guide # tooling

AI Coding Tools Setup

TCSS 460 — Client/Server Programming

AI coding assistants can accelerate your learning and productivity in this course. This guide walks you through setting up one (or more) of three tools that are **free for students**. You have already read *Using a Coding Agent* and understand what these tools do conceptually — now it is time to get one running.

---

1 Your Options

This guide covers four AI coding tools — three free and one paid. You do not need to pick just one — many developers use a combination — but we recommend starting with **one** so you can build fluency before adding complexity.

Tool	Access	Where It Runs	Best For	Cost
Google Gemini (CLI + Code Assist)	Personal Google account	Terminal + VS Code	Terminal-first workflows, understanding starter code	Free
GitHub Copilot	GitHub Student Developer Pack	VS Code (built-in)	Inline completions, seamless VS Code integration	Free
Cursor	Student verification	Standalone editor (VS Code fork)	All-in-one AI editor, agent-driven development	Free (1 year)
Claude Code	Claude Pro subscription	Terminal	Deep codebase understanding,	$20/month

Tool	Access	Where It Runs	Best For	Cost
			long-running agent tasks	

All four support **agent mode** — the ability to give a high-level instruction and let the tool plan, edit files, and run commands autonomously. All four understand TypeScript and can work with Express, Prisma, and Next.js projects.

> **Recommendation**
>
> If you have never used an AI coding tool before, **GitHub Copilot** is the easiest starting point because it lives inside VS Code with no extra editor to learn. If you prefer working in the terminal, **Gemini CLI** is a strong free choice. If you want an editor built from the ground up around AI, try **Cursor**. If you want the most capable terminal agent and do not mind paying $20/month, **Claude Code** is the strongest option in that category.

---

2 Google Gemini CLI — Terminal Agent

Gemini CLI is Google's open-source AI agent that runs directly in your terminal. It can read your project files, run shell commands, search the web, and suggest code changes — all from the command line.

2.1 Free Tier

Gemini CLI is free to use with a personal Google account. The free tier provides:

- **60 requests per minute** and **1,000 requests per day**
- Access to Gemini models with a **1M token context window**
- All built-in tools: file operations, shell commands, web search, and Google Search grounding

No credit card is required. You authenticate with your Google account via OAuth.

2.2 Installation

Gemini CLI requires **Node.js** (which you already have for this course). Choose one of these installation methods:

Option A — Run without installing (npx):

```
npx @google/gemini-cli
```

This downloads and runs the latest version each time. Good for trying it out.

Option B — Install globally via npm:

```
npm install -g @google/gemini-cli
```

After installation, run the tool with:

```
gemini
```

Option C — Homebrew (macOS):

```
brew install gemini-cli
```

2.3 First Run and Authentication

The first time you run `gemini`, it will prompt you to authenticate with your Google account:

1. Run `gemini` in your terminal.
2. A browser window opens asking you to sign in with your Google account.
3. Grant the requested permissions.
4. Return to the terminal — you are now authenticated.

Once signed in, you can start giving natural language instructions:

```
> Explain the folder structure of this project
> Add a new GET route to src/routes/hello.ts that returns { message: "Hello,
world!" }
> Run the test suite and fix any failures
```

Try It Yourself

1. Open your terminal and navigate to your TCSS 460 project directory
2. Run `npx @google/gemini-cli`
3. Authenticate with your Google account
4. Type: `Explain what this project does based on the file structure`
5. Review the output — it should describe your Express project accurately

---

3 Gemini Code Assist — VS Code Extension

While Gemini CLI runs in the terminal, **Gemini Code Assist** brings Google's AI directly into VS Code as an extension. It provides code completions, chat, and agent mode inside the editor.

3.1 What You Get (Free)

Gemini Code Assist for individuals is free and includes:

- **Code completions** as you type (inline suggestions)
- **Chat-based assistance** for explaining code, generating tests, and debugging
- **Multi-file editing** from natural language instructions
- **Agent mode** for autonomous multi-step tasks

The free tier has daily request limits. You can increase these limits with a Google AI Pro or Ultra subscription, but the free tier is more than enough for coursework.

3.2 Installation

1. Open VS Code.
2. Open the Extensions view (`Cmd+Shift+X` on macOS, `Ctrl+Shift+X` on Windows/Linux).
3. Search for "**Gemini Code Assist**" (publisher: Google).
4. Click **Install**.
5. After installation, click the **Gemini Code Assist** icon in the activity bar (left sidebar).
6. Click **Login to Google** in the chat window.
7. A browser window opens — sign in with your Google account.

8. When asked if you downloaded Gemini Code Assist from Google, click **Sign In**.

You are now ready to use Gemini Code Assist in VS Code.

3.3 Using Agent Mode

Agent mode in Gemini Code Assist goes beyond simple completions. It can:

- Plan multi-step changes across multiple files
- Execute shell commands in your terminal
- Review its own output and iterate until the task is complete

To use agent mode, open the Gemini Code Assist chat panel and describe what you want:

```
Add input validation middleware to the /auth/register route that checks
for a valid email format and a password with at least 8 characters
```

The agent will identify which files to modify, make the changes, and show you diffs to review before applying.

---

4 Gemini CLI Companion — Bridging Terminal and Editor

If you use both Gemini CLI (terminal) and VS Code, the **Gemini CLI Companion** extension bridges them. When Gemini CLI detects that it is running inside a VS Code integrated terminal, it can:

- **Read your open files and cursor position** for better context
- **Show diffs in VS Code's native diff viewer** instead of the terminal
- **Access workspace context** so suggestions are more targeted

4.1 Setup

The companion extension is typically installed automatically. When you run `gemini` inside VS Code's integrated terminal for the first time, Gemini CLI detects your environment and asks if you want to connect. Answering **Yes** installs the companion extension and enables the integration.

You can also install it manually:

1. Open the Extensions view in VS Code.

2. Search for "**Gemini CLI Companion**" (publisher: Google).
3. Click **Install**.

Once connected, you get the best of both worlds: the power of a terminal agent with the visual feedback of your IDE.

4.2 VS Code Commands

With the companion installed, you can access Gemini CLI from the Command Palette (`Cmd+Shift+P`):

- **Gemini CLI: Run** — Start a Gemini CLI session
- **Gemini CLI: Accept Diff** — Accept a proposed code change
- **Gemini CLI: Close Diff Editor** — Dismiss a proposed change

> **Gemini Ecosystem Summary**
>
> Google offers two complementary tools: **Gemini CLI** (terminal agent) and **Gemini Code Assist** (VS Code extension). They share the same underlying models and free tier. Use the CLI for terminal workflows and Code Assist for in-editor assistance. The **Companion** extension connects them when you run the CLI inside VS Code.

---

5 Gemini Tips for TCSS 460

When using Gemini (CLI or Code Assist) with your course projects:

- **Point at your project root.** Run `gemini` from the top-level directory of your Express project (where `package.json` lives). This gives the agent the best context for understanding your code.
- **Use it to understand starter code.** When you receive a new starter repo, ask Gemini to explain the project structure, describe what each route does, or walk through the middleware chain.
- **Ask about unfamiliar patterns.** If you encounter a TypeScript pattern you have not seen before (generics, mapped types, decorators), ask for an explanation with a Java comparison.

> **Don't Skip Understanding**
>
> AI tools are excellent for productivity, but this is a learning course. If Gemini writes code for you, make sure you understand every line before submitting. Check-off assignments require you to **explain your code** to an instructor — "the AI wrote it" is not a valid explanation.

---

6 What Is GitHub Copilot?

GitHub Copilot is an AI coding assistant built into VS Code (and other editors). It provides inline code completions, a chat interface, and a full agent mode — all without leaving your editor. For students with the GitHub Student Developer Pack, Copilot is free.

