



GenAI Tools on FASRC Clusters

Dr. Manasvita Joshi

Harvard - FAS Research Computing



Who We Are?

- Team of network, software, hardware engineers & scientists to
 - Support data-intensive & computational research across disciplines
 - Enable & facilitate collaboration, innovation, and reproducibility
- Provide research support to faculty, students, and research staff
 - Collaborate with departments, labs, and libraries to support full research lifecycles requirements.
- Expertise across high performance computing (HPC), data management, and scientific software management
 - Focus on performance, reliability, and secure research infrastructure



Our Mission

- Develop a large-scale computing infrastructure that helps researchers with the big data challenges of the 21st century
- Reduce time2science (T2S) - Lower technical barriers so researchers can focus on science, not infrastructure
- Data driven decision making to build and support our services
- Provide scalable, cost-effective computing and storage solutions to empower research at scale
- [Services Catalog | FAS Research Computing](#)



What we offer?



Learning objectives

- FASRC AI Training Material
- Harvard's GenAI Guidance
- Accessing Tools - Rest in FASRC Training Material
 - Anthropic - Harvard Subscription (Claude Code CLI)
 - Setting up [CLAUDE.md](#)
- Prompting Examples
- Summary



AI Training Material

- FASRC: Launching GenAI Tools on FASRC Clusters
 - Training Materials:
<https://docs.rc.fas.harvard.edu/kb/training-materials/>
 - Session 1:
<https://docs.rc.fas.harvard.edu/wp-content/uploads/2026/09/Gen-AI-Tools-FASRC-Clusters-Session-1.pdf>
 - Session 2:
<https://docs.rc.fas.harvard.edu/wp-content/uploads/2026/09/Gen-AI-Tools-FASRC-Clusters-session-2.pdf>
- Harvard GAI Weekly Workshops for FAS Staff
 - <https://hu.sharepoint.com/sites/FAS-GAI-StaffTrainingResources/SitePages/Weekly-AI-Workshops.aspx>



Materials Covered in Sessions

Session 1

- AI terminology: models, tools, assistants, & agents
- Harvard/HUIT GenAI guidance
- Claude Code, Codex, OpenAI API, & AWS Bedrock
- [CLAUDE.md](#)/[AGENTS.md](#) instruction files
- API keys, authentication, environments, & storage
- Cluster best practices, costs, privacy, & ethics
- Safer operating model: **Approve** → **Isolate** → **Authenticate** → **Monitor** → **Review**

