Coding with AI

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction

What is GenAI?

Generative AI is the broad cateagory of AI systems that can generate content. This includes LLMs, but also specialized systems that can generate audio, video, images, code and so on. These models work by learning patterns from large amounts of training data, and using using them to generate novel content.

A large language model (LLM) is a particular type of model that works with natural language text.

Features of LLMs that make them interesting in this course:

- **Simple interaction**: You communicate with models using plain language. Describe what you want, provide relevant data and get a reply (image, video, text as appriate to the model and prompt).
- **Novel content creation**: Models can produce original content that didn't exist in the training data, rather than just classifying existing content.
- **Arbitrary tasks**: Models can answer arbitrary questions and even perform tasks, within some specific constraints.

Choosing a Model

There are a large number of models that are available for consumer use.

- Most high-performance models run `remotely`, in the cloud (usually on clusters of GPUs). Vendors charge for access to top-models.
  - e.g., Copilot, Gemini, ChatGPT, Claude
- Low-medium performance models can be run ‘locally’, but are generally less capable and slower to execute. These are suitable for simpler, limited tasks.
  - e.g., Qwen 3, Gemma 3

> **Tip**
>
> We are mostly interested in specialized coding models, which we will discuss in the next section.

Writing Effective Prompts

Interacting with a model looks something like this:

Step 1: Pick a model suitable for your task.
- e.g., Coding model e.g., Claude or a generic model e.g., Copilot.

Step 2: Create a new session & prompt.
- Models generally will have a CLI or web interface. Create a new session in that interface, and supply your prompt.
- e.g., “List the requirements for a minesweeper game…”.

Step 3: Review the answer and adjust.
- Ask the model to “expand” for more information, or “refine” the previous answer towards more useful information.
- “e.g., Expand the previous question to include the following requirements…”

Writing Effective Prompts

Models get their context from your prompt; the more precise and complete the prompt is, the more useful the result will be. GenAI is very helpful for open-ended questions like this one.

