SCIENTIFIC SOFTWARE CENTER

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

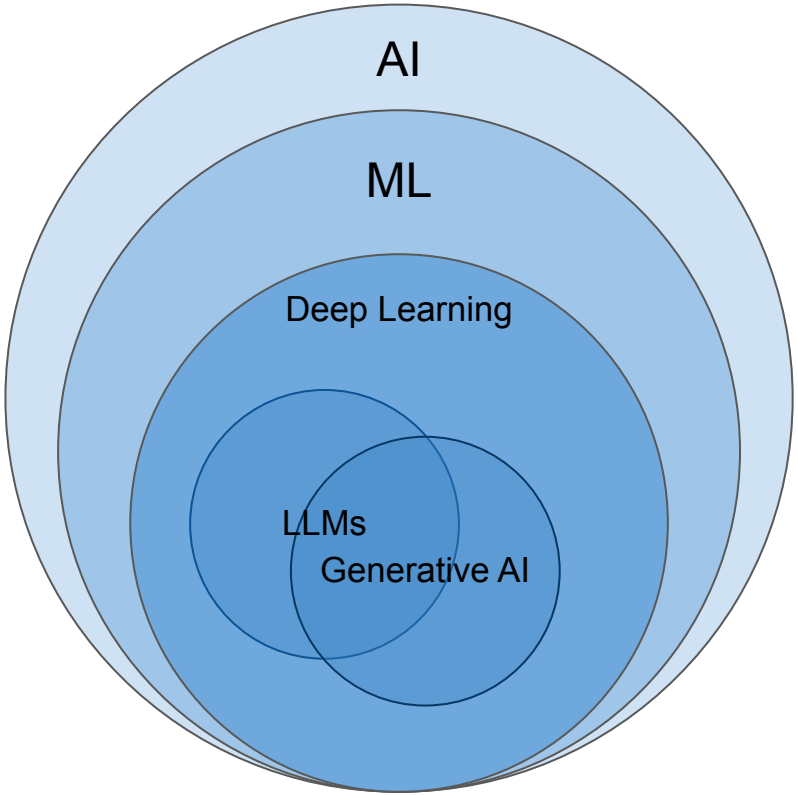
Getting started with generative AI coding tools

Dr. Inga Ulusoy, Dr. Harald Mack, Scientific Software Center

26th May, 2026

Generative AI and software development

What is generative AI?



Discriminative model

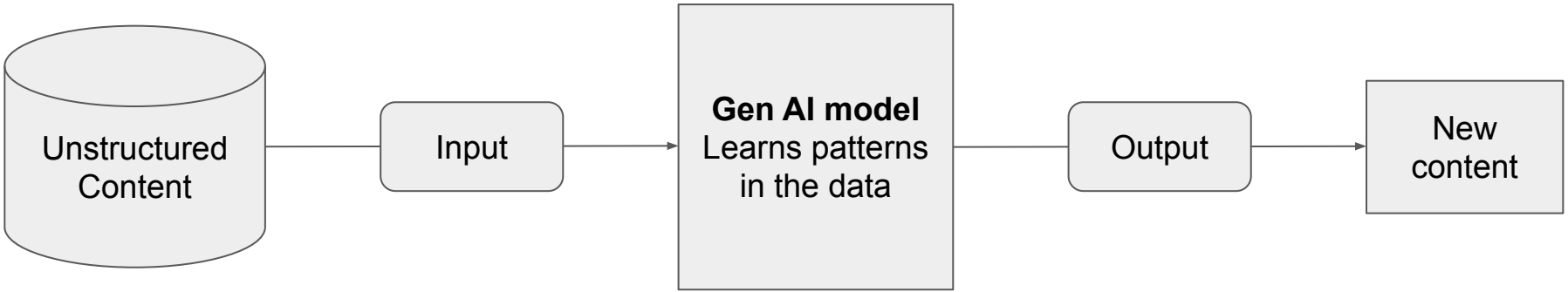
This is a cat.