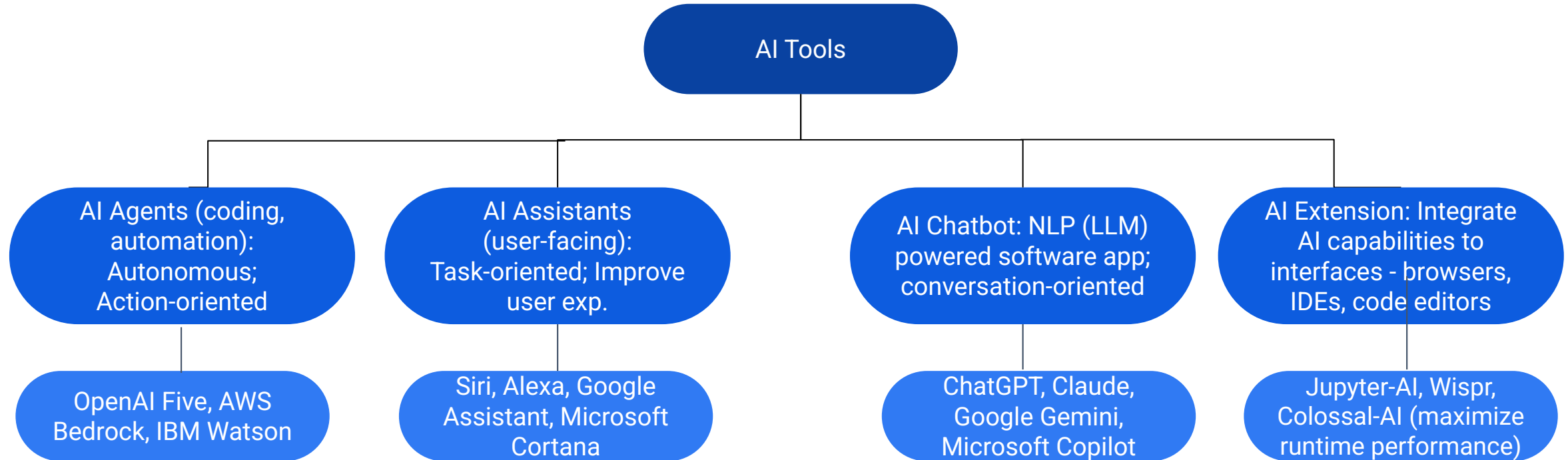
Session 2

- JupyterLab/Jupyter AI through Open OnDemand
- RStudio Server/Posit Assistant
- VS Code with Claude Code and Codex extensions
- Workspace directories and trust settings
- Permission modes and sandboxing
- Dev Containers
- GitHub Copilot Business, Ollama, and local models

AI Tool vs AI Model: Car & Engine

- AI Model (The “Engine”)
 - Algorithms/computer programs trained on large datasets to predict or generate content
 - Learn patterns, analyzes inputs, produces meaningful outputs
 - Not visible to the end user but powers the AI tool from “under the hood”
 - GPT, Llama, Claude, BERT, etc,
- AI Tool (The “Car”)
 - User-facing application that utilizes the model to solve the problem initiated by the user
 - Provides an interface to the users to interact with the underlying model
 - ChatGPT, Notion AI, Jasper, etc.
- Use AI tool to interact with AI model to get work done
- Checkout:
 - <https://medium.com/@akinbondabefe/ai-tools-vs-ai-models-think-car-engine-ef0071eabd77>
 - <https://www.linkedin.com/pulse/ai-tools-vs-models-whos-driving-future-work-arnab-bhor-8cpec/>

AI Tools



- Finer Distinction: Table in [Understanding the differences between AI agents, assistants, and other intelligent tools - Anima Blog](#)

AI Coding Agent



- Chatbot usually returns text, an agent can r/w files, execute commands, install packages, call APIs, modify repo/folders, etc.
- Agent should operate with user's permissions
- Account holder responsible for actions of their agents
- [AI Agents on the FASRC Clusters](#)

GenAI Tools @ HU

- **Harvard-wide:** <https://www.huit.harvard.edu/ai/tools>
- **General use** - No cost
 - Don't need API access
 - Proprietary closed-weight LLMs
 - Hosted on the cloud
- **Advanced use** - Additional cost
 - Don't need API access
 - Proprietary closed-weight; Hosted on the cloud
- **Specialist use** - Additional cost
 - Need API access
 - Proprietary closed-weight &/or open-weight
 - Hosted on the cloud
- Not related to FASRC clusters but good to know
- **For FAS:** <https://atg.fas.harvard.edu/ai-at-fas>



What Harvard Provides

General Tools

- Harvard AI Sandbox
- Google Gemini
- Other tools available through HUIT

Request-Based Tools

Depending on current availability and authorization:

- ChatGPT Edu
- Claude Cowork/Code
- Google Gemini Pro
- Microsoft Copilot Pro

Developer & API Access

For custom development & integration:

- Harvard API Portal
- OpenAI Direct API
- AWS Bedrock
- Harvard-approved API keys and gateways

Harvard's Generative AI Guidelines

Data Classification

Protected data (**Level 2 & above**) is **NOT** allowed for publicly available GenAI tools.

FASRC Clusters Support:

- **Cannon:** Up to Level 2
- **FASSE:** Up to Level 3

Strict Policy: Do not use personal AI accounts or subscriptions with public GenAI tools on FASRC clusters.

Workflows & Security

- Review all AI-generated content before propagation and remain vigilant against phishing risks.
- Using a Harvard email address does **NOT** convert a personal AI account into an approved research workflow.
- Always check and comply with your specific division, school, or lab policies regarding GenAI tools.

Official Resources

Access official guidelines and computing documentation:

HUIT Guidelines:

[HUIT GenAI Guidelines](#)

Research Computing:

[FASRC Cluster Computing](#)

FAS Weekly AI Workshops:

<https://hu.sharepoint.com/sites/FAS-GAI-StaffTrainingResources/SitePages/Weekly-AI-Workshops.aspx>

Key Mandate: Use Harvard-approved tools & registered API keys for all Harvard work.

Anthropic - Harvard Subscription

Use this option when

- You have an approved [Harvard Claude license](#).
- You want interactive Claude Code access.
- You do not need programmatic API billing to a research account - **AWS Bedrock** (shown in [Session 1](#))
- Your usage fits within the subscription's limits.

