Agentic Coding

CS 346 Application Development

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

Contents

Introduction

What is Generative AI?

“Generative AI refers to a category of artificial intelligence systems designed to generate new content based on patterns learned from training data. These systems can create various types of outputs including text, images, audio, video, code, and more.”

-- Claude Haiku 4.5 (2026) [1]

Key Characteristics

- **Content creation**: They can produce original content that didn’t exist in the training data, rather than just classifying or analyzing existing content.
- **Learning from data**: Generative AI models are trained on large datasets to understand patterns, relationships, and structures.
- **Probabilistic approach**: They work by predicting the most likely next element (word, pixel, etc.) based on what came before.

Generative AI has become increasingly prominent since the release of ChatGPT in 2022. It’s now used extensively for content creation and revision.

What is Generative AI?

Common Types

- **Large Language Models** (LLMs): Generate human-like, conversational text e.g., ChatGPT, Gemini, Deepseek, Copilot.
- **Image generation**: Create images from text descriptions e.g., DALL·E.
- **Audio synthesis**: Generate speech or music e.g., ElevenLabs.
- **Video generation**: Create video content e.g., DaVinci.
- **Code generation**: Write programming code. e.g., Claude, Codex.

How It Works Generative AI typically uses deep learning architectures like:

- **Transformers**: The foundation for most modern language models
- **Diffusion models**: Used for image generation
- **Generative Adversarial Networks** (GANs): Pit two networks against each other to improve output quality

What is Generative AI?

In traditional AI-assisted coding (like pair programming with AI), you're actively involved—typing code while the AI suggests improvements.

