

How to Make Your Science Easier with AI Support

Tianyuan Liu

Date: October 3rd, 2025

Time: 10:00–12:00

Location: I2SysBio Seminar Room

by our PhD student: **Tianyuan Liu**

October 3rd, 2025

Genomics 
 of Gene
Expression Lab

Outline

1. Introduction to AI-Assisted Development
2. Context Engineering
3. Demo Highlights
4. Best Practices and Resources
5. Conclusion

LLM vs. AI Agent

Large Language Model

- ▶ Cannot use tools

Example: ChatGPT (basic mode)

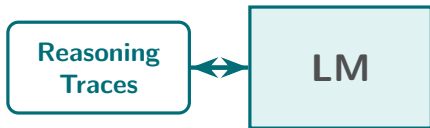
AI Agent

- ▶ Tools: Execute actions
- ▶ Memory: Context retention
- ▶ Planning: Multi-step tasks

Example: Cursor, Gemini CLI

$\text{Agent} = \text{LLM} + \text{Tools} + \text{Memory} + \text{Planning}$

Reason Only and Act Only



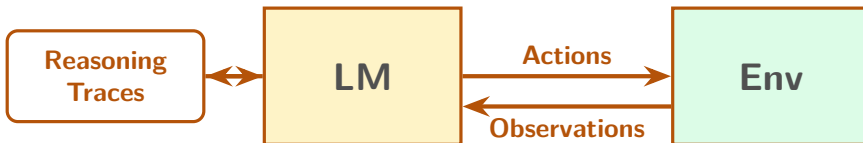
Reason Only



Act Only

ReAct (Reason + Act)

ReAct



(Reason + Act)

Why Do We Need MCP?

The Missing Link

ReAct requires **tools** to interact with the environment — but how does the LM find and use them?

The Challenge

- ▶ Each tool has unique API
- ▶ Manual integration required
- ▶ LM can't discover tools

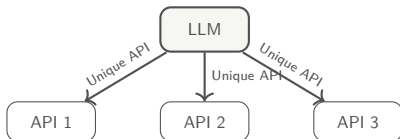
The Solution: MCP

- ▶ Standardized protocol
- ▶ Auto tool discovery
- ▶ LM acts with environment

MCP = The bridge enabling ReAct to work in practice

API vs MCP: Before and After

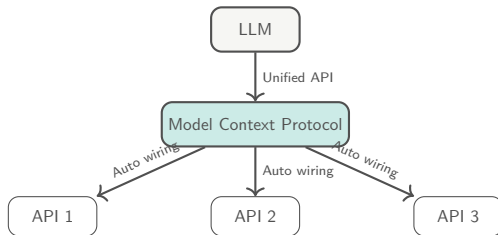
Before MCP



Traditional API

You write code

After MCP



MCP

AI uses tools

MCP = APIs that AIs can find and use by themselves

API vs. MCP: Concrete Example

Task: “What’s the weather in Valencia?”

Using Traditional API

You must code:

```
1 import requests
2
3 def get_weather(city):
4     url = "https://api.weather.com"
5     params = {"city": city}
6     response = requests.get(url, params)
7     data = response.json()
8     return data["temp"]
9
10 # Call it manually
11 temp = get_weather("Valencia")
12 print(f"Temperature: {temp}C")
```

**Problem: You write & maintain all
integration code**

Using MCP

MCP Server provides:

```
1 {
2     "name": "get_weather",
3     "description": "Get current weather",
4     "parameters": {
5         "location": {
6             "type": "string",
7             "description": "City name"
8         }
9     }
10 }
```

AI automatically:

- ✓ Reads the schema
- ✓ Knows when to call it
- ✓ Passes correct parameters
- ✓ Interprets the response

How LLMs Learn Tool Use

Pretraining

- ▶ Massive web + code
- ▶ Predict next tokens
- ▶ Learn world priors



Instruction Tuning

- ▶ Prompt-response demos
- ▶ Tool schemas inline
- ▶ Structured formats



Alignment / RLHF

- ▶ Rewarded feedback
- ▶ Safety objectives
- ▶ Reliable tool calls

Takeaway

Layered training primes agents for confident, auditable tool calls.

