



Coding with GenAI: The Current Landscape

How both technical and non-technical users are rapidly finding value from the hottest new programming language: English.

By [Curtis Michelson](#), Contributing Writer, InnoLead

Is generative AI on the verge of becoming a powerful force in software development?

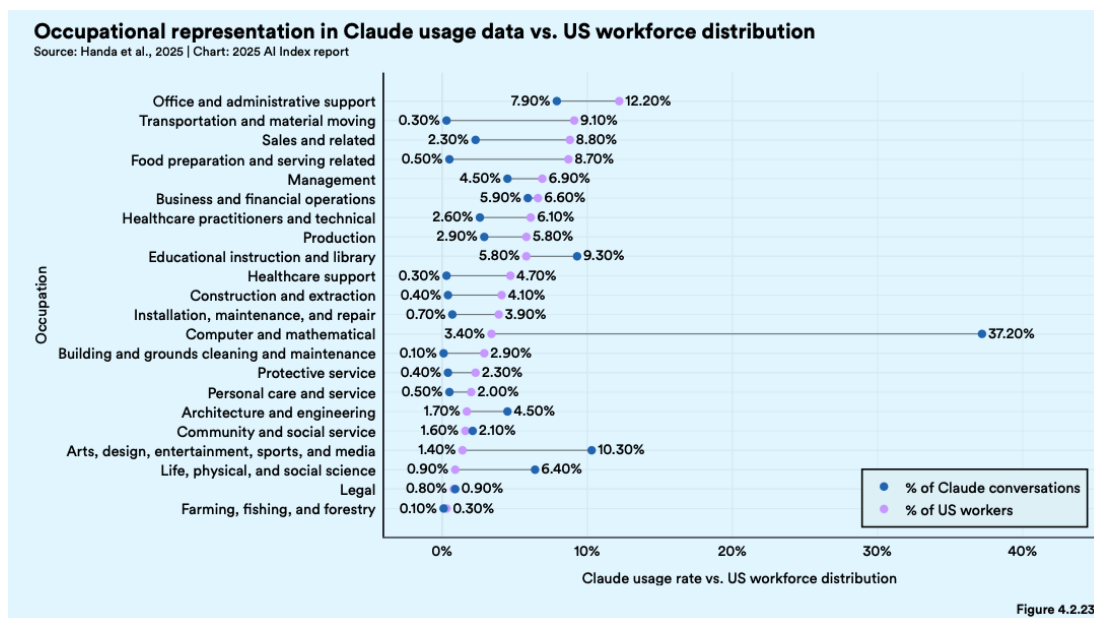
In 2025, coding and code generation have emerged as one of the most prominent and rapidly growing use cases for generative AI. The code-generation sector is projected to experience a remarkable 53 percent compound annual growth rate (CAGR) from 2024 to 2029, outpacing other generative AI modalities, according to a [June 2025 S&P Global report](#). Startups like Cursor, building coding assistants, [sport multi-billion-dollar valuations](#) and hundreds of millions in revenue. And [InnoLead research](#) earlier in the year found coding to be one of the top five ways that innovation and R&D groups in large organizations are using generative AI.

For both professional developers and ordinary folks like myself who want to set up a website quickly or build a data dashboard, the benefits of reduced development time, radical ease-of-use, and the ability to move quickly on one's own without the inertia of a whole team are obvious. And, the risks commonly associated with large language models (LLMs), like hallucinations, are markedly reduced. (More about why that's the case in a moment.)

I took a deep dive into how people are using these tools today; the risks and potential return on investment; some of the most popular and powerful platforms; and a few of the open questions about where this all goes.

The Current Landscape

Like all aspects of genAI, codegen is rapidly evolving. The market roughly breaks into tools for professionals and tools for everyone else. The [2025 AI Index Report](#) from Stanford University's Human-Centered AI center found that professional developers make up just 3.4 percent of the U.S. workforce but already constitute 37 percent of the conversations in Anthropic's Claude. Anecdotally, many developers, including ones I interviewed for this article, give Claude an edge (for the moment) over Gemini or chatGPT or others when it comes to general software development.



Anthropic just released their [Economic Index Report](#), looking at the impact of AI on software development, noting that far more people are using its Claude Code AI tool to automate the work of coding — do the work — rather than to augment their own work. The term “vibe coding” was coined by one of the OpenAI founders and former head of AI at Tesla, Andrei Karpathy, who used the phrase offhandedly in [a tweet](#) that went viral. It refers to the zen-like flow state one can enter by talking to a coding agent and seeing your app rendered on-the-fly. Traditional software development typically had a strong separation of specialists: designers in one department and coders in the other. Today, vibe coding is eroding or at least blurring these lines, as design teams whip up code and coders deliver designs, and both camps do so in minutes, days or weeks.

If you can just type, “Build me a slick, cool B2C website using Ruby On Rails, with a look-and-feel that is a cross between low-fi retro Craigslist and shiny Spotify” and 2-3 minutes later (a typical turnaround time for a tool like Claude) you have something halfway decent, then why not? That fast bypass of the “blank page” or “cold start problem” may be the thing that unblocks any team’s imagination and gets a project moving.

Where Is The ROI?

A value point for all users but especially for software teams is the fast troubleshooting capabilities these tools offer, helping developers not get bogged down reading software documentation. A good example of this is something [Gavin Hupp](#), VP Technology at United Parks & Resorts (which operates SeaWorld, Busch Gardens, and other theme parks) shared with me. Recently his team was a bit stuