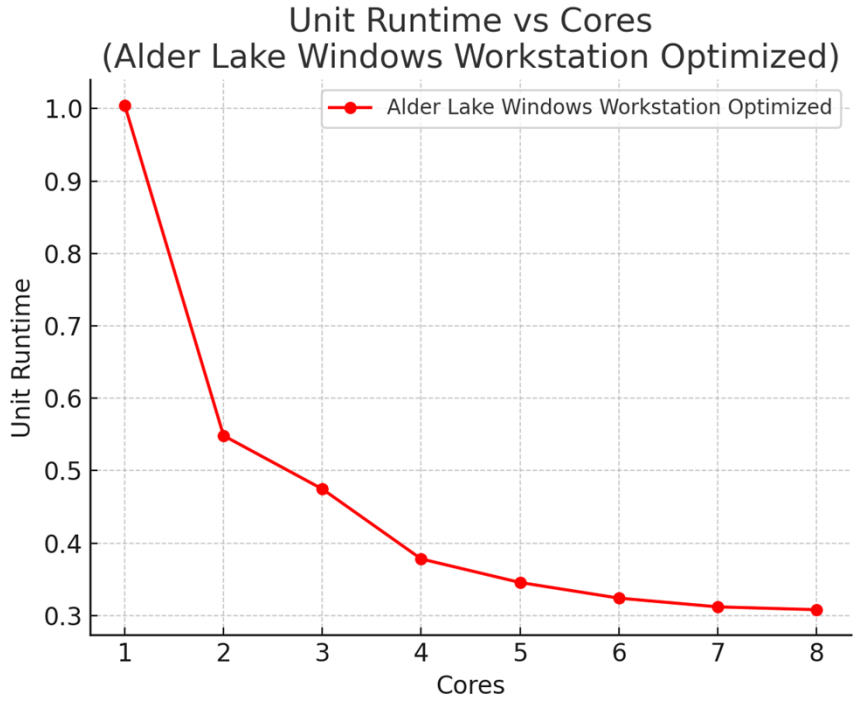
8 cores = 0.31 days

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

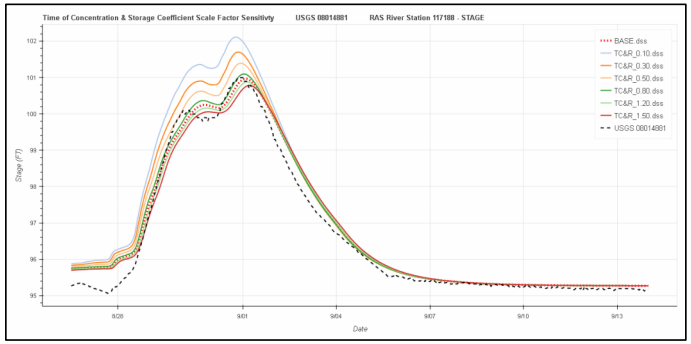
A Better Solution was Needed
So I coded it myself with AI

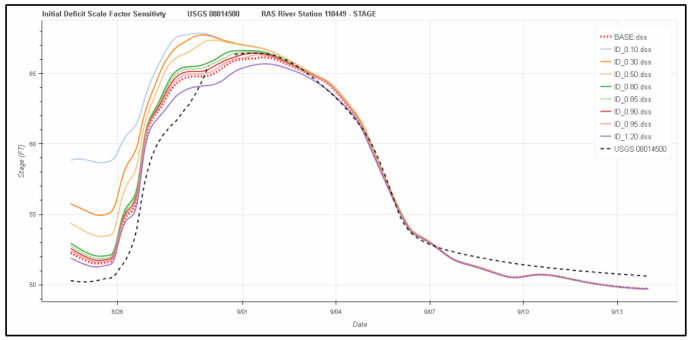
Case Study: West Fork Calcasieu Model

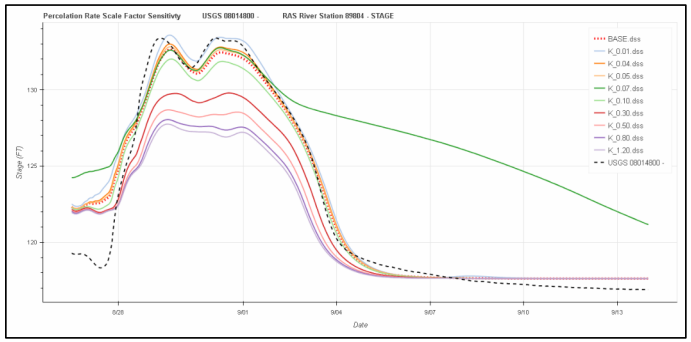
Region 4, Louisiana Watershed Initiative

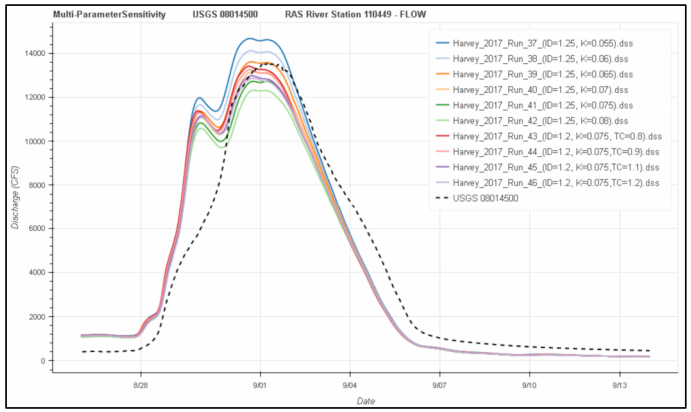
LOUISIANA WATERSHED INITIATIVE
working together for sustainability and resilience

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

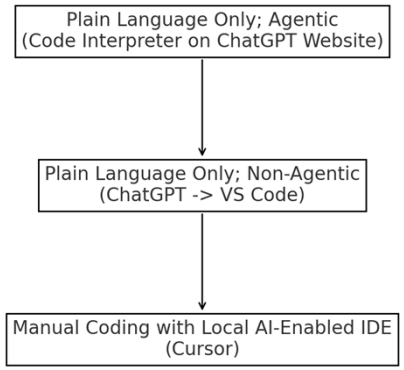
We will come back to this again in the Case Study at the end of the presentation.

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

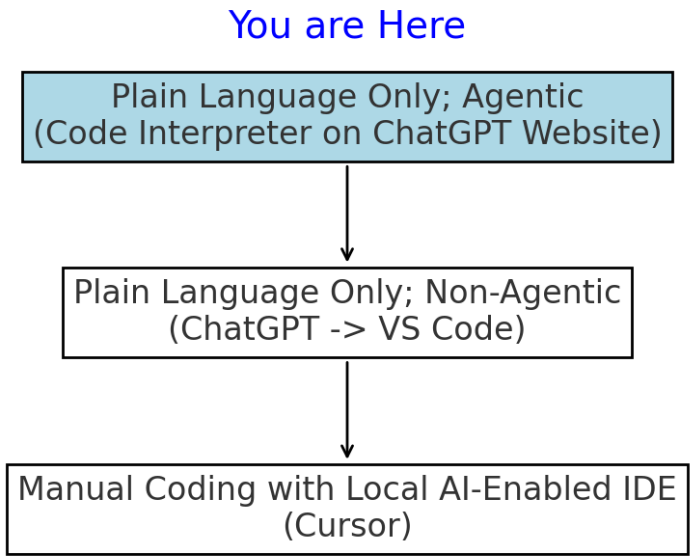
AI can assist beginner users with:

- 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.