6.1 What Students Get

The **GitHub Copilot Student plan** (updated March 2026) includes:

- **Unlimited code completions** — inline suggestions as you type
- **Chat with Copilot** — ask questions, explain code, generate tests
- **Agent mode** — autonomous multi-file editing, terminal commands, error recovery
- **Next edit suggestions** — predicts where you will edit next and proposes the change
- **300 premium requests per month** — for agent mode, code review, and premium model access

> **March 2026 Changes**
>
> Starting March 12, 2026, GitHub restructured student access under a new Copilot Student plan. Students can no longer manually select premium models (such as GPT-5.4 or Claude Opus/Sonnet). Instead, the **Auto mode** intelligently selects from available models from OpenAI, Anthropic, and Google. All core features remain free — the change only affects manual model selection.

6.2 Premium Requests

Your 300 monthly premium requests are consumed by:

- Each **agent mode** message (1 request per message, multiplied by model cost)

- **Coding agent** sessions (1 request per session regardless of internal steps)
- **Code review** requests

The counter resets on the 1st of each month at 00:00 UTC. For typical coursework, 300 requests is sufficient. If you run out, you can still use basic completions — only premium features pause.

---

7 Getting GitHub Copilot — Student Access

GitHub Copilot is free through the **GitHub Student Developer Pack**. Here is how to apply:

7.1 Apply for the Student Developer Pack

1. Go to education.github.com/pack.
2. Click **Get your pack** (or **Join**).
3. Sign in with your GitHub account (create one if you do not have one).
4. Select **Student** as your academic status.
5. Verify your academic affiliation using **one** of these methods:
   - **School email** — Use your `@uw.edu` email address. This is the fastest method.
   - **Student ID** — Upload a photo of your UW student ID.
   - **Enrollment document** — Upload a transcript or enrollment verification letter.
6. Submit the application and wait for approval.

> ⚠ **Approval Timeline**
>
> Verification typically takes **a few hours to a few days** but can take up to 30 days in some cases. **Apply early** — do not wait until you need Copilot for an assignment. Using your `@uw.edu` email speeds up the process significantly.

7.2 Verify Copilot Access

After your Student Developer Pack is approved:

1. Go to github.com/settings/copilot.
2. You should see that you are enrolled in the **GitHub Copilot Student** plan.

3. No payment information is required.

---

8 Setting Up Copilot in VS Code

8.1 Install the Extension

1. Open VS Code (version 1.75 or later required).
2. Open the Extensions view (`Cmd+Shift+X` on macOS, `Ctrl+Shift+X` on Windows/Linux).
3. Search for **"GitHub Copilot"** (publisher: GitHub).
4. Click **Install**. This installs both the Copilot extension and the Copilot Chat extension.
5. You may be prompted to reload VS Code — do so.

8.2 Sign In

1. After installation, VS Code prompts you to sign in to GitHub. Click **Sign In**.
2. A browser window opens. Log in with your GitHub credentials (the account with the Student Developer Pack).
3. Authorize the VS Code extension when prompted.
4. Return to VS Code — you should see the Copilot icon in the status bar (bottom right).

8.3 Verify It Works

Open any TypeScript file and start typing. You should see gray "ghost text" suggestions appear. Press `Tab` to accept a suggestion or `Esc` to dismiss it.

To open Copilot Chat, click the Copilot icon in the activity bar or press `Cmd+Shift+I` (macOS) / `Ctrl+Shift+I` (Windows/Linux).

Try It Yourself

1. Open your TCSS 460 project in VS Code
2. Create a new file `test-copilot.ts`
3. Type `// Function that reverses a string` and press Enter
4. Watch Copilot suggest the implementation
5. Press `Tab` to accept, then delete the file

---

9 Three Modes of GitHub Copilot

Copilot offers three distinct modes, each suited to different tasks.

