The Transition Accelerator | L'Accélérateur de transition

AI Adoption Training

Workshop#5: Advanced AI Topics – Coding, Models and Agents

December 12th, 2025

Today’s Agenda

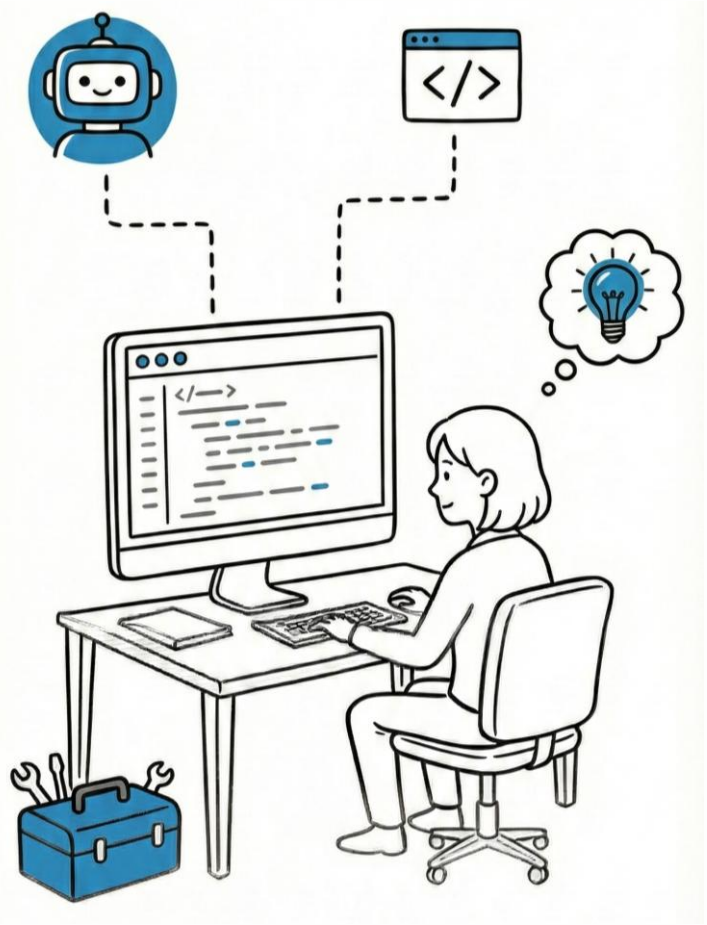
1. **Intro & Context** (5 mins)
2. **Understanding LLM Coding & Agents** (20 mins)
3. **LLM Coding Techniques** (30 mins)
4. **Agents Beyond Coding** (10 minutes)
5. **Learnings and Additional Resources** (5 mins)
6. **Wrap Up** (5 mins)

Materials are available at: https://github.com/mullinsean/ai_training

Goals for Today's Session

Exposure to AI coding tools, not mastery

- Goal: Understand what is possible with advanced applications of LLM tools
    - Admittedly, more complex than previous four sessions
    - For those that have some coding knowledge: provide a roadmap for how to use LLM tools to dramatically enhance your productivity
    - For those who do not have prior coding knowledge: provide an overview of what is possible and how to edge into this space
    - Provide resources for further learning



The LLM coding revolution

No industry is changing faster due to LLMs than software engineering

- In 2024, 25% of all code at Google was generated by LLM
- Cursor (AI coding tool) has grown from $0 to >$1B in annual revenue since 2023
- Anthropic has disclosed that more than half of their revenue (estimated at $10B for 2025) is coming from coding and software engineering tasks
    - Claude Code (launched in Feb 2025) is already generating more than $1B in annual revenue
- Early evidence that demand for *junior* software engineers is declining
- Performance on software benchmarks are becoming key factors in evaluating new frontier models (eg: Claude Sonnet 4.5, Gemini 3.0, ChatGPT 5.1)

"The future is already here – it's just not evenly distributed."

