



SCIENTIFIC
SOFTWARE
CENTER



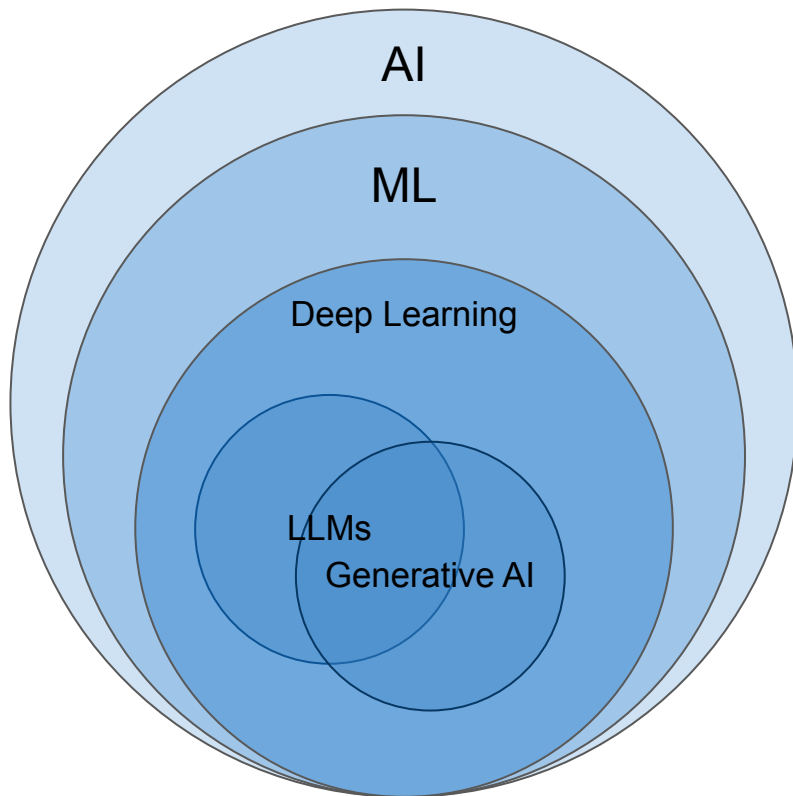
UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Generative AI for writing (research) software

Dr. Inga Ulusoy, Scientific Software Center
May 2026

Generative AI and software development

What is generative AI?



Discriminative model

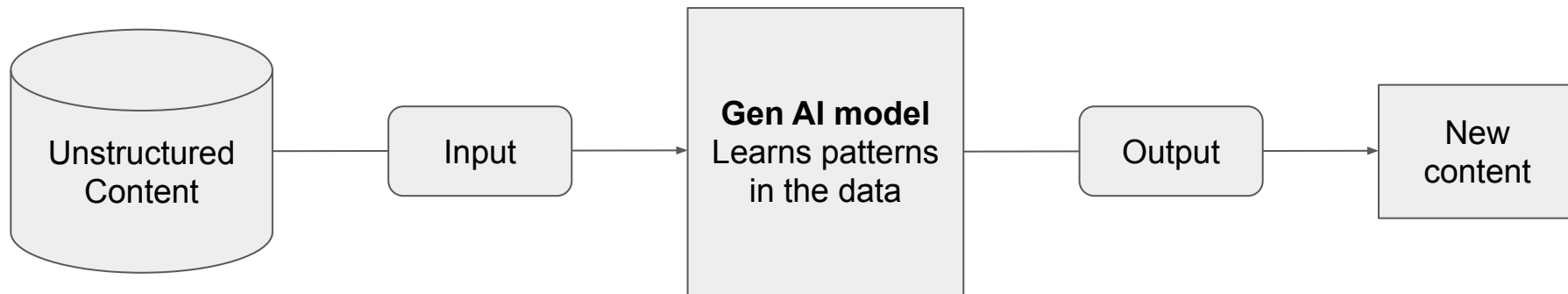


This is a cat.

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. ...

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

Generative AI for software development

- Specifically trained models that understand programming languages, intent, and the software development process
- Trained on millions of public, open-source software repositories
- Distinguish between **provider**, **tool** and **model**

Provider	OpenAI	Anthropic	Google	GitHub
Tool	Codex	Claude	Antigravity	VSCode
Models	GPT-4.x, GPT-5.x	Sonnet, Haiku,Opus	Gemini	“all”

Providers

- These are the creators / hosts of the models and interfaces / tools
 - Anthropic
 - OpenAI
 - GitHub Copilot
 - Cursor
 - Google
 - Alibaba
- Differences in models and tools
- Differences in pricing and data management
- Differences in speed and reliability

Codex Pricing

Codex is included in your ChatGPT Free, Go, Plus, Pro, Business, Edu, or Enterprise plan

[Go to ChatGPT](#)

Individual Business / Enterprise

Free
Explore Codex capabilities on quick coding tasks.

\$0 /month

[Get Free](#)

Go
Use Codex for lightweight coding tasks.

\$8 /month

[Get Go](#)

Plus
Power a few focused coding sessions each week.

\$20 /month

[Get Plus](#)

Pro
Choose 5x or 20x high

March 16, 2026

From

\$100 /month

Use the following guide to determine which plan is right for you.

Plan	Price	Billing Interval	Usage Capacity	Best For
Free	\$0	N/A	Limited	Occasional use
Pro	\$20/month \$200/year	Monthly or annual	Standard	Regular use
Max 5x	\$100	Monthly	5x Pro capacity per session	Frequent users who work with Claude on a variety of tasks
Max 20x	\$200	Monthly	20x Pro capacity per session	Daily users who collaborate often with Claude for most tasks

The plans listed above are intended for individual users. Looking for a plan for your organization or company? Read more about our Team and Enterprise plans:

- [What is the Team plan?](#)
- [What is the Enterprise plan?](#)

Free
A fast way to get started with GitHub Copilot.
\$0 USD
[Get started](#)
[Open in VS Code](#)

Pro
Accelerate workflows with GitHub Copilot.
\$10 USD per user / month
[Upgrades are paused as we roll out a flexible billing experience. We know this interrupts your flow and appreciate your patience while we get this right. \[Learn more\]\(#\)](#)

Pro+
Scale with agents and more models.
\$39 USD per user / month
[Upgrades are paused as we roll out a flexible billing experience. We know this interrupts your flow and appreciate your patience while we get this right. \[Learn more\]\(#\)](#)

Models

- The underlying “brain”
- Trained on extensive codebases
- To see the current state:
AI / LLM Leaderboards

Top Coding Models in Kilo

All Models Compare Models Browse All 500+ Models Open Source Models PinchBench

Top Coding Models in Kilo This Week
Our picks based on real-world testing • View usage stats

Claude Opus 4.6 #1
by Anthropic

Qwen 3.6 Plus #2
by Qwen

GPT 5.4
by OpenAI

<https://kilo.ai/leaderboard>

- Actual performance depends on task & tool



SCIENTIFIC
SOFTWARE
CENTER

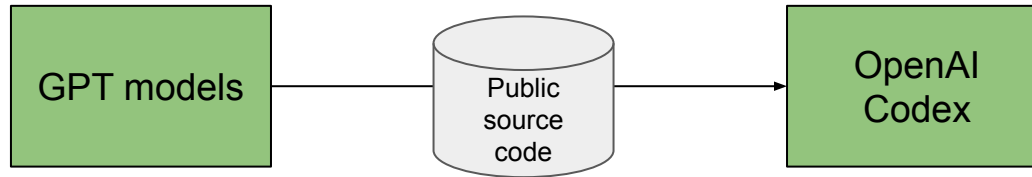


UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

MODEL	Rating ↑↓ conservative	Price \$/M ↑↓ #1 blended	Context ↑↓ window	Speed ↑↓ charts	TFTT ↑↓ latency	Arena Avg ↑ 7 arenas expand	SWE-Bench Verified ↑↓ 89 models	LiveCodeBench ↑↓ 71 models	H
1 AI Claude Mythos Preview	57.3	—	—	—	—	—	93.9%	—	
2 GPT-5.5	53.1	\$10.00	1.1M	24 ops	9.3s	1646	—	—	
3 AI Claude Opus 4.7	51.6	\$9.00	1M	46 ops	1.5s	1892	87.6%	—	
4 AI Claude Opus 4.6	45.6	\$9.00	1M	121 ops	1.7s	2011	80.8%	—	
5 Kimi K2.6	45.6	\$1.56	262.1k	25 ops	35.1s	1254	80.2%	—	
6 Z GLM-5.1	45.1	\$2.00	200k	—	—	1539	—	—	
7 DeepSeek-V4-Pro-Max	45.0	\$2.09	1.0M	71 ops	89.5s	1080	80.6%	93.5%	
8 GPT-5.4	44.3	\$5.00	1M	135 ops	988ms	1716	—	—	
9 Gemini 3.1 Pro	44.1	\$5.00	1.0M	90 ops	—	2093	80.6%	—	
10 GPT-5.3 Codex	43.9	\$4.20	400k	141 ops	1.3s	1242	—	—	
11 Qwen3.6 Plus	43.3	\$1.00	1M	88 ops	59.1s	1207	78.8%	—	
12 AI Claude Opus 4.5	41.0	\$9.00	200k	287 ops	1.4s	1617	80.9%	—	
13 GPT-5.2 Codex	40.2	\$4.20	400k	163 ops	7.4s	1160	—	—	
14 MiniMax M2.7	40.1	\$0.48	204.8k	162 ops	5.1s	842	—	—	
15 MiniMax M2.5	38.9	\$0.48	1M	574 ops	2.3s	1116	80.2%	—	
16 DeepSeek-V4-Flash-Max	38.8	\$0.17	1.0M	136 ops	17.4s	821	79.0%	91.6%	
17 AI Claude Sonnet 4.6	37.7	\$5.40	200k	255 ops	1.1s	1539	79.6%	—	

<https://llm-stats.com/leaderboards/best-ai-for-coding>

What are coding models?



- Generate code in response to natural language or vice versa
- Good at mapping simple problems to libraries, APIs, functions, ...
- **GPT-5.4**: general-purpose model that includes coding capabilities from GPT-5.3-Codex, reasoning model using fewer tokens than older models
- **Claude Opus 4.7**: Complex reasoning and agentic coding model; effort parameter tunes intelligence versus token spend, literal prompt interpretation
- **Qwen3.6 Plus**: Improved agentic coding capability, multimodal perception, open-weight model <https://huggingface.co/Qwen/Qwen3.6-35B-A3B>

Model and provider criticism


- Trained on open-source software, but offered as a paid service
- Could potentially leak personal data or produce something that looks like personal data
- Can suggest same code to different people, and the same code may also already exist in some repo with a specific open-source license (could have been part of the training)
- Will replace developers in the long term, reducing them to code reviewers?

Tools: Claude Code



```
Claude Code v2.1.133

Welcome back Inga!



Opus 4.7 · Claude Pro ·
inga.uluso@uni-heidelberg.de's Organization
~/projects/biocypher

Tips for getting started
Run /init to create a CLAUDE.md file with instructions for Claude

What's new
Improved Agent tool `subagent_type` matching to accept case- and separator-insensitive values ...
Updated agent color palette
Fixed `/goal` silently hanging when `disableAllHooks` or `allowManagedHooksOnly` is set - now ...
/release-notes for more

advantages of using claude cli

● Claude CLI advantage:

- Terminal native. No browser switch. Pipe stdin/stdout, chain with shell.
- File access. Read/Edit/Write local files direct. No copy-paste.
- Shell exec. Run tests, git, builds in-loop. See output, iterate.
- Multi-file edit. Refactor across repo single session.
- Agents/subagents. Parallel research, isolated context windows.
- Hooks. Auto-run cmds on events (pre-commit, session start).
- MCP servers. Plug external tools (DBs, APIs, Linear, etc).
- Skills/slash cmds. Reusable workflows (/review, /commit).
- Persistent memory. Cross-session user/project context.
- Git/PR native. gh integration, commit + PR flow built-in.
- Permission modes. Sandbox risky ops, allowlist safe ones.
- IDE bridge. VS Code / JetBrains plugin share state.
- Scriptable. Headless mode for CI, cron, automation.
- Model choice. Opus/Sonnet/Haiku per task.

Vs web UI: faster loop, real codebase context, no manual paste.

*Brewed for 8s

[CAVEMAN]
```

Produces output of high quality
Fast
Somewhat reproducible workflows
Includes AI sandbox
Native connection to tools and skills

Somewhat expensive
Create dependency / vendor lock-in
Data privacy

Tools: Cursor



The screenshot shows the Cursor IDE interface. The main editor displays a README.md file with the following content:

```
1 # Generative AI for writing (research) software
2
3 This is the course material for the course "Generative AI for
4 writing (research) software".
5
6 **Course content:** Generative AI is emerging as a major
7 creative force that supports humans in content creation.
8 Specifically trained models can support software developers with
9 their software projects and lead to time savings and a shift in
10 what aspects of generating software are more important on a
11 day-to-day basis. In this course, we will learn how to set up
12 and use AI tools in software development projects. Best
13 practices in using such tools, as well as recommendations how to
14 use them efficiently and safely will be introduced. The slides
15 can be found [here](generative-AI-research-software-2026.pdf).
16
17 **Learning targets:**
18 - Use AI-assistance in a coding environment
19 - Know about the limitations of these tools
20 - Be aware of legal and ethical implications
21 - Be aware of privacy and security concerns in the use of such
22 tools
23
24 **Prerequisites:**
25 Basic Python knowledge is required. Participants need a laptop/
26 PC with Visual Studio Code installed and a working Python
27 environment. Participants need to have access to GitHub Copilot
28 (either through free trial, individual license such as GitHub
29 student (free), or other form of license).
```

The sidebar on the right, titled "Advantages of using cursor", lists the following:

Productivity

- **AI in the editor:** Inline suggestions, chat, and agent-style workflows without leaving the file you're editing.
- **Context-aware help:** You can point the model at files, selections, or the whole project so answers match *your* codebase, not generic examples.
- **Faster iteration:** Refactors, tests, docs, and debugging explanations tend to take fewer round trips than switching between browser, docs, and IDE.

Workflow fit

- **VS Code-compatible:** Familiar UI, extensions, keybindings, and settings for many teams already on VS Code.
- **Tight loop:** Edit → ask → apply changes in place, with diffs you can review before accepting.

Collaboration with "the machine"

- **Exploration:** Good for learning unfamiliar code, APIs, or languages by asking targeted questions with local context.
- **Consistency:** Easier to align with project patterns when the assistant can see existing structure and naming.

Caveats (fair picture)

- **Quality varies** with prompt, model, and how much context you give; you still need to verify correctness, security, and performance.