9.1 Ask Mode

Ask mode is the simplest. You type a question in the Copilot Chat panel, and Copilot answers it — without touching your code. Use ask mode to:

- **Explain** what a piece of code does
- **Understand** an error message or stack trace
- **Get a code snippet** for a specific task
- **Learn** about a TypeScript feature or Express pattern

Example prompt:

```
What does app.use(express.json()) do and why do I need it?
```

9.2 Edit Mode

Edit mode lets you specify which files Copilot can modify. You provide instructions, Copilot proposes changes, and you review and accept or reject each edit. This gives you granular control.

Use edit mode when you want to:

- Refactor a specific function
- Add error handling to an existing route

- Update types across a few related files

9.3 Agent Mode

Agent mode is the most powerful. You give a high-level instruction, and Copilot autonomously:

1. Plans the steps needed to complete the task
2. Identifies and opens the relevant files
3. Makes edits across multiple files
4. Runs terminal commands (with your permission)
5. Detects errors and iterates until the task is complete

To start agent mode, open Copilot Chat and select **Agent** from the mode dropdown (or type `@workspace` followed by your instruction).

Example prompt:

```
Add a new POST /api/books route that accepts { title, author, isbn } in
the request body, validates all fields are present, and returns the
created book with a 201 status code
```

---

10 Using Copilot Agent Mode Effectively

Agent mode is powerful but works best when you guide it well.

10.1 Provide Context

- **Open relevant files** before starting. Agent mode uses open files as context.
- **Be specific** about file paths and function names when possible.
- **Describe the end state**, not just the first step.

10.2 Review Every Change

Agent mode shows you a diff for every file it modifies. **Always review the diff** before accepting:

- Does the code match your project's patterns?

- Are the TypeScript types correct?
- Did it modify files you did not intend?

10.3 Iterate

If the first result is not right, tell Copilot what to fix rather than starting over. Agent mode maintains conversation context, so saying "use an interface instead of a type alias" or "add error handling for the database call" will refine the existing work.

> ⚠ **Check-Off Assignments**
>
> During check-off assignments, you must be able to explain every line of code. Use agent mode to **learn patterns** and **save time**, not to bypass understanding. An instructor asking "why did you use `async` here?" expects a real answer.

---

11 What Is Cursor?

Cursor is an AI-first code editor built as a fork of VS Code. It looks and feels like VS Code (your extensions and settings carry over), but it has AI capabilities deeply integrated into every interaction. Cursor is not just an extension — it is a reimagined editor where AI is a first-class feature.

11.1 Three Ways to Interact

Interaction	Shortcut	What It Does
Tab	`Tab`	Inline completions as you type — accepts AI suggestions keystroke by keystroke
Cmd+K	`Cmd+K`	Inline edit — select code, describe what to change, Cursor rewrites it
Agent	Chat panel	Full agent mode — plans, edits files, runs commands, fixes errors

11.2 Key Differences from VS Code + Copilot

- **Tab completions** in Cursor are multi-line and context-aware across your entire project, not just the current file.
- **Cmd+K** lets you select a block of code and describe a change in plain English. Cursor rewrites just that block.
- **Agent mode** can create files, run terminal commands, and iterate on errors automatically.
- **Auto mode** selects the best model for each request, and Auto mode requests are unlimited on paid plans.

---

12 Getting Cursor — Student Access

Verified university students get **one free year of Cursor Pro** (normally $20/month).

12.1 Apply for Student Access

1. Go to cursor.com/students.
2. Sign up for a Cursor account if you do not have one.
3. Verify your student status (typically with a `.edu` email address).
4. Your account is upgraded to **Cursor Pro** for one year.

12.2 What You Get

The student plan gives you full **Cursor Pro**, which includes:

- **Unlimited Tab completions** — no cap on inline autocomplete
- **Extended Agent requests** — significantly more than the free Hobby plan
- **$20 monthly usage credits** — for premium model requests (frontier models like GPT-4.1, Claude Sonnet)
- **Unlimited Auto mode** — Cursor selects the best model automatically, and these requests do not consume credits
- **Background Agents** — for long-running tasks
- **MCP (Model Context Protocol) support** — for custom tool integrations

> ⚠ **After the Free Year**
>
> When your student year expires, Cursor reverts to the **Hobby plan** (limited completions and agent requests) unless you subscribe to Pro at $20/month. Be aware of this timeline so you are not surprised by reduced functionality mid-quarter.