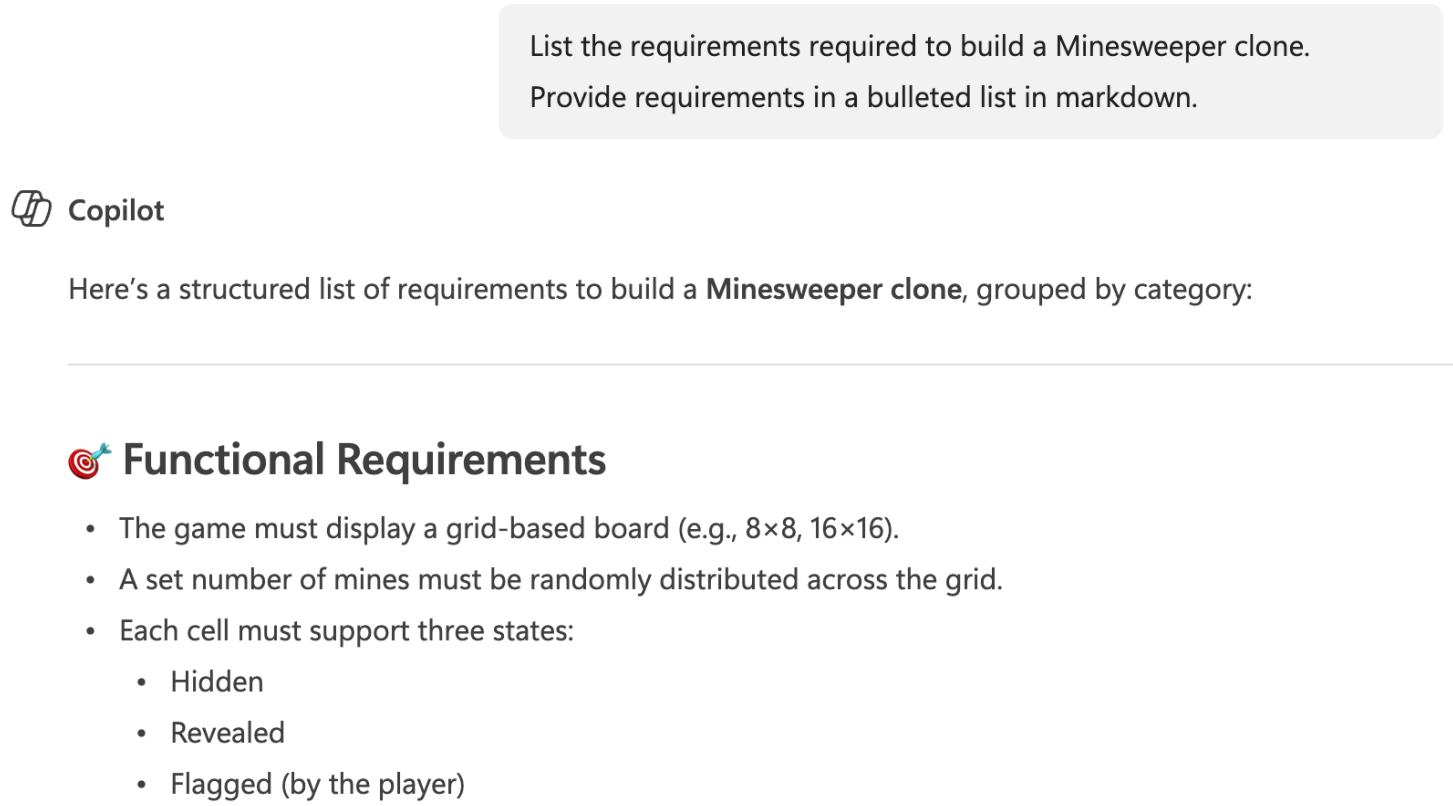

Figure 1: Using Copilot through a web interface to generate application requirements for a project. GenAI is great for things like brainstorming product requirements.

Writing Effective Prompts

We want to build effective prompts that get the best possible result quickly.

> Prompt engineering is a relatively new discipline for developing and optimizing prompts to efficiently use language models (LMs) for a wide variety of applications and research topics. Prompt engineering skills help to better understand the capabilities and limitations of large language models (LLMs).
>
> — Prompt Engineering Guide [1]

Why is this important?

- Processing a prompt can take time - from minutes to hours. A more effective prompt can save significant time.
- You pay for usage! The fewer prompts you need, the cheaper the session.

Prompt Format

The PLOC framework describes one way to format your prompts [2].

- **Persona**: The perspective you want the AI to take (this limits AI to specific domain/subject knowledge)
- **Level**: The level of complexity would you like the output to have (1st yr student, graduate, instructor.)
- **Outcome**: What specifically do you want AI to give you (e.g., format, inclusions, exclusions, etc.)?
- **Context**: Any additional context that will help provide more relevant output.

Use elements of each of these components in your instructions to the agent.

- Talk to it in the first-person!
- Be explicit. Don't be afraid to repeat yourself to reinforce important points.
- Be polite. They might become sentient one day 😳.

Example:

> You are a mid-level software developer, with extensive application development experience. (**Persona**)
>
> You are producing a Minesweeper game that you hope to publish on the app store. Generate a complete list of requirements that would need to be met to match the Windows Minesweeper game's functionality. (**Outcome**) Keep the requirements simple and relatively easy to implement. (**Level**)
>
> This game will need to run on Windows desktop and be publishable on the Microsoft Store. (**Context**)

> **Tip**
>
> Competing with Microsoft's free version of Minesweeper, on their own online store, may not be the smartest marketing play...

Concerns About AI Use

Privacy & Security

The University has strict policies around AI use, for a number of reasons:

1. **Privacy of data**: Users may unknowingly provide sensitive information in their queries, prompts, and inputs. Since GenAI is trained on text that is entered into it, the tool may disclose sensitive information in an output.
2. **Loss of intellectual property** (IP): Entering sensitive or proprietary information into a GenAI tool opens you up to the loss of confidentiality and IP rights (e.g., research data, patented or copyrighted data, source code).
3. **Threat or "bad" actors** have been known to harvest sensitive information to impersonate individuals or spread false information. They can also steal IP data quickly and in bulk, and use it to create targeted cyberattacks.

> ⚠ **Warning**
>
> NEVER upload private or confidential data to a model. It's too risky.

Concerns About AI Use

Hallucinations

Large language models (LLMs) hallucinate, a concept popularized by Google AI researchers in 2018.

Hallucination refer to “mistakes in the generated text that are semantically or syntactically plausible but are in fact incorrect or nonsensical.” [3]

In other words, the text that a model generates may not be factually correct. LLMs have been known to cite links to web sites and online materials that don’t exist. This includes citing made-up legal quotes and citations which were then used in a court case.

> ⚠ **Warning**
>
> Particularly in the case of generating text, you need to fact-check! Don't assume the LLM is correct.

Concerns About AI Use

Human-Labor Costs

Tech companies often exploit workers to reduce the costs of developing AI.

- Rowe, N. (2023). Millions of Workers are Training AI Models for Pennies. Wired.
- Perrigo, B. (2023). OpenAI Used Kenyan Workers on Less Than $2 Per Hour to Make ChatGPT Less Toxic. Time.

Most tech companies are experimenting with AI, or using it in targeted ways. Optimistic companies are “investing in AI” by laying off human developers.

- Jamali, L. (2025). Microsoft to cut up to 9,000 more jobs as it invests in AI. BBC Online.
- Brynjolfsson, E., Chandar, B., & Chen, R. (2025). Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence. Stanford Digital Economy Lab.

Concerns About AI Use

Environmental Costs

The environmental costs of AI (data centers) is staggering. We don't fully understand the implications.

- Zewe, A. (2025). Explained: Generative AI's environmental impact. MIT News.
- Aczel, M., Chamanara, S., Matin, M., Farsi, A., Marwala, T., Madani, K. (2026). Environmental Cost of AI's Energy Use: Carbon, Water and Land Footprints. United Nations University Institute for Water, Environment and Health (UNU-INWEH)

Guiding Principles

There's a lot to consider. Here's my guiding principles for this course.

1. **AI cannot be ignored**. We cannot ignore the growing importance of GenAI. We need to consider it's impact on software development.

2. **AI should be used strategically**. Understand where AI adds value e.g., "explain this code to me", "add feature X using pattern Y to achive Z". Avoid the temptation to vibe code, which often leads to a Ball-of-Mud instead of a well-designed application.

3. **AI should be used ethically**. When and where you can, mitigate the environmental and human costs. Don't use GenAI except where it adds significant value. Use local models to reduce environmental impact.

4. **AI is just a tool**. AI works best when you can guide it through careful code refinement. You, the developer, need to know what the end-product should look like; what patterns should be used, how the code should be structured. Ultimately, you are responsible for what is produced.

Coding with GenAI

Introduction

We're going to focus on using AI for development tasks. GenAI can provide support for most of your activities.

1. During the requirements gathering stage, use it to help brainstorm features or refine how your features work.
2. Provide it with a detailed list of tasks and ask it to generate a Gantt chart or burndown chart for project planning.
3. Explain some demo code, or something your teammate produced.
4. Ask it to fix build problems when you don't remember build script syntax.
5. Ask it to generate unit tests for a new feature.

Coding Models

For coding tasks, use coding-specific models. The top models at this time are:

- The `Claude` series produced by Anthropic.
  - Their agentic coding interface is Claude Code. Claude code has a CLI, or a web-interface that can be used to feed it source code and interact with the model. Claude has an API that allows integration with IDEs and editors.
- `ChatGPT`, produced by OpenAI.
  - Their agentic coding interface is Codex. It also has a CLI, and it can be integrated with other tools and editors. e.g., IntelliJ can integrate with it.
- `Gemini` (Google) and `Copilot` (Microsoft) are less commonly used for coding, but are still quite capable.

> **Tip**
>
> Frontier models are very expensive. You can usually get great results without using the most-expensive model every time.

AI-Assisted Programming

AI-assisted programming is an approach to software development where programmers “interact with LLMs to generate code in an iterative fashion… the developers are not just focused on the end result of the implementation, but also on hands-on guiding the implementation” [4], [5]