Generative model

```
A cat is a small
domesticated carnivorous
mammal (Felis catus) that
is commonly kept as a pet.
Cats have been domesticated
for thousands of years and
have adapted to live
alongside humans in various
environments around the
world. They are known for
their agility,
gracefulness, and
independent nature. …
```

Photo by Krysten Merriman: https://www.pexels.com/photo/short-coated-gray-cat-20787/
Response by ChatGPT 3.5: https://chat.openai.com/share/846ff979-4eff-47a6-954c-dd2f2943fead

Generative AI Models



Generative AI in academia

- ✔ Proposal writing
- ✔ Paper writing
- ✔ Visualization of research data
- ✔ Research software development
- ✔ For learning
- ✔ For teaching

! Openly state and reference use of GenAI

! Ensure no plagiarism occurs

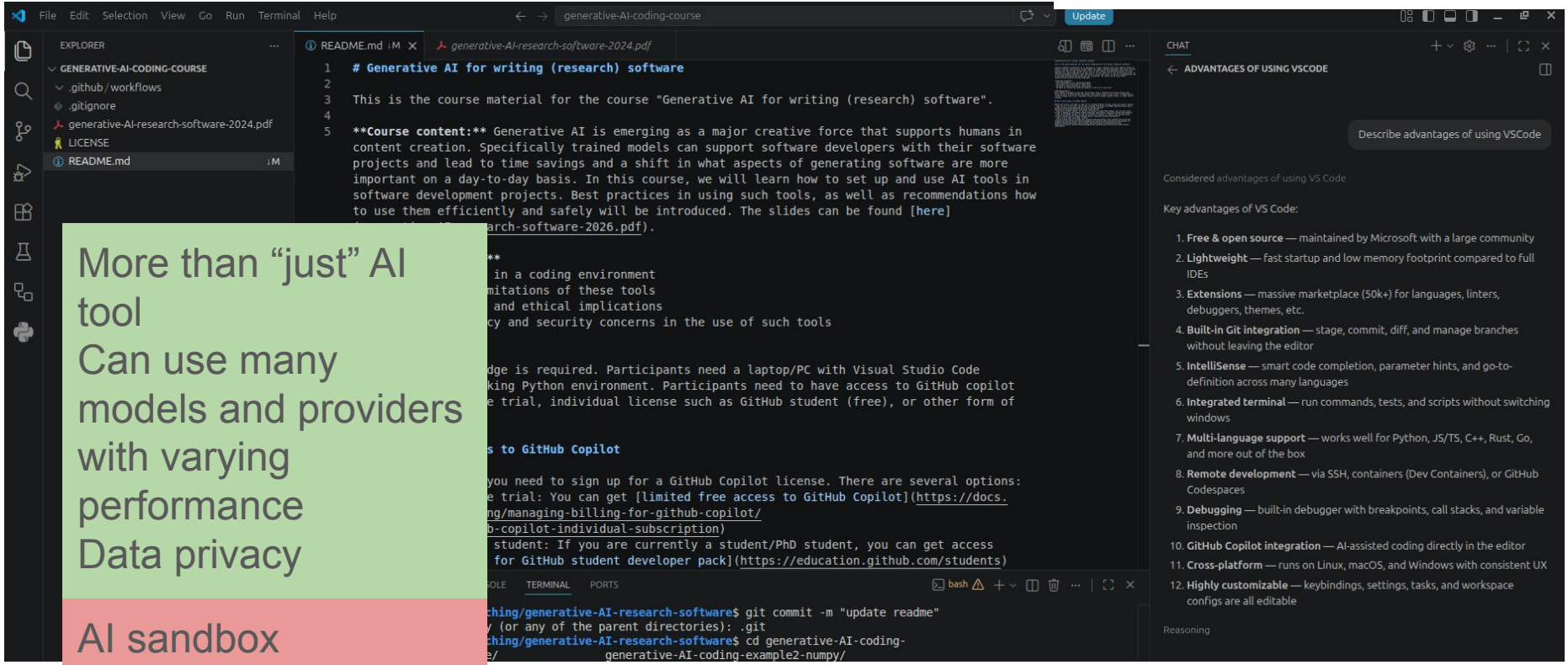
! No infringement of intellectual property rights

