

# Git, GitHub, and VS Code

## Agentic AI for Project Management and Research Productivity

Tomas Larroucau

Arizona State University

January 9, 2026

# Workshop Overview

- 1 Introduction
- 2 VS Code: Your Research Environment
  - Project Structure
- 3 Git: Version Control Fundamentals
- 4 GitHub: Collaboration Platform
- 5 Agentic AI for Research
- 6 Putting It All Together

# Why This Workshop?

- Modern research requires computational reproducibility
- Collaboration demands version control and project management
- AI tools are transforming how we write code and documents
- Integration of these tools creates powerful workflows

**Goal:** Learn to combine VS Code, Git/GitHub, and AI assistants for efficient, reproducible research.

**“We’ll let the AI roam free... but set up proper guardrails.”**

# The “Jur-Al-ssic Park” Approach to Research

## **The Park**

*(VS Code)*



The  
Environment

The island where everything lives

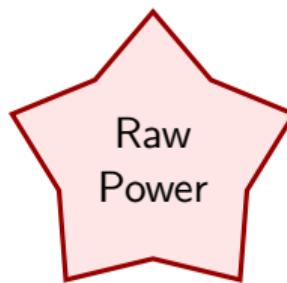
# The “Jur-Al-ssic Park” Approach to Research

## The Park (VS Code)



The island where everything lives

## The Dinosaurs (AI Agents)



Powerful, fast, but chaotic

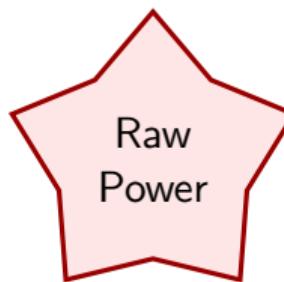
# The “Jur-Al-ssic Park” Approach to Research

## The Park (VS Code)



The island where everything lives

## The Dinosaurs (AI Agents)



Powerful, fast, but chaotic

## The Guardrails (Git & GitHub)



Keeps the chaos contained

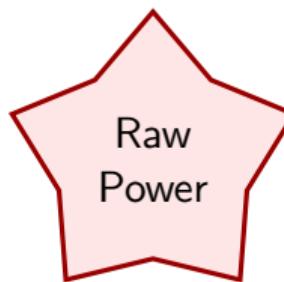
# The “Jur-Al-ssic Park” Approach to Research

## The Park (VS Code)



The island where everything lives

## The Dinosaurs (AI Agents)



Powerful, fast, but chaotic

## The Guardrails (Git & GitHub)



Keeps the chaos contained

*“Your scientists were so preoccupied with whether or not they could, they didn’t stop to think if they should... Git lets you undo if they shouldn’t.”*

# What You'll Learn

## Module I: Foundations

- VS Code as integrated development environment
- Git fundamentals: commits, branches, merges
- GitHub workflows: issues, pull requests, project boards

## Module II: Agentic AI

- VS Code Chat: Ask, Edit, and Agent modes
- GitHub Copilot: local and cloud workflows
- AI-assisted code review and refactoring
- Complementary tools: Refine, NotebookLM, Elicit

# Workshop Materials

- **Template Repository:** Complete research project structure
- **Sample Data:** CSV files for demonstration
- **Python Scripts:** Analysis, plotting, table generation
- **LaTeX Templates:** Paper and slides
- **Makefile:** Automated workflow
- **Documentation:** README, agent instructions

Repository available at: [https://github.com/tlarroucau/AI\\_workshop](https://github.com/tlarroucau/AI_workshop)

# Workshop Prerequisites

## 1. Accounts

- **GitHub Account:** Create at [github.com](https://github.com)
- **Education Benefits:** Apply for student/educator pack at [education.github.com](https://education.github.com) (Upgrade to Pro!)