ReAct in Action: Example

Task: “What’s the weather in Valencia today?”

1. **Thought:** “Need weather data, use MCP tool.”
2. **Action:** `get_weather(location="Valencia, Spain")`
3. **Observation:** “Temperature: 24°C, Sunny”
4. **Thought:** “Got data, formulate response.”
5. **Action:** Return response to user

Result: “Valencia today is sunny, 24°C.”

Definition

ReAct loops for planning, writing, testing & refining code autonomously.

ReAct Cycle

1. Think → Code
2. Test → Adjust
3. Repeat until done

Examples

- ▶ Cursor
- ▶ GitHub Copilot
- ▶ Gemini CLI

How does this loop play out in the lab?

Live Demo: Coding Agent for Research

Goal Turn the epigenome review into polished slides with a ReAct loop.

Setup

- ▶ Workspace: `examples/demo1/`
- ▶ Read `prompt.txt`
- ▶ Use Conesa theme

Loop

1. Think: ground in prompt + paper
2. Act: write slides, manage assets
3. Observe: compile, adjust

Checklist

Compile cleanly; figure rule respected; log key decisions.

Coding Agent Example: RNA-Seq

Scenario

Raw FASTQs → polished expression report by morning.

ReAct Loop in Practice

1. **Think:** Review sample sheet & QC goals
2. **Act:** Run FastQC + STAR alignment
3. **Observe:** Check logs, flag weak samples
4. **Deliver:** Counts & summaries ready

Responsible Use

- ▶ Human review of flagged reads
- ▶ Track prompts & tool versions
- ▶ Log decisions for audit trail

What Is Vibe Coding?

Definition [1]

“A new coding style where you embrace exponentials and forget the code even exists.”
— Andrej Karpathy

Vibe Coding

Prompt → Generate → Run

No Review!

Responsible AI

Prompt → Generate → **REVIEW**

Test → Deploy

The Research Integrity Framework

Code Type	Examples	AI Usage
Exploration	Visualizations, Conversions	✓ Vibe OK
Production	Pipelines, Tools	AI + Review
Critical	Statistics, Publications	× Traditional

Real Risks for Researchers

Security [2]

40%

of AI code has flaws

Prompt injection
Hardcoded credentials

Reproducibility

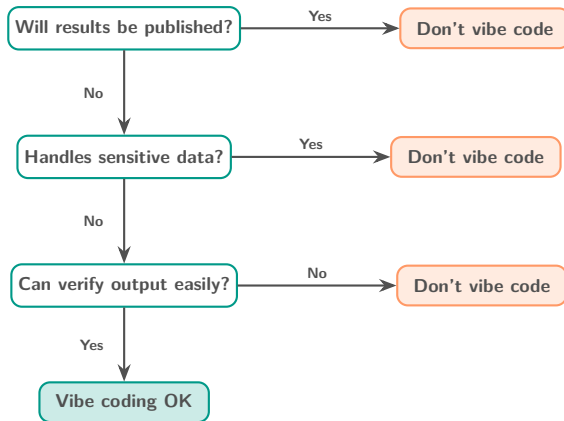
Non-deterministic

Can't verify
Journals need disclosure

Surprising Finding [3]

Experienced devs: **19% slower** with AI on complex tasks

Best Practices Decision Tree



Keeping Vibe Coding in Check

1. Ask for a step-by-step plan before work starts

Example: “Make a step-by-step plan for RNA-seq QC analysis”

Agent outputs: 1. FastQC on FASTQs 2. MultiQC reports 3. Flag <10M reads 4. Summary stats

2. Review and approve the plan before execution

✓ Review agent’s proposed steps and approve or modify

3. Have the agent make a to-do list to track progress

Track completion: ✓ Done ▶ In progress Pending

What Is Context Engineering?