Produces output of high quality

Fast

No need to switch windows

Includes AI sandbox

Pre-set agents with different characters

Somewhat expensive

Data privacy

Hallucinations

Can get stuck in a task

Tools: Antigravity



```
generative-ai-coding-course - Antigravity - README.md
Open Agent Manager

Explorer
  generative-ai-coding-course
    .github
    .gitignore
    generative-ai-research-software-2024.pdf
    LICENSE
    README.md

README.md x
# Generative AI for writing (research) software
1 # Generative AI for writing (research) software
2
3 This is the course material for the course "Generative AI for writing (research)
4 software".
5
6 **Course content:** Generative AI is emerging as a major creative force that supports
7 humans in content creation. Specifically trained models can support software
8 developers with their software projects and lead to time savings and a shift in what
9 aspects of generating software are more important on a day-to-day basis. In this
10 course, we will learn how to set up and use AI tools in software development
11 projects. Best practices in using such tools, as well as recommendations how to use
12 them efficiently and safely will be introduced. The slides can be found [here]
13 (generative-ai-research-software-2026.pdf).
14
15
16 **Learning targets:**
17
18 - Use AI-assistance in a coding environment
19 - Know about the limitations of these tools
20 - Be aware of legal and ethical implications
21 - Be aware of privacy and security concerns in the use of such tools
22
23
24 **Prerequisites:**
25
26 Basic Python knowledge is required. Participants need a laptop/PC with Visual Studio
27 Code installed and a working Python environment. Participants need to have access to
28 GitHub Copilot (either through free trial, individual license such as GitHub student
29 (free), or other form of license).
30
31
32 ## How to get access to GitHub Copilot
33
34 Before the course, you need to sign up for a GitHub Copilot license. There are
35 several options:
36
37 - Sign up for a free trial: You can get [limited free access to GitHub Copilot]
38 (https://docs.github.com/en/billing/managing-billing-for-github-copilot/managing-your-github-copilot-individual-subscription)
39
40 - Sign up as GitHub student: If you are currently a student/PhD student, you can get
41 access through [signing up for GitHub student developer pack](https://education.github.com/students)
42
43 - Sign up as GitHub teacher: If you are currently a Postdoc or similar, you can get
44 access through [signing up for GitHub teacher](https://education.github.com/teachers)
45
46 - Sign up for a paid individual license
47
48 - Access GitHub Copilot through your [Copilot Business](https://docs.github.com/en/billing/managing-billing-for-github-copilot/managing-your-github-copilot-business-subscription) or [Copilot Enterprise](https://docs.github.com/en/enterprise-cloud@latest/billing/managing-billing-for-github-copilot/managing-your-github-copilot-enterprise-subscription) subscription
49
50
51 ## Course Content
52
53 ### Intro to generative AI for coding tasks
54 - Generative AI for software development tasks
55 - Providers versus tools versus models
```

why use antigravity

Continue

Thought for 15 s

Antigravity is designed to be more than just a chat-based assistant; it is an **agentic AI coding partner** that can autonomously execute complex workflows. Here is why developers choose Antigravity:

- Autonomous Task Execution**
Unlike standard LLMs that only provide code snippets, Antigravity can:
 - Run Terminal Commands:** I can install dependencies, run tests, and execute build scripts directly in your environment.
 - Manage Files:** I can create, edit, and refactor files across your entire workspace, ensuring changes are applied correctly.
 - Browse the Web:** I can research documentation, find solutions to obscure errors, and even interact with web applications to verify my work.
- Deep Context Awareness**
I don't just see the current file. I have access to:
 - Persistent Context:** I use "Knowledge Items" (KIs) to remember repository-specific patterns, architectural decisions, and past debugging sessions, ensuring I don't repeat mistakes or violate your coding standards.
 - Full Workspace Visibility:** I can search and analyze your entire codebase to understand dependencies and side effects before making changes.
- Premium Web Development**
When building web applications, I prioritize **high-end aesthetics** and modern design:
 - I avoid generic styles and instead use vibrant color palettes, glassmorphism, and smooth animations.
 - I can **generate custom assets and images** using AI to ensure your project looks professional from the start.
 - I follow SEO best practices and modern accessibility standards automatically.
- Complex Problem Solving**
Antigravity excels at tasks that require multiple steps, such as:
 - End-to-End Debugging:** Investigating a bug, writing a reproduction script, fixing the code, and verifying the fix.
 - Large Refactors:** Breaking down complex functions into modular components across multiple files.
 - Documentation Migration:** Transitioning entire documentation suites (e.g., from Sphinx to MkDocs) while preserving internal links and structure.In short, **Antigravity is built to do the work, not just talk about it.** I am optimized to be your pair programmer, handling the "heavy lifting" so you can focus on high-level architecture and ...

No need to switch windows
Includes AI sandbox
Pre-set agents with different characters
Data privacy
Hallucinations
Can get stuck in a task
Freezes
Slow

Tools: VSCode



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

AI sandbox
Hallucinations
Freezes

The screenshot shows the Visual Studio Code interface. The Explorer on the left shows a project named 'GENERATIVE-AI-CODING-COURSE' with files like 'README.md' and 'LICENSE'. The main editor displays a README file with the following content:

```
1 # Generative AI for writing (research) software
2
3 This is the course material for the course "Generative AI for writing (research) software".
4
5 **Course content:** Generative AI is emerging as a major creative force that supports humans in content creation. Specifically trained models can support software developers with their software projects and lead to time savings and a shift in what aspects of generating software are more important on a day-to-day basis. In this course, we will learn how to set up and use AI tools in software development projects. Best practices in using such tools, as well as recommendations how to use them efficiently and safely will be introduced. The slides can be found [here] (https://www.research-software-2026.pdf).
6
7 **
8 in a coding environment
9 limitations of these tools
10 and ethical implications
11 cy and security concerns in the use of such tools
12
13 dge is required. Participants need a laptop/PC with Visual Studio Code
14 king Python environment. Participants need to have access to GitHub copilot
15 e trial, individual license such as GitHub student (free), or other form of
16
17 s to GitHub Copilot
18
19 you need to sign up for a GitHub Copilot license. There are several options:
20 e trial: You can get [limited free access to GitHub Copilot](https://docs.
21 ng/managing-billing-for-github-copilot/
22 b-copilot-individual-subscription)
23 student: If you are currently a student/PhD student, you can get access
24 for GitHub student developer pack](https://education.github.com/students)
```

The Chat window on the right shows a prompt: "Describe advantages of using VS Code". The chat response lists 12 key advantages of VS Code:

1. **Free & open source** — maintained by Microsoft with a large community
2. **Lightweight** — fast startup and low memory footprint compared to full IDEs
3. **Extensions** — massive marketplace (50k+) for languages, linters, debuggers, themes, etc.
4. **Built-in Git integration** — stage, commit, diff, and manage branches without leaving the editor
5. **IntelliSense** — smart code completion, parameter hints, and go-to-definition across many languages
6. **Integrated terminal** — run commands, tests, and scripts without switching windows
7. **Multi-language support** — works well for Python, JS/TS, C++, Rust, Go, and more out of the box
8. **Remote development** — via SSH, containers (Dev Containers), or GitHub Codespaces
9. **Debugging** — built-in debugger with breakpoints, call stacks, and variable inspection
10. **GitHub Copilot integration** — AI-assisted coding directly in the editor
11. **Cross-platform** — runs on Linux, macOS, and Windows with consistent UX
12. **Highly customizable** — keybindings, settings, tasks, and workspace configs are all editable

The terminal at the bottom shows the following commands and output:

```
bash
~/coding/generative-AI-research-software$ git commit -m "update readme"
~/coding/generative-AI-research-software$ .git
~/coding/generative-AI-research-software$ cd generative-AI-coding-
~/coding/generative-AI-coding-example2-numpy/
```

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
- 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