## 2. Software

- **VS Code:** Download from [code.visualstudio.com](https://code.visualstudio.com)
- **Git:** Install from [git-scm.com](https://git-scm.com)
- *Optional:* Python & LaTeX distribution

## 3. Extensions & Setup

- **Link GitHub:** Sign in to GitHub within VS Code (Accounts menu)
- **Copilot:** Install "GitHub Copilot" extension (Required for Module I & II)
- **MCP Server:** Will be configured in Module II (Agentic AI)

### Note on Access

GitHub Copilot is now free! However, upgrading to Copilot Pro (via Education pack) offers better models and limits. Verification can take a few days.

## WORKSHOP SECTION

# Module I

---

## Foundations: VS Code, Git & GitHub

- ① VS Code as integrated development environment
- ② Git fundamentals: commits, branches, merges
- ③ GitHub workflows: issues, pull requests, project boards

# What is VS Code?

## Visual Studio Code

- Free, open-source editor by Microsoft
- Cross-platform (Windows, Mac, Linux)
- Extensible via marketplace
- Integrated terminal
- Built-in Git support
- AI assistant integration

## Why VS Code for Research?

- Write code *and* papers in one place
- Manage entire project lifecycle
- Collaborate seamlessly
- Automate repetitive tasks
- Leverage AI for productivity

# Essential VS Code Features

## Navigation & Interface:

- **Ctrl/Cmd+Shift+P**: Command Palette
- **Ctrl/Cmd+P**: Quick file open
- **Ctrl+`**: Toggle terminal
- **F5**: Start debugger

## Multi-Cursor Magic:

- **Ctrl/Cmd+D**: Select next occurrence
- **Alt+Click**: Add cursor anywhere
- **Ctrl/Cmd+Shift+L**: Select all occurrences

**Pro tip:** Multi-cursor editing (Ctrl+D) is a game changer for renaming variables!

## Code Editing:

- **Alt+↑/↓**: Move line up/down
- **Ctrl/Cmd+/**: Toggle comment
- **Ctrl/Cmd+Shift+K**: Delete line
- **Ctrl/Cmd+Shift+D**: Duplicate line

## Code Navigation:

- **Ctrl/Cmd+Click**: Go to definition
- **Alt+←/→**: Navigate back/forward

# Essential Extensions for Research & Productivity

Extension	Purpose
<i>AI &amp; Productivity</i>	
GitHub Copilot	AI code assistant
Copilot Chat	AI chat interface
Codex	Cloud AI coding agent
Continue	Local AI assistant
Cline	Autonomous AI agent
Prompt Flow	MCP integration
VS Code Speech	Speech-to-text input
<i>Data Science</i>	
Python	Language support
Pylance	IntelliSense, type check
Jupyter	Notebook support
R	R language support
Stata Enhanced	Stata syntax
<i>Documentation</i>	
LaTeX Workshop	Compile LaTeX
LTeX	Grammar/spell check
Markdown All in One	Markdown preview

Extension	Purpose
<i>Project Management</i>	
GitHub PR & Issues	Manage PRs/issues
GitLens	Git visualization
Git History	View git log
Project Manager	Organize projects
Todo Tree	Track TODOs
<i>Data &amp; Utilities</i>	
Rainbow CSV	CSV colorization
Excel Viewer	View Excel files
PDF Viewer	Preview PDFs
Code Snap	Code screenshots
Auto Align	Align code formatting
<i>Development Tools</i>	
Remote SSH	Remote development
Docker	Container support
Error Lens	Inline diagnostics
Live Server	Local web server

# VS Code Workspace Setup

## Recommended Workspace Structure:

- Open entire project folder as workspace
- Use multi-root workspaces for complex projects
- Configure settings per workspace (Python path, linters, etc.)
- Save workspace file (.code-workspace) for team sharing

## Settings Sync:

- Sync extensions and settings across machines
- Use GitHub or Microsoft account
- Maintain consistency in team environments

## Real-Time Collaboration:

- **Live Share extension:** Co-edit files with co-authors in real-time
- Share terminals, debuggers, and servers
- Great for remote pair programming and debugging sessions