Emerging Use Cases of Large Language Models

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:

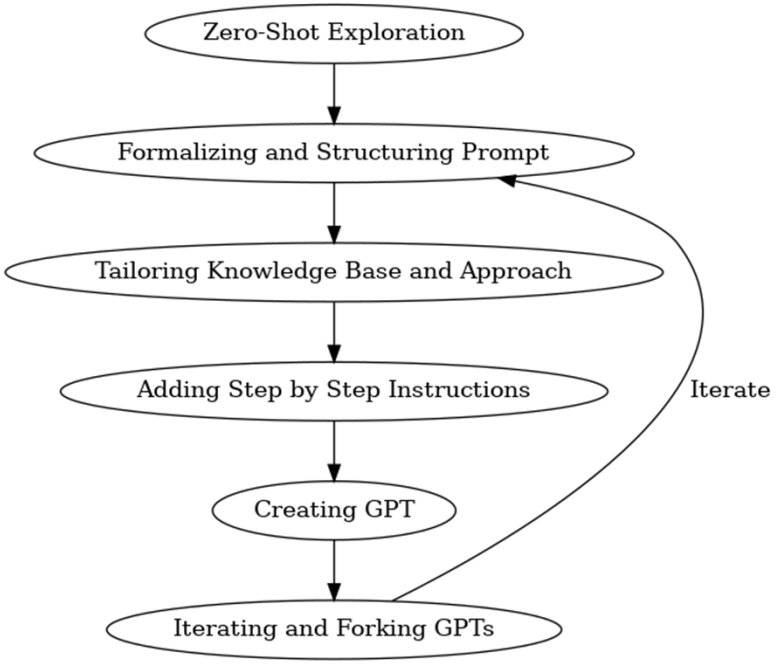
- Basic GIS Operations
- Sort/Filter/Display Datasets.
- Create Charts and Graphs
- Assemble simple code operations to solve problems Excel can't solve.
- Input/Output files up to ~1GB

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:

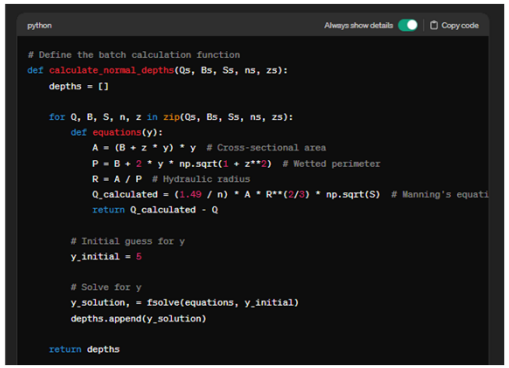
- Role (Persona)
- Constraints
- Contextual Data
- Instructions
- Desired Output
- Examples

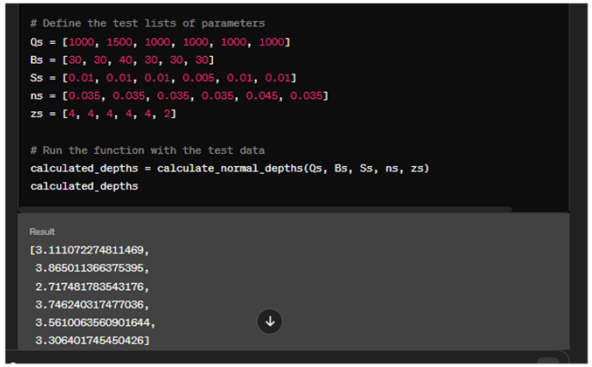
Prompts can also be structured as code

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

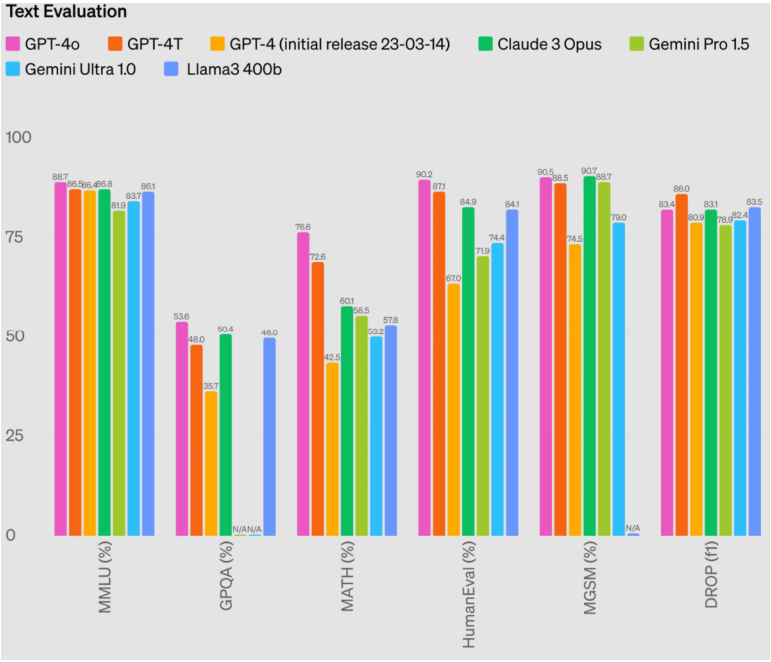
AI-Assisted Python Scripting for Beginners: Notebook Based, Code-Forward Approach

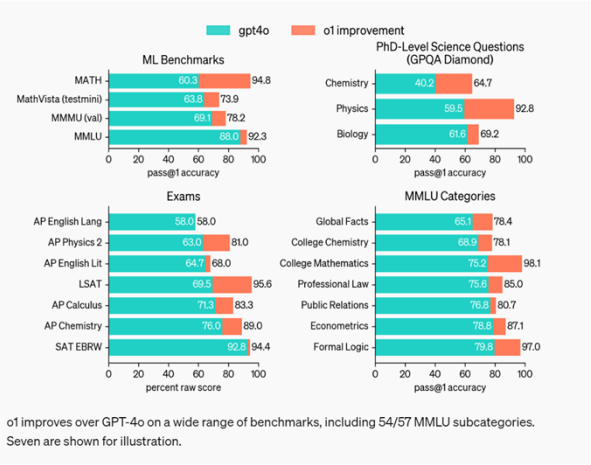
GPT can just as easily write a script for you to execute locally. Since the backend for Code interpreter is a Jupyter Notebook, the format suddenly became very useful for small to medium complexity scripts due to the ability to have robust AI assistance.





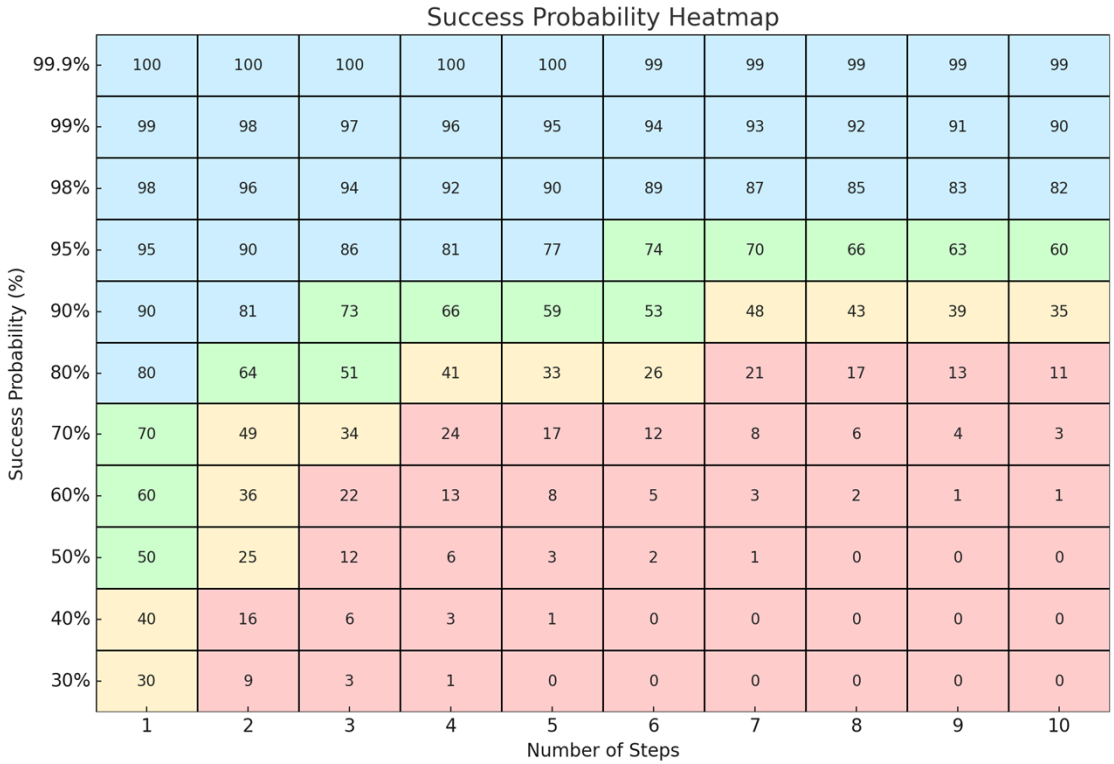
By starting with small, useful operations that execute flawlessly within the code interpreter environment, non-coding users can begin chaining simple functionalities together within a local notebook, then iterating with GPT to achieve their workflow automation.