x Writing reviews

confidentiality
transparency
quality assurance
responsibility

https://www.dfg.de/en/news/news-topics/announcements-proposals/2026/ifr-26-18

Tools: VSCode



More than “just” AI tool
Can use many models and providers with varying performance
Data privacy

AI sandbox
Hallucinations
Freezes

AI agents

- Can autonomously perform tasks
- Use workflows
- Use tools
- Capable of NLP tasks like decision-making, problem-solving, interacting with environments and performing actions, use loops

- Brain: LLM for reasoning/planning (stateless!)
- Tools: functions agent invoke (shell, HTTP, MCP, file I/O)
- Memory: context window + optional persistent store
- Loop: observe → think → act → observe

Differ from plain chatbot: take actions in world, not just emit text. Differ from script: pick steps dynamically, not hardcoded.

Example: Claude Code = agent. Tools = Read/Edit/Bash. Loop = read code, plan fix, edit, run tests, retry.

`claude`

Give your AI agent a personality

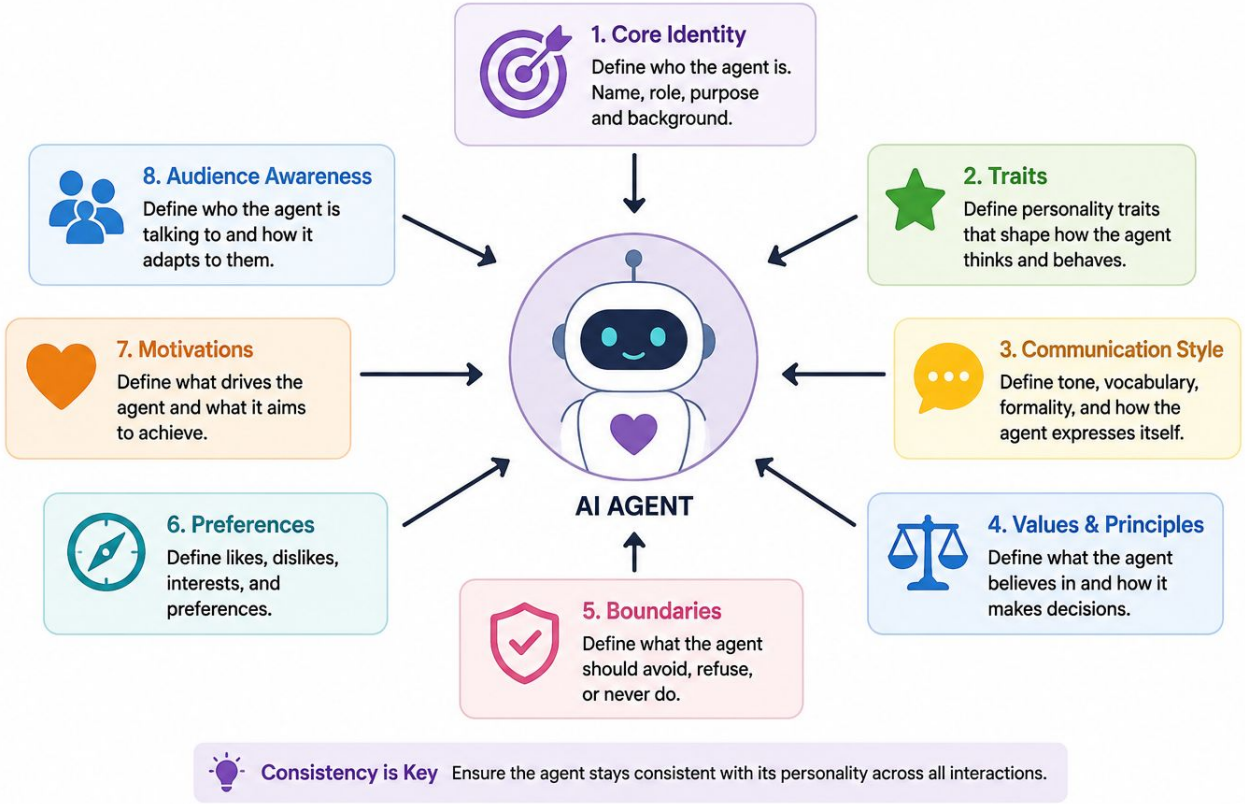
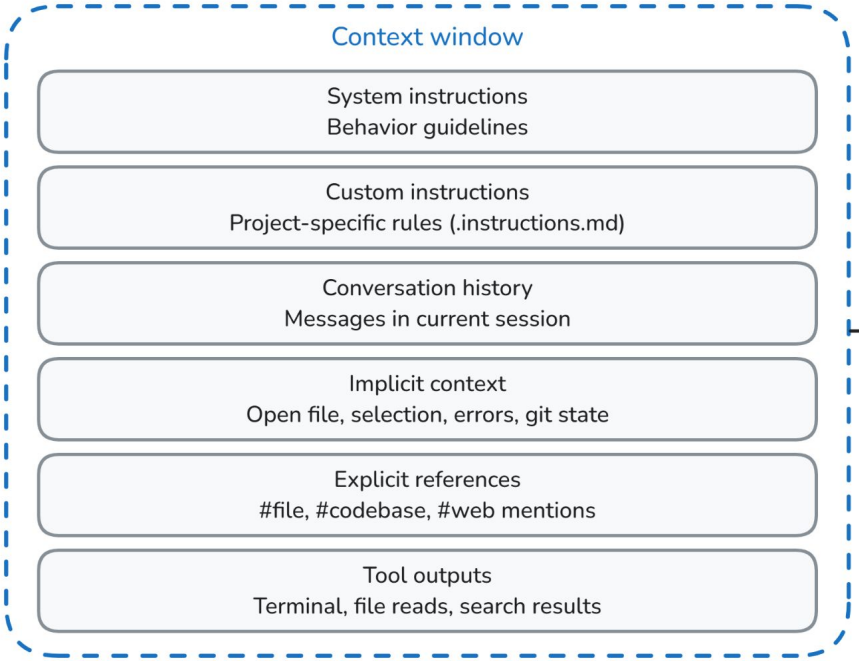


Image credit ChatGPT May 2026

Via the system prompt (“always consider memory footprint …”)
Context injection (memory files like CLAUDE.md, AGENTS.md)
Multi-agent pipelines with planner, coder, reviewer, tester

Agent context



https://code.visualstudio.com/docs/copilot/concepts/context

- New task = new session
- Select useful content and provide explicit reference
- Use persistent rules