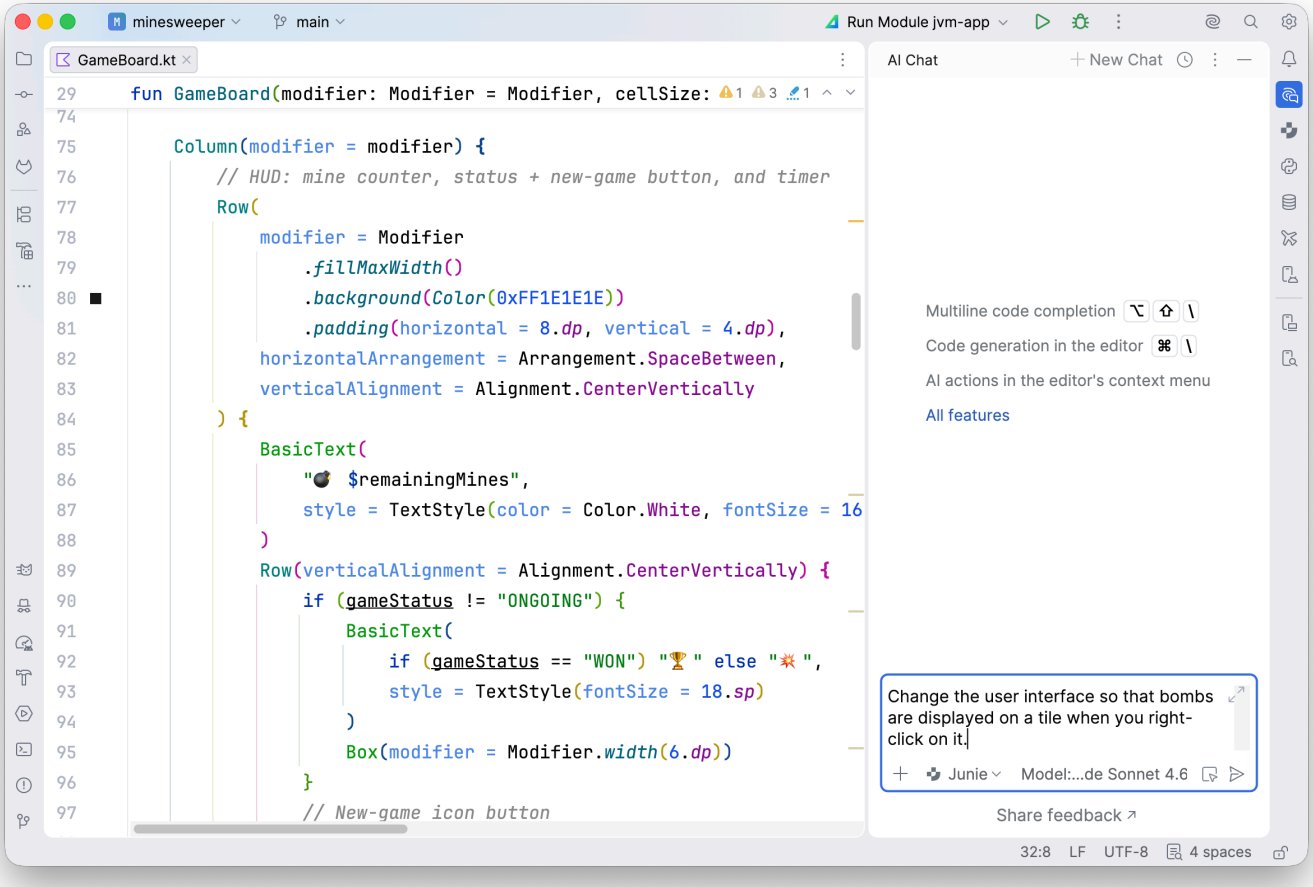

Figure 1: Asking Junie to add a feature. Well-supported in IntelliJ IDEA. [2]

What is 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) [3]

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 failures, and iterate on solutions without waiting for human feedback at each step

What is Agentic Coding?