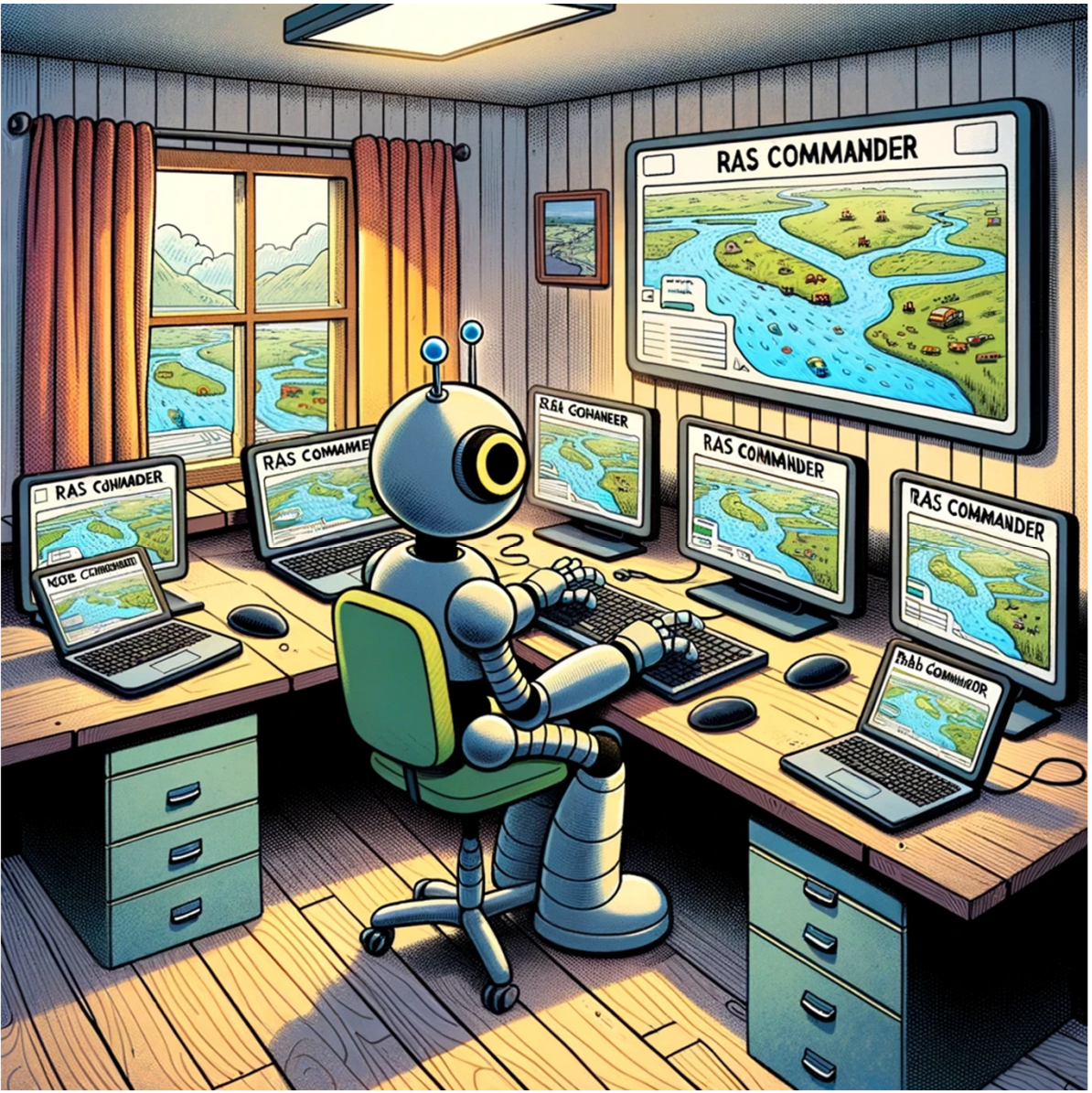
o1 improves over GPT-4o on a wide range of benchmarks, including 54/57 MMLU subcategories. Seven are shown for illustration.

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

RAS Commander Library

Scheduled for Release: Q1 2025

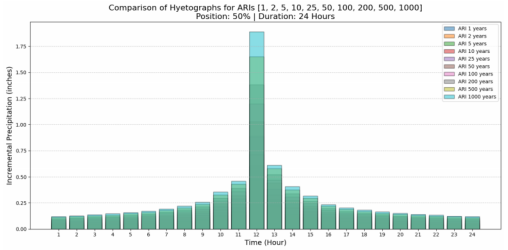
- Automation Library for HEC-RAS using Python
  - Version 6.x supported
  - HEC-RAS 2025 Support with API Documentation Release
  - Built with Large Language Models

- Integrated AI Tools
  - Context Window Driven Modularity
  - Cursor Rules and GPT Instructions
  - Knowledge Bases
  - AI Library Assistant
  - GPT with ability to load and visualize HEC-RAS data

- Goals:
  - Data Accessibility
  - Enabling Deeper Data Analysis
  - Agentic Research Assistance
  - Functional Replacement for HECRASController
  - Embracing the Future with Large Language Model-driven Code


```
Plan Parameters DataFrame:
{'1D Cores': 0,
 '1D Flow Tolerance': nan,
 '1D Maximum Iterations': 20,
 '1D Maximum Iterations Without Improvement': 0,
 '1D Maximum Water Surface Error To Abort': 100.0,
 '1D Methodology': 'Finite Difference',
 '1D Storage Area Elevation Tolerance': 0.02,
 '1D Theta': 1.0,
 '1D Theta Warmup': 1.0,
 '1D Water Surface Elevation Tolerance': 0.02,
 '1D-2D Flow Tolerance': 1.0,
 '1D-2D Gate Flow Submergence Decay Exponent': 1.0,
 '1D-2D IS Stablity Factor': 1.0,
 '1D-2D LS Stablity Factor': 2.0,
 '1D-2D Maximum Iterations': 0,
 '1D-2D Maximum Number of Time Slices': 20,
 '1D-2D Minimum Flow Tolerance': nan,
 '1D-2D Minimum Time Step for Slicing(hours)': 0.0,
 '1D-2D Number of Warmup Steps': 0,
 '1D-2D Warmup Time Step (hours)': 0.0,
 '1D-2D Water Surface Tolerance': 0.02,
 '1D-2D Weir Flow Submergence Decay Exponent': 1.0,
 '2D Advanced Convergence': array([0], dtype=uint8),
 '2D Boundary Condition Ramp Up Fraction': array([0.5], dtype=float32),
 '2D Boundary Condition Volume Check': array([b'False'], dtype='|S5'),
 '2D Cores (per mesh)': array([12]),
```

RAS-Commander Library Features

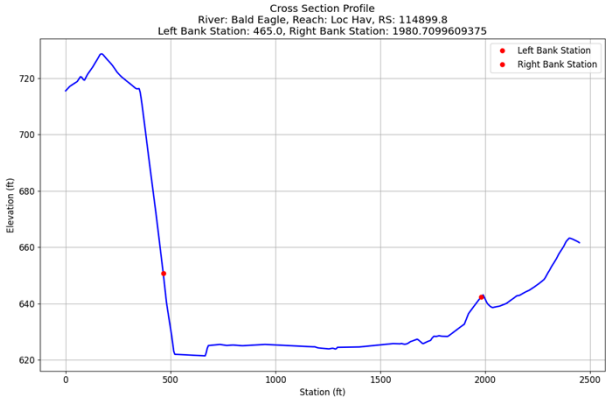


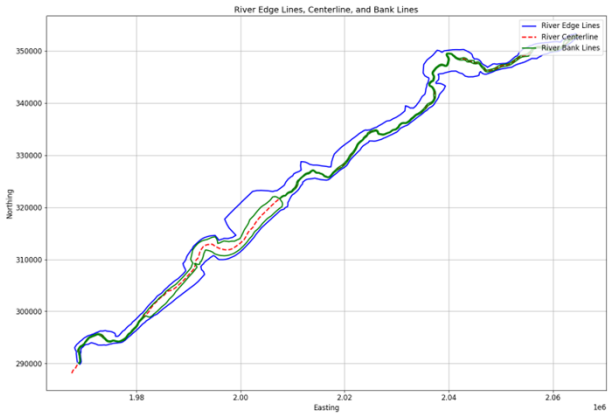
- Access All HDF Datasets as Pandas and GeoPandas Dataframes
- Common Automation/ Execution
- Functional Replacement for HECRASController

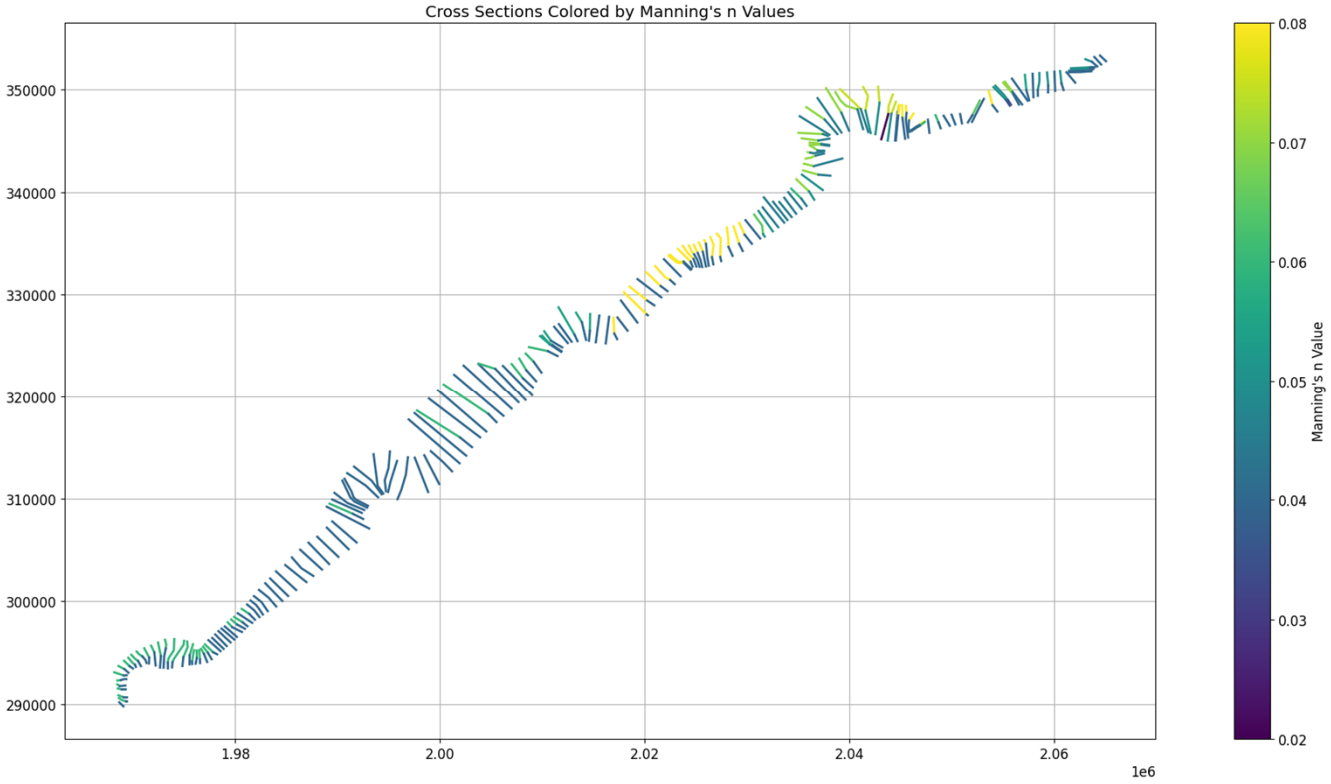
```
Cross Section 20:
River: Bald Eagle
Reach: Loc Hav

Geometry:
LINESTRING (1975868.58 295827.2, 1975727.52 295333.46, 1975545.3 294751.54, 1975533.54 293875.74)

Station-Elevation Points:
```

#	Station	Elevation	#	Station	Elevation	#	Station	Elevation	#	Station	Elevation	#	Station	Elevation
1	0.00	660.64	27	320.00	654.87	53	575.00	641.21	79	1055.00	639.99	105	1570.00	648.41
2	25.00	658.61	28	355.00	653.36	54	580.01	641.13	80	1080.00	640.40	106	1575.00	648.46
3	30.00	658.28	29	360.00	653.18	55	620.00	641.03	81	1085.00	640.44	107	1620.00	648.66
4	45.00	657.38	30	405.00	652.85	56	645.00	640.52	82	1123.28	640.43	108	1670.00	650.05
5	50.00	657.12	31	415.00	653.03	57	650.00	640.54	83	1150.00	640.26	109	1675.00	650.12
6	55.01	656.80	32	425.00	653.09	58	700.00	641.79	84	1155.00	640.27	110	1690.00	650.13
7	60.00	656.61	33	430.00	653.08	59	725.00	640.94	85	1200.00	641.46	111	1740.00	649.49
8	85.00	656.06	34	435.00	653.01	60	730.00	640.88	86	1205.01	641.51	112	1745.00	649.48
9	90.00	656.15	35	445.00	651.67	61	740.01	641.00	87	1260.00	640.96	113	1775.00	649.70
10	95.00	656.19	36	450.00	650.92	62	745.00	640.63	88	1270.00	640.93	114	1820.00	649.78
11	100.00	656.26	37	455.00	650.29	63	750.00	640.35	89	1315.01	641.18	115	1860.00	649.59
12	110.00	656.35	38	460.00	649.73	64	755.00	640.27	90	1365.00	642.03	116	1865.01	649.45
13	115.00	656.42	39	465.00	648.81	65	760.00	640.38	91	1367.42	642.09	117	1870.01	649.35
14	125.00	656.30	40	475.01	645.93	66	765.00	640.56	92	1385.00	642.50	118	1875.01	649.97
15	130.00	656.42	41	485.00	643.35	67	770.00	640.77	93	1395.00	642.66	119	1905.00	658.80
16	145.00	656.94	42	490.00	642.12	68	775.01	641.06	94	1400.00	642.50	120	1910.00	660.31
17	150.00	657.08	43	495.00	641.92	69	780.00	641.22	95	1405.00	640.12	121	1915.00	661.54
18	155.00	657.03	44	500.00	641.81	70	785.00	640.84	96	1410.00	638.23	122	1930.00	662.21
19	160.00	656.92	45	513.50	641.85	71	820.00	639.88	97	1415.00	637.54	123	1935.00	662.40
20	180.01	656.32	46	515.00	641.85	72	835.00	640.41	98	1510.00	636.88	124	1940.00	662.19
21	195.00	655.98	47	520.00	641.80	73	840.00	640.36	99	1515.00	636.99	125	1945.00	661.32
22	225.00	655.61	48	545.01	641.41	74	875.00	639.38	100	1540.01	646.36	126	1950.00	660.51
23	250.00	654.99	49	550.00	641.38	75	880.00	639.40	101	1545.00	647.47	127	1955.00	659.87
24	255.00	654.91	50	555.00	641.43	76	915.00	640.10	102	1554.14	647.87	128	1990.00	657.08
25	300.00	655.27	51	560.00	641.45	77	920.00	640.16	103	1560.00	648.12	129	1999.16	656.61
26	305.01	655.22	52	565.00	641.43	78	1050.00	639.94	104	1565.00	648.28			