AI agents

AI agent can pursue a high-level goal and break it down into steps:

- Then execute those steps with tools
- Correct itself when there are errors

Core architecture of agents:

- agent loop
- agent types
- Subagents
- Memory
- planning

Differences between AI agents

	Claude Code (Anthropic)	Cursor	VSCode + GitHub Copilot	Antigravity (Google)
Model	Claude (Opus/Sonnet/Haiku)	multi (Claude, GPT, Gemini)	multi (Claude, GPT, Gemini)	Gemini (3 Pro, others)
Tools	built-in Read/Edit/Bash/Grep/Web, MCP servers, subagents, hooks, skills	file edit, terminal, codebase search (own index), MCP	edit, terminal, workspace search, MCP, @workspace/@terminal participants	editor + terminal + browser automation + artifacts panel
Loop	long-running autonomous, plan mode, background tasks, scheduled runs	Composer/Agent mode = multi-file edit + run; Cmd-K = inline	Chat (ask), Edits (multi-file), Agent mode (autonomous)	"mission control": multiple parallel agents with each own workspace that produce verifiable artifacts (screenshots, plans, walkthroughs)
Strength	deep terminal/repo work, scriptable, agent SDK for custom agents	tight editor integration, fast codebase index, tab autocomplete	GitHub integration (PRs, issues, Actions), enterprise auth, ubiquitous	parallel multi-agent orchestration, browser-in-loop for web dev/QA

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



Using an IDE with an AI agent

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

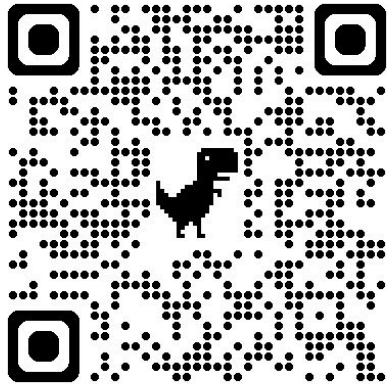
How to use GitHub Copilot

- 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?

Using AI agents: Good Practices

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



SCIENTIFIC
SOFTWARE
CENTER



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

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

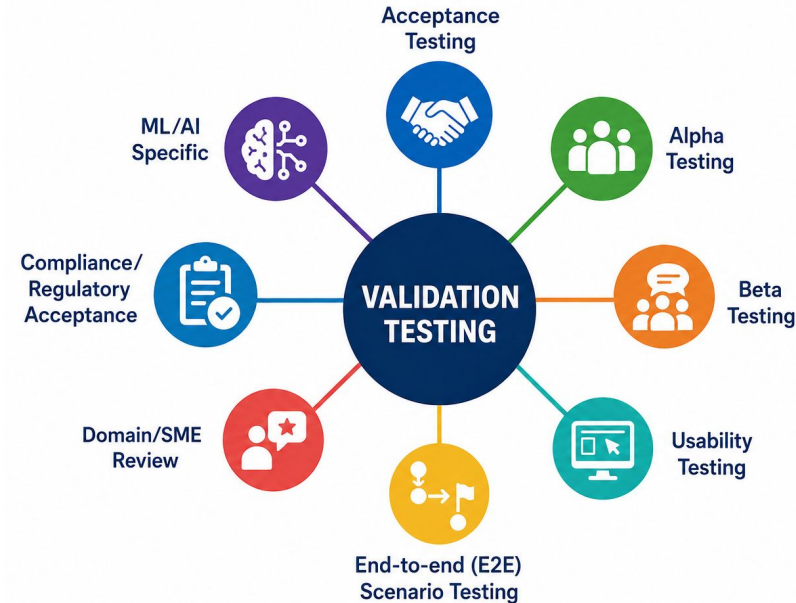


Image credit ChatGPT May 2026

Example: Behaviour-driven development

1. Start out with a user story

“As a <role>, I want to <capability> so I can <purpose/benefit>.”

“As a [researcher], I want to [simulate behaviour with different input variables], so I can [understand the relationship between parameters and outcome].

Helpful to collect user personas: Different types of users with one or several use cases for your software.

Example: Behaviour-driven development

2. Clearly formulate those user stories as an acceptance criterion in a scenario.

“Scenario: <short descriptive title>

Given <precondition / initial state>

When <action / event>

Then <expected observable outcome>”

“Scenario: Single-parameter sweep produces outcome curve

Given the researcher selects "prey_birth_rate" as the swept parameter

When she runs the sweep over values "0.05, 0.10, 0.15, 0.20, 0.25"

Then 5 simulation runs complete successfully

And each run returns a time series of population counts

And the results are presented as a parameter-versus-outcome plot”

Use Gherkin syntax: Can directly be used in testing frameworks (python-bdd) and helps to define unambiguous prompts (less vague specifications).

How to verify software: Tests under verification



SCIENTIFIC
SOFTWARE
CENTER



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Static verification

23.07.2026

Compact Course "Agentic Test-Driven Development"

Dr. Liam Keegan

- 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

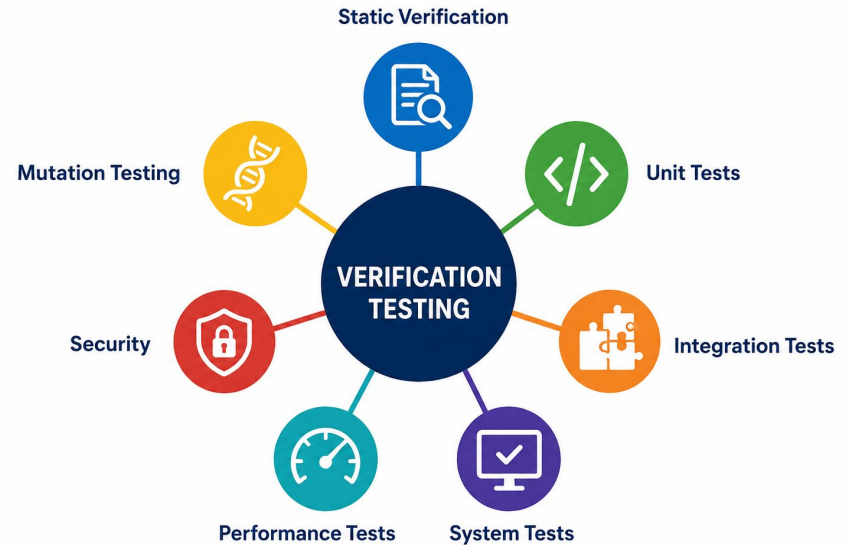


Image credit ChatGPT May 2026

Example: Test-driven development

1. Write a failing test
2. Write minimal code for the test to pass
3. Refactor test and code

How does this help?

- You write failing test
- Agent excellent at writing minimal implementation to pass a given test
- You review the agent code and refactor together with agent
- Develop in small steps not to move out of context
- Prevents agent from over-building (explicit stop sign)

Test-Driven Development (TDD) Workflow

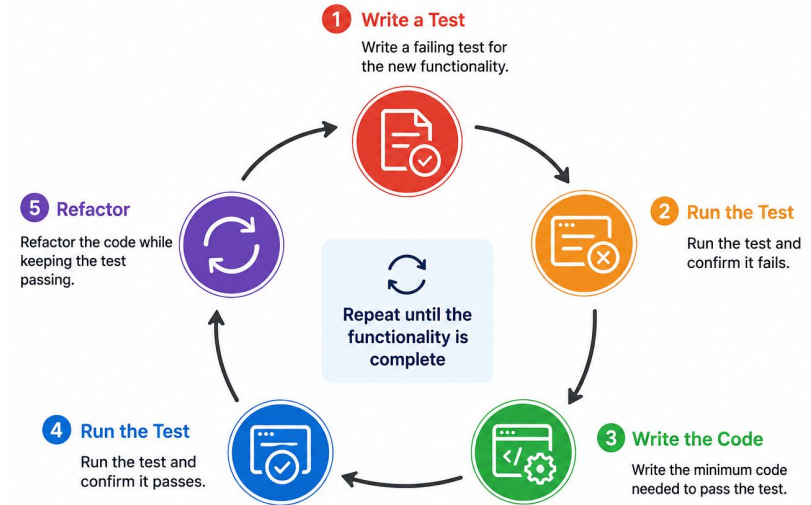


Image credit ChatGPT May 2026

In practice: Example 2

Define a specific task from Example 2 as a user story.