Typical flow

Request/obtain Harvard Claude license



Install Claude Code



Run claude



Complete Harvard/Claude sign-in



Launch from the approved project environment

GenAI Tools & Agents on Shared Cluster

- Can accelerate research on a supercomputing cluster
- You need to carefully scope, authenticate, monitor, & review their output/work to utilize them effectively & optimally
- Safe & optimal use depends on
 - Selecting the right tool
 - Providing the right context
 - Limiting agent permissions
 - Protecting data by knowing your data security level, if any
 - **Validating & verifying results** before propagating them further

FASRC Clusters: Cannon vs FASSE

Environment	Appropriate use	Data level	Interactive workflow
Cannon	General FASRC research computing	DSL1–DSL2	SSH, <code>salloc</code> , batch jobs, Open OnDemand
FASSE	Secure enclave for sensitive research	DSL3	Batch jobs and Open OnDemand/VDI; no <code>salloc</code>
DSL4–DSL5	Not supported by ordinary FASRC environments	DSL4+	Contact University Research Computing

- Cannon is approved for DSL1 and DSL2 data
- FASSE is the environment for approved DSL3 projects. DSL4 and DSL5 data are not supported by the ordinary FASRC environments. (<https://docs.rc.fas.harvard.edu/kb/data-security-levels/>)
- **Do not treat Cannon & FASSE as interchangeable. Data must not be moved from FASSE into Cannon merely because Cannon is easier to use.**
- FASSE does not permit interactive jobs through `salloc`; interactive work should use FASSE Open OnDemand/VDI instead. (<https://docs.rc.fas.harvard.edu/kb/fasse/>)

Claude Code CLI

- **Install Claude Code CLI**

```
# Login to Cannon
ssh <username>@login.rc.fas.harvard.edu

# Install Anthropic's native installer in your user space
curl -fsSL https://claude.ai/install.sh | bash

# Executable location: ~/.local/bin/claude
# Add the executable to your path (if necessary):
export PATH=$HOME/.local/bin:$PATH

# Verify installation (Advanced setup - Claude Code Docs)
claude --version (for CLI version number)
claude doctor (for environment, configuration, context usage for errors &
token waste)
```

Claude Code CLI

- **Launch, Authenticate, & Setup Claude Code CLI**

```
# Go to a project folder: cd /<path/to/a/safe/project/folder>
# Launch & Authenticate Claude:
claude
# claude auth login (if url copy-paste doesn't work from terminal)
# Follow prompts & select Enterprise subscription to login using HarvardKey
# Initialize a per-project CLAUDE.md (optional but best practice):
/init
(creates by scanning repo's file structure, dependencies, build scripts)
```

- **Installation not equivalent to authorization**

- Doesn't establish user has approved data access, API access, or permission to run the agent on a particular dataset

Claude Code Version Accumulation



Automatic Updates

Claude code auto-updates every few days, causing old versions to accumulate over time.



Manual Clean-up

Older versions are not purged automatically and can be manually removed to free up disk space.



Session & Cache Data

Be mindful of cumulative storage from old session logs, orphaned project caches, and temp files.

```
bash — Claude Code Storage Check

[jharvard@boslogin07 ~]$ which claude
~/.local/bin/claude

[jharvard@boslogin07 ~]$ ls -l ~/.local/bin/claude
lrwxrwxrwx. 1 jharvard j_lab 59 Sep 9 13:01
~/.local/bin/claude -> ../share/claude/versions/2.1.266

[jharvard@boslogin07 ~]$ ls -lh
~/.local/share/claude/versions/
total 612M
-rwxr-xr-x. 1 jharvard j_lab 206M Sep 4 16:05 2.1.261
(old)
-rwxr-xr-x. 1 jharvard j_lab 206M Sep 7 08:55 2.1.263
(old)
-rwxr-xr-x. 1 jharvard j_lab 206M Sep 9 13:01 2.1.266
[active]
```

- **Enable** /sandbox for bashtools
- **Note:** .credentials.json file stores cached tokens; disappear after session logout

What are CLAUDE.md and AGENTS.md?