# Common Frictions in Academic Research

## Integration Challenges:

### Statistical Software

- **Stata/MATLAB**: Not natively integrated in VS Code
- **Solution**: Run through integrated terminal with commands
- Extensions available for syntax highlighting
- Execute code blocks via terminal shortcuts

### Overleaf Integration

- **Via Dropbox**: Sync local folder with Overleaf project (some lag)
- **Via Git**: Clone Overleaf project, push/pull changes (more control)
- **Alternative**: Use LaTeX Workshop extension in VS Code directly and Copy/Paste files
- Trade-off: Real-time collaboration vs. local control

**Bottom line:** Not perfect, but workable with some adjustments!

# Reproducible Research Template

## Repository Structure:

- `data/`: Raw and processed data
- `scripts/`: Analysis code
- `output/`: Generated figures and tables
- `tex/`: LaTeX documents (paper, slides)
- `Makefile`: Automation workflow
- `README.md`: Project documentation
- `.gitignore`: Excluded files

**Key Principle:** Everything generated from scripts, nothing manual!

# The Makefile Approach

## Why Makefile?

- Automate entire workflow
- Document dependencies
- One command rebuilds everything
- Reproducibility guarantee

### Example Targets

```
make all      # Run complete pipeline
make data     # Generate/process data
make analysis # Run statistical analysis
make figures  # Create plots
make tables   # Generate LaTeX tables
make paper    # Compile PDF
make clean    # Remove generated files
```

**Demo:** Running the complete workflow

# Data Management

## Raw Data

- Store in `data/raw/`
- Never modify original files
- Commit to Git (if reasonable size)

## Processed Data

- Save to `data/processed/`
- Generate from scripts
- Add to `.gitignore` (reproducible)

## Output Files

- Figures: PDF + PNG in `output/figures/`
- Tables: LaTeX in `output/tables/`
- Rebuild via Makefile

# Code Organization

## Python Scripts Best Practices:

- **Modular:** Separate utility functions from main analysis
- **Documented:** Docstrings for all functions
- **Typed:** Use type hints for clarity
- **Styled:** Follow PEP 8 conventions
- **Tested:** Include basic validation

## Example Structure:

- `utils.py`: Helper functions, plotting utilities
- `analysis.py`: Main analysis script
- `requirements.txt`: Python dependencies

**Demo:** Code structure in template repository

# LaTeX Integration

## Automated Table Generation:

- Python scripts create .tex files
- Use `pandas.DataFrame.to_latex()`
- Format with `booktabs` package
- Include via `\input{}` in main document

## Figure Inclusion:

- Save plots as PDF (vector) and PNG (preview)
- Use `\includegraphics{}`
- Relative paths from tex directory
- Captions and labels for cross-reference

## Benefits:

- No manual copy-paste errors
- Update data → regenerate everything
- Always in sync with analysis

# VS Code Chat Modes

## ① Ask Mode (Chat Panel)

- Answer questions about code
- Explain complex functions
- Suggest best practices

## ② Edit Mode (Inline)

- Modify existing code
- Refactor functions
- Apply changes directly

## ③ Agent Mode (@workspace)

- Multi-file operations
- Workspace-wide changes
- Project scaffolding

## ④ Plan Mode (New!)

- High-level reasoning
- Break down complex tasks
- Create implementation plans

**Advanced Feature:** *Agent Sessions* maintain context across multiple interactions, allowing for iterative refinement of complex tasks without losing history.