– William Gibson

Code + Tools: The Language of Computers

LLMs as the bridge between intention and execution

- Problem: Often, prompts written in English are not specific enough for LLMs to implement sophisticated, detailed tasks (eg: build an economic model of X)
    - How do you know what the LLM is doing “under the hood” of a task? Important for verification and reproducibility
    - For repeated problems, we may not want to reinvent the solution each time
- LLMs writing code can solve for these problems:
    - Code is the “language” of how we specify to computers how to execute tasks
    - LLMs are language experts → this especially includes software code!
    - Code be reviewed, edited, shared and reused; it also allows you to work on problems step-by-step
- LLMs also write code for themselves to solve problems and enhance tool use (eg: how Claude creates and manipulates Excel files)

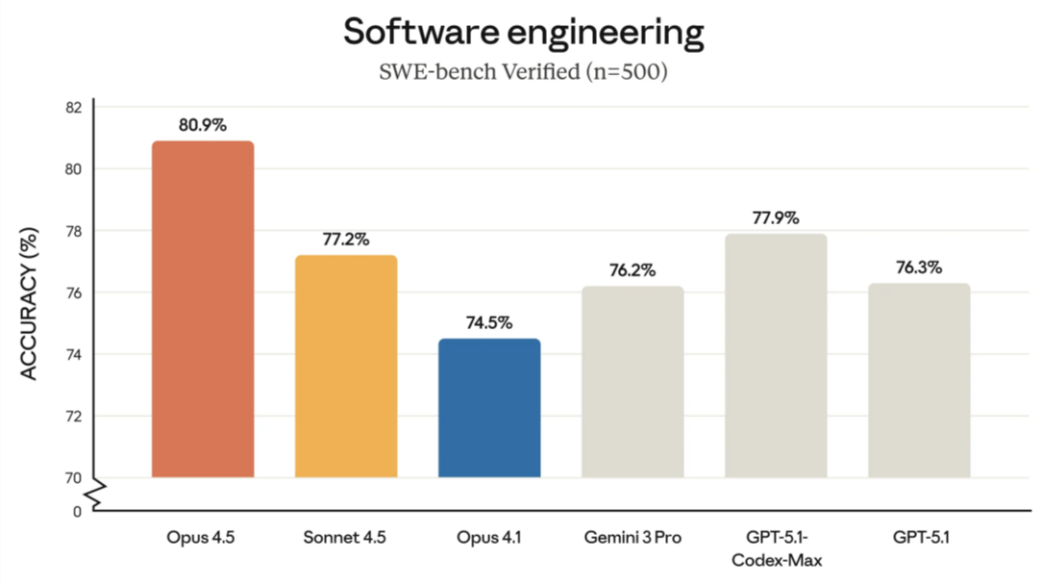
G7 gdp chart · JSX

```jsx
import React, { useState, useEffect } from 'react';
import{LineChart,Line,XAxis,YAxis,CartesianGrid,Tooltip,Legend

const G7GDPChart = () => {
  const [chartData, setChartData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // G7 country codes, names, and colors
  const countries = {
    'USA': { name: 'United States', color: '#1f77b4' },
    'CAN': { name: 'Canada', color: '#ff7f0e' },
    'GBR': { name: 'United Kingdom', color: '#2ca02c' },
    'DEU': { name: 'Germany', color: '#d62728' },
    'FRA': { name: 'France', color: '#9467bd' },
    'ITA': { name: 'Italy', color: '#8c564b' },
    'JPN': { name: 'Japan', color: '#e377c2' }
  };

  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true);

```

The Race to Build the Best Coders…

Frontier models are being increasingly optimized for coding

- All major AI companies are now training and optimizing frontier models for coding:
  - OpenAI: GPT5.2 and GPT5.2-Codex (agentic coding)
  - Anthropic: Claude Sonnet 4.5 and Claude Opus 4.5 (complex problems)
  - Google: Gemini Pro 3.0
- Why it works so well:
  - Vast amount of publicly available code for training data; verifiable benchmarks for performance
  - Code is more structured, less ambiguous than regular language (by design)
  - AI labs use these models themselves → huge incentive to improve performance
- Emergence of specialized tools: Claude Code, Codex, Github Copilot, Cursor, Windsurf