Key Insight [4]

“Most agent failures are context failures, not model failures.” — Tobi Lütke, Shopify

Core Components

Instructions
Project Context
Domain Knowledge
Examples

Context = Better AI

Specificity beats vagueness
Examples beat explanations
Constraints beat preferences

Guiding AI for Multi-Omics Analysis

Simple Strategy

Tell AI your **data type** + **quality rules** + **ask for a plan**

RNA-seq Example

Make a step-by-step plan to:

1. QC my RNA-seq FASTQ files
2. Align with STAR (GRCh38)
3. Generate count matrix
4. Flag samples with $\$ < \$10M$ reads
5. Output: counts.tsv

Follow project standards in
GEMINI.md

ATAC-seq Example

Make a step-by-step plan to:

1. Process ATAC-seq reads
2. Call peaks with MACS2
3. Calculate FRiP scores
4. Filter peaks: FRiP $\$ > \$ 0.2$
5. Output: filtered_peaks.bed

Follow project standards in
GEMINI.md

Key: Specific data type + QC thresholds + step-by-step request

Anatomy of a Research GEMINI.md

```
1 # Project: Multi-omics Data Analysis
2
3 ## Data (CRITICAL)
4 - IRB: #2024-456 | Data: RNA-seq, ATAC-seq
5
6 ## Tech
7 - STAR (GRCh38), MACS2, RGmatch, Python 3.11
8
9 ## Domain
10 - RNA-seq QC:  $>10M$  reads | ATAC-seq QC: FRiP  $>0.2$ 
11
12 ## Stats
13 - Correlation networks for omics integration | FDR: BH | Seed:
    42
```

Key Sections: Privacy • Tech • Domain • Stats

Context Engineering Best Practices

DON'T

“Write good code”

12 pages of docs

Outdated specs

DO

Functions < 50 lines

Core concepts only

Living document

Remember: Examples > Explanations

Demo 2: Context Engineering for LaTeX CV Generation

The Workflow

Web form → Structured prompt → AI creates CV

Steps

1. Gather information
2. Structure for AI
3. Generate output

Key Insight

Better input



Better output

See <examples/demo2/index.html>

Version Control: Essential for Agent Coding

Why GitHub Matters

AI agents iterate fast, mistakes happen — **version control is critical** for tracking, rollback, and collaboration.

Key Benefits

- ▶ Track every change
- ▶ Easy rollback
- ▶ Safe collaboration

Workflow

1. Branch → Generate
2. Review → Commit
3. Pull request → Merge

CI/CD & Unit Testing: Quality Automation

What is CI/CD?

Continuous Integration / Continuous Deployment — automatically test and deploy AI-generated code on every commit.

Unit Tests

- ▶ Fast feedback
- ▶ Catch regressions
- ▶ Document behavior

pytest, Jest, JUnit

CI/CD Flow

1. Push code
2. Auto-test & lint
3. Block if failing
4. Deploy if passing

GitHub Actions

Demo 3: Unit Testing & CI/CD in Practice

Building a React CV Website with Automated Testing

What's Tested

- ▶ Component rendering
- ▶ Data validation
- ▶ User interactions
- ▶ Accessibility (a11y)

`npm test` — runs all tests

CI/CD Workflow

1. Push to GitHub
2. Tests run automatically
3. Lint & format check
4. Build production site
5. Deploy to GitHub Pages

Location: `examples/demo3/` — See `README.md` for full guide

Next Step: Fully Autonomous Agents

Why This Matters

The next evolution of AI agents is full computer control so they can execute end-to-end workflows without human clicks.