- Shared context & workflow instructions for Claude agents
 - Provides guidance to agents
 - Acts as the AI's long-term memory and persistent instruction manual
 - Not a security layer; not an operating-system control, permission boundary, secret store, or a substitute for human review
- They can document:
 - Project purpose; Directory structure
 - Build and test commands; Coding conventions
 - Data-handling rules; Files that must not be changed
 - Approval requirements; Safe operational procedures; How to validate a result
- FASRC global [CLAUDE.md](#) currently underway - Additive to HUIT's administered settings

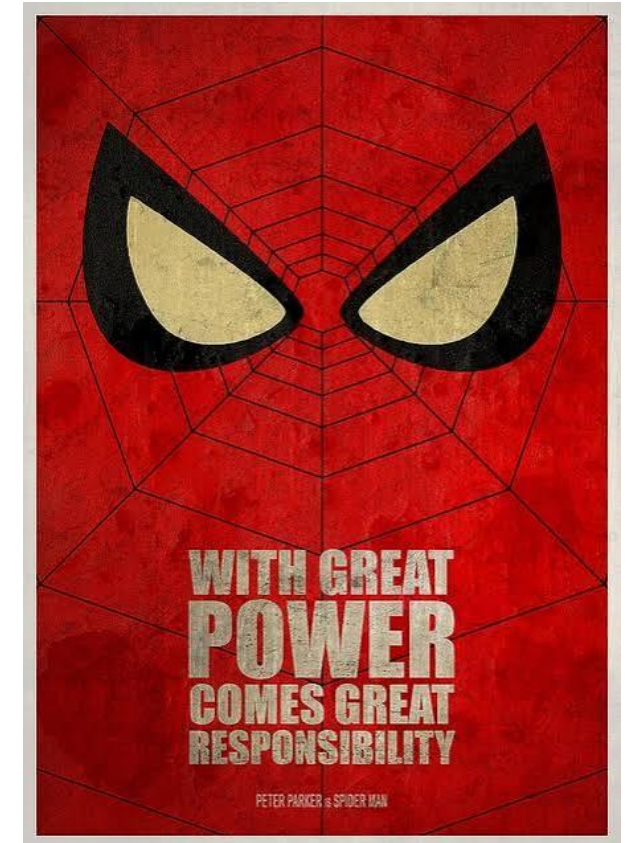
Agents Recap

An agent on a FASRC cluster may have access to:

- Research code; Shared lab directories
- Restricted or regulated data; Git repositories
- Long-running jobs; API credentials
- Large computational resources
- Other users' shared project files, depending on permissions

You need to:

- Monitor agents actively; Limit agents on login nodes;
Terminate unexpected processes



The Safer Operating Model

Approve

- Confirm that the model/API is covered by a Harvard agreement.
- Confirm that your data may be used with that service.
- Consult your PI or data steward when data-use restrictions apply.

Isolate

- Work in a project directory or temporary checkout.
- Use least-privilege file permissions.
- Prefer containers or sandboxed execution where practical.
- Do not run agents with elevated privileges.

Authenticate

- Use Harvard-provided access.
- Do not use personal API keys for work on the FASRC cluster.
- Never paste secrets into prompts or repositories.

Monitor

- Watch commands, files changed, processes, network activity, and cost.
- Stop unexpected behavior immediately.

Review

- Inspect diffs.
- Run tests.
- Check outputs for fabricated claims, data leakage, or unsafe commands.

Save on cost - optimize on prompts

- Prompt Engineering

- #  Vague prompts - "Fix my code"
- #  Clear, specific prompts

```
"""
```

```
I have a pandas DataFrame with columns ['date', 'value',  
'category'].
```

```
I need to:
```

1. Filter for category 'A' only
2. Calculate 7-day rolling mean
3. Handle NaN values

```
Current error: TypeError on groupby  
Please provide corrected code.
```

```
"""
```

- Specific details over keywords (it's not a Google search!)

Effective Prompting: PTCF+ Framework

Effective prompt is a structured brief. Organize your instruction pipeline into 2 tiers:

Tier 1: Defining assignment using setup parameters (Persona, Task, Context, Format) | **Tier 2: Defining quality** using guardrails (Plus)

Persona

Tier 1: Assignment

What role or expertise
should the system use?

Who is writing?

Task

Tier 1: Assignment

What action/task should
it perform?

Context

Tier 1: Assignment

What background and
audience information
matter?

Who it is for?

Format

Tier 1: Assignment

What should the output
look like or how to
present the results?

Plus (+)

Tier 2: Quality

External guardrails,
sources, constraints, and
success criteria.

Tip: Use what's needed for the task at hand

Better prompts do not compensate for

- Missing or poor source material
- An overly broad task
- Tooling limitations
- A model mismatched to the task
- Missing expert verification
- More elaborate prompt does not always guarantees a correct answer

Prompting Demos

SCENARIO 1: Vague / Unstructured

“Fix my analysis code.”

SCENARIO 2: Structured PTCF+ Prompt

“You are assisting with a Python data-analysis repository. First inspect the repository structure and report the files relevant to the analysis. Do not read **.env** files, SSH keys, credentials, or files outside the project directory. Do not edit files yet. Explain the likely cause of the error, propose a plan, and wait for approval before making changes. After approval, modify only the analysis script, run the existing tests on sample data, and summarize the changes and validation results.”

Highlights & Principles

FRAMEWORK ALIGNMENT

PTCF+ framework; Ethos of [CLAUDE.md](#) or [AGENTS.md](#);

SECURITY & ISOLATION

Least privilege; Workspace isolation;
Secret protection;

GOVERNANCE & QUALITY

Approval modes; Human review;
Testing & validation

From Research Question to Cluster Job

- Can AI generate a technically valid script that's also appropriate, efficient, safe, & compliant?
- How a research task becomes a cluster job?
- How to specify CPU, memory, time, partition, and GPU resources?
- How to ask GenAI to generate a Slurm script?
- How to distinguish **interactive exploration** from **batch execution**?
- How to validate an AI-generated job script before submission?
- How to diagnose pending, failed, or inefficient jobs?

Typical Workflow

1 Define the scientific or computational task.

2 Identify the program, script, or notebook to run.

3 Determine software and environment requirements.

4 Estimate CPU, memory, GPU, and time needs.

5 Select an appropriate Slurm partition.

6 Generate or write a batch script.

7 Test interactively on a small allocation.

8 Submit with **sbatch**.

9 Monitor the job.

10 Inspect outputs and resource efficiency.

Role of GenAI

What it can help with

- Translating requirements into Slurm directives
- Explaining scheduler options
- Drafting a first version of a script
- Reviewing shell syntax
- Suggesting logging and error handling
- Interpreting common job errors

What it should not decide on its own

- Which data are approved for processing
- Which lab account should be charged
- Whether a job should use GPUs
- Whether a destructive command is safe
- Whether scientific results are valid

Job-generation Prompts

Compute Resources

Partition, Number of nodes, Number of CPU cores, Memory, GPU count (if needed), Wall-clock time

Software Environment

Programming language, Module or environment, Package manager, Virtual environment name, Required libraries

Execution Details

Working directory, Input files, Output files, Logging location, Command to run, Expected exit behavior

Validation Details

Test command, Expected output, Small test dataset, Resource checks after completion

Defaults applied when options are omitted:

https://docs.rc.fas.harvard.edu/kb/running-jobs/#Slurm_Global_Limits_and_Defaults

- 1 core, 100 MB memory, no GPU, and the **serial_requeue** partition; time must be explicitly declared

Weak vs Strong Job-generation Prompt

Weak: “Create a Slurm script to run my Python code.”

- **AI guesses:**
 - Which partition? How much memory? How many cores? Is a GPU required?
 - How long will it run? Which Python version? Which environment?
 - Where are the input files?
 - Where should output and errors go?
 - Which lab account should be charged?

Stronger: “Create a Slurm batch script for FASRC Cannon.”

- **Task:** Run analysis.py once on one input file.
- **Environment:** Use the site Python module and the existing virtual environment named analysis_env.
- **Resources:** Use the test partition, 1 CPU core, 4 GB memory, and 30 minutes.
- This is a CPU-only job.
- **Files:** Working directory: /n/netscratch/<lab>/Lab/<user>/project
- **Input file:** data/sample.csv
- **Output directory:** results/
- **Requirements:**
 - - Use #!/bin/bash.
 - - Keep all #SBATCH directives together at the top.
 - - Write stdout to logs/job_%j.out.
 - - Write stderr to logs/job_%j.err.
 - - Do not include API keys or secrets.
 - - Do not delete or overwrite existing results without confirmation.
 - - Print the hostname, date, and Python version.
 - - Use `set -euo pipefail`.
 - - Include a dry-run or validation step before submission.

AI-generated Batch Script

What you would need to inspect - Don't just ask the agent to submit!!