12.3 Hobby Plan (Fallback)

If you do not verify student status or after your free year ends, the Hobby plan provides:

- Limited Tab completions (roughly 2,000 per month)
- Limited Agent requests (roughly 50 slow premium requests)
- No credit card required

The Hobby plan is functional for light use but most active developers find they exhaust the quotas within a week or two of regular coding.

---

13 Setting Up Cursor

13.1 Download and Install

1. Go to cursor.com and click **Download**.
2. Install the application for your operating system (macOS, Windows, or Linux).
3. Open Cursor.

13.2 Import VS Code Settings

Because Cursor is a VS Code fork, it can import your existing setup:

1. On first launch, Cursor offers to import your VS Code extensions, themes, and keybindings.
2. Accept the import — your familiar environment carries over.
3. Your existing VS Code extensions (ESLint, Prettier, etc.) will work in Cursor.

Use Both Editors

You do not have to choose between VS Code and Cursor permanently. Many developers use Cursor for AI-heavy work and VS Code for tasks where they want a leaner editor. Both can open the same project directory.

13.3 Sign In

1. Click the gear icon (Settings) in the bottom left.
2. Navigate to your account settings.
3. Sign in with the account you used for student verification.
4. Verify that your plan shows **Pro** (student).

---

14 Using Cursor's Agent Mode

Cursor's Agent mode is its most powerful feature. Open the chat panel (sidebar) and describe what you want.

14.1 Agent Mode

Agent mode handles complex, multi-step tasks:

```
Create a new middleware function in src/middleware/validate.ts that
checks if the request body contains a valid email field. If invalid,
return a 400 error. Then apply it to the POST /auth/register route.
```

The agent will:

1. Create the middleware file
2. Write the validation logic
3. Find your route file
4. Import and apply the middleware
5. Run any relevant tests if configured

14.2 Tab Completions

Tab completions work passively as you code. Unlike simple autocomplete, Cursor's Tab completions:

- Predict multi-line completions based on your project context
- Suggest entire function bodies, not just the current line
- Learn from patterns in your codebase

Press `Tab` to accept, `Esc` to dismiss.

14.3 Inline Edit (Cmd+K)

For targeted changes:

1. Select the code you want to change.
2. Press `Cmd+K` (macOS) or `Ctrl+K` (Windows/Linux).
3. Describe the change: "Add TypeScript types to this function" or "Convert this callback to async/await."
4. Review the proposed edit and press `Enter` to accept.

Cmd+K is scoped to the selected code block — it will not modify other parts of your file.

14.4 Plan Mode

For larger tasks, Cursor offers a **Plan mode** (select "plan" in the chat panel). Instead of jumping straight to code changes, Cursor:

1. Crawls your project structure
2. Reads relevant files and documentation
3. Generates an editable Markdown plan with file paths and to-do items
4. Waits for your approval before executing

This is especially useful at the start of a sprint when you need to think through architecture before coding.

---

15 Claude Code — Terminal Agent (Paid)

Claude Code is Anthropic's AI coding agent that runs directly in your terminal. Like Gemini CLI, it operates outside of any specific editor — you run it in your project directory and

interact with it through natural language. Unlike the free options above, Claude Code requires a paid subscription, but it is the most capable terminal-based coding agent available as of early 2026.

15.1 What You Get

Claude Code is included with a **Claude Pro** subscription ($20/month). The subscription gives you:

- **Claude Code CLI** — the terminal agent
- **Access to Claude's latest models** including Opus 4.6 and Sonnet 4.6
- **1M token context window** — available with Opus 4.6. No per-token surcharge for the larger window — standard rates apply whether you use 9K or 900K tokens.
- **All built-in tools** — file read/write, terminal commands, web search, Git operations
- **IDE integrations** — VS Code and JetBrains extensions that connect to the CLI agent
- **Auto-updates** — the native installer updates in the background

There is no free tier for Claude Code. The $20/month Claude Pro subscription is required.