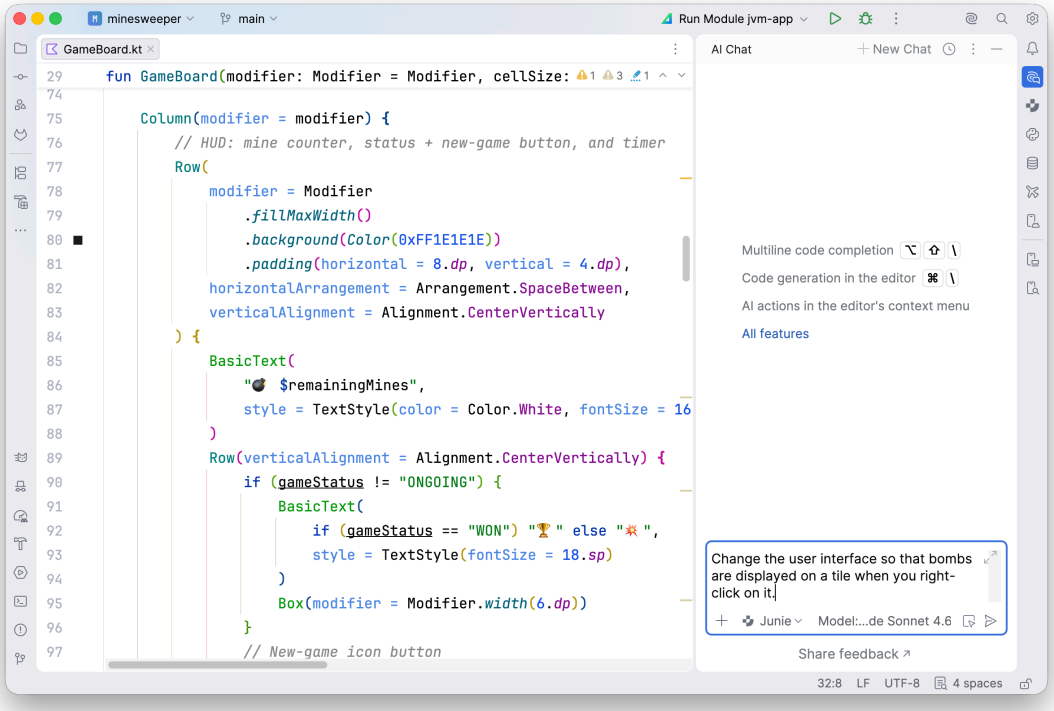

Figure 2: Asking Junie to add a feature. Well-supported in IntelliJ IDEA. [6]

AI-Assisted Programming

This approach generally requires:

- integration between your model and your editor,
- the ability for your model to interact with other tools e.g., to run unit tests and check results.

Most commercial models have some means of integrating, or a separate agent that can facilitate e.g., Junie.

What's the appeal of AI-assisted programming?

1. Your LLM has context from your source code. You don't need to manually upload thousands of files to work on your coding tasks.
2. You are leveraging the expertise of the developer, who has to provide step-by-step instructions on what should be done, and often how it should be done (without necessarily writing the code).
3. You are more likely to produce code that is readable, maintainable, and fits your organization or project constraints.

AI-Assisted Programming

Coding Editors

Many coding editors have pivoted towards integrating AI into your development workflow. The main advantages of doing this over working from a prompt, is that your editor can provide contextual information to the LLM. i.e., function graph, details on dependencies and modules and so on.

Examples include:

- Cursor
- Zed
- VS Code
- IntelliJ IDEA

> **Tip**
>
> If you want to use AI, I strongly recommend that you use an editor that has a deeper integration. I'll use IntelliJ IDEA and the Junie agent.

Coding Prompts

Coding is about achieving some goal, so your prompt will tend to be a *description of the tasks to perform*. The more specific your instructions are, the better the results will be. What should you include in your prompts? [1], [5]

- Details on programming language, supported platform, project configuration.
- Details on how your source files should be structured.
- Programming style e.g., “prefer a functional style when possible”.
- Architectural patterns or design patterns that you want used or respected.
- What NOT to change e.g., “do not touch this imported module”.
- Whether or not to generate unit tests.

e.g., “Create a Kotlin project in the style supported by the Kotlin toolkit. This project should support desktop/JVM and Android targets, and include the dependencies for a shared Compose user interface. Include a simple screen that shows `Hello World` in the center of the screen, and which runs on all platforms.”

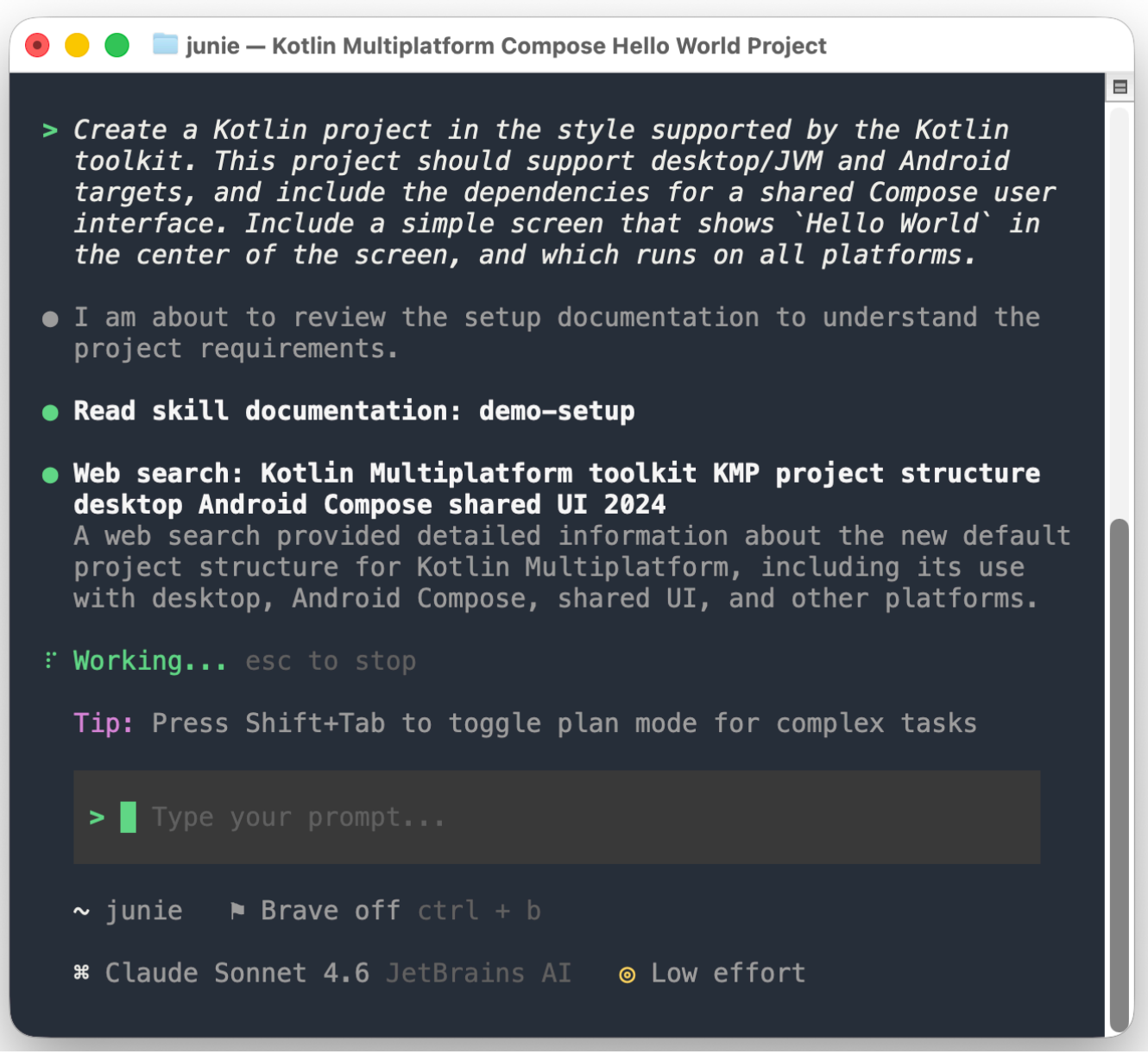

Figure 3: Junie running this prompt from the CLI.

Setup & Configuration

Models and agents have their own specific standards on how they should be configured. However, all agents and models use markdown files to store project information e.g., `specification.md` or `skills.md`.

Let's identify some standards that have emerged.

Agents.md

Agents.md is an emerging standard configuration file (a "readme" for agents) that you save in your project [7]. It's basically an input that is fed to the model at the start of a session that provides a foundation for further prompts.

```
## Project Identity
- **Package**: com.example.snacksquad
- **Language**: Kotlin only — no Java files
- **Min SDK**: 24, Target SDK: 34, Compile SDK: 34
- **Build System**: Kotlin Toolchain (project.yaml).
```

Skills

Agent Skills are a "lightweight, open format for extending AI agent capabilities with specialized knowledge and workflows" [8]. Unlike prompts, which are

applied to every query, agent skills are only used when they match the needs of the current task. There is a format specification and directories of skills that developers have published.

A skill is a directory containing, at minimum, a SKILL.md file:

```
skill-name/
├── SKILL.md          # Required: metadata + instructions
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
├── assets/           # Optional: templates, resources
└── ...               # Any additional files or directories
```

Here's a detailed example of a Junie skill file that follows the standard:

```
---
name: my-skill-name
description: A short description of what this skill provides
---

# My Skill Name
```

```
Use this skill when [describe when Junie should use this skill].

## Key Principles
- Principle 1
- Principle 2

## Guidelines
- Guideline 1
- Guideline 2

## Examples

[Include code examples, patterns, or references to project files]

## Checklist

See `checklists/review.md` for a detailed checklist.
```

Spec-Driven Development

When working with prompts, you need disciplined workflows to ensure that:

- you are providing the model with all required background information,
- any code that is produced reflects and fulfils your requirements,
- your model keeps track of what it “learns” each time it executes, so that you don’t have to keep repeating the same queries.

We’ll address this by using GenAI to manage all of our development artifacts [9]

> **Tip**
>
> It's helpful to think of the model as an inexperienced but enthusiastic developer that you are managing. They are eager and will do what you ask to the best of their ability, but they are limited to what you tell them. Clear instructions are important!

Spec-Driven Development

This is the practice of focusing on delivering a high-quality detailed specification to the agent before you start any implementation.

- Collaborate with the AI to build a detailed specificaation. Review it and make manual adjustments.
- If you are working with an existing codebase, ask your model to generate the specification from existing code! Tweak it and save it in a specifications file.
- Once you have a specification for the work that needs to be done, as your model to generate a plan based on that specification. Review and adjust.
- Finally, use the plan as the basis for your prompt. [10].

Test-Driven Development

As you iterate on your code, you should also ask your agents to produce corresponding unit tests and run them with every iteration.

- This reduces hallucinations or non-working code.
- It ensures that you don’t break working code.

Spec-Driven Development

Track Project Status

When using an AI coding agent, continuity across coding sessions is really important. However, your agent has limited information that it tracks between sessions. For this reason, you should externalize the project state into a set of files that you can have your agent reference and update. [9]

In this course, we recommend that you create a sub-folder in your project to store useful files for your agent e.g., `docs`. We recommend the following files:

- `docs/requirements.md`
  - a structured requirements document that describes your project in detail.
- `docs/plan.md`
  - an execution plan for your agent that you will generate from your requirements.
- `docs/tasks.md`
  - a detailed list of tasks to complete, generated from your plan.

Agentic Coding

“Agentic coding is a software development approach where autonomous AI agents plan, write, test, and modify code with minimal human intervention. Unlike traditional AI coding assistants that provide suggestions or wait for user prompts, agentic coding agents take high-level instructions and independently execute them to completion.”

- Claude Haiku 4.5 (2026) [11]

Key characteristics:

- **Autonomous execution** - The AI agent reads a goal, plans a sequence of actions, executes those actions using tools (file edits, shell commands, API calls), and iterates until the task is complete
- **Tool use** - Agents can call multiple tools in a loop to achieve objectives, functioning more like a skilled contractor than a passive consultant
- **Context awareness** - They maintain state and understanding across entire development sessions, adapting to your codebase
- **Iterative refinement** - Agents can verify results, identify failusres, and iterate on solutions without waiting for human feedback at each step