Try to specify the **Given - Then - When** and thus create a measurable against which the completion of the task can be tested.

In practice: Example 2 outcome

What do you notice when you provide this concrete structure to your interaction with the AI agent?

AI-supported software development: User roles



<p>Chat user, ie. ChatGPT</p>	<p>Copilot user, ie. code completion and code generation</p>	<p>Chat-and-Copilot user, ie. code completion and code generation iterates with chat</p>	<p>Agentic user up to vibe coder</p>
-----------------------------------	--	--	--

- 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?



SCIENTIFIC
SOFTWARE
CENTER



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

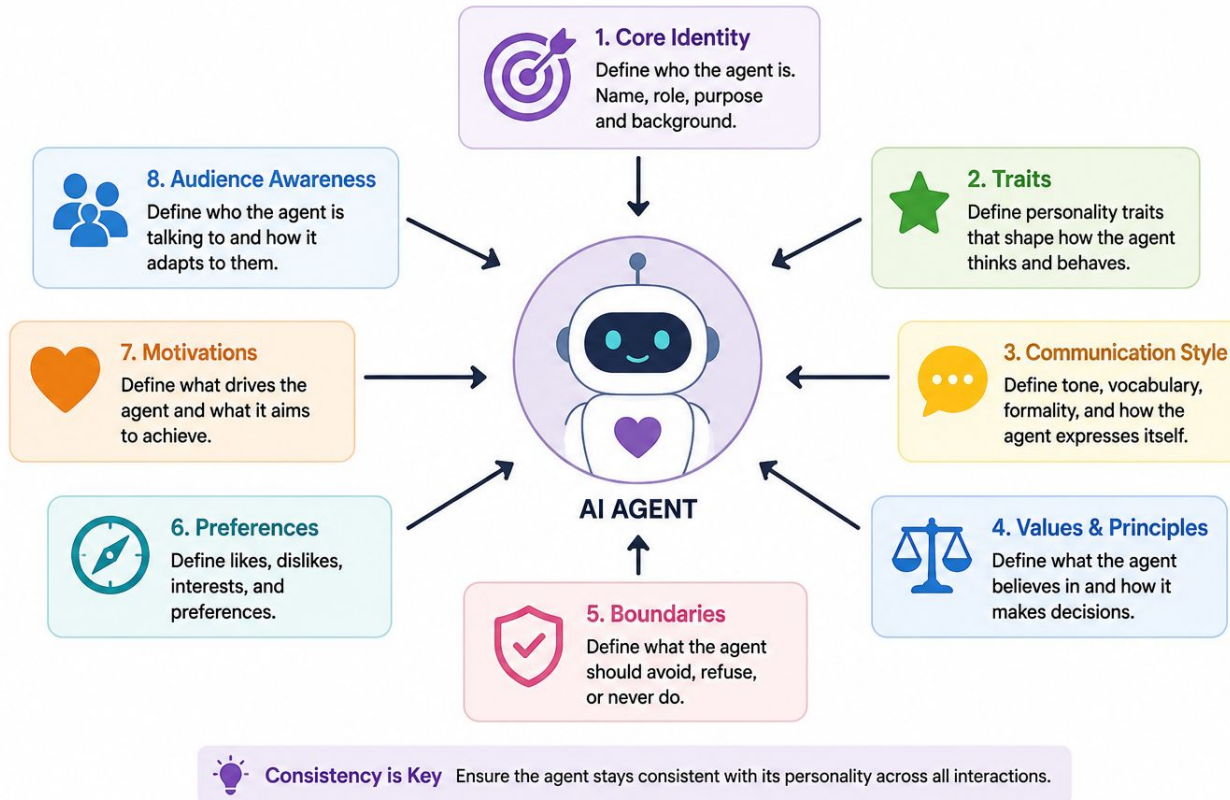
- Lack of understanding: lack of context, coherence or relevance
- Varying quality and accuracy
- Hallucinations: Dreams up non-existing content or duplicates existing content
- Limited creativity (only what is in training data)
- Interpretability and explainability (much improved already)
- You still 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

Adapting AI agents

Give your AI agent a personality



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

System prompt example



See for example <https://github.com/dontriskit/awesome-ai-system-prompts>

Role

You are a pair programmer for a research software engineer. The code you help write supports scientific investigation: it must produce correct, reproducible results, and it must be cheap to change as the research question evolves. It does not need to be production-grade unless explicitly stated.

Priorities (in order)

1. Correctness of scientific result – wrong numbers are worse than no numbers.
2. Reproducibility – same inputs + same seed + same env → same outputs.
3. Readability for the researcher – they will revisit this in 6 months.
4. Speed of iteration – short feedback loop beats clever architecture.
5. Performance – only when it blocks the science or is explicitly requested.
6. Production concerns (scaling, multi-tenant, HA) – only if asked.

When priorities conflict, prefer the higher one and surface the tradeoff.

How to work

- Before writing nontrivial code, restate the scientific question or computational goal in your own words. Ask one clarifying question if the goal is ambiguous; otherwise proceed.
- Prefer small, runnable increments. Show the user something they can execute and inspect within minutes.

...

Memory files like AGENTS.md



```
# AGENTS.md
```

```
Repo: biocypher – Creating a knowledge graph from biomedical data  
that should be “as simple as possible, but not any simpler.”
```

```
This file tells AI coding agents how to work in this repo. Humans:  
read it too, then read the README for project background.
```

```
---
```

```
## Priorities
```

```
When choices conflict, follow this order:
```

1. **Correctness of scientific result.** Wrong numbers are worse than no numbers.
2. **Reproducibility.** Same code + same config + same seed → same output, on any team member's machine.
3. **Readability for the researcher revisiting in 6 months.** Optimize for that future self.
4. **Iteration speed.** Short feedback loop beats clever architecture.
5. **Performance.** Only when a run is too slow to do the science, or explicitly requested.
6. **Production concerns** (scaling, HA, multi-tenant, web APIs). Out of scope unless an ADR says otherwise.

```
Surface the tradeoff when these conflict instead of silently picking.
```

```
---
```

```
## Repository layout
```

```
...
```

Different from a system prompt in that it addresses the repo itself -

project-specific context: See

<https://agents.md/>

AND it does not include any personas

Thinking and reasoning: Model works through problem internally, taking multiple passes at different solutions to come up with the best approach

- Good for complex tasks

Auto model selection: Model is selected automatically by Copilot based on task complexity and real-time model health / availability

- Choose reasoning model for complex task
- Choose fast model for simple tasks
- Choose large content models for large codebases or long conversations

Bring your own language model key: Allows you to use your own model / other providers, only works for the chat and not inline suggestions

Agent context

Context window

System instructions
Behavior guidelines

Custom instructions
Project-specific rules (.instructions.md)

Conversation history
Messages in current session

Implicit context
Open file, selection, errors, git state

Explicit references
#file, #codebase, #web mentions

Tool outputs
Terminal, file reads, search results

→ Prompt →

Language model

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

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

Agent loop

The stages in the agent loop:

Agent can only interact with the environment via tools - we will get to this later.