How can AI coding tools support you?

- Plan
  - Architecture
  - Design
  - Best approach
  - Prototype
- Create content
  - Code
  - Documentation
  - Tutorials
- Provide language assistance
  - Translation (programming languages or different versions of the same programming language)
  - Content summarization (explain code)
  - Content curation (explain keywords, key concepts)
  - Writing assistance (style checking, best practices, identify errors and inconsistencies)
- Debugging
- Refactoring
- Reviewing
- Testing
- Software maintenance
- Code performance

How to set up GitHub Copilot

- You need a Copilot license. Free tier comes with registering for GitHub
- Open VSCode, go to “Marketplace” and search for “Copilot”
- Install the “GitHub Copilot Chat” extension
- Follow the prompts to sign in to GitHub (lower left in VsCode)
- Go to 'Copilot Settings' on GitHub to change the permissions and settings and see your current usage
  - Enable or disable code duplication (allow or block code completion suggestions that match publicly available code)
  - Enable or disable prompt and suggestion collection
- Change the behavior of copilot in vscode via the extension settings
- You should be able to see Copilot code suggestions now

https://docs.github.com/en/copilot/using-github-copilot/getting-started-with-github-copilot
https://docs.github.com/en/copilot/configuring-github-copilot/configuring-github-copilot-settings-on-githubcom