- Is the partition appropriate?
- Is the time limit realistic?
- Is memory sufficient?
- Is the environment path correct?
- Do the input paths exist?
- Will `mkdir -p` affect anything unexpectedly?
- Will the output overwrite an existing file?
- Are logs written somewhere useful?

```
#!/bin/bash

#SBATCH --job-name=analysis
#SBATCH --partition=test
#SBATCH --cpus-per-task=1
#SBATCH --mem=4G
#SBATCH --time=00:30:00
#SBATCH --output=logs/analysis_%j.out
#SBATCH --error=logs/analysis_%j.err

set -euo pipefail

echo "Host: $(hostname)"
echo "Start: $(date)"

module load python
source /path/to/analysis_env/name activate

mkdir -p logs results

python --version
python analysis.py \
    --input data/sample.csv \
    --output results/sample_results.csv

echo "End: $(date)"
```

Safer Prompt - Batch Script

- Generate a batch script in a safer manner
- Agent drafts the job script but **you** remain the resource & submission authority
- Validate the script using Slurm's native option:

```
#SBATCH --test-only
```

or

```
sbatch --test-only <batchscript>
```

You are assisting with a Slurm job script for FASRC Cannon.

Before writing the script:

1. List the assumptions you need to make.
2. Identify any missing information.
3. Ask questions instead of guessing about:
 - partition
 - CPU count
 - memory
 - runtime
 - GPU requirement
 - lab account
 - environment path
 - input/output paths

After I answer:

- Generate a complete bash script.
- Put all #SBATCH lines together at the top.
- Do not include secrets, API keys, or personal tokens.
- Do not add destructive commands.
- Do not submit the job.
- Explain every resource request.
- Provide commands to validate the script before submission.

AI-generated Interactive Job Command

- Interactive allocation used for
 - Iterative data exploration; Debugging
 - Compiling software
 - Testing resource requirements
 - Interactive R, Python, or Jupyter work
 - Running a small sample before batch submission
- Disabled on FASSE; Use Open OnDemand ([https://docs.rc.fas.harvard.edu/kb/running-jobs/#Summary of Slurm Commands](https://docs.rc.fas.harvard.edu/kb/running-jobs/#Summary_of_Slurm_Commands))
- Prompt asks the AI to explain & validate command, not produce a string of flags

Construct an interactive salloc command for FASRC Cannon.

Goal:

Test analysis.py on data/sample.csv before submitting a batch job.

Requirements:

- CPU-only
- One CPU core
- 4 GB memory
- Maximum runtime: 30 minutes
- Use the test partition
- Do not submit or execute the command
- Explain each option
- Include the commands I should run after the allocation starts
- Include the command to exit the allocation
- Warn me if any requested option is inconsistent with FASRC policy

Using Prompt to Convert Interactive to Batch Job

Convert the following successful interactive test into an sbatch script.

Interactive command:

```
salloc -p test -c 1 --mem=4G -t 0-0:30
```

Commands run:

```
module load python  
python analysis.py \  
  --input data/sample.csv \  
  --output results/test_results.csv
```

Observed behavior:

- Runtime: approximately 10 minutes **(Do not submit runs lasting < 10mins on cluster)**
- Peak memory: approximately 2.1 GB
- CPU-only
- No GPU required

Requirements:

- Use a production-appropriate partition rather than test.
- Request 1 CPU core, 4 GB memory, and 20 minutes.
- Include stdout and stderr log files with %j.
- Do not delete existing output.
- Include a clear job name.
- Explain which assumptions require confirmation before submission.
- Do not submit the job.

Prompt to Diagnose Your Job

```
This FASRC Slurm job <JOBID> failed.
```

```
Job state:  
FAILED
```

```
Relevant stderr:  
[paste only non-sensitive error text]
```

```
Submission script:  
[paste the script, excluding secrets]
```

```
Please:
```

1. Identify the most likely failure point.
2. Separate scheduler errors from application errors.
3. Suggest the smallest safe correction.
4. Do not recommend increasing resources unless the evidence supports it.
5. Do not modify or resubmit the job.

Prompting is an iterative process

Prompting to Improve Job Efficiency

- Weak: “Make my job faster.”
- Stronger: Relevant specifications provided
- Be aware of your job efficiency:
<https://docs.rc.fas.harvard.edu/kb/job-efficiency-and-optimization-best-practices/>

Review this Slurm script and the completed-job resource report.