> ⚠ **Pro Tier Usage Limits**
>
> The Pro plan includes Claude Code access, but with **rolling usage limits**. Anthropic does not publish exact numbers, but the practical reality for Pro users is roughly **10-40 prompts per 5-hour rolling window**, depending on model and codebase size. There are also weekly caps on top of the 5-hour windows.
>
> All Claude products (claude.ai, Claude Code, Claude Desktop) share the same usage bucket — heavy use of the chat interface eats into your Claude Code allocation and vice versa.
>
> For light to moderate use — asking questions, generating individual route handlers, debugging errors — the Pro allocation is usually sufficient. For heavy agent sessions — long multi-file refactors or extended coding sprints — you may hit the limit. When you do, you can wait for the window to reset or enable **extra usage**, which bills at standard API rates on top of the $20 subscription.
>
> Higher tiers provide significantly more capacity: Max 5x ($100/month) provides roughly 50-200 prompts per window, and Max 20x ($200/month) provides 200-800. These plans also unlock Opus model access with separate weekly hour caps. For most students, Pro is enough if you use the tool intentionally — batch your requests, scope your prompts tightly, and avoid leaving long idle sessions open.

15.2 Why Consider a Paid Tool?

The three free tools covered earlier are genuinely good and sufficient for this course. You do not *need* Claude Code. But if you are considering it, here is what sets it apart:

- **Deep project understanding.** Claude Code reads your entire project structure, configuration files, and documentation at the start of every session. It builds a rich mental model of how your codebase fits together before it writes a single line.
- **Long-running agent tasks.** Claude Code excels at multi-step tasks that require planning, executing, testing, and iterating — the kind of work where you describe a feature and the agent implements it across multiple files.
- **Context engineering support.** Claude Code reads project-level documentation files and memory systems that persist across sessions. If you invest in describing your project's conventions, architecture, and intent, the agent applies that knowledge to every task. (See the *Using a Coding Agent* guide, Section 5 for more on context engineering.)
- **Terminal-native workflow.** If you prefer working in the terminal rather than inside an editor, Claude Code is the most polished experience in that category.

15.3 Installation

Claude Code uses a **native installer** — no Node.js or npm required.

macOS / Linux:

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows (PowerShell):

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
irm https://claude.ai/install.ps1 | iex
```

After installation, run:

```
claude
```

The native installer auto-updates in the background, so you always have the latest version without manual intervention.

Migrating from npm

If you previously installed Claude Code via npm (`npm install -g @anthropic-ai/claude-code`), that method is deprecated. Install the native binary using the commands above, then remove the old installation: `npm uninstall -g @anthropic-ai/claude-code`.

15.4 Authentication

The first time you run `claude`, it will prompt you to authenticate:

1. Run `claude` in your terminal.
2. A browser window opens asking you to sign in with your Anthropic account.
3. If you do not have an account, create one at claude.ai and subscribe to **Claude Pro** ($20/month).
4. Grant the requested permissions.
5. Return to the terminal — you are now authenticated.

15.5 Basic Usage

Navigate to your project directory and run `claude`:

```
cd ~/my-express-project
claude
```

Claude Code reads your project and you can start giving instructions:

```
> Explain the folder structure of this project
> Add a new GET route for /users/:id that returns the user as JSON
> Run the tests and fix any failures
```

Claude Code will read files, propose edits, run terminal commands, and iterate on errors — all within the terminal session.

Setting Up Project Context with `/init`

The first thing to do in a new project is run the `/init` command inside a Claude Code session:

