# How-To: Context-Aware LLM-assisted coding workflows for science

**Jesse Jones, 06.01.2026, GöAID Seminar Series**

# Overview of contents

- Point of View
- Scientific Coding Workflow and DevOps Cycles
- Context-aware LLM models
- Model Context Protocol (MCP)
- Context-aware LLM assisted scientific coding workflow + „prompt engineering“
- Conclusion

# Point of View

## About me


German
Research Software Engineers

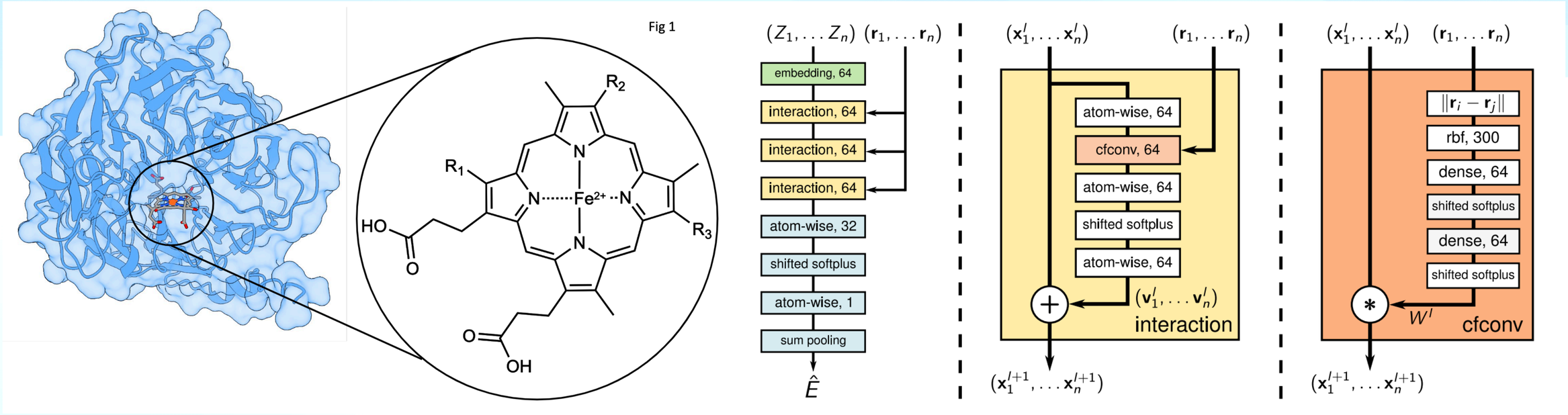

German Biophysical Society


German Physical Society

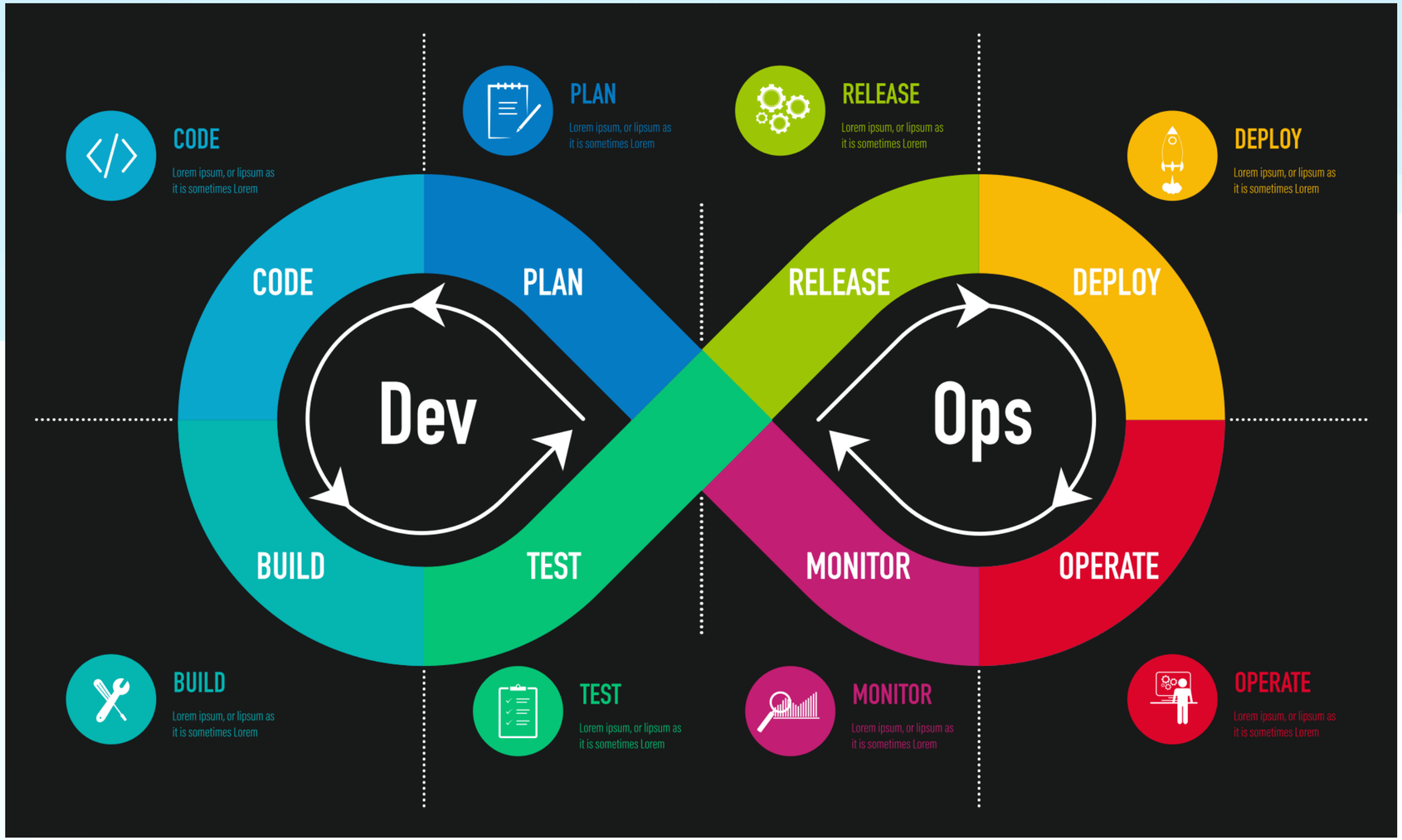
Domain Scientist with Research Software Engineering Aspirations