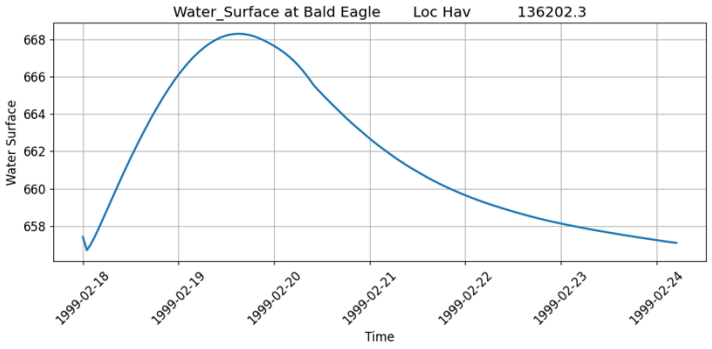
1D HDF Data Extraction

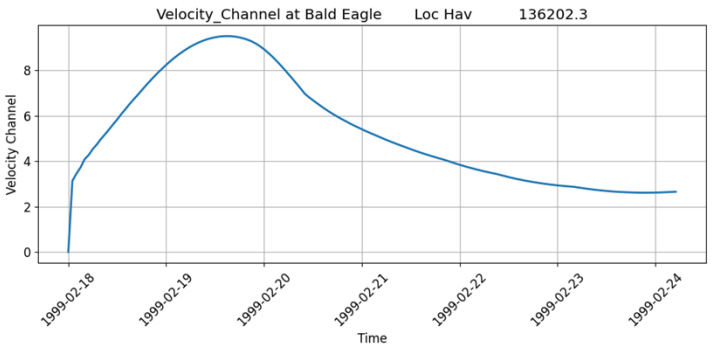


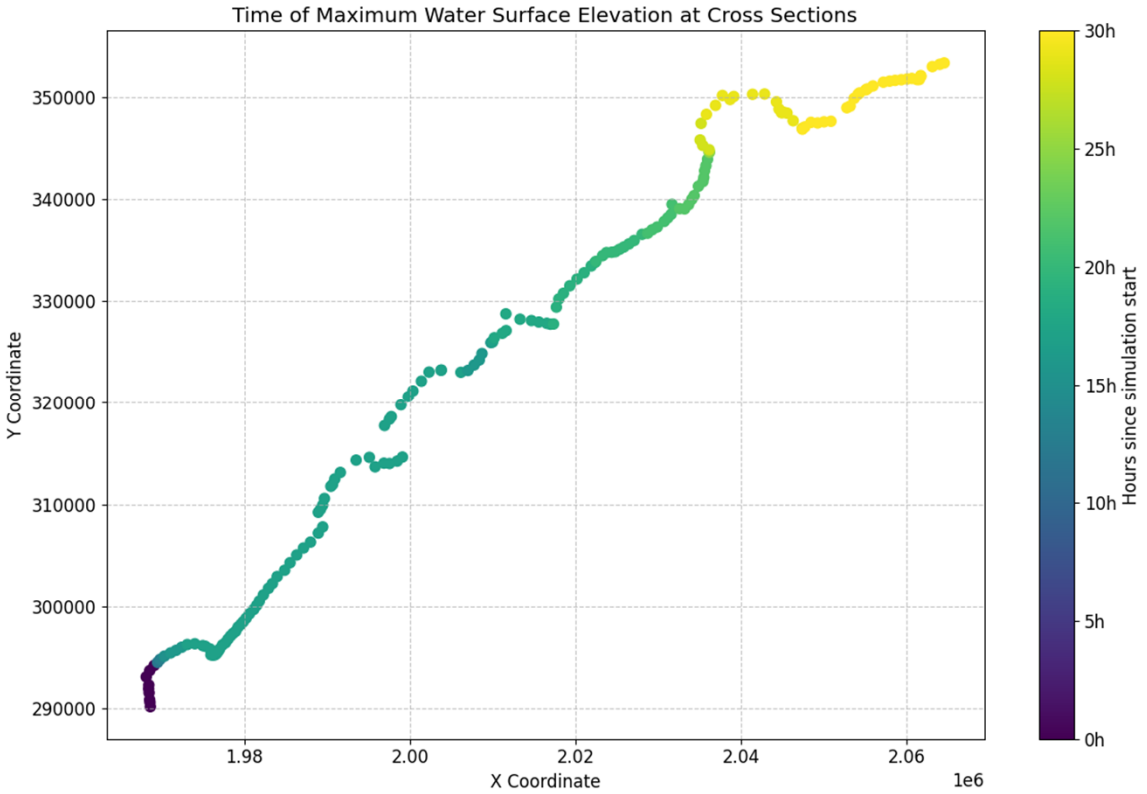




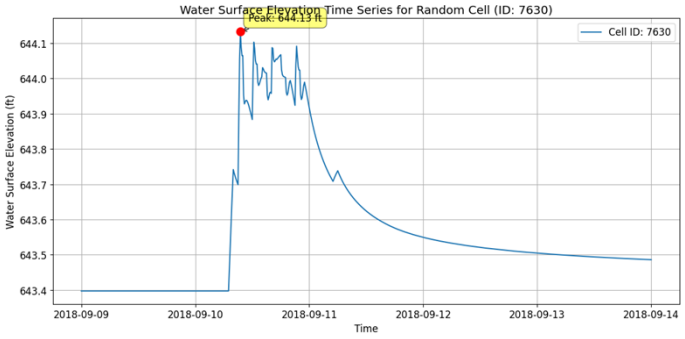
1D HDF Data Extraction

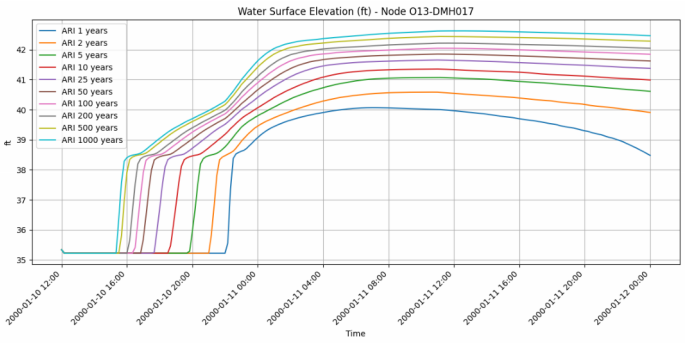


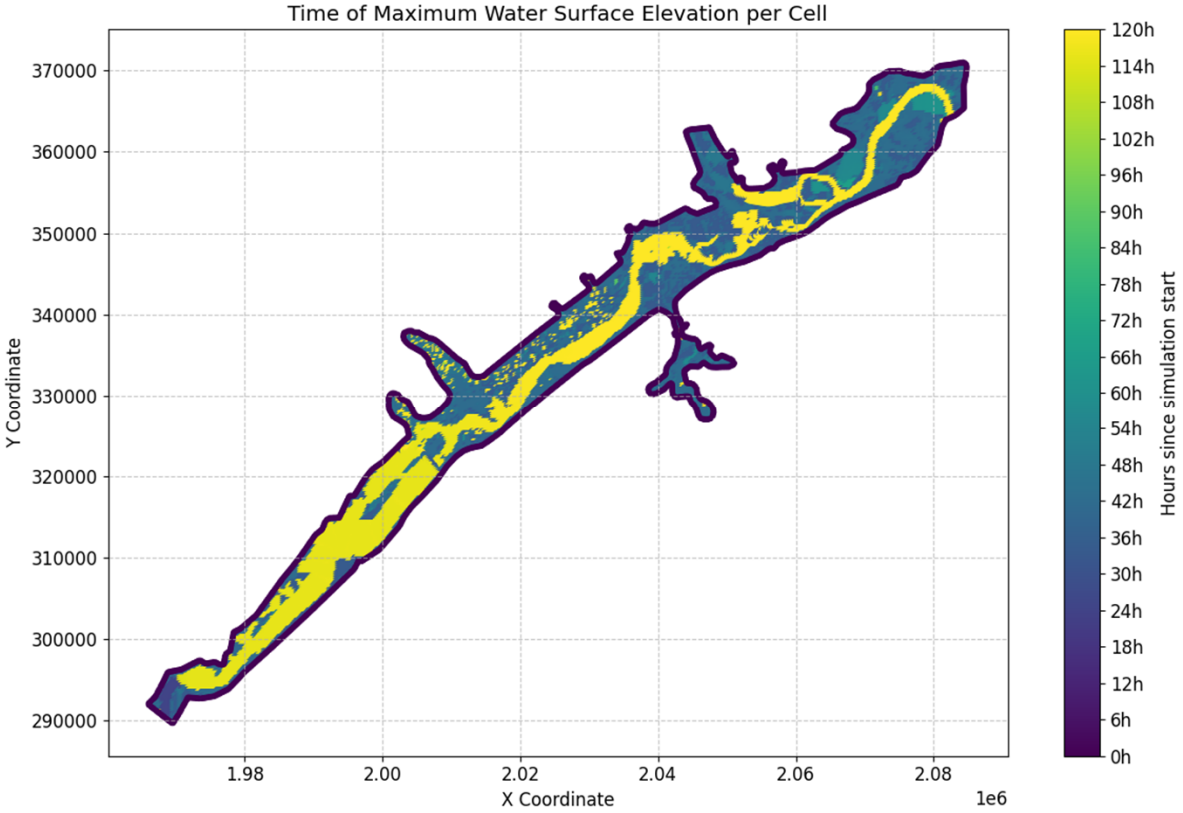




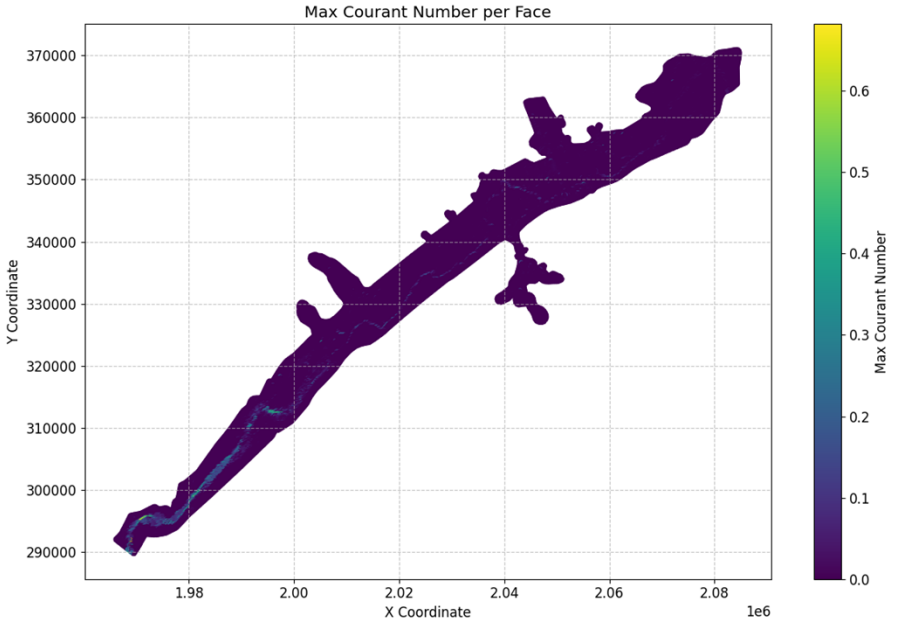
2D HDF Data Extraction

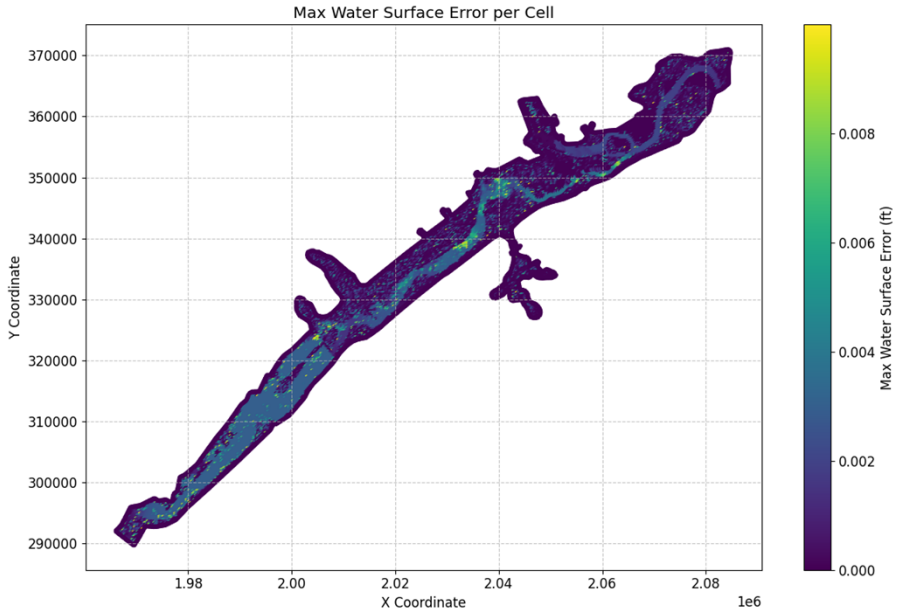






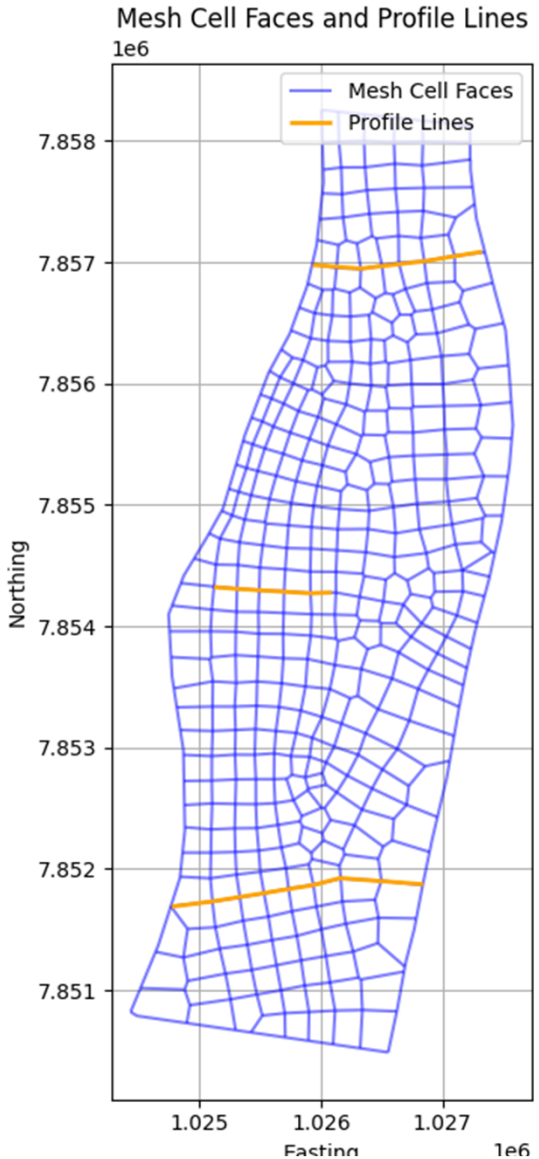
2D HDF Data Extraction

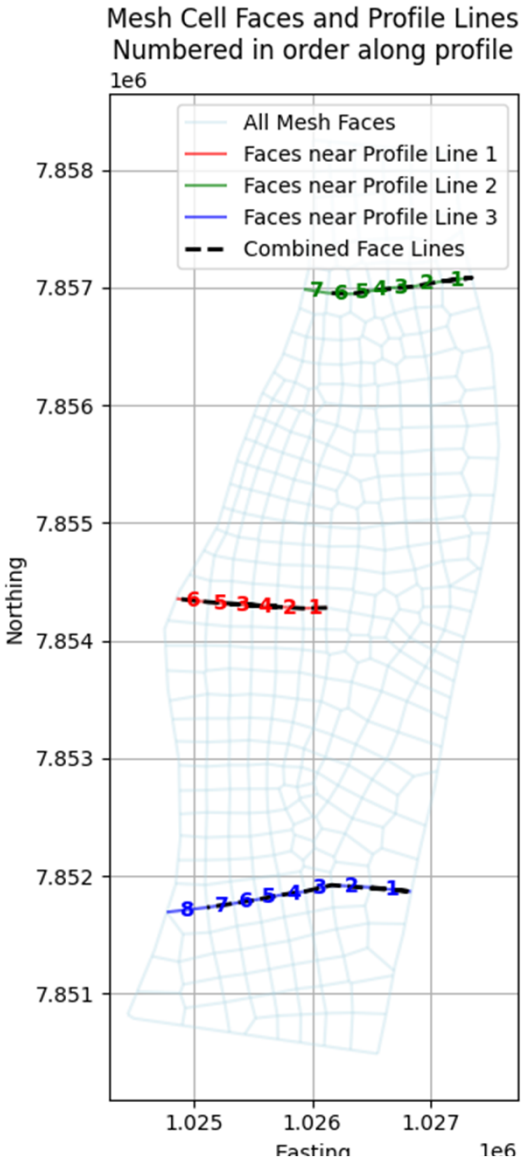


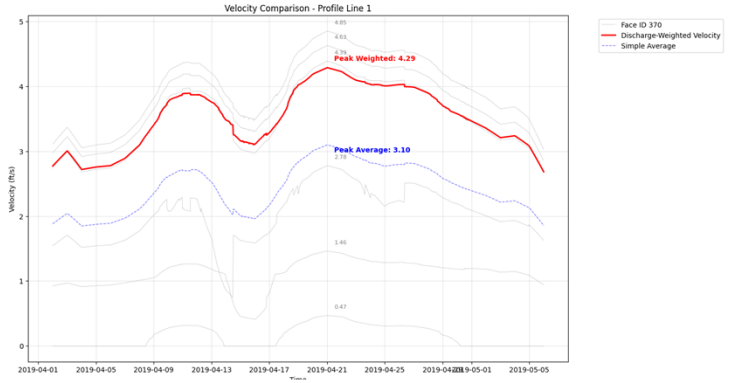


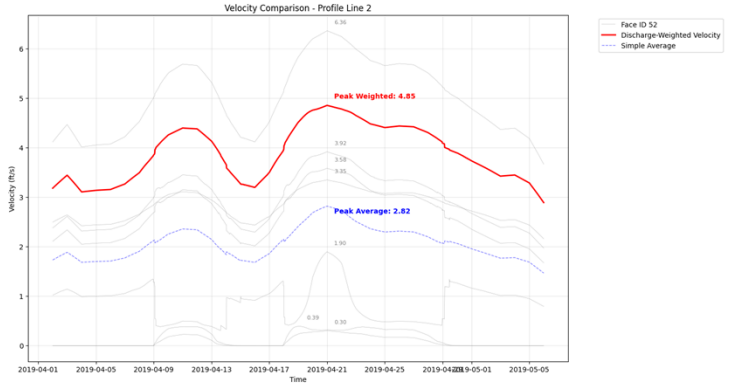
Enabling Novel Data Analysis and Research