Agentic Coding

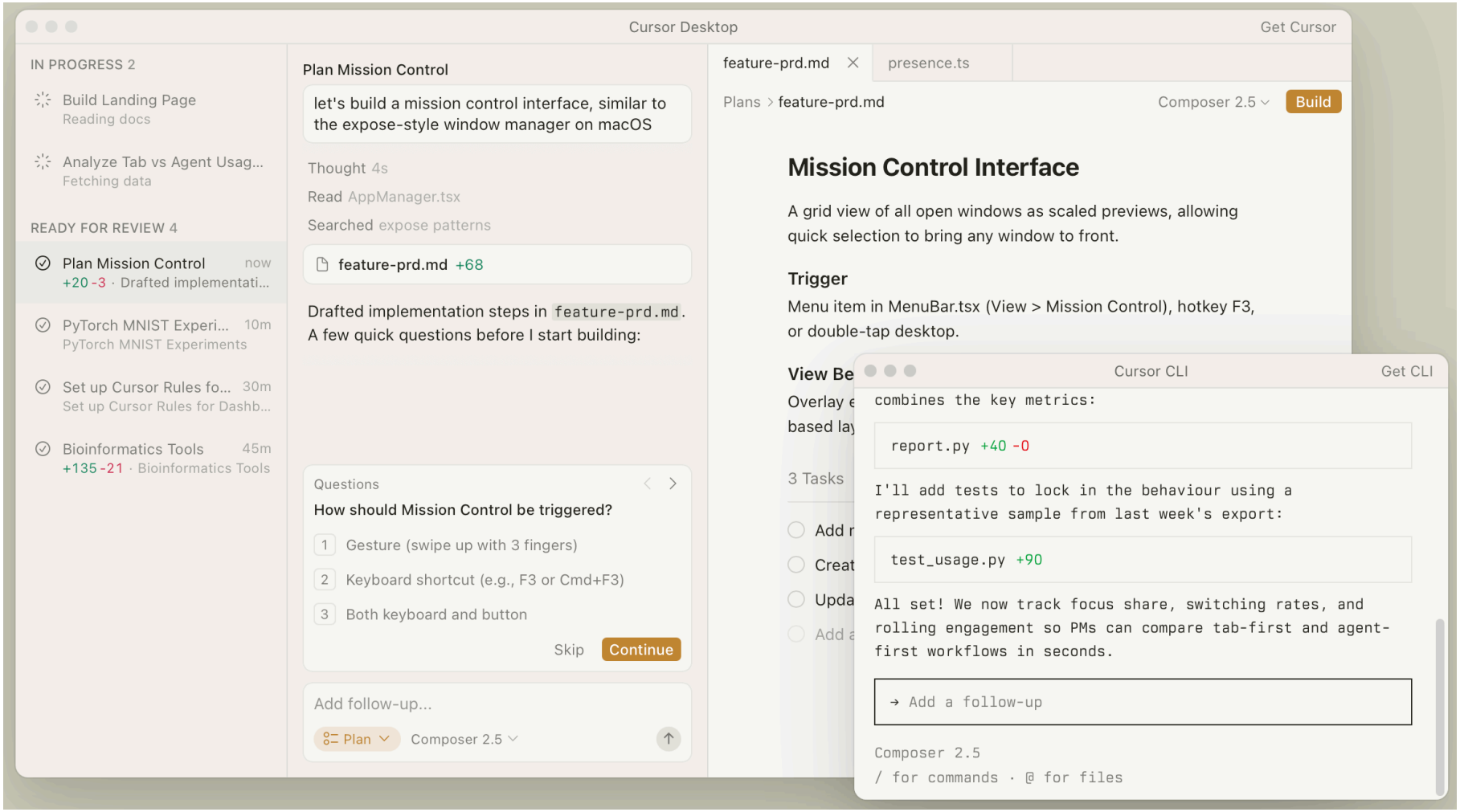

Figure 4: Cursor running agents in the background while the user can edit code, or revise prompts.

Agentic Coding

What is the appeal of agentic-coding?

1. Reduces time to deliver software. Potential for more “radical leaps forward”, since agents are empowered to automatically take action e.g., creating tests, making code changes directly.
2. Since the agent is empowered, you can let it iterate for a longer period of time on more complex problems.
3. Changing how developers work. Agents become tools for a developer; their job becomes a coordinator of work instead of the person writing it. [12]

Workflows

Here are some tips and tricks for using AI effectively when coding [13].

1. Use IDE + CLI Search out and install official plugins for whatever IDE and model combination you are using.

- e.g., Junie for IntelliJ IDEA (which runs any model), or Copilot in VS Code.

The CLI interface for your model is likely more powerful than the IDE plugin. Open a terminal directly in your IDE to access advanced configuration and commands.

2. Use containers! Consider running your agents in a Docker container to safeguard your development environment. You probably don't need this for small changes or simple prompts, but if you enable "trusted mode" (where an agent can do things without asking you first for permission) then this may be necessary.

3. Install & use skills Skills are preconfigured capabilities that you can give your agent. Install these into your project based on project details. See skills.sh.

4. Planning mode Use plan mode to have the agent prototype and plan for you before you have it make changes. This gives you opportunities to fine-tune your prompt, it's faster and uses fewer tokens.

5. Rubber-duck your plan Take your plan and feed it into a different model for critique. Cross-validation more likely to produce a correct result than a single-pass.

6. Be careful with context Clear your context between planning and implementation passes to ensure that only the planning output is being used. Clear caches between planning and implementing. e.g., new session, `implement plan.md`.

7. Use sub-agents and local models When appropriate, use small models or sub-agents to speed up the implementation. Most CLI tools can do this for you i.e. act as agent coordinators.

9. Multi-agent review Have your agent do a review pass! Ask it to review the output and look for areas to improve. Cross-validate with second model.

Agentic Coding

e.g.,

“Review this project for completeness and accuracy, improvements. Run that review by GPT5.5 and get its input into your findings. Display findings in a list ordered by priority. Fix every item in the list. Then repeat this process until both you and GPT5.5 agree that he only remaining items have diminishing returns.”

Examples

Minesweeper

Let’s build a simple game to demonstrate using Junie for spec-driven development.

We’ll follow the workflow we described above, where we write detailed requirements and then use a model to generate a plan file and task list from the specification [10]:

`Requirements -> Plan -> Task list`

This example is stored in our GitLab public repository - see demos/minesweeper.

Figure 5: “Minesweeper running on macOS”

Minesweeper

We started by generating an empty project from the command-line
e.g., `kotlin init` and followed the instructions for the platforms we wanted.

Next, we set up high-level guidelines for Junie to follow:

`AGENTS.md`

```
# A short bullet list of the most critical rules the agent must
follow before doing anything

- [ ] Read this file and `README.md` before acting.
- [ ] Update `CHANGELOG.md` with a summary of user-facing changes.

# Feature development and decision-making

- All code should be written in Idiomatic Kotlin.
- Make small, targeted changes instead of building for future needs.
- If something is unclear, ask before making assumptions.
- Avoid using global variables and mutable state whenever possible.
- Prefer functional code over imperative code.
```

Next, we defined our high-level specification.

docs/requirements.md

```
# High-level requirements

Layout

- The game must display a grid-based board (e.g., 8×8, 16×16).
- Mines must be randomly distributed across the grid.
- Each cell must support three states:
  - Hidden
  - Revealed
  - Flagged (by the player)

When a player clicks a cell:

- If it contains a mine → game ends (loss).
- If it is empty → reveal the number of adjacent mines.
- If it has no adjacent mines → recursively reveal neighbors.
```

Minesweeper

We used this prompt to have Junie review requirements and generate a plan.

```
Junie> Read the `docs/requirements.md` file and generate a detailed
development plan in `docs/plan.md` to implement those requirements.
Include the following in the development plan:

- A short overview of the project goals.
- The main steps required to complete it, listed in order.
- Any dependencies, risks or considerations to keep in mind.
```

> **Note**
>
> Junie supports Plan mode by pressing Shift-TAB. This doesn't save anything, but instead just outputs the plan to the screen. We recommend NOT doing this, but having Junie save to output files instead. It's helpful to have everything documented.

Minesweeper

We reviewed the plan file and then let Junie execute it. This generated our `docs/plan.md` which is the main development plan for this project! Next we want to break this down into smaller steps that Junie can execute. We used this prompt to convert the `plan.md` into `docs/tasks.md`:

```
Junie> Read the contents of `docs/plan.md` and generate a detailed,
enumerated task list based on it.

- Each task should be clear, actionable, and written in a way that
allows it to be marked as completed once done.
- Maintain a logical order of execution. Group the tasks logically
into phases.
- Save the result as `docs/tasks.md`, formatted as a numbered list
with checkboxes (for example, - [ ] Implement data repository).
- Do not start implementing any tasks yet—only create the list.
```

Minesweeper

Finally, review the plan:

`docs/tasks.md`

```
# Minesweeper — Task List
## Phase 1 — Core Data Models

- [ ] 1. Create `GameConfig.kt` with `ROWS`, `COLS`, and `MINES`
- [ ] 2. Create `CellState.kt` as a sealed class with `Hidden`,
`Revealed(adjacentMines)`, `Flagged`, and `Exploded` variants
- [ ] 3. Create `Cell.kt` as a data class with `hasMine: Boolean`
and `state: CellState`
- [ ] 4. Create `GameState.kt` as a sealed class with `Idle`,
`Playing`, `Won`, and `Lost` variants
```

Ask Junie to execute a subset of tasks. Test, check results, then repeat.

```
Junie> Proceed with the implementation steps for Phase 1 according
to the tasks listed in `docs/tasks.md`.  Mark the tasks as done
[x] upon completion
```

Bibliography

[1] DAIR.AI, “Prompt Engineering Guide.” [Online]. Available: https://www.promptingguide.ai/

[2] E. R. Molick and L. Molick, “Assigning AI - Seven Approaches for Students, with Prompts,” 2023, *The Wharton School Research Paper*. [Online]. Available: http://dx.doi.org/10.2139/ssrn.4475995

[3] C. S. Smith, “Hallucinations Could Blunt ChatGPT's Success,” *IEEE Spectrum*, pp. 41–52, 2023, [Online]. Available: https://spectrum.ieee.org/ai-hallucination

[4] C. Longo, “Best Practices I Learned for AI Assisted Coding.” [Online]. Available: https://statistician-in-stilettos.medium.com/best-practices-i-learned-for-ai-assisted-coding-70ff7359d403

[5] T. Taulli, *AI-Assisted Programming*. O'Reilly Media Inc., 2024.

[6] JetBrains Inc., “Getting Started with Junie.” [Online]. Available: https://junie.jetbrains.com/docs/get-started-with-junie.html

[7] L. LF Projects, “Agents.md specification.” [Online]. Available: https://agents.md/

[8] Anthropic Inc., “Agent Skills specification.” [Online]. Available: https://agentskills.io/home

[9] Modus Create, “Agentic Coding Handbook.” [Online]. Available: https://tweag.github.io/agentic-coding-handbook/

[10] JetBrains Inc., “How to Use a Spec-Driven Approach for Coding with AI.” [Online]. Available: https://blog.jetbrains.com/junie/2025/10/how-to-use-a-spec-driven-approach-for-coding-with-ai/

[11] “Prompt for 'What is Agentic Coding'?.” [Online]. Available: https://www.anthropic.com/claude/haiku

[12] M. Shumer, “Something Big Is Happening.” [Online]. Available: https://shumer.dev/something-big-is-happening

[13] B. Holland, *10 AI coding tips that actually work*, (2026). [Online Video]. Available: https://www.youtube.com/watch?v=7hAUbsHox20&t=6s