# Point of View

## About me

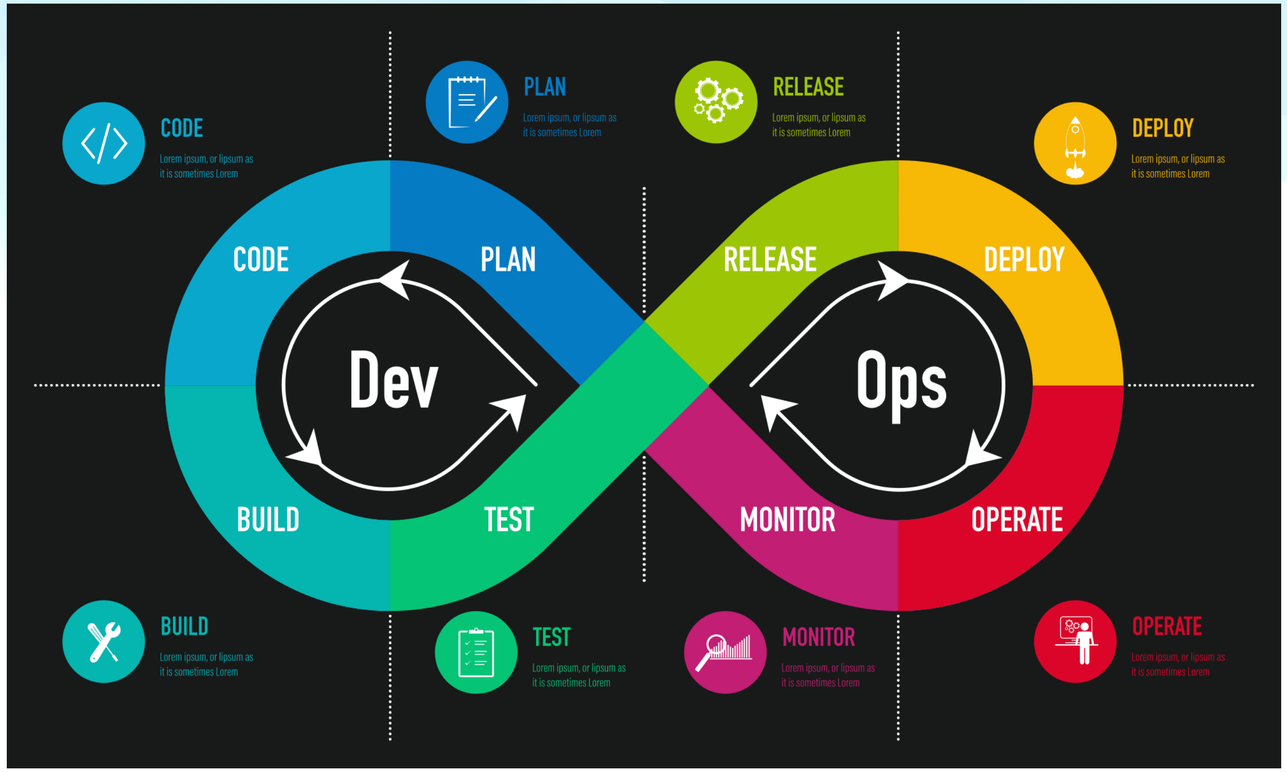


# Scientific Coding Workflow and