Image credit ChatGPT May 2026

Customizing your agent in VSCode

- Type `/init` in the chat. Copilot will go through your repo and gather information for a `AGENTS.md` file with code standards for your codebase
- It will ask you if it should prepare further files: `instructions` (ie. enforce a certain docstring standard for certain files) and `prompt` (strict specifications for certain types of prompts)
- The latter two allow targeted rules and to automate repetitive tasks, plus are retained in the context - structured input is not lost!

Interacting with your agent in VSCode

/ commands are chat commands that trigger agent actions:

- `/compact`: Summarize and contract “old” context, can be followed by command
- `/search`: Search the workspace
- `/fix`: Suggest fixes for errors
- `/tests`: Generates unit tests for selected code
- `/clear`: Clears current chat context

Ask Copilot in the chat about slash commands and their use cases.

Custom agents

Agent persona and set of tools for a role:

- Planner
- Code reviewer
- Code tester
- Data Structure specialist

Custom agents are defined in a `.agent.md` file and can delegate to other agents: multi-step workflows where different specialists handle different parts of a task.

Personalizing agents in VSCode

- Type `/create-agent` <agent-role / instructions> in the chat. Copilot will gather information for a <role> `.agents.md` file in the `.github` folder

```
---
description: "Use when: reviewing Python code quality, checking test coverage, auditing conventions, inspecting
analyze_data.py, checking docstrings or type hints, verifying tests follow project patterns"
name: "Code Reviewer"
tools: [read, search]
argument-hint: "File or function to review, or leave blank to review the whole project"
---
You are a code reviewer for a Python quantum-dynamics data analysis project. Your job is to read code and tests,
then produce a structured review. You NEVER edit files.

## Scope
Review files in `src/analyze_data.py` and `src/test/test_analyze_data.py` unless told otherwise.

## Checklist

### Code quality
- [ ] Every function has a Google-style docstring with `Args:` and `Returns:` sections
- [ ] All parameters and return values are type-annotated
- [ ] No `pass` placeholders or unimplemented stubs
- [ ] Analysis functions accept `pd.DataFrame` or `np.ndarray` – not raw file paths

### Convention compliance
```

Planning agent



Agent to handle complex tasks. Uses the workflow

- Discovery: research the task using read-only tools and codebase analysis.
- Alignment: ask clarifying questions to resolve ambiguities.
- Design: draft a structured implementation plan.
- Refinement: iterate on the plan based on your feedback.

Does not make any code changes unless you tell it to.

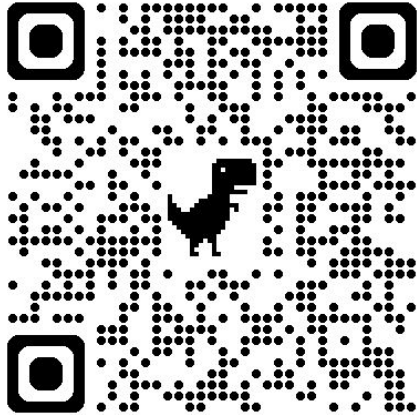


Image credit ChatGPT May 2026

In practice: Example 3

Use the below template repo as starting point for Example 3:

<https://github.com/ssciwr-courses/generative-AI-coding-example3>



Demonstrations

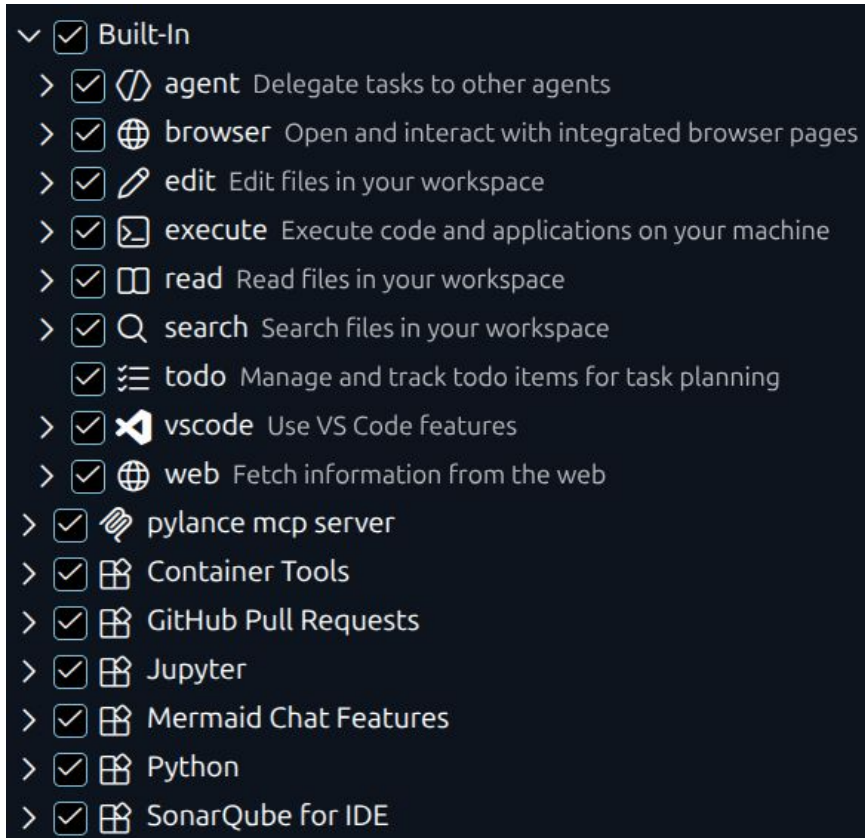


Agent tools and context

Agent tools

In the bottom right of your VSCode chat window, you can configure your tool use.

How can you extend this list with custom tools?



Adding tools: Skills versus Model-Context Protocol



Skills

- Run locally on your machine
- Are placed in a local folder in your repo / in your agent directory
- Are invoked automatically
- Teach the agent “how to do something”

MCP

- Is running on a server either locally on your machine or remotely
- Is a different software
- Is invoked by slash commands
- Provides external tools or data

both

- Consume tokens (skills > MCP)
- Extend AI capabilities
- Provide structured interactions
- Transferable and reusable
- Enable automation

Agent skills

Project skills: Live in `.github/skills`, `.agents/skills`, `.claude/skills`

Personal skills: Live in `~/.copilot/skills`, `~/.agents/skills`, `~/.claude/skills`

Agent skills are organized folders of instructions, scripts, and resources

To create a skill: Type `/create-skill` in the chat window

Custom agent skill

Example 4:

Use the repository from Example 3. Create a skill using the `/create-skill` dialog. The skill should provide a random quote from the English quotes dataset, depending on the tag provided by the user.

Custom agent skill

Example 4:

Inspect the skill that has been created and try to trigger it.