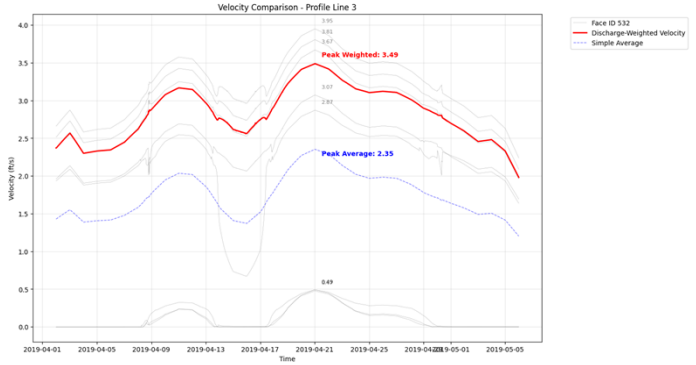
- Discharge-weighted average velocity is not readily available in HEC-RAS. Reference lines are used to extract summary results along 2D profile lines, but do not currently support the calculation of discharge-weighted average velocity.
- This is an important limitation on 2D sediment transport modeling
- One of the use cases that will be published with the library is the use the results arrays to directly calculate discharge-weighted average velocity using a user-drawn profile line which can be placed after results are calculated (unlike reference points and lines).


Mesh Cell Faces and Profile Lines