DevOps Cycles



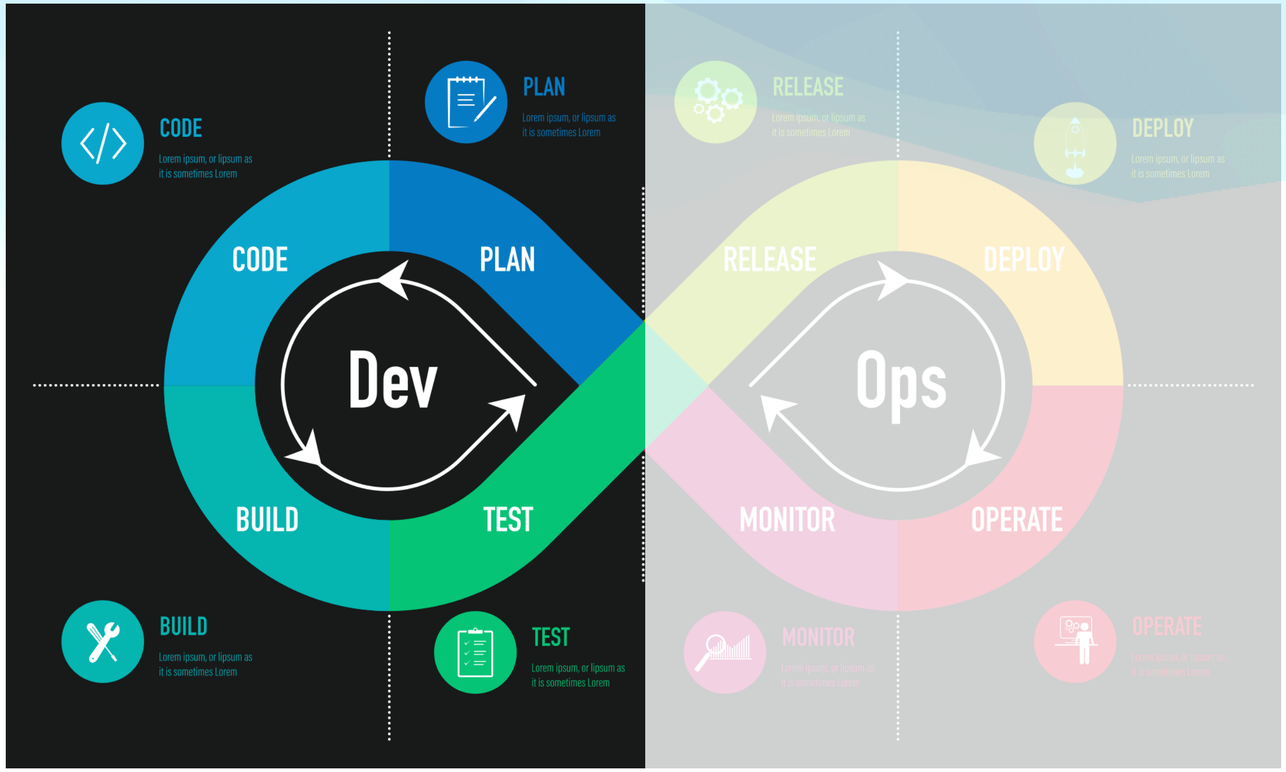
# Scientific Coding Workflow and DevOps Cycles