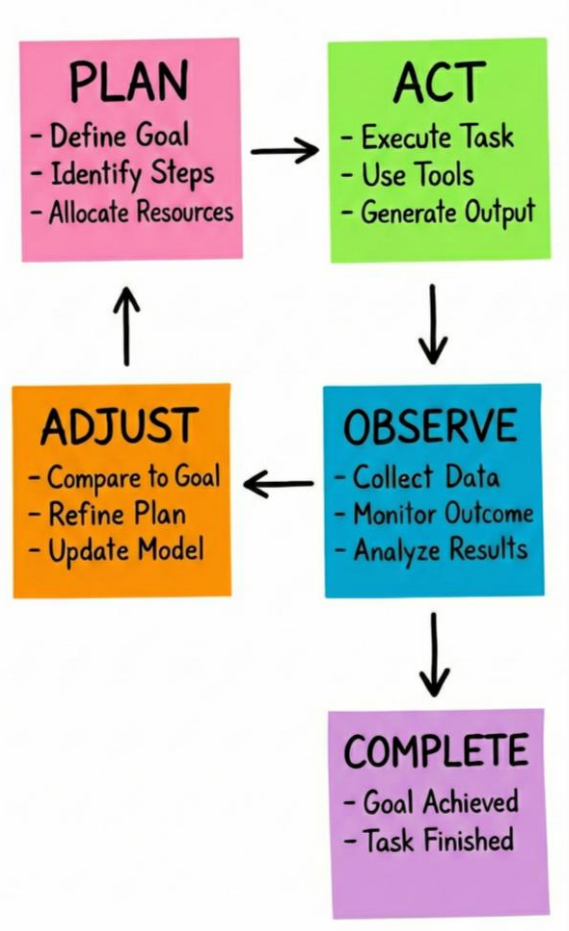
(Source: Anthropic)

What Are AI Agents?

"LLM + tools in a loop"

- Tremendous source of hype right now: "2025 is the year of AI agents"
  - More realistically, we are looking at the "decade of agents" (Karpathy)
- AI Agent = **LLM + tools + ability to act autonomously to achieve a goal**
  - Agents can plan multi-step tasks, use tools, delegate to "sub-agents", decide when task is finished
  - Deep Research feature is an example of an agent: submit a research request, tool goes off and conducts research, notifies you when complete
- Conceptually:
  - LLMs are *stateless*; the appearance of an interactive chat session is an illusion; in practice, they are series of API calls
  - The output of one LLM call can generate input for the next LLM call; allows for branching, monitoring outcomes and deciding when goal has been achieved.