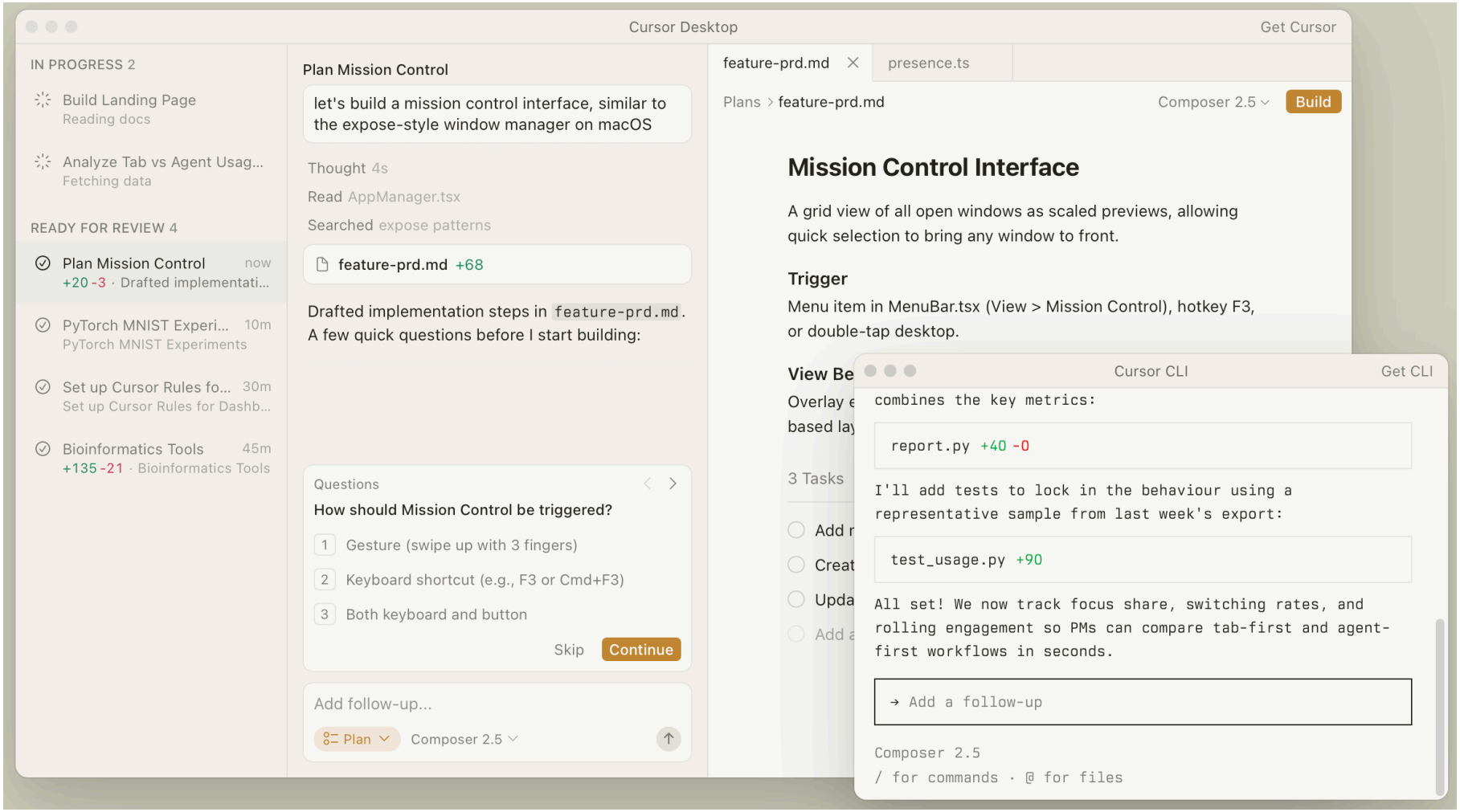

Figure 2: Cursor running agents in the background while the user can edit code, or revise prompts.

What is Agentic Coding?

What is the appeal of agentic-coding?

- **Reducing time required to deliver software**, by optimizing any/all of the required tasks e.g., design, development, testing, deployment.
- **Changing how developers work**. Let agents perform simple but labor intensive tasks to make developers more efficient, and free up their time. Agents become tools for a developer. [4]

What is it being used?

- Most companies are experimenting with AI, or using it in targeted ways e.g., prototyping, fixing bugs, adding features, writing unit tests.
- 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 with 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**
>
> Always limit the access and data you provide a model.

Concerns with 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.” [5]

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 with 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.

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**. It's already having a huge impact in many different fields, including software development. We cannot ignore it, any more than we can ignore other major shifts in how software is developed and delivered.

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. You won't learn anything, and you'll end up with a ball-of-mud instead of a well-designed application.

3. **AI is not a substitute for learning**. AI works best when you can critically examine what it produces and guide it through small refinements. You, the developer, need to know what the end-product should look like; what patterns should be used, how the code should be structured.

4. **AI is just a tool**. You are ultimately responsible for what it produces.

GenAI for Development

Models & Agents

You have many options. The most significant coding models are:

The `Claude` suite e.g., `Claude Opus`, `Sonnet`, 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.

Google produces Gemini, and **Microsoft** produces Copilot. There are also free/local models that can be installed e.g., Ollama.

> **Tip**
>
> Which model should I use? Claude and Codex are top-tier coding models, but can be expensive. Others may be more accessible/affordable.

Coding Environments

Also, there are coding environments that attempt to support richer interaction with these models. These environments support multiple agents running, possibly against multiple remote models.

1. Cursor: The first major coding environment designed around managing agent and agentic coding. NOTE: it handles model interactions just fine but the editor doesn't work well with Kotlin code.
   - Free to use with a limited model. Subscribe to get access to better models.

2. Zed: A competing editor built for collaborative (multi-user) and agentic coding. It handles Kotlin reasonably well.
   - Bring your own model subscription. No model provided.