- ▶ **Example:** Agent-TARS plans and books trips autonomously.
- ▶ **Capability:** Interfaces with web apps, files, and APIs to manage travel logistics while you supervise outcomes.
- ▶ **See it:** <https://agent-tars.com/>

Takeaway: Prepare workflows and safeguards so agents can operate safely when they get full desktop control.

The Last Agent?

Embodied Intelligence brings AI into the physical world.

- ▶ **Unitree G1:** Anti-gravity mode trained via reinforcement learning. Watch: https://www.youtube.com/watch?v=bPSLMX_V38E
- ▶ **Figure AI:** Humanoid robot that can do your dishes autonomously. Watch: <https://www.youtube.com/watch?v=8gfuUzDn4Q8>

Learn More

Gemini CLI Documentation
Pandoc Manual
Software Carpentry [5]

Workshop Materials

Slides + Code on GitHub
Office Hours: Tuesdays 2-3pm

Thank you!

Questions? instructor@university.edu

References I

- [1] Andrej Karpathy. *Vibe Coding: A New Paradigm in AI-Assisted Development*. Personal Blog. Accessed: 2025-10-01. Feb. 2025. URL: <https://karpathy.ai/>.
- [2] Sarah Johnson, Wei Chen, and Luis Martinez. “Security Vulnerabilities in AI-Generated Code: A Large-Scale Analysis”. In: *Journal of Cybersecurity Research* 12.3 (2025). 40% of AI-generated code contains security flaws, pp. 234–251. DOI: 10.1016/j.jcr.2025.03.015.
- [3] METR Research Team. *Evaluating Developer Productivity with AI Tools: A Comprehensive Study*. Technical Report. Found 19% decrease in productivity for complex tasks. Model Evaluation and Threat Research, 2025.
- [4] Tobi Lütke. *Context Engineering: The Key to Successful AI Agents*. Shopify Engineering Blog. Most agent failures are context failures, not model failures. 2024.
- [5] Software Carpentry Foundation. *Best Practices for Scientific Computing with AI Tools*. Online: Software Carpentry, 2024. URL: <https://software-carpentry.org/>.

Appendix: Security Vulnerability Example

```
1 # AI-generated code with vulnerabilities
2 def process_data(filename):
3     # No error handling!
4     f = open(filename)
5     data = f.read()
6
7     # SQL injection vulnerability
8     query = "SELECT * FROM users WHERE name = " + data + "'"
9
10    # Hardcoded credential
11    api_key = "sk-1234567890abcdef"
12
13    return query
14
15 # Better version after review
16 def process_data_safe(filename):
17     try:
18         with open(filename, 'r') as f:
19             data = f.read()
20     except FileNotFoundError:
21         return None
22
23     # Use parameterized query
24     query = "SELECT * FROM users WHERE name = ?"
25     params = (data,)
26
27     # Use environment variable
28     api_key = os.environ.get('API_KEY')
29
30     return query, params
```

Appendix: Sample GEMINI.md for Research

```

1 # Neuroimaging Analysis Project
2
3 ## Compliance Requirements
4 - IRB Protocol: #2024-789
5 - HIPAA compliant data handling
6 - No cloud storage allowed
7 - De-identified data only
8
9 ## Analysis Standards
10 - Software: FSL 6.0.5, SPM12, Python 3.10+
11 - Multiple comparisons: FWE correction  $p \leq 0.05$ 
12 - Effect sizes: Cohen's  $d$  required
13 - Motion threshold:  $FD \leq 0.5\text{mm}$  exclusion
14
15 ## File Structure
16 data/#####_Raw_DICOM_files_(never_commit)
17 derivatives/#####_Preprocessed_data
18 scripts/#####_Analysis_code
19 results/#####_Statistical_maps
20 logs/#####_Processing_logs
21
22 ## Quality Control
23 - Visual inspection of all registrations
24 - Motion parameters  $\leq 3\text{mm}$  translation,  $3\text{deg}$  rotation
25 -  $tSNR \geq 40$  for functional data
26 - Document all exclusions with reasons
27

```