```
> /init
```

Claude scans your project — reading `package.json`, configuration files, existing code, and documentation — and generates a `CLAUDE.md` file tailored to your project. This file captures build commands, test instructions, key directories, coding conventions, and project-specific details.

Claude reads `CLAUDE.md` at the start of every conversation, so everything in it becomes persistent context. This is context engineering in practice (see the *Using a Coding Agent* guide, Section 6) — you invest once in describing your project, and every future session benefits.

Think of `/init` as a starting point, not a finished product. The generated `CLAUDE.md` captures obvious patterns but may miss nuances specific to your workflow. Review what Claude produces and refine it over time. You can also run `/init` again later — Claude will review the existing file and suggest improvements based on what it learns from exploring your codebase.

Try It Yourself

1. Navigate to your TCSS 460 project directory
2. Run `claude`
3. Type `/init` to generate a `CLAUDE.md` for your project
4. Review the generated file — does it accurately describe your project's structure, build commands, and conventions?
5. Edit the file to add anything Claude missed (e.g., "use `request`/`response` instead of `req`/`res`" or "routes follow the pattern in `src/routes/v1/`")
6. In your next Claude Code session, notice how the agent already knows your project conventions without being told

15.6 IDE Integration

While Claude Code is terminal-first, it also offers extensions for **VS Code** and **JetBrains IDEs** (WebStorm, IntelliJ). These extensions connect to the Claude Code agent and provide an in-editor chat panel, inline diff review, and the ability to accept or reject proposed changes without switching to the terminal.

Install from the VS Code marketplace by searching for "Claude Code" or from JetBrains Marketplace for WebStorm.

---

16 Which Should I Choose?

All four tools are capable. The best choice depends on your workflow preferences and budget.

16.1 Feature Comparison

Feature	Gemini (CLI + Code Assist)	GitHub Copilot	Cursor	Claude Code
Cost for students	Free (Google account)	Free (Student Developer Pack)	Free for 1 year (student verification)	$20/month (Claude Pro)
Runs in	Terminal + VS Code extension	VS Code (built-in)	Standalone editor (VS Code fork)	Terminal + VS Code/JetBrains extensions
Inline completions	Yes (Code Assist)	Yes	Yes (multi-line, project-aware)	No (agent-based, not inline)
Chat	Yes (Code Assist)	Yes (Copilot Chat)	Yes	Yes (terminal + IDE panel)
Agent mode	Yes (both CLI and Code Assist)	Yes	Yes	Yes (primary interaction model)
Terminal agent	Yes (Gemini CLI)	Limited (Copilot CLI)	No (agent runs commands in integrated terminal)	Yes (terminal-first)

Feature	Gemini (CLI + Code Assist)	GitHub Copilot	Cursor	Claude Code
Model selection	Gemini models	Auto mode (multiple providers)	Auto mode + manual selection with credits	Claude Opus, Sonnet
Monthly limits	1,000 requests/day	300 premium requests/month	Unlimited Auto; $20 credits for frontier models	~10-40 prompts per 5-hour window (Pro); shared with claude.ai
Context window	1M tokens	Varies by model	Varies by model	1M tokens (Opus 4.6, standard pricing)
VS Code extensions	Works alongside	Works alongside	Imports VS Code extensions	Works alongside
Verification	Google account	GitHub Education application	Student email verification	Credit card (paid subscription)
Approval time	Instant	Hours to days	Varies	Instant

16.2 Decision Guide

Choose Gemini CLI + Code Assist if you...

- Prefer working in the terminal
- Want the highest free-tier request limits (1,000/day)
- Already use Google services
- Want an open-source tool you can inspect and extend

Choose GitHub Copilot if you...

- Want the simplest setup (one extension in VS Code)
- Already have a GitHub account (you will need one for this course anyway)
- Prefer inline completions that feel like a natural part of typing
- Want the "industry standard" tool that most companies use

Choose Cursor if you...

- Want AI deeply integrated into every part of the editor
- Prefer multi-line, project-aware Tab completions
