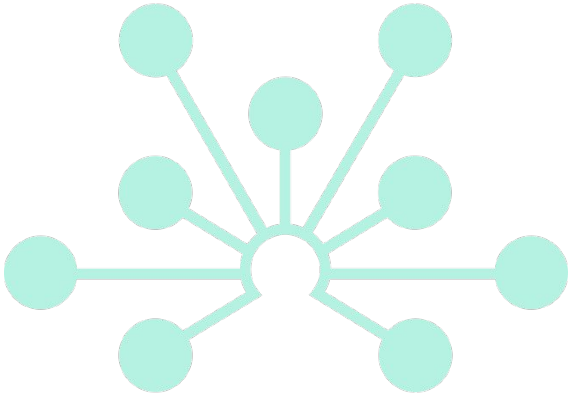

How I Use LLMs at the OMSF

Presented by:
Ethan Holz - Research Software Engineer

OMSF Ecosystem Infrastructure

2025

Some caveats

1. Test your code
2. AI doesn’t replace expertise
3. Always be careful about managing your data

Context is king

- LLMs are predictive machines, not thinking ones
- Provide as much context as possible
  - Documentation
  - Code
  - Papers*
- We have tools/methods to make getting context easy
  - Retrieval Augmented Generation (RAG)
  - Model Context Protocol (MCP)
  - Repomix
- llms.txt

My preferred workflow

1. Provide the context of the problem you have
    - MCP Server
    - Code snippet
    - An entire repo (Repomix)
2. Limit the scope of the solution
3. Provide a feedback loop
    - LLM-native editor
    - MCP

What I Use

- My editor - Zed
- My model of choice - Claude 3.5/4
  - For brainstorming, Claude Desktop
  - For code editing, GitHub Copilot Chat + Zed Agent
- MCP Tools
  - Linear MCP Server
  - Context7
  - GitHub MCP Server
- Included tools
  - File editing
  - Shell commands (helpful for git)

What hasn’t done gone well

- Blind trust of LLMs leading to project setback
- Unmaintainable code
- Over-engineering

What has worked well

- Helped for me to iterate quickly on an established web dev codebase
  - Lots of context (large codebase, well documented)
- Automated test generation for simple tests
- Enhanced feature development on an LLM-native codebase
  - llms.txt file for enabling easy documentation ingest
- Faster PR reviews
  - GitHub Copilot for a first pass review
- First pass documentation generation

Summary

- AI has helped, but it falls short without context
- Useful in well developed codebases
- Not a replacement for expertise