**Demo:** Using different chat modes

# Copilot Workspace Instructions

## What are workspace instructions?

- File: `.github/copilot-instructions.md`
- Guide Copilot about your project
- Define coding standards and patterns
- Explain project structure
- Specify workflows and conventions

## Benefits:

- Consistent AI suggestions
- Project-specific context
- Reduced need for prompting
- Better code generation

**Demo:** Copilot instructions in template repo

# AI-Assisted Workflows: Local

## Local AI Agent Workflow:

- ① Open repository in VS Code
- ② Use Copilot Chat to understand codebase
- ③ Generate code with suggestions
- ④ Refactor existing code
- ⑤ Write tests and documentation

## Example Tasks:

- "Add a function to compute standard errors"
- "Write docstrings for all functions in this file"
- "Explain what this regression model does"

**Demo:** Live coding with Copilot

# What is Git?

- **Distributed version control system**
- Tracks changes to files over time
- Enables collaboration without conflicts
- Essential for reproducible research

## Why Git for Research?

- Complete history of your work
- Experiment safely with branches
- Collaborate with co-authors
- Publish code alongside papers
- Recover from mistakes

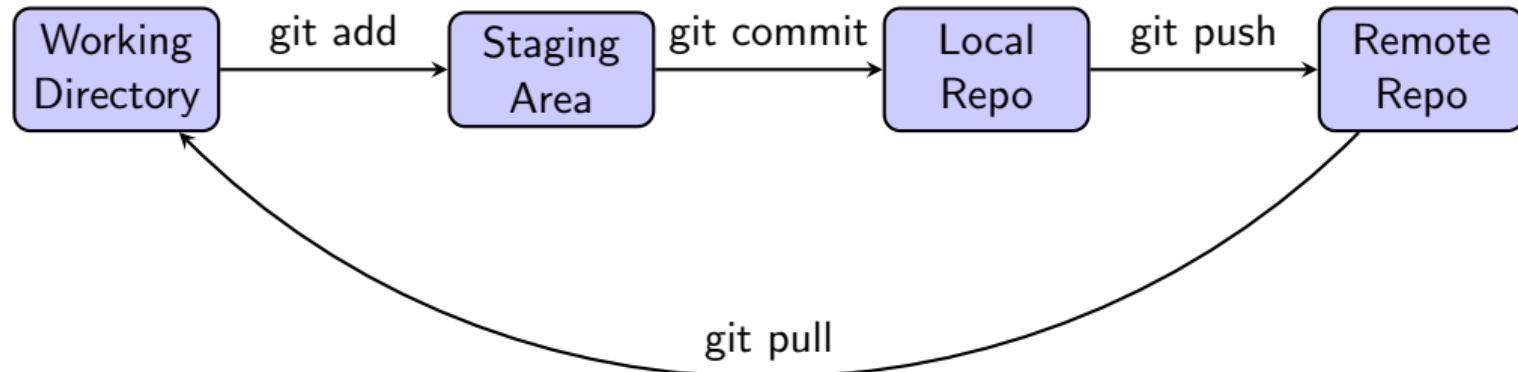
# Git Core Concepts

- ① **Repository (repo)**: Project folder tracked by Git
- ② **Commit**: Snapshot of your project at a point in time
- ③ **Branch**: Parallel version of your repository
- ④ **Merge**: Combine changes from different branches
- ⑤ **Remote**: Repository hosted online (e.g., GitHub)

## Basic Workflow

```
git add file.py          # Stage changes
git commit -m "message" # Save snapshot
git push                  # Upload to remote
```

# The Git Workflow



- Edit files in working directory
- Stage changes you want to commit
- Commit creates permanent snapshot
- Push shares with collaborators

# Essential Git Commands

## Repository Setup

```
git init                      # Initialize new repo  
git clone <url>              # Copy remote repo
```

## Daily Workflow

```
git status                     # Check current state  
git add <file>                # Stage specific file  
git add .                      # Stage all changes  
git commit -m "msg"            # Commit with message  
git fetch                      # Download remote changes  
git pull                       # Fetch + merge changes  
git push origin main          # Push to remote  
git log                        # View commit history  
git diff                       # See unstaged changes
```