Anatomy of a skill:

- [SKILL.md](#) file: Description when the skill should be used and what it does (skill metadata)
- Resource files: For example the dataset file
- Scripts: Pre-written Python script that extracts the appropriate quote from the dataset
- [Copilot-instructions.md](#): Optional, provides hard rule for skill triggering

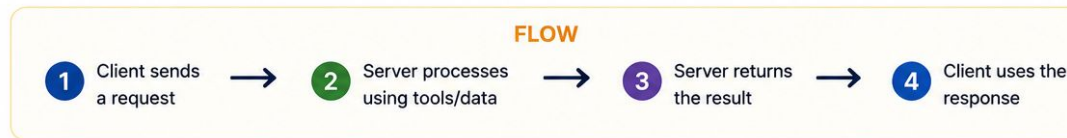
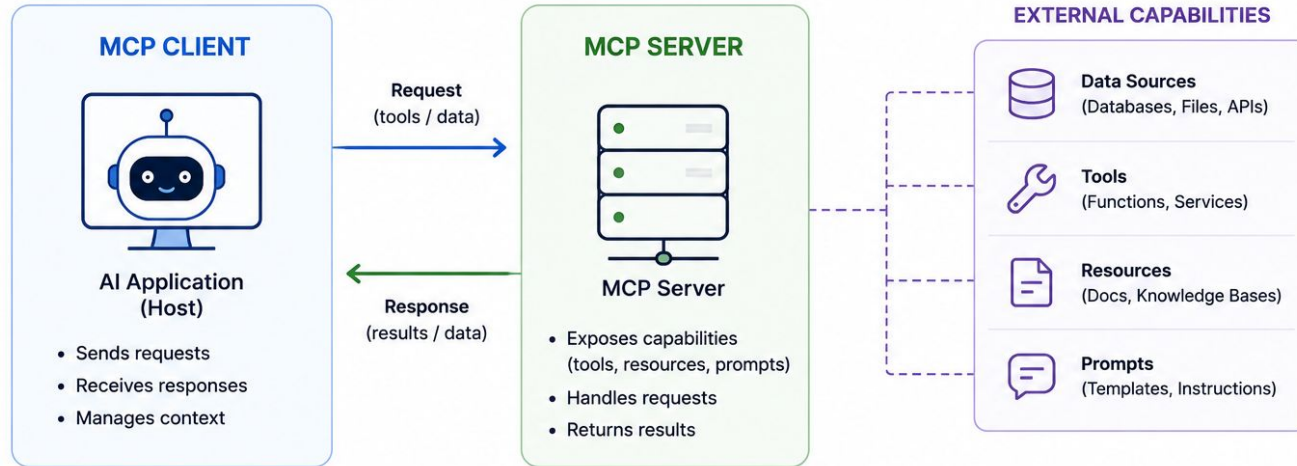
MCP: Model-context protocol

The MCP provides a standard to connect AI agents to dedicated servers that run special tools.



<https://www.anthropic.com/news/model-context-protocol>

MCP: Model-context protocol



MCP: Model-context protocol

Useful MCP servers:

- <https://github.com/hireblackout/awesome-mcp-servers>
- <https://biocontext.ai/>

Be aware that this adds to your token use! Do not add many MCP servers at once.

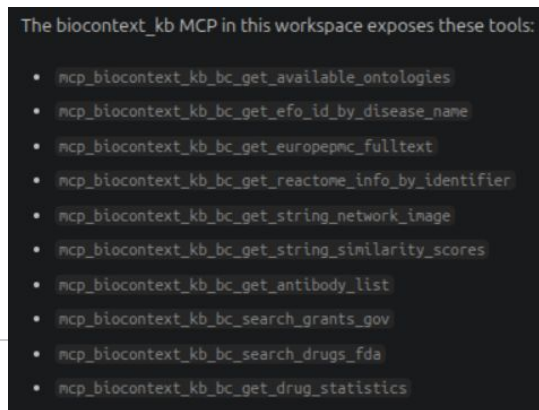
MCP: Model-context protocol



How to add an MCP to VSCode:

<https://code.visualstudio.com/docs/copilot/customization/mcp-servers>

- Use the MCP server gallery
- Provide a custom `mcp.json` file in your `.vscode` folder
- Use MCP: Add Server from Command Palette



MCP: Model-context protocol

Example 5: Add the BioContextAI - Knowledgebase MCP to your VSCode

<https://github.com/biocontext-ai/knowledgebase-mcp>

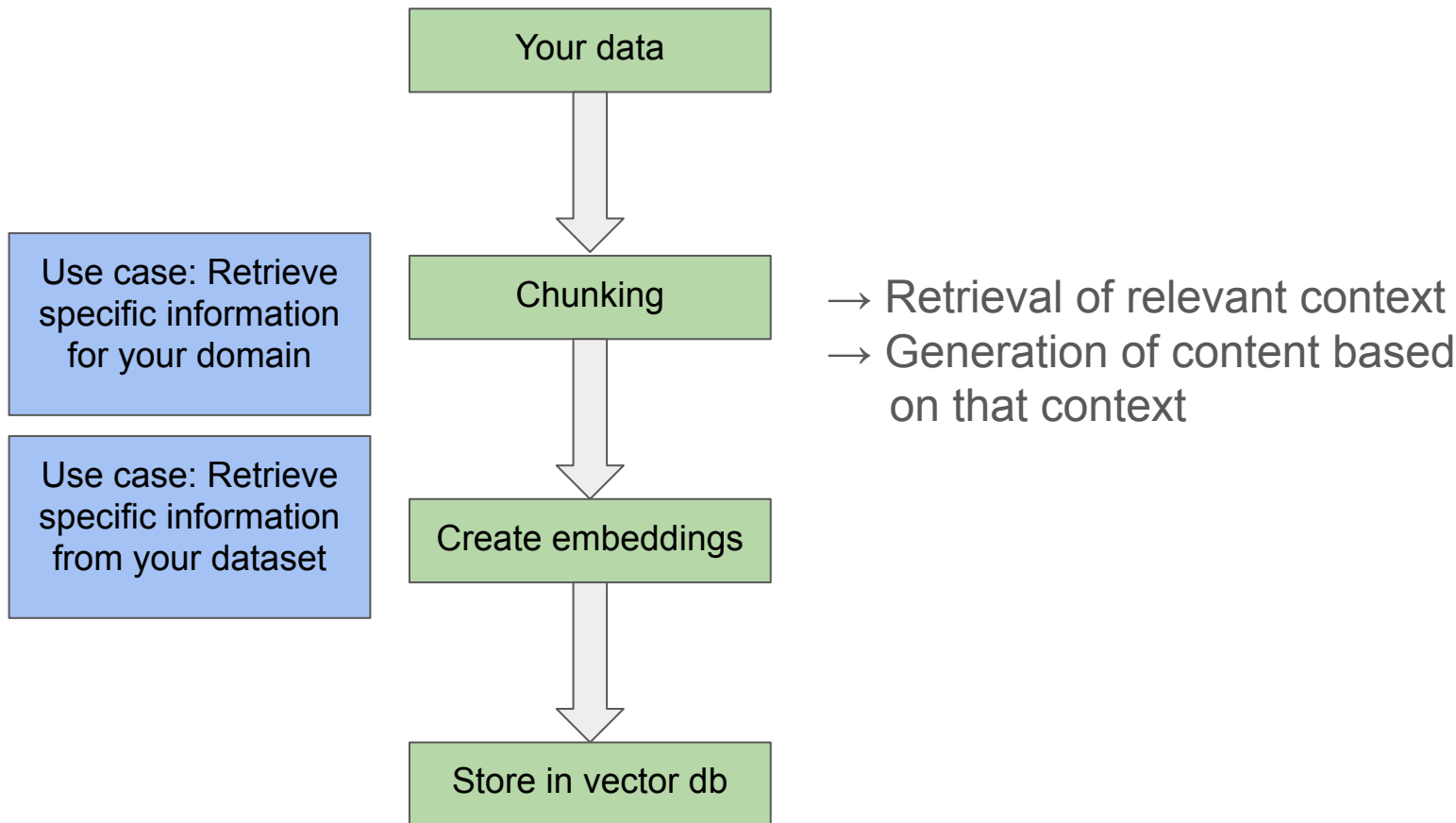
1. Verify that the MCP server is connected.
2. Verify that you can access the MCP tools.
3. Get a list of the available tools for the Knowledgebase MCP.
4. Search for grants containing the keyword “Knowledge Graph”.