How to use GitHub Copilot

- Start writing code to see suggestions
- Press `Tab` to accept suggestions, `Esc` to dismiss
- To see alternative suggestions, press Alt+] ([) (Windows/Linux), Option (⌥) or Alt+] ([) (macOS) or hover with the mouse above the suggested code
- Use Ctrl+I to open the editor inline chat
- Accept only the next word by pressing Control+→ (←) (Windows/Linux), Command+→ (←) (macOS)

Example 1: Basic usage

The starter repository is here:
https://github.com/ssciwr-courses/generative-AI-coding-example1

Click on “Use this template”, then clone the repository locally

Example 1: Basic usage

- Start typing code
- Start typing comments
- First write code, then the comments
- Write the comment and have Copilot write the code for you
- You can also try this in an interactive environment (jupyter notebook) within VSCode, if you wish
- Ask the chat for specific advice and explanations
- Navigate and accept suggestions, and command execution by the agent
- The context of the agent is determined by what windows you have open in the IDE

Example 2: More advanced code, numpy

Use the same repo and code as in Example 1.

Create a plot of the <z> and <H> data column.

Convert the dataframe columns <z> and <h> into two numpy arrays.

Create a moving average of the values.

Plot the moving average.

Write tests for the added functions.

Compute the Fourier transform of the <z> values and plot the result.

Example 2: Outcome

Reflect on the code that you have just generated. What is good, what is bad? Have you learned anything new?

What is your level of understanding of the code?

When does the agent perform well?

- Helps you create content fast
  - Clear specification
  - Tight feedback loop
  - Well-structured prior content
- Can derail your project:
  - Vague goal
  - No validation of goals
  - New domain that does not have much training data

How can we set clear goals?

Software validation versus software verification

Software validation:

“Are we building the right product?”

Example1:

- Software should provide image content summarization
- Images and models are too large to be able to process with available resources

Example2:

- Software should simulate bond formation of chemical reactions
- Method not accurate enough compared to the energy differences between compounds

Software verification:

“Are we building the product right?”

Example1:

- Software should calculate space vehicle trajectories
- Wrong data column used in equations leading to error in actual trajectory

Example2:

- Software should analyze pathways based on gene names
- Input gene names were converted to dates in Excel leading to wrong assumptions

How to validate software: Testing under validation

Acceptance testing
- Stakeholder/user run against requirements

Alpha testing
- Internal users, pre-release with real environment and in communication with a developer

Beta testing
- External users with a limited release to provide feedback on real usage

Usability testing
- Real users do real tasks - success rate, time, errors, satisfaction