# Branching Strategy

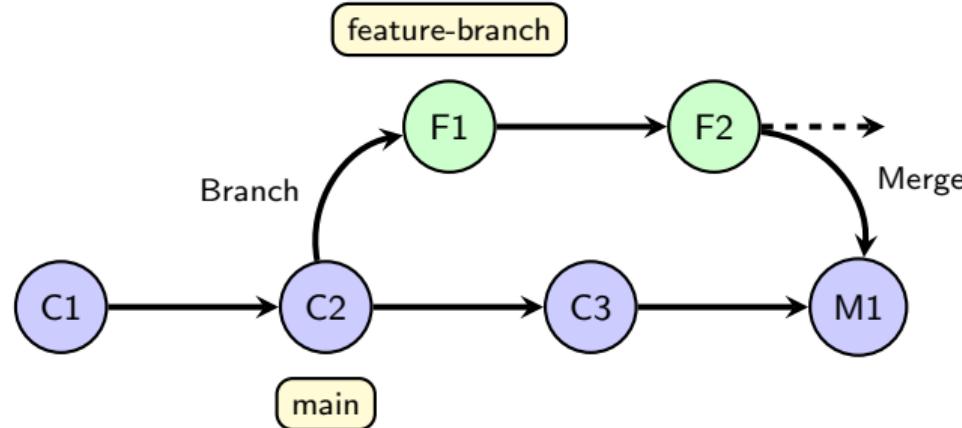
## Why branches?

- Isolate experimental work
- Develop features independently
- Keep main branch stable

## Common workflow:

- ① Create branch for new feature: `git checkout -b feature-name`
- ② Make changes and commit
- ③ Push branch: `git push -u origin feature-name`
- ④ Create pull request on GitHub
- ⑤ Review and merge
- ⑥ Delete branch after merge

# Visualizing Branches



- **Branch:** Create a parallel timeline (`git checkout -b`)
- **Commit:** Work on the feature branch (F1, F2)
- **Merge:** Combine feature history back into main (`git merge`)
- **Note:** *The branch remains open after merging unless deleted*

# Git Best Practices

## Commit Messages

- Be descriptive: "Add regression analysis for Model 2"
- Use imperative mood: "Fix data loading bug"
- Reference issues: "Closes #42"

## What to Commit

- **DO:** Source code, scripts, documentation
- **DO:** Raw data (if reasonable size)
- **DON'T:** Generated outputs (rebuild from scripts)
- **DON'T:** Large binary files (use Git LFS if needed)
- **DON'T:** Passwords or API keys

# The .gitignore File

## What is .gitignore?

- Tells Git which files to ignore (never commit)
- Prevents committing generated files, credentials, or large binaries
- One per repository (in root directory)

## Example .gitignore for Research Projects:

```
# Python                                # LaTeX
__pycache__/
*.pyc
.ipynb_checkpoints/
*.egg-info/

# R                                     *.log
.Rhistory                               *.out
.RData                                    *.fls
.Rproj.user/                             *.synctex.gz

# Stata                                 # Generated outputs
*.dta~                                  output/
*.log                                    *.pdf
                                         *.png
```

# What is GitHub?

## GitHub is...

- Web-based Git hosting
- Social coding platform
- Project management tools
- Collaboration infrastructure
- Portfolio for researchers

## Key Features:

- Remote repository hosting
- Pull requests
- Issues and project boards
- Actions (CI/CD)
- Pages (documentation)
- Copilot (AI assistant)

**Free for academic use!** ([github.com/education](https://github.com/education))

# GitHub Workflow: Issues

## What are Issues?

- Track tasks, bugs, feature requests
- Organize work with labels and milestones
- Assign to team members
- Reference in commits and PRs

## Issue-Driven Development:

- ① Create issue: "Add robustness checks"
- ② Create branch: `git checkout -b issue-42-robustness`
- ③ Work on feature, commit with "Addresses #42"
- ④ Create pull request
- ⑤ Merge and close: "Closes #42"