Goal:

Reduce requested resources without increasing the risk of failure to improve job efficiency.

Compare:

- Requested CPU cores
- Requested memory
- Elapsed time
- Maximum resident memory
- Total CPU time
- Job state

Return:

1. A table comparing requested versus observed resources.
 2. Any evidence of over-requesting.
 3. Any evidence of under-requesting.
 4. A revised script only if justified.
 5. Risks and assumptions.
- Do not change the partition or add GPUs without evidence.

What an Agent Must Not Do Automatically

- Needs explicit approval before or after validation/verification
 - Submitting or canceling job arrays: Running `sbatch`; `salloc`; `scancel`
 - Requesting GPUs or large-memory nodes
 - Modifying shared or production data
 - Changing file permissions
 - Installing software under your user profile (system-wide not allowed to users)
 - Making network requests
 - Reading credentials or `.env` files
 - Deleting or overwriting results
- Conduct **dry runs** to verify/validate your runs before sending them through a pipeline

What a Gen-AI Assisted Workflow Could Look Like

1. Describe the scientific task
 2. Ask GenAI to identify missing resource information
 3. Provide environment and file details
 4. Ask for an sbatch script draft & inspect the script
 6. Run bash syntax checks
 7. Use --test-only where appropriate
 8. Test a small sample with salloc
 9. Verify requested resources
 10. Submit with sbatch where appropriate
 11. Monitor job(s); Inspect output, errors, & results.
 13. Adjust the script only after evidence-based review.
- You are **responsible** for
 - Data authorization
 - Account selection
 - Resource approval
 - Job submission
 - Scientific interpretation
 - Final validation
 - ***Help the agent to help you***

Keep in Mind

GenAI can help:

- Translate requirements into Slurm syntax
- Draft batch scripts
- Explain scheduler flags
- Debug shell and environment errors
- Convert interactive tests into batch jobs
- Interpret job reports
- Improve resource requests

GenAI cannot replace:

- Cluster policy knowledge
- Data governance
- Security judgment
- Scientific expertise
- Human approval
- Validation of results

Generate with AI. • Submit deliberately. • Monitor actively. • Validate independently

Guardrails Emphasized Throughout

1. Approve

- Confirm that the model, API, and account are approved.
- Confirm that the data may be used with that service.
- Consult the PI or data steward when restrictions apply.

2. Isolate

- Use a project directory or temporary checkout.
- Do not start an agent in `$HOME`.
- Prefer a sandbox or dev container when available.
- Use least-privilege access.

3. Authenticate

- Use Harvard-managed subscriptions or registered API keys.
- Never use personal API keys for Harvard work on the cluster.
- Never paste secrets into prompts or repositories.

4. Monitor

- Watch commands, files changed, processes, network, GPU use, and cost.
- Do not leave memory-intensive tools running unnecessarily.
- Stop unexpected behavior immediately.

5. Review

- Inspect diffs.
- Run tests.
- Check Slurm output & resource usage.
- Review scientific assumptions and results.
- Look for hallucinated references, unsafe commands, and data leakage.

To Summarize: Asking Before Doing

1. Approval

Is this tool and account approved?

2. Data Exposure

What data will leave the cluster or be exposed to the tool?

3. Access Scope

What files and commands can the agent access?

4. Resources

What resources will this workflow consume?

5. Validation

How will I validate the code, job, and scientific result?

Core Principle

Use GenAI to accelerate your workflow—not to outsource responsibility for your data, cluster account, computational resources, or scientific conclusions

Be Mindful Of

Login-node use, accidental delete permissions, prompt-injection attacks, data leakage, and data-security levels when running AI workflows

(<https://docs.rc.fas.harvard.edu/kb/fasrc-ai-facilitation/>)

FASRC Support

- FASRC Docs: [FASRC DOCS](#)
- Training
 - Calendar: [Training Calendar | FAS Research Computing](#)
 - Material: [Training Materials – FASRC DOCS](#)
- Getting help
 - Office hours: [Virtual Office Hours | FAS Research Computing](#)
 - Ticket:
 - HUIT Service Portal -> Submit Ticket: [Submit Ticket - IT Help](#)
 - Email: rchelp@rc.fas.harvard.edu



Thank You (& thanks to Harvard AI Sandbox OpenAI GPT-5.6 Luna model)!
Questions? Comments?