Domain review
- Subject expert check outputs make sense

Compliance/regulatory acceptance
- Audits with an external validator

ML/AI specific
- Eval sets vs benchmarks
- Real-world application

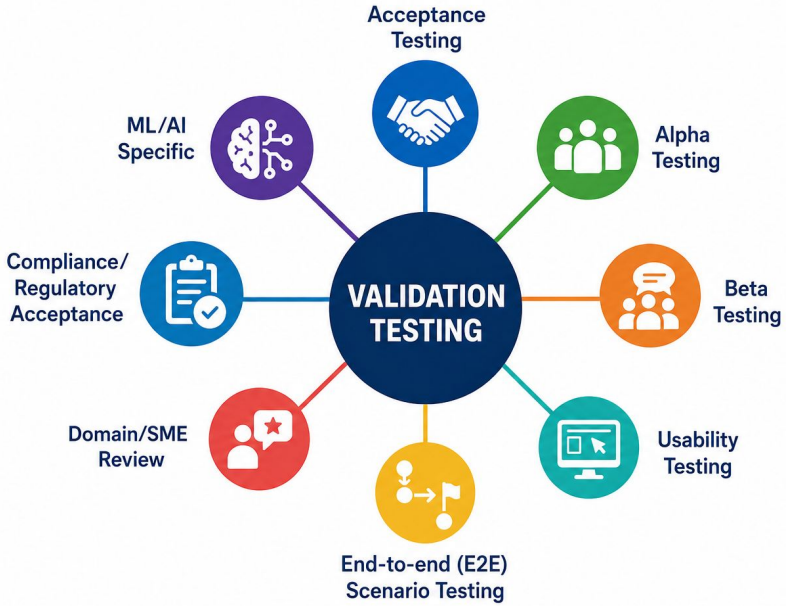


Image credit ChatGPT May 2026

How to verify software: Tests under verification

Static verification
- Type checking
- Linting
- Static analysis (sonarcube)
- Code and design review

Unit tests
- Single function, class / method

Integration tests
- Multiple units together

System tests
- Whole system with all components

Performance tests
- Latency and throughput, scaling with data size

Security
- Pen testing, fuzz testing

Mutation testing
- Mutate code, check that tests catch it

23.07.2026

Compact Course **"Agentic Test-Driven Development"**

Dr. Liam Keegan

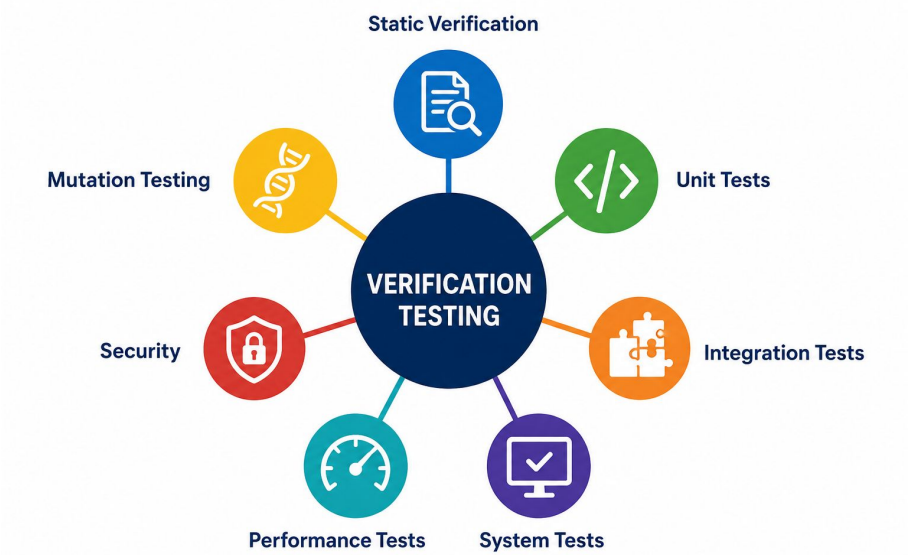


Image credit ChatGPT May 2026

AI-supported software development: User roles

Chat user, ie. ChatGPT	Copilot user, ie. code completion and code generation	Chat-and-Copilot user, ie. code completion and code generation iterates with chat	Agentic user up to vibe coder
- Get help and learn through explanations and summarization	- Increase efficiency through passing over cumbersome tasks such as boilerplate code, docstring generation or straightforward pieces of code	- Increase efficiency and learn through interacting more closely with the LLM, refactoring suggestions in an iterative manner	- Pass of responsibility for pieces of code, or complete parts of the code to the AI agent without following the AI implementation in detail

AI agents good at or helpful for:

- Following patterns already in the code-base using currently open tabs and context around cursor
- Python or other much-used programming language of public repos, like JavaScript
- Adhering to best practices and community coding standards
- Writing more readable and generalized/reusable code
- Speeding up code review
- Maintaining flow state through fewer context switches (stackoverflow/google search)
- Documenting your code
- Thinking about requirements beforehand
- Behaviour-driven development
- Test-driven development
- Creating boilerplate code
- Summarizing and explaining code