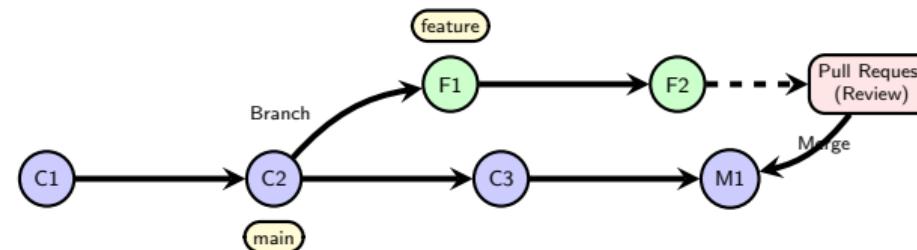
**Demo:** Creating and managing issues

# GitHub Workflow: Pull Requests

## What are Pull Requests (PRs)?

- Propose changes to repository
- Enable code review before merging
- Discuss implementation details
- Run automated tests

## PR Workflow:



- ① Create and push feature branch
- ② Open Pull Request on GitHub
- ③ Review, discuss, and fix
- ④ Merge when approved

**Demo:** Creating and reviewing pull requests

# GitHub Project Boards

## Organize Research Projects:

- Kanban-style boards
- Columns: To Do, In Progress, Done
- Link to issues and PRs
- Automate card movement

## Example Workflow:

- ① Create project board for paper
- ② Add columns for analysis, writing, revisions
- ③ Create issues for each task
- ④ Move cards as work progresses
- ⑤ Track progress visually

**Demo:** Setting up a project board

# Collaboration Best Practices

## Repository Setup

- Clear README with setup instructions
- .gitignore for generated files

## Team Workflow

- Never commit directly to main
- All changes via pull requests
- Require reviews before merging

## Communication

- Issues for tasks and discussions
- PR comments for code feedback
- Wiki for documentation

## WORKSHOP SECTION

# Module II

---

## Agentic AI for Research Productivity

- ① VS Code Chat: Ask, Edit, and Agent modes
- ② GitHub Copilot: local and cloud workflows
- ③ AI-assisted code review and refactoring
- ④ MCP: Connecting AI to external tools

# What is Agentic AI?

## Traditional AI Assistants

- Respond to queries
- Generate code snippets
- Provide suggestions

## Agentic AI

- Autonomous task completion
- Multi-step reasoning
- Context-aware assistance

## Key Tools:

- GitHub Copilot (local & cloud)
- VS Code Chat modes
- Cline + Continue (offline capable!)
- Cursor, Windsurf

# GitHub Copilot Overview

## Copilot Features:

- Code completion
- Chat interface
- Inline suggestions
- Whole function generation
- Documentation writing
- Code explanation

## Use Cases:

- Write boilerplate code
- Debug errors
- Refactor functions
- Generate tests
- Write docstrings
- Translate code

**Free for students and educators!**

Apply at: [education.github.com](https://education.github.com)

# AI-Assisted Workflows: Cloud

## **GitHub Copilot Workspace / Claude Code / Codex / Jules (Cloud):**

- Work on issues directly in browser
- AI proposes implementation plan
- Review and refine suggestions
- Create PR automatically
- Collaborate asynchronously!

## **Use Cases:**

- Quick fixes from mobile/tablet
- Delegate tasks to AI agent
- Review AI-generated solutions

**Demo:** Creating issue and using Copilot Workspace and Codex

# Complementary Research Tools

Tool	Purpose
<b>Refine</b> (refine.ink)	AI-powered writing revision and style improvement Interactive editing with suggestions
<b>NotebookLM</b> (Google)	Structured reading and note-taking Create study guides from papers Create academic podcasts!
<b>Elicit</b>	Literature discovery and synthesis Extract data from papers
<b>GPT/Claude/Gemini</b>	General research assistance Draft writing, brainstorming

**Integration:** Use alongside Git/GitHub workflow for complete research pipeline

# AI Best Practices

## DO:

- Review all AI-generated code
- Test suggestions before committing
- Understand what the code does
- Use AI to learn new techniques
- Iterate on prompts for better results