Agentic AI Loop

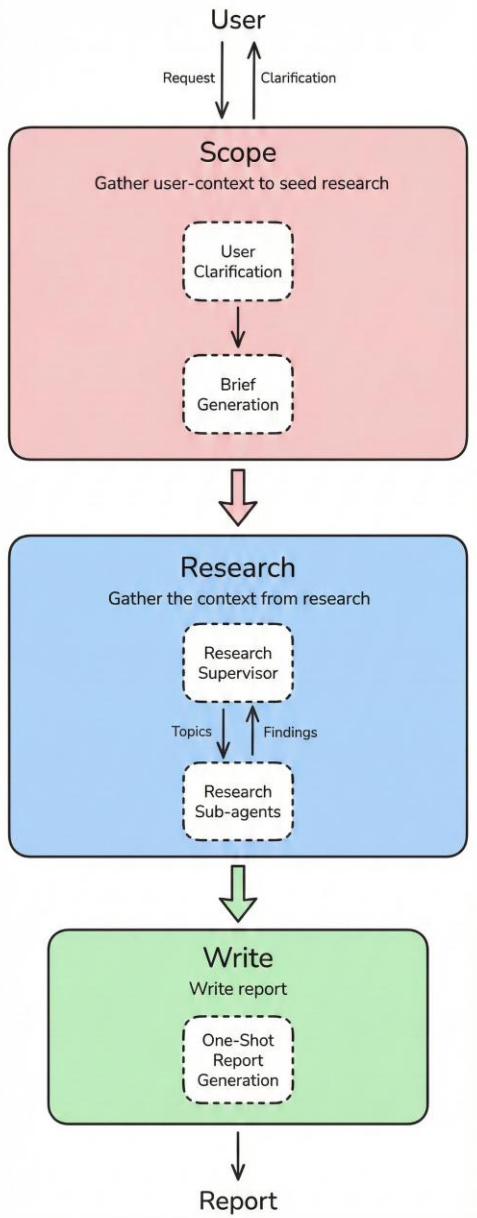


AI Agents in Action

Parallel processing and problem decomposition

- Under the hood, AI agents are just sequences of calls to LLMs:
  - They can break problems into subproblems and call multiple “subagents” in parallel
  - Each LLM call can have its own instructions and context, isolated from the rest of the process
  - Generally, one “supervisor” LLM maintains general context across the task
- Example: Open Deep Research tool by LangChain:
  1. Refine and clarify research brief with user; generate research brief
  2. Use research brief to break down task into multiple, smaller research queries; stop research when sufficient amount of information has been collected
  3. Combine findings from each subagent and write final research report

Example: Open Deep Research Agent



In Practice: Coding with LLMs

From “vibe coding” to professional software development

- Coding with LLMs is rapidly changing as the technology is continuously evolving

- Right now, three “levels” of how to code with LLMs:
  - Level 1: “Vide Coding” – coding in a chat window → copy/pasting code
  - Level 2: IDE integration – VS Code, Cursor, etc. → coding with autocomplete and light LLM tool use
  - Level 3: Advanced Agentic Coding – Claude Code/Codex, IDE for review, Git/Github for version control

Level 1: “Vibe Coding”

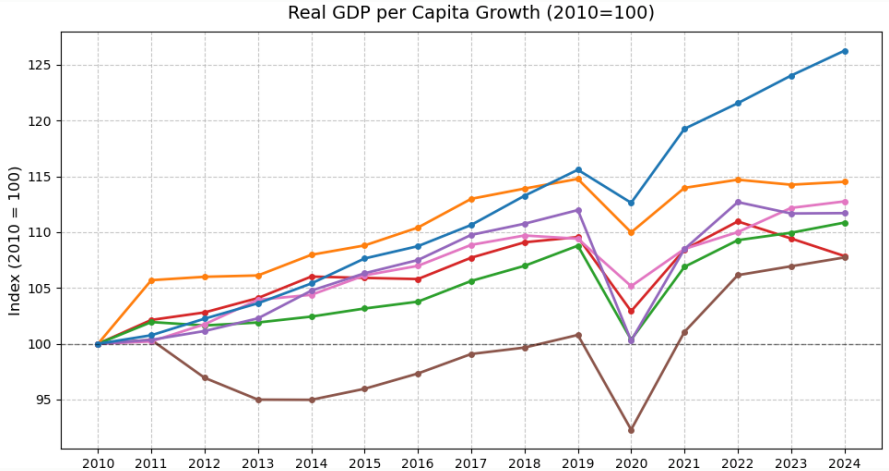
Quick scripts in the chat window

- Simplest entry point for coding: prompting the LLM to generate code directly in the chat window
  - Ask LLM to write code (in language X) to solve a particular problem
  - Inspect code
  - Copy/paste code into your environment and run
  - Interpret any error messages or bugs
  - Iterate via chat conversation
- **Good for:** small, discrete tasks; one-off scripts; testing ideas or simple prototypes
- **Limitations:** no version control; clunky to iterate and fix bugs; rapidly runs into context window limits; can only work on one file at a time;
- While useful in certain circumstances, pure “vibe coding” is generally not feasible for larger, more complex projects.

Example:

Write code in Python to plot real GDP per capita for all G7 countries from 2010 to 2025. Normalize data so 2010=100.