Best practices

- Ask to have code explained and to give reasons why this implementation is better than others or what would be alternatives
- Ask to have code critically reviewed in aspects of correctness, tests, security, performance, reliability, maintainability, design, compatibility … and use new contexts for each review
- Ask for test suggestions including edge cases, negative testing, mocking
- Request suggestions for code improvements
- Use meaningful names to help your agent understand your code better
- Be ready to explain your implementation and why you did it like that
- Don’t let the agent write long stretches of code, always little pieces to make sure to stay on the task (“vibe coding”, handing over complete control to agent)
- Do not create slop: unnecessarily lengthy or complicated code

Day-to-day work

- You will write less code but review (read) more code and approve changes
- Coding is not a continuous flow anymore, but a lot of reading
- A lot of prompting and using agents of different characters / personas
- Works better if you use meaningful names for variables, functions, …
- Frees you from writing obvious code and you are the architect, doing the real intellectual work (design and problem solving)
- You learn new things and have someone to discuss technical problems and possible solutions with, and to double-check your work

What are the limits of AI agents as of May 2026?

- Lack of understanding: lack of context, coherence or relevance
- Varying quality and accuracy
- Hallucinations: Dreams up non-existing content or duplicates existing content
- Limited creativity (only what is in training data)
- Interpretability and explainability (much improved already)
- You still need to understand code and requirements to write code - and be good at it if you want to write good code

Caveats

- Might spread bad code and lead to self-amplification (more public bad code, retraining on worse code samples, even worse code)
- If you are a very junior developer, it may take the joy of coding for you
- May replace junior developers
- May render frameworks obsolete
- May increase technical depth
- Authorship and licensing complications, uncharted territory
- Understands context quite well, but not always the exact goal
- Cutting-edge or non-standard procedures do not receive great suggestions
- Creates dependencies and reduction of open knowledge

Reproducibility

- Two kinds of reproducibility
  - 1): Will the agent come up with an equivalent solution if I ask it the same thing twice?
  - 2): Does the code produced by the agent confirm to scientific reproducibility standards?
  - Hard reproducibility not possible with current LLMs -> Reproduce behavior, not details
  - Skills, prompt templates, specialized agents, workflows
  - Empty context window when you start work on a new problem
  - Define acceptance criteria first and measure result
  - "Explore -> Plan -> Code -> Commit" workflow
  - Document decisions that where taken
  - Agent-written code is typically not reproducible out of the box
  - Reviewing code can become a bottleneck
  - Define what correctness means first
  - Specifications and tests become more important…
    - … because the source of truth is your engineered intent
  - Intent expressed in specification documents as anchor for agent (and for you)
  - Specifications expressed in tests for cross-checking

Context engineering

Spec-driven development

Sustainability

- Sustainability: "the capacity of a system to remain understandable, modifiable, and verifiable and validatable over time"
- Sustainability implicitly refers to intent -> *How do I want my system to behave?*
- Sustainability in agentic coding is enabled by specifications
- It's easy to lose control over project complexity: "Comprehension Debt"
  - Which part influences which?
  - Where are side effects?
  - Is my code still save and compliant with my restrictions?
  - Are my dependencies save, maintained and needed?
- Specifications can help, but only if they express intent fully
  - Architecture
  - User classes
  - performance requirements
  - Behavioral requirements
  - Security limits