## DON'T:

- Blindly accept all suggestions
- Share sensitive data with AI
- Rely on AI for critical decisions
- Use AI to write entire papers

**Remember:** AI is a tool, not a replacement for thinking!

# Data Privacy & Security Considerations

## Follow Your Institution's Guidelines:

- Always comply with ASU (or your institution's) data security policies
- Review AI tool terms of service for data retention policies!
- Be cautious with sensitive, proprietary, or confidential data

## When Cloud AI is Not an Option:

- If data privacy or code confidentiality is a major concern
- If institutional policies prohibit cloud-based AI tools
- If working with regulated data (HIPAA, FERPA, etc.)

## Fully Local AI Solutions:

- **Continue + Ollama:** Run local LLMs (Llama, CodeLlama, etc.)
- **Cline + local models:** Autonomous agent without cloud
- Complete offline workflow: No data leaves your machine
- Trade-off: Lower performance than cloud models, but complete privacy

# Using AI APIs in Your Code

## When to Use AI APIs Directly:

- Automate repetitive data processing tasks (classify text, extract entities)
- Generate synthetic data for testing or augmentation
- Create automated reports or summaries from analysis results

### Example: OpenAI API

- Text generation/completion
- Code generation
- Data classification
- Embeddings for similarity

### Example: Anthropic Claude

- Long document analysis
- Complex reasoning tasks
- JSON mode for structured output
- Vision for image analysis

**Best Practice:** Store API keys in environment variables, never commit them to Git!

# MCP: Model Context Protocol

## What is MCP?

- Protocol that gives AI agents access to external tools and services
- Enables *direct interaction* with APIs (not just generating code for you)

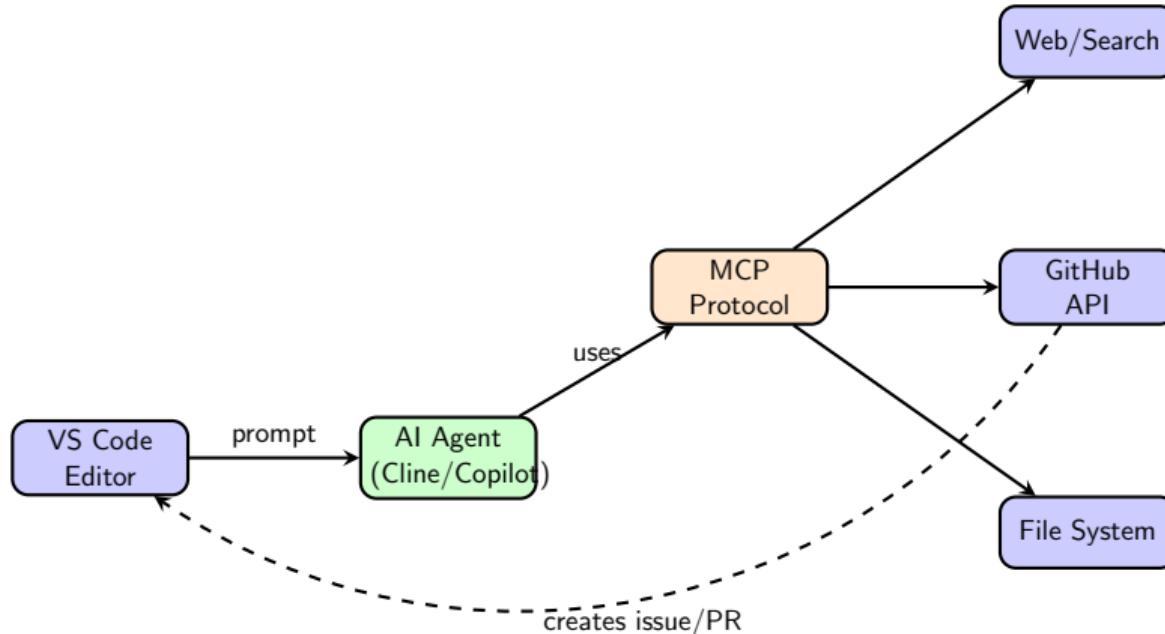
## Key MCP Servers:

- **GitHub**: Create issues, PRs, manage boards
- **Filesystem**: Access external files/folders
- **Brave Search**: Web search with sources
- **Fetch**: Download/parse web content

## Setup (GitHub MCP):

- 1 Install MCP server (`npx`)
- 2 Get GitHub token
- 3 Configure in Copilot settings
- 4 Done!