- Different phases in the DevOps Cycle have different requirements and target code quality
- Dev cycle is often more experimental, code less refined and less tested
- Ops cycle is more established, code is refined and tested, run on a large scale



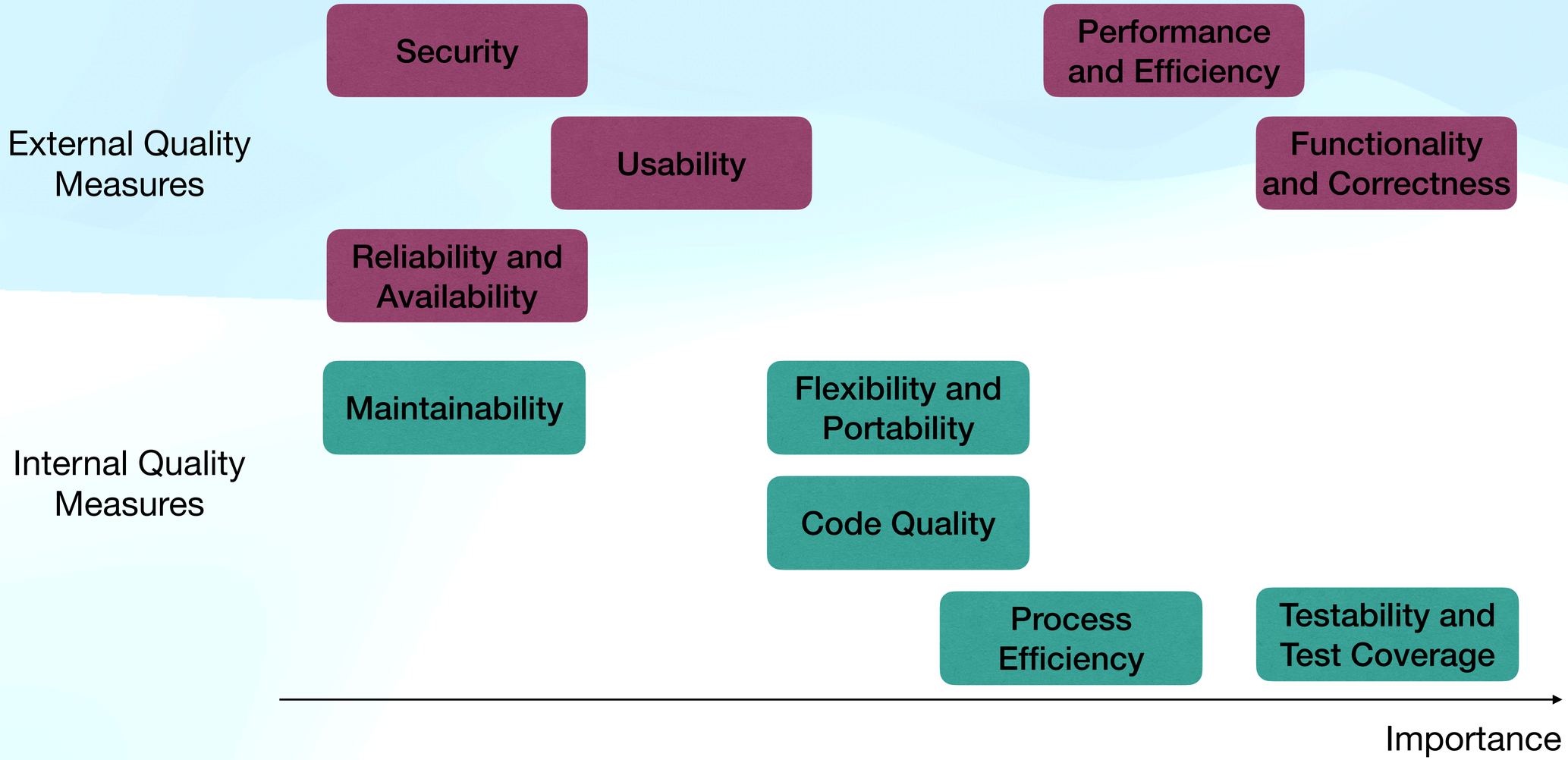
# Scientific Coding Workflow and DevOps Cycles