1e6
Mesh Cell Faces
Profile Lines
7.858
7.857
7.856
7.855
7.854
7.853
7.852
7.851
Northing
1.025
1.026
1.027
Easting
1e6

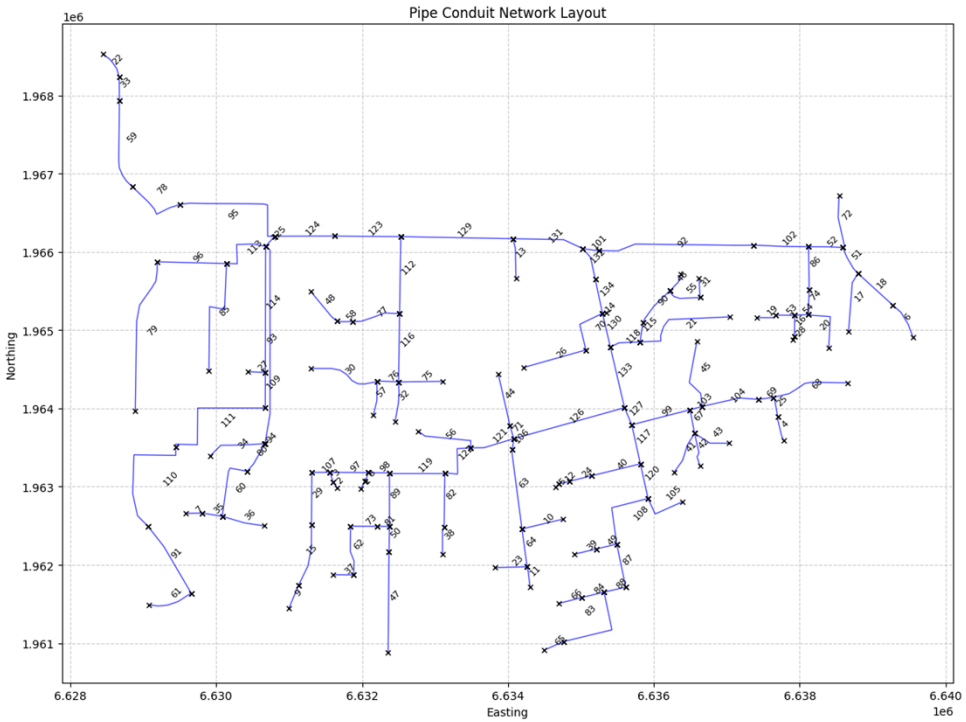

Mesh Cell Faces and Profile Lines
Numbered in order along profile
1e6
All Mesh Faces
Faces near Profile Line 1
Faces near Profile Line 2
Faces near Profile Line 3
Combined Face Lines
7.858
7.857
7.856
7.855
7.854
7.853
7.852
7.851
Northing
1.025
1.026
1.027
Easting
1e6

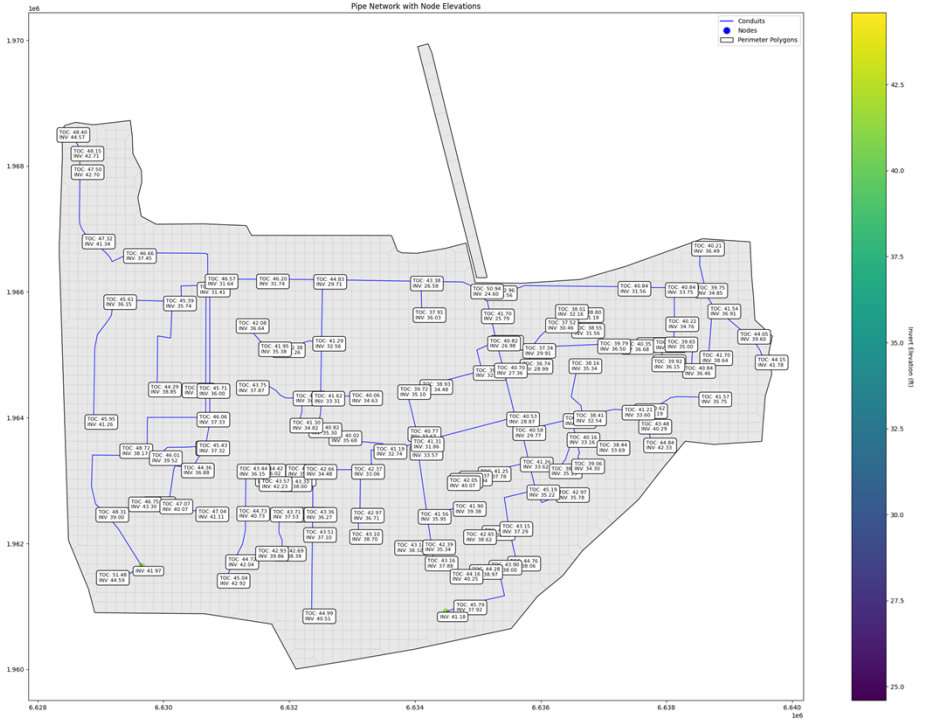

Velocity Comparison - Profile Line 1
Face ID 370
Discharge-Weighted Velocity
Simple Average
Peak Weighted: 4.29
Peak Average: 3.10
Velocity (ft/s)
Time