# How AI Tools Connect: The Big Picture



## Example Flow:

- 1 You: "Create issue for clustered SE analysis and add to project board"
- 2 AI Agent → MCP → GitHub API → Issue created + added to board
- 3 VS Code shows notification with issue link
- 4 Agent can then create branch, write code, commit, create PR automatically!

# Complete Research Workflow

## 1. Setup

- Clone template repository
- Set up Python environment
- Configure VS Code

## 2. Development

- Create issues for tasks
- Work in feature branches
- Use AI for code generation
- Commit regularly

## 3. Collaboration

- Push branches to GitHub
- Create pull requests
- Review code
- Merge to main

## 4. Publication

- Run `make all`
- Compile paper and slides
- Share repository with paper

# Live Demo: End-to-End Example

**Scenario:** Add a new robustness check using MCP-enabled AI agent

- ① Ask AI agent: “Create issue for adding clustered SE robustness check”
- ② Agent uses GitHub MCP to create issue and add to project board
- ③ Agent creates branch: feature/clustered-se
- ④ Agent uses Copilot to generate implementation code
- ⑤ Update analysis script with AI assistance
- ⑥ Run `make all` to regenerate outputs
- ⑦ Agent commits changes with descriptive message
- ⑧ Agent creates PR linking to original issue
- ⑨ Review PR and merge to main

**We'll work through this together - from idea to merged code in minutes!**

# Common Challenges & Solutions

## Understanding Merge Conflicts:

- Occur when Git cannot automatically determine which version to keep
- Happen when the same lines of code/text are edited in different branches
- VS Code highlights conflicts and lets you choose which version to keep
- AI agents can suggest the best resolution strategy!

## Backing Up Your Work:

- You can keep your local repo in Dropbox for automatic backup
- **WARNING:** Do not share that Dropbox folder with co-authors!

Challenge	Solution
Merge conflicts	Use VS Code conflict resolver, ask AI agent for help
Large files	Use .gitignore, Git LFS if needed
Slow Git operations	Use .gitignore for generated files
Lost work	Commit often, use branches
Unclear AI output	Refine prompts, add context
Reproducibility	Use Makefile, document dependencies

# Next Steps

## First:

- ① Fork/clone workshop template repository
- ② Install VS Code and extensions
- ③ Set up GitHub account (education benefits)
- ④ Practice basic Git commands
- ⑤ Run `make all` to build template
- ⑥ Ask your favorite AI agent to explain any concepts from the slides you don't understand

## Second:

- Apply workflow to a small project
- Create repository for current research
- Experiment with Copilot
- Set up project board

# Resources

## Documentation

- Git: [git-scm.com/doc](http://git-scm.com/doc)
- GitHub: [docs.github.com](https://docs.github.com)
- VS Code: [code.visualstudio.com/docs](https://code.visualstudio.com/docs)

## Learning

- GitHub Skills: [skills.github.com](https://skills.github.com)
- Software Carpentry: [software-carpentry.org](https://software-carpentry.org)

## Workshop Materials

- Template: [github.com/tlarroucau/AI\\_workshop](https://github.com/tlarroucau/AI_workshop)
- Slides: [tex/slides/](#)

**This ENTIRE workshop was created with ONE prompt and  
FOUR hours of edits!**

# Questions?

Thank you for attending!

**Contact:** Tomas.Larroucau@asu.edu

**Template Repository:**

[github.com/tlarroucau/AI\\_workshop](https://github.com/tlarroucau/AI_workshop)