(chat, code, chart)

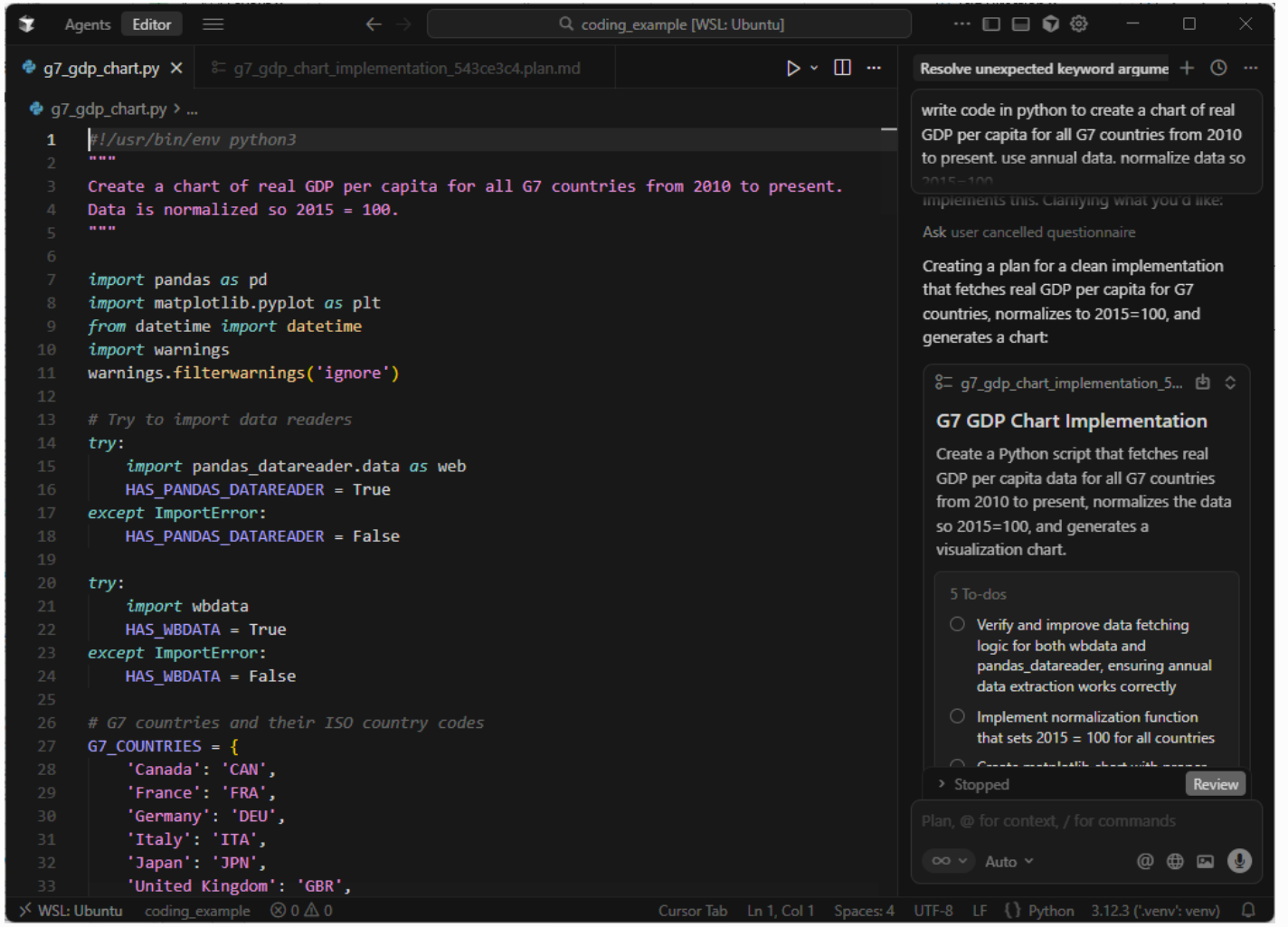


Level 2: IDE Integration

LLM-enhanced development environments

- Code in a professional IDE, but with LLM-enhanced tools: VS Code, Cursor, Github Copilot, Windsurf, Google's Antigravity
- Key features:
  - Autocomplete: simplest use; start typing code and use tab key to accept autocomplete suggestions
  - Generate code via chat window
  - Use "Plan Mode" for complex tasks
  - Accept/reject edits via "diffs"
  - Can help setup environments and other "coding overhead" requirements
- **Good for:** Medium-sized projects, learning to code, augmenting productivity of experienced coders