3. IntelliJ: Our favorite IDE has plugins to integrate models e.g., `Codex`, or `Claude`. It also has Junie, a standalone agent that can be run from the CLI.
   - Subscription, but lets you choose which models to use.

Models & Agents

We'll use Junie for demos, since it integrates nicely with IntelliJ IDEA and Kotlin project. Examples are CLI, but it can be run within IntelliJ as well.

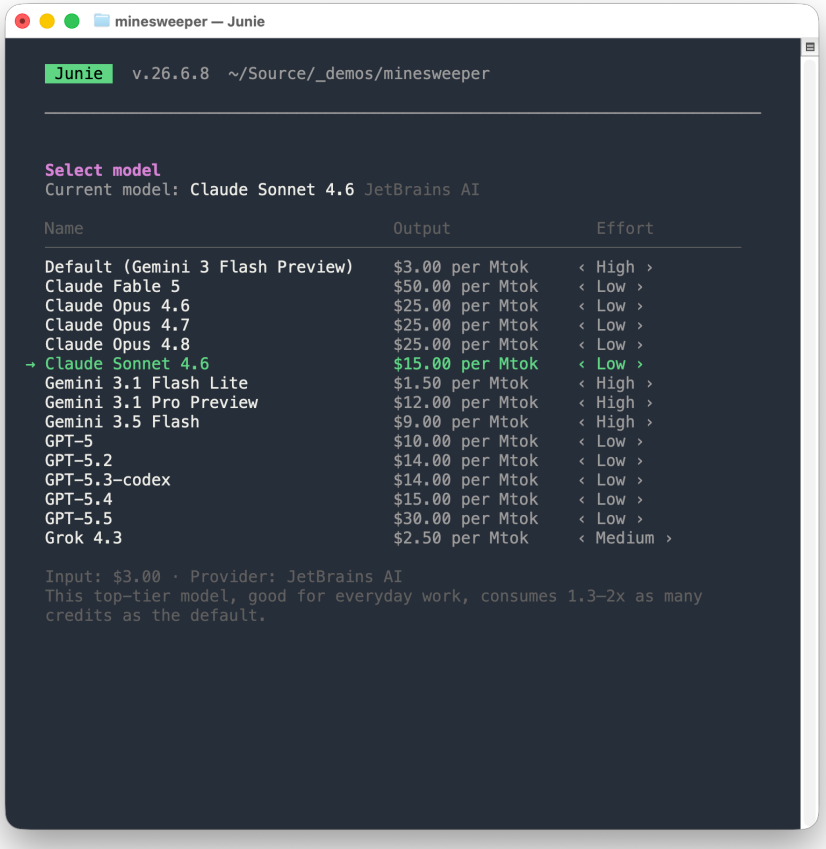

Figure 3: Choosing a model in Junie.

Figure 4: Using Junie to explain source code.

Building Effective Prompts

You interact with an agent by providing it with tasks to perform. The more specific your instructions are, the better the results will be.

Foe example:

- Poor prompt: “Build me a minesweeper game.”
- Better prompt: “Build me a minesweeper game that follows the same rules and design as the Windows minesweeper game.”
- Even better: “Build me a minesweeper game that behaves identically to the Windows 95 minesweeper game. It should follow the same rules as that game, and the layout of the screen should be the same. It should be built using the most recent version of Kotlin and execute on the Java/JVM. You should prefer functional programming style when possible. You are allowed to read and write files in the src/ directory but do not touch the project configuration files without first asking me.”

Building Effective Prompts

What to include in your prompts? [6]

- Details on programming language, supported platform, project configuration.
- Details on how your source files should be structured.
- Programming style e.g., “given the choice, you should prefer to use a functional style over an object-oriented style”.
- 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 and if they tests should be executed before proceeding.

Bibliography

[1] “Prompt for 'What is Generative AI'?.” [Online]. Available: https://www.anthropic.com/claude/haiku

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

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

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

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

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