- In most scientific work, only the Dev cycle is important, as the code is not (yet) part of the core contribution of the work
- Most of the Code is only completely executed a very low number of times (e.g. <10 times) over a projects duration, only users are the developers
- Hence, the quality measures change



# Scientific Coding Workflow and DevOps Cycles

## Quality Metrics for scientific coding workflows in physical sciences



# Context-aware LLM models

- Context-aware relates to the context of files, databases, codebases in your current project which are added to the prompts received by the LLM (and therefore loaded into the context window)
- Example implementation: Claude Code CLI
- Core life quality improvements include:
  - CLAUDE.md file and Initialisation
  - Tool Usage including file search capacities, file writing, git interactions
  - Custom commands, skills and agents

# Context-aware LLM models

**CLAUDE.md - project documentation automatically loaded into the context**

- Document:
  - Core files and utility functions
  - Common bash commands
  - Code style guidelines and (git)-repository etiquette
  - Environment setup (e.g. pyenv & compiler setups)
  - Other project specific information to remember
- `/init` will automatically create a base sketch of a CLAUDE.md file

# Context-aware LLM models

## Tool Usage

- Available tools out of the box are e.g. mv, rm, file writing and reading, as well as searching
- Access specific files for the model to read into context window or search for context window additions with `@` operator
- Also available git and GitHub commands with the GitHub CLI tool
- Interaction between LLM inputs and any tools are done via Model Context Protocol (MCP)

# Context-aware LLM models

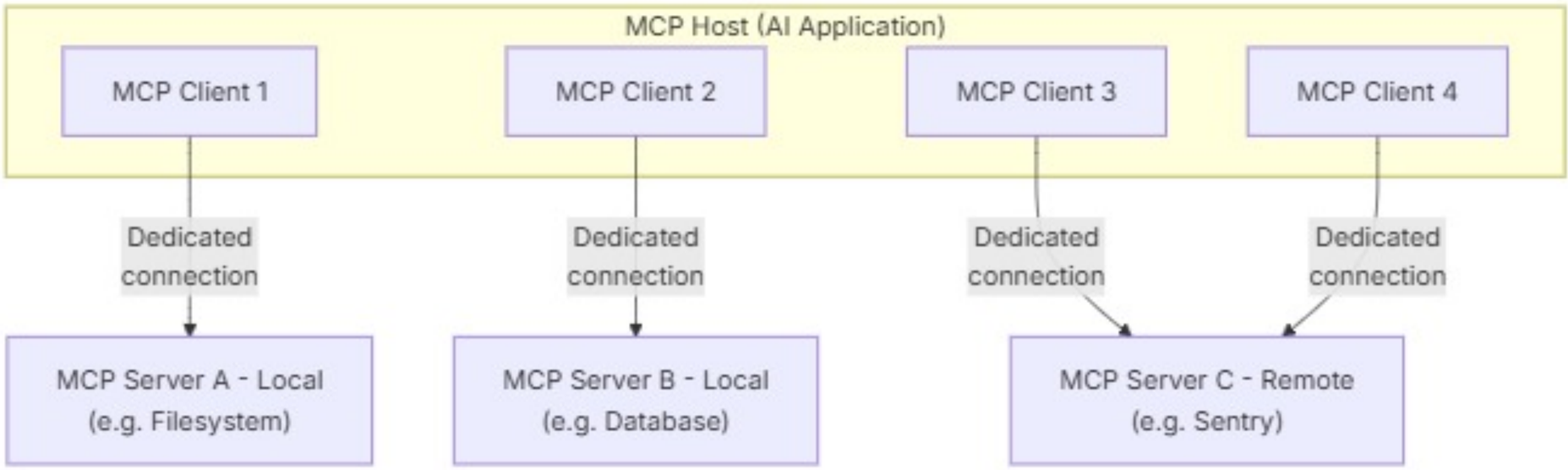
## Custom commands, skills and agents

- Shortcuts for repetitive text can be created via `/`commands, e.g. /debug
- `/`commands can include keyword $ARGUMENTS
- Skills are shortcuts not explicitly called by the user but autonomously invoked when matching a task context
- Agents are „personalities“ which the LLM can become, typically limiting the instance of the LLM to work on one specific task, sparing out the context of the different instances

# MCP

## Model Context Protocol

- Open-Source standard for connecting AI applications to external systems (data sources, tools, workflows)
- Server-Client architecture where MCP-host (LLM) establishes connection(s) to MCP server(s) maintained by MCP client



# MCP

## Model Context Protocol

Data Layer
JSON-RPC 2.0

Lifecycle Management

Connection Init./term.