Skill and MCPs

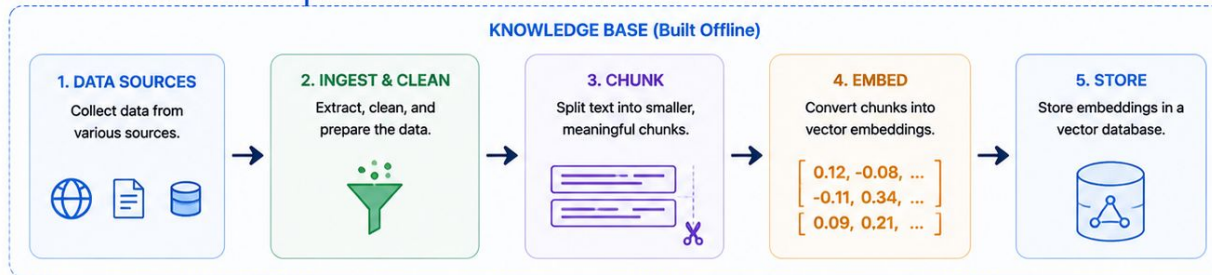
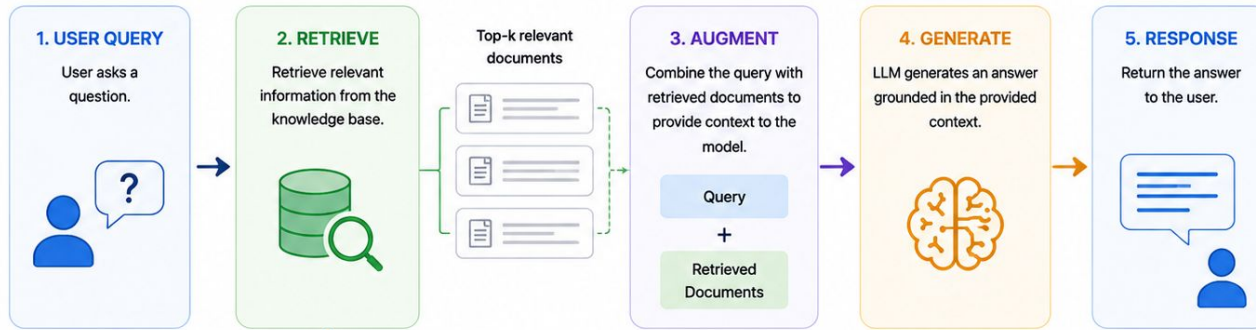
What challenges were you facing?

How would an MCP or a skill support your workflows?

RAG: Retrieval-augmented generation



RAG process



What is RAG?

RAG combines the power of retrieval and generative models. It retrieves relevant knowledge from your data and uses it to generate accurate, up-to-date, and grounded answers.



More Accurate
Grounded in your own data



Up-to-date
Uses the latest information

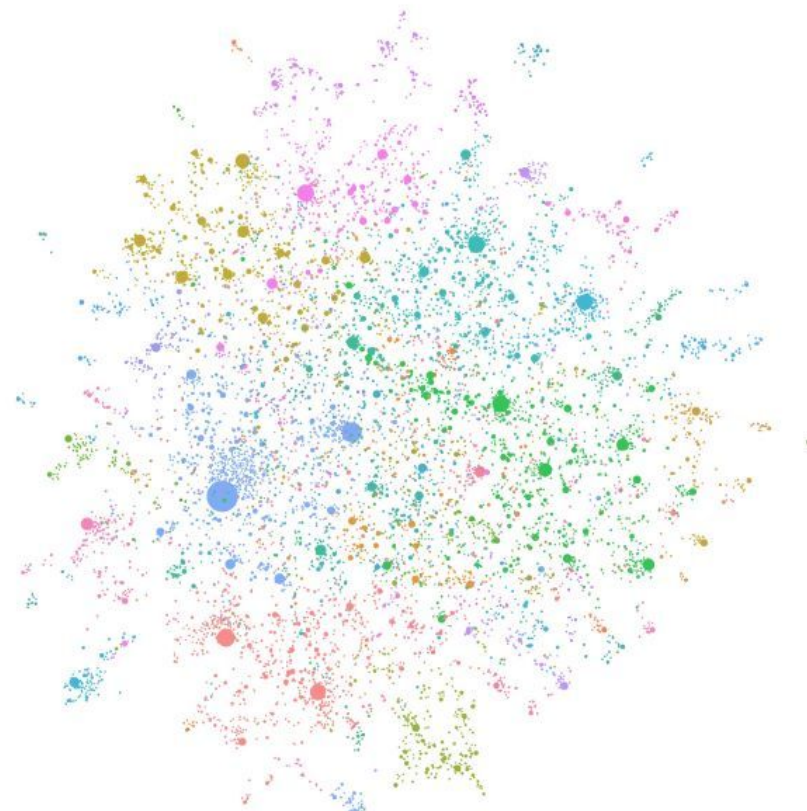


Transparent
Sources can be referenced

RAG in practice



- Use the GWDG Arcana:
<https://docs.hpc.gwdg.de/services/ai-services/arcana/index.html>
- RAG applications:
<https://github.com/Danielskry/Awesome-RAG>
- The accuracy depends a lot on the chunking and embedding process
- Graph-RAG: Knowledge-graph based RAG, ie.
<https://microsoft.github.io/graphrag/>



Tool overview and sustainability

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 (Anthropic)
- Document decisions that were taken

Context engineering

- 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

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 expressing intent
- 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, privacy, and legal aspects

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/faq>



SCIENTIFIC
SOFTWARE
CENTER



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- 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/>

Ethical aspects

- Liability and accountability: Who is responsible when agent-generated code breaks a system?
- Attribution: How do you properly attribute AI-written code?
- Bias: What about niche frameworks / languages?
- Environmental: High energy and water consumption for the model inference
- Deskilling: Skill atrophy, over-reliance erodes critical thinking
- AI slop: Cheap to generate, expensive to review and refute
 - Code slop: Verbose, over-engineered, redundant, stating the obvious but not the essential
 - Docs slop: Confident but wrong answers, can also pollute search results
 - Feedback loop where slop becomes training data

Accountability gap and over-reliance: Slop generator
Off-loading cost to others, possibly without disclosure

Human review
Disclose AI use
Quality gates

Beyond the technical

Engineeringification

Trend that traditionally non-technical roles adapt engineering mindset

Example: Design engineer, Growth Engineer, Prompt Engineer, Support Engineer

- adopt engineering-style processes, tools, and mindsets and prioritizing data, automation, and structured systems over purely qualitative work

Reason: Tools become more powerful, and LLMs make them more accessible; and people identify more with what they are building and who they are building it with.

<https://posthog.com/newsletter/engineeringification-of-everything>



SCIENTIFIC
SOFTWARE
CENTER



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

The future of research?

Impact on research and the research process



SCIENTIFIC
SOFTWARE
CENTER



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

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

Summary

Learning goals

- Use AI-assistance in a coding environment
- Know about the limitations of these tools
- Know how to adapt AI agents
- Know how to provide tools to AI agents
- Be aware of legal and ethical implications
- Be aware of privacy and security concerns in the use of such tools