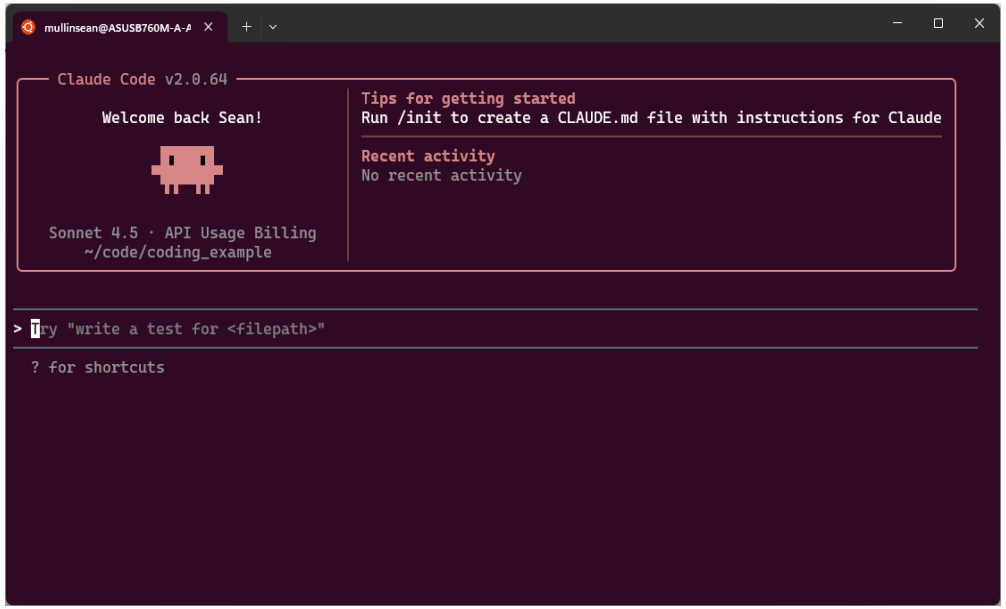
Example: Cursor IDE



Level 3: Advanced Agentic Coding

Command-line agents with version control

- Command-line interface (CLI) coding agents have emerged as most powerful tools for complex, production-level codebases: Claude Code and Codex (OpenAI) are the two leading models
    - Can work over multiple files, making edits across entire code bases, run tests and debug errors
- Use Git/Github for version control – mistakes can be easily rolled back
- Features:
    - Planning mode maps out workplan for your approval before coding
    - CLAUDE.MD (or AGENTS.MD) maintains state of project and any coding style patterns or other customizations
    - Explicit management of context window
    - Choose which model to use (eg: Sonnet, Haiku, Opus)
    - Directly edits files in your filesystem; can backup to Github; run command line tools



Workflow:

Generate feature → Inspect → Test → Commit

Deep Dive into Claude Code

Professional agentic coding in action

- Example 1: Setting up a new project in Claude Code
  - Initializing environment
  - Navigating Claude Code
  - Generating code to solve a problem
  - Committing to Git
- Example 2: Making a simple change to a file
  - Fetching repo from Github
  - Requesting edits
  - Reviewing and pushing to Github
  - Merging changes
- Example 3: Adding a new sophisticated feature to existing code base
  - Using Claude Code on the web to plan and implement code change