Velocity Comparison - Profile Line 2
Face ID 52
Discharge-Weighted Velocity
Simple Average
Peak Weighted: 4.85
Peak Average: 2.82
Velocity (ft/s)
Time


Velocity Comparison - Profile Line 3
Face ID 532
Discharge-Weighted Velocity
Simple Average
Peak Weighted: 3.49
Peak Average: 2.35
Velocity (ft/s)
Time

Pipes and Conduits Data Extraction



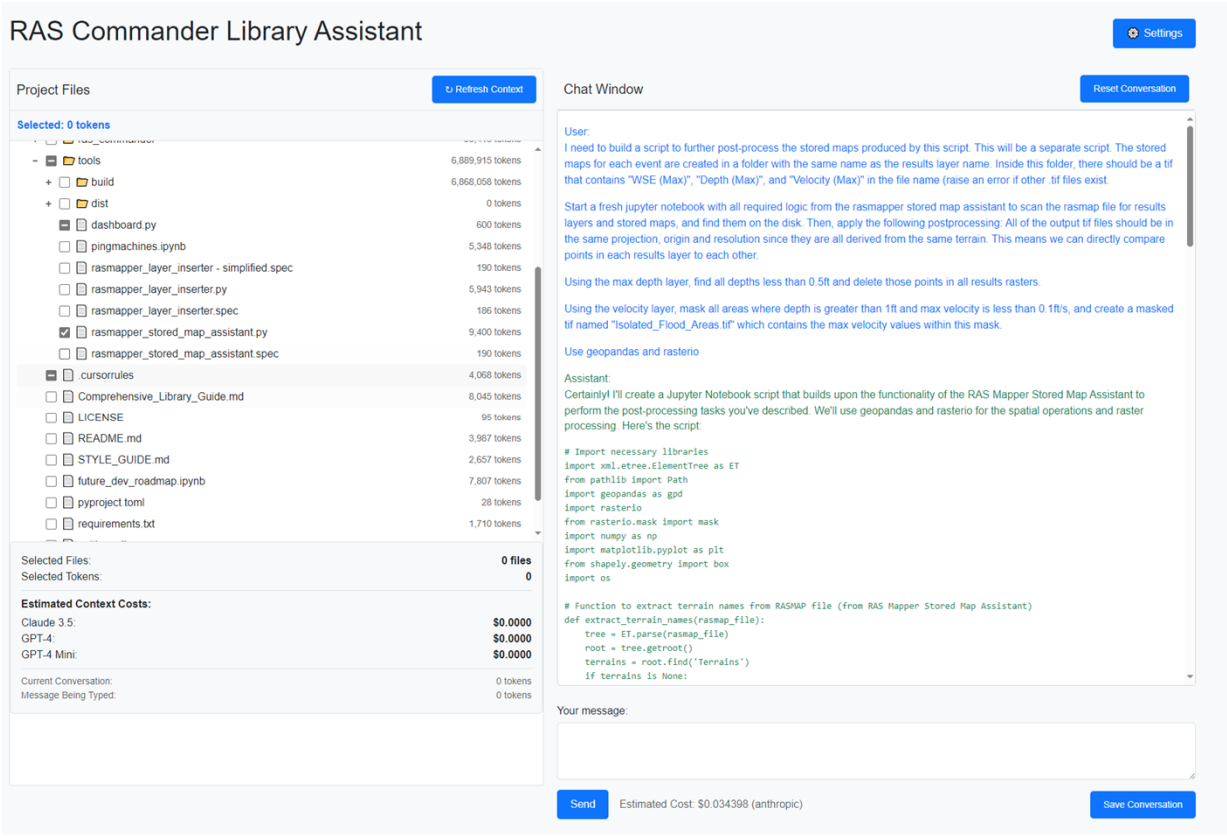


AI Library Assistant

- Bring your own API Keys
- Repo Explorer

Roadmap:

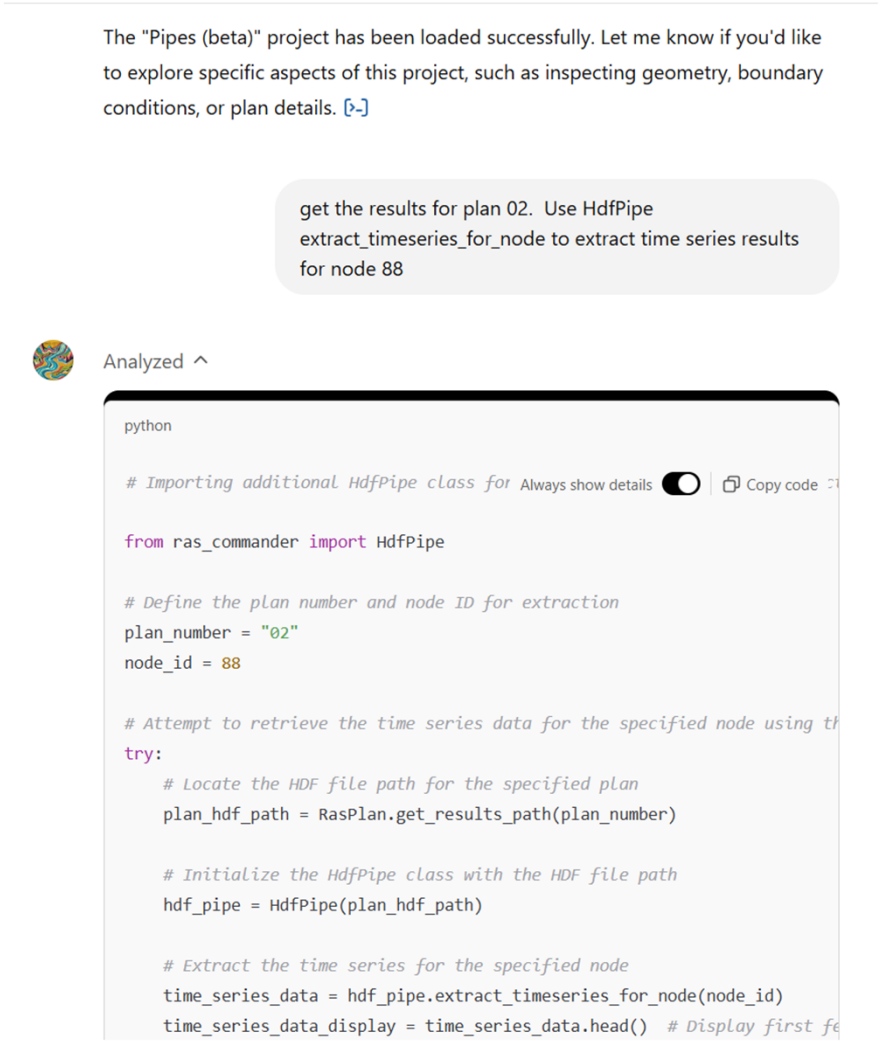
- RAG Mode (coming soon)
- Agentic Code Execution (coming soon)



RAS-Commander GPT

- Limited by OpenAI's GPT framework, but still a novel way of interacting with the library
  - Load Small Projects with Code Interpreter
  - Visualize Results and Perform Data Analysis
  - Ask General Questions
  - Short Context is Best for Now

Ask for Early Access to the RAS Commander Repository or wait for Public Release in Q1 2025



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