Conn. Capability Negotiation

Core Server Features

Tools for AI actions

Resources for context data

Prompts and interaction templates

Core Client Features

Server to client LLM sample requests

Elicit user input

Client logging

Utility Features

Notifications for RT updates

Progress tracking

Transport Layer

Transport Mechanisms:

Stdio transport

Streamable HTTP transport

# Context-aware LLM assisted scientific coding workflow

## Workflows for coding in science

- Early stages in the project:
  - **Explore, plan, code, execute, refine, commit**

- Larger software projects:
  - **Write tests, commit; code, iterate, commit**

- General workflow best practice (or if unknown workflow requirements):
  - **Ask LLM to make a plan on how to proceed, then iterate until satisfied with plan**

# „Prompt Engineering“

## Checklist for a good prompt

- Before writing a prompt, use the (mental) checklist:
  - Is this prompt for a LLM-agent? If so, reduce any action requests to things the agent would be in charge of
  - Are there any specific predefined workflows to use?
    - Specifically request a workflow through a /command
    - Have the LLM figure out which workflow to use by referencing skills
    - Define a specific alternative workflow in the prompt

# „Prompt Engineering“

## Checklist for a good prompt

- Can I give context via file-references (direct: „read @logfile.txt“, indirect: „read the log file“)
  - Direct file references are usually better, as the user often understands the big picture and the problem better
- Does my prompt contain every additional information I want to add manually?
- Is my prompt written concise and succinct? Too much information might lead to undesired results
- Do I give emphasis on which part of the task I want to have handled in which way and in which order?

# Advanced Strategies

## Intentional Compaction of Context Windows

- Regular Prompting Workflow: Prompt - Code - Evaluate - Prompt until you reach a loop often indicated by „You’re absolutely right.“
- Hitting the „You’re absolutely right“ you can refresh context by starting a new session and including in the prompt the learnings of the last session.
- Intentional Compaction: Have original LLM agent write a progress file which onboards the fresh-session agent
    - Removes large context operations from current context like file-searching, code flow understanding, file-edits, test/build output, MCP tool responses

# Advanced Strategies

## Subagents - Searcher Agent

- Subagents (i.e. multiple LLM instances working in parallel on different tasks in the same project) are really about context control
  - Less and more focused information leads to less unnecessary context, which leads to better results for this task
  - Example:
    - Searching Subagent - called through the prompt „find where xyz is handled, use the subagent searcher“ within the parent-agent
    - The searcher subagent/child-agent finds the files, returns filenames without cluttering the context window of the parent agent

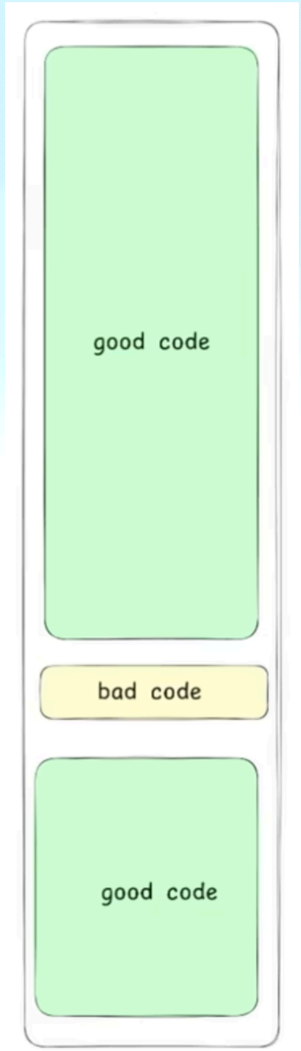
# Advanced Strategies

## Subagents - Searcher Agent

- Split Workflow Steps into subagents to increase quality of individual steps
- For example: **Explore, plan, code, execute, refine, commit** as **research, plan, implement**
- Research Subagent: understand how the system works, find all relevant files, explore causes of *xyz* bug
  - return document with problem specific information, and code references with filenames and line numbers, function names etc.
  - HUMAN REVIEW STEP
- Planning Subagent: Outline Exact implementation steps, include filenames, lines and snippets, add explicit testing steps
  - HUMAN REVIEW STEP
- Implementing Subagent: Straightforward from this point on