Requirements engineering

Sustainability

- Agents only solve local problems
  - Without guidance, they reach for complex solutions: *Understand your 'intent' before writing Specs*
  - Agents don't see 'the big picture' unless you tell them and keep them on track: *Specs can supply it*
  - Agents don't ask if something 'is a good idea' unless you make them: *Use an 'oracle' to question decisions, find holes, highlight alternatives*
  - Agents sometimes solve problems by removing the check that ensures their detection. *Cross check boundaries in review*
- Having an agent write tests of its own code is not enough
  - Behavior driven development: define a set of user stories that define how software should behave, test against those to find deviations: *Acceptance tests for the 'what'*
  - Tests for the functionality of code units: *Unit tests for the 'how'*
  - Mutation testing: Check test suite by introducing code mutations -> is it sensitive enough?
  - Rebuild testing: Delete src/, check if code behavior can be reproduced from specs and tests pass

Security aspects

- Privileges: With AI agents using tools, they can execute destructive actions on your PC
  - rm -rf / ?? - Antigravity https://news.ycombinator.com/item?id=46103532 , https://support.google.com/gemini/thread/415523107?hl=en
- Tool and API manipulation:
  - Modify tool / API to trigger unauthorized action, ie. an MCP server bcc'd email content to an external domain https://snyk.io/blog/malicious-mcp-server-on-npm-postmark-mcp-harvests-emails/
- Skill manipulation:
  - Skill file appears legitimate but contains malicious payload (Typosquatting, Memory poisoning, ClickFix prompts), ie. targeting crypto wallets and ssh credentials https://owasp.org/www-project-agentic-skills-top-10/ast01
- Prompt injection
  - For example, payload splitting - malicious content is divided in multiple prompts
- Secrets leakage
  - Can happen without intent - .env file in your project folder?

Privacy: https://copilot.github.trust.page

- User Engagement Data: accepted or dismissed completions, error messages, system logs, and product usage metrics (stored for two years)
- Prompts: inputs for chat or code along with context (not retained)
- Suggestions: AI-generated code lines or chat responses provided to users (not retained)
- Feedback Data: real-time user feedback, including reactions (e.g., thumbs up/down) and optional comments, along with feedback from support tickets (“stored for as long as needed for its intended purpose”)
- Data from individual licenses (ie, student licenses) is used for training (not the case for GitHub Enterprise or Business) - same as with other providers

Legal aspects

- If you incorporate third-party code, you need to make sure there are no license incompatibilities
- However, with Copilot you may use another person's code unaware
- How will you provide attribution/copyright?

GitHub is being sued because of this
https://hackernoon.com/doe-vs-github-ammended-complaints-on-copyright-infridgement-open-source-licenses-and-more

You may block suggestions that match public code, but this only works from 150+ characters (ignoring whitespace).

Feature providing a reference to matching public code
https://github.blog/2022-11-01-preview-referencing-public-code-in-github-copilot/

Legal aspects

Who owns the code that your agent created?

Copyright: Protects your intellectual property right in software creation:

But: If the agent wrote the software, is it still *your* code?

>>> the US Copyright Office has confirmed that copyright protection requires human authorship, and that material generated by AI without sufficient human creative control is not registrable. In particular, they have noted that when a complex work is made by a prompt, “these prompts function more like instructions to a commissioned artist”, with the result uncopyrightable.<<<

https://www.twobirds.com/en/insights/2026/belgium/ai-coding-can-you-protect-what-your-agents-create
https://www.copyright.gov/ai/

Impact on research software development

Powerful tools put even more knowledge at your fingertips - and can also break it down for you!

There are multiple levels of tools to support your work, from a chatbot, to an AI agent, to an AI agent using YOUR tools - for example, via an MCP.

+ Could allow you to spend more time on the creative process
+ Could allow you to think more across disciplines and approaches
+ Could make your science more accessible to others
- Could restrict the creative process by implicit rules that are present
- Could limit your learning and in-depth understanding
- Workflows, architecture and structure become more important
- Go from engineering of solution to engineering of intent: Spec-driven-dev.
- Limited control with external providers: keep privacy- and legal aspects in mind

Learning goals

- Use AI-assistance in a coding environment
- Basic idea how to adapt AI agents
- Basic idea of legal implications
- Basic idea of privacy and security concerns in the use of such tools