- Like the Cmd+K inline edit workflow
- Want unlimited Auto mode requests on a Pro plan

Choose Claude Code if you...

- Are willing to pay $20/month for the most capable terminal agent
- Want deep project understanding and strong multi-file agent tasks
- Value context engineering support (persistent project knowledge across sessions)
- Prefer a terminal-first workflow with optional IDE integration

> **Claude Code Is Not Required**
>
> The three free tools are more than sufficient for every assignment in this course. Claude Code is listed here because it is an excellent tool and some students may already have a Claude Pro subscription or want to invest in one. Do not feel pressured to pay for a tool — the free options will serve you well.

16.3 Can I Use More Than One?

Yes. Common combinations:

- **Gemini CLI + GitHub Copilot** — Use the CLI for terminal tasks (understanding projects, running commands) and Copilot for inline completions while coding in VS Code.
- **Claude Code + GitHub Copilot** — Use Claude Code for agent tasks (feature implementation, debugging, multi-file changes) and Copilot for inline completions while typing.
- **Cursor as primary editor** — Cursor replaces VS Code entirely, so Copilot and Cursor extensions do not coexist. But you can use Gemini CLI or Claude Code in Cursor's integrated terminal.

Start Simple

Pick one tool, use it for a full week, and then decide if you want to add another. Switching between tools constantly is less productive than getting proficient with one.

---

17 Summary

Concept	Key Point
Three free options	Gemini (Google account), Copilot (Student Developer Pack), Cursor (student verification)
One paid option	Claude Code ($20/month Claude Pro subscription)
Gemini CLI	Terminal-based agent; 1,000 free requests/day; open-source
Gemini Code Assist	VS Code extension; completions, chat, agent mode; free for individuals
Gemini CLI Companion	Bridges CLI and VS Code for workspace-aware suggestions
GitHub Copilot	VS Code extension; ask, edit, and agent modes; 300 premium requests/month for students
Copilot student access	Apply at education.github.com with your `@uw.edu` email
Cursor	AI-first editor (VS Code fork); Tab, Cmd+K, and Agent interactions
Cursor student access	One free year of Pro at cursor.com/students
Claude Code	Terminal-first agent; deep project understanding; strong multi-file tasks; VS Code + JetBrains extensions
All tools	Support agent mode for autonomous, multi-file coding tasks

Concept	Key Point
Course policy	Use AI tools to learn, not to bypass understanding — you must explain your code

---

18 References

Official Documentation:

- Gemini CLI — GitHub Repository — Source code, installation, and documentation
- Gemini CLI — Quotas and Pricing — Free tier details
- Gemini Code Assist — Setup for Individuals — VS Code extension setup
- Gemini Code Assist — VS Code Marketplace — Extension listing
- Gemini CLI Companion — VS Code Marketplace — Companion extension
- GitHub Copilot — Plans and Pricing — Plan comparison
- GitHub Copilot — Features — Feature overview
- GitHub Copilot — Setup in VS Code — Installation guide
- GitHub Student Developer Pack — Student benefits application
- GitHub Copilot — Student Plan Updates (March 2026) — Recent changes
- Cursor — Pricing — Plan details and comparison
- Cursor — Students — Student access application
- Cursor — Agent Overview — Agent mode documentation
- Claude Code — Documentation — Official Claude Code documentation
- Claude Pro — Pricing — Subscription plans and pricing

---

19 Further Reading

External Resources

- Copilot Ask, Edit, and Agent Modes (GitHub Blog) — Detailed walkthrough of Copilot's three modes
- Gemini CLI + VS Code: Native Diffing and Context-Aware Workflows (Google Developers Blog) — How the CLI Companion integrates with VS Code
- Gemini Code Assist Agent Mode Guide (Gemini Lab) — Deep dive into Code Assist agent mode
- GitHub Copilot Agent Mode 101 (GitHub Blog) — Comprehensive agent mode tutorial
- Apply to GitHub Education as a Student (GitHub Docs) — Step-by-step application guide

---

This guide is part of TCSS 460 — Client/Server Programming, School of Engineering and Technology, University of Washington Tacoma.