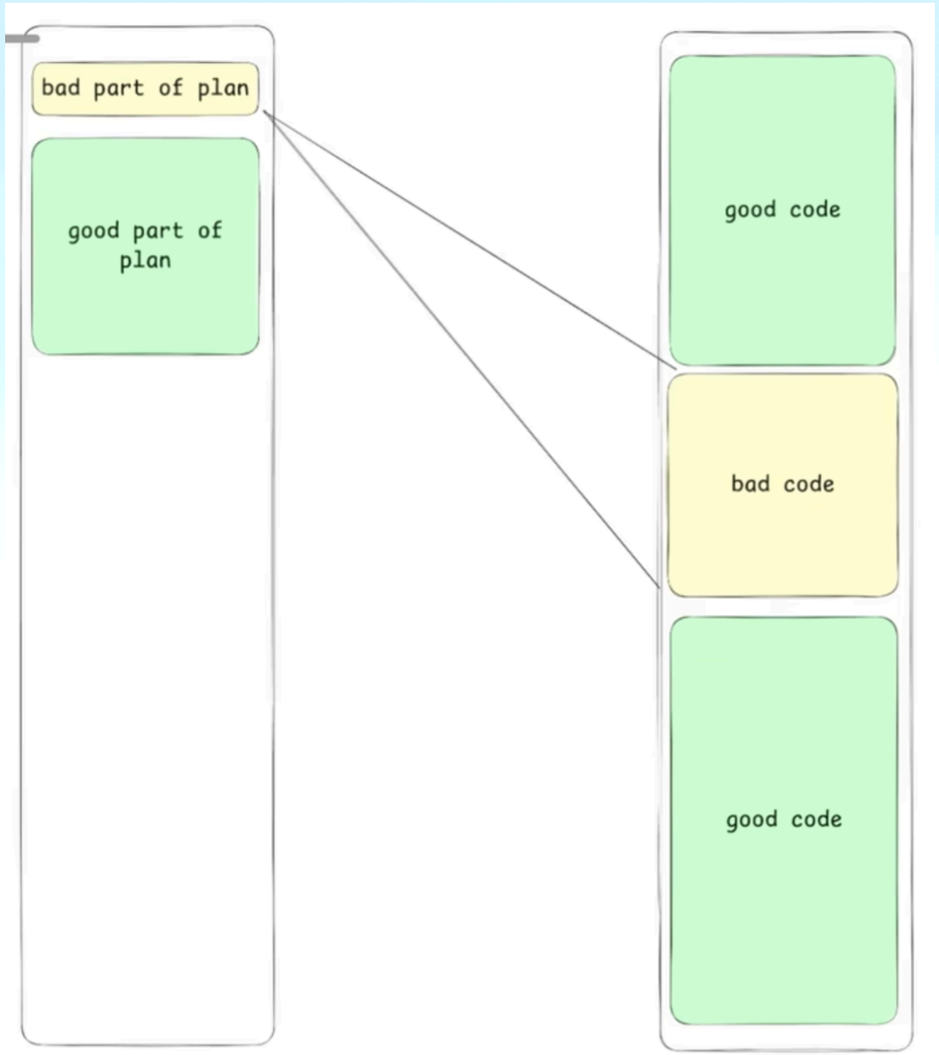
# Human Intervention

## Where to spend your time most efficiently?



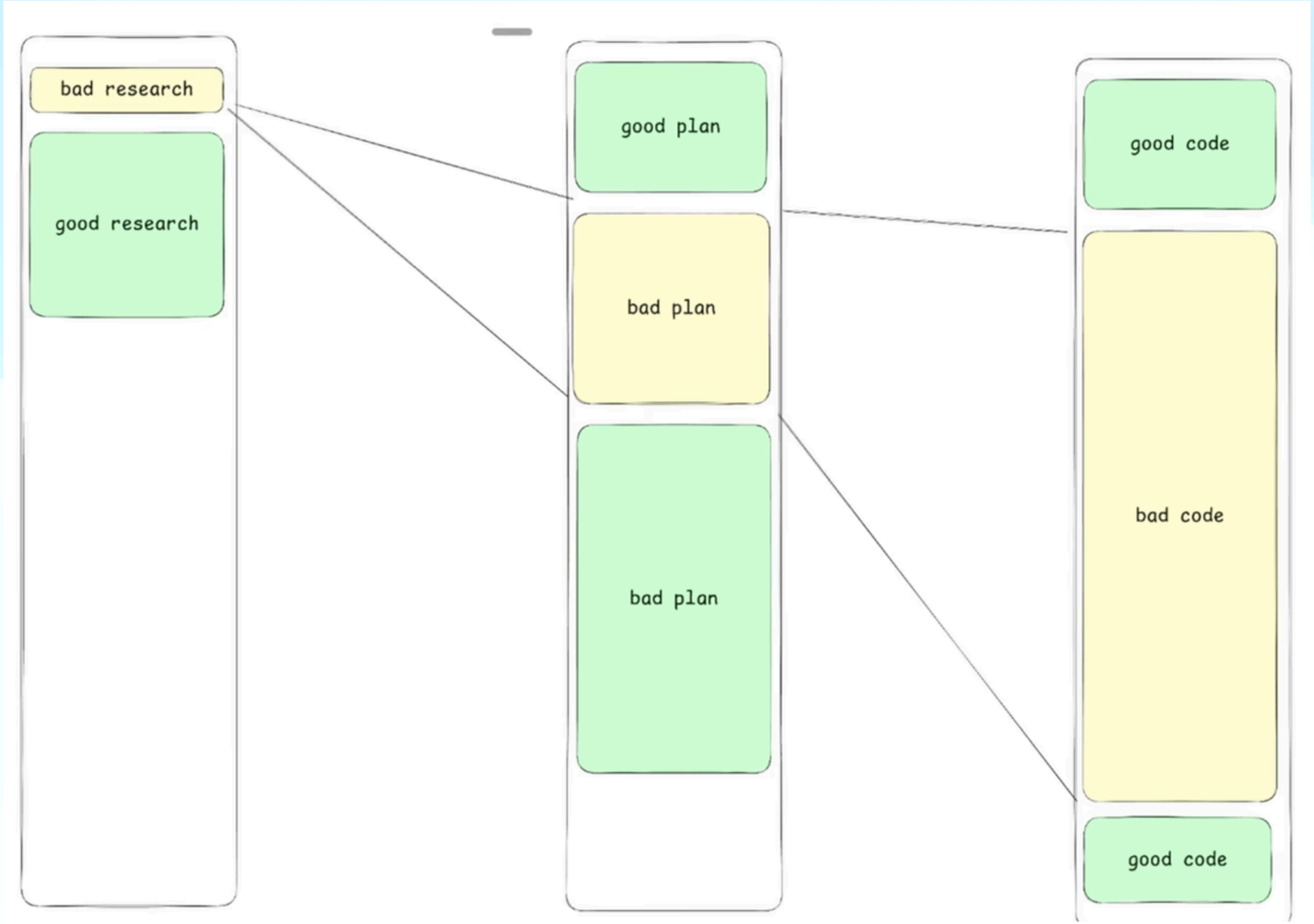
# Human Intervention

## Where to spend your time most efficiently?



# Human Intervention

## Where to spend your time most efficiently?



# Demonstration

# Conclusion

- Using Context-Aware LLM coding assistants and controlling the context is helpful in saving time for researchers
- Controlling context actively: giving all necessary information, but not more
- Correct application includes identifying the importance of internal and external quality measures for software and including this evaluation into the work
- MCP acting like a USB-standard for LLM-tool interactions helps tremendously, as LLMs need only be supervised
- Human supervision and good planning is still key for good results

# Acknowledgements

**Contact: j.jones@tu-berlin.de**

unisyscat

Technische Universität Berlin

SFB 1078


Mroginski Group: Biomodeling @ TU Berlin
https://github.com/biomodeling-tub