Documentation: Claude Code Overview

Techniques for Effective LLM Coding

Best practices across all three levels

- ☑ **Create "bite-sized" features:** Build incrementally, test as you go
- ☑ **Frequent commits:** Version control lets you easily roll back LLM errors
- ☑ **Use the planning feature with the most advanced models:** Let the LLM first write out a plan for approval before starting coding
- ☑ **Parse error messages:** Use LLM to troubleshoot and explain what went wrong; use this to build your knowledge as you go
- ☑ **Stay within your understanding:** Don't try to build things beyond your knowledge level (but experimentation is okay!)
- ☑ **Focus inspection on most critical parts of the code:** eg: review the core economic model code, not the chart generation or file saving boilerplate
- ☑ **Stick to popular languages:** Python, Javascript/Typescript, Java, C/C++ all have vast amounts of training data on the web
- ☑ **Build your knowledge gradually:** LLMs are great learning tools; expand your knowledge as you test more complex use cases for coding

Cost and Token Usage:

Using LLM-based tools to generate code can rapidly hit usage limits for most plans.

Many professional coders create API keys and get billed for coding based on per token usage.

This allows costs to be allocated *per project*, if desired.

LLM Coding: Shift in Mentality

From writing code to software architect

- Coding in the AI era is now much more about **co-developing the architecture for software** with an LLM and delegating the actual coding to the model.
  - Old mentality: Write code line by line
  - New mentality: Architect solutions, guide AI implementation
- Spend more time thinking through the problem you are trying to solve and what are the potential steps to solve it, rather than coding.
- Because writing code is cheap, experiment more: try new features, build multiple prototypes, create something just to learn
  - You can always throw away code or roll back mistakes that don't work.
- Final, "mission critical" code should be carefully tested

LLM Coding Mental Process:

- Define the problem clearly
- Break it into logical pieces
- Build features one at a time
- Test and evaluate
- Integrate and refine
- Move to the next step of the problem

Using Agents Beyond Coding

Practical applications for LLM-based agents

- Agents are not just for coding tasks and have a wider range of general-purpose uses.
- Claude Code is actually a general-purpose command-line tool. For example, it can:
  - Rename all the files in a folder or find all the PDF files in range of folders and move them
  - Summarize a series of PDFs in a folder and write the output to a text file
- Platforms like Zapier or Gumloop allow you to connect “AI agents” with existing tools to automate workflows
- MS Office Co-Pilot has advanced agent features for data analysis, task automation and research within the Office suite.
- Can be built into messaging/work communication tools like Slack or MS Teams to automate notifications, communication and provide an easy way to interact with agents.

Limits of Agents

- Errors can compound over a long horizon task; reducing likelihood of a successful outcome
- Context can degrade over long horizon tasks, limiting complexity of an agentic task
- Retrieving relevant data for task is still a work in progress
- Can be expensive (ie: use a lot of tokens)

Additional Resources

Resources for further exploring LLM-assisted coding

Tools:

- Claude Code, Codex
- Cursor, Windsurf
- Google AI Studio, Vercel, Replit
- VS Code, Git, GitHub

Guides:

- Claude Code
- Codex

Articles and People:

- Simon Williamson (link)
- Andrej Karpathy (YouTube and X)
- Claude Code in Action (course)
- Writing a good CLAUDE.md file (link)
- How to Use Cursor - for non-technical people (link)
- Social Science Tech Stack (Kevin Bryan) (link)

Final Thoughts

Using LLMs for success and happiness… :)

- LLMs are powerful tools that, if used properly, can significantly **augment** your productivity
  - The are especially helpful for knowledge work.
- LLMs are not a replacement for thinking:
  - Used properly, they can allow you to spend **more time thinking critically** about the most important aspects of your work;
  - But careful review of their output is frequently necessary
- They are incredible **teaching machines**:
  - Experiment often, continuously test their capabilities and use LLMs to learn
- The technology is **changing very fast**:
  - What is not possible now, may be a simple feature in 3-6 months
- Everyone is learning what they are capable of; no one is really an expert

Thank You!

Sean Mullin

sean.mullin@gmail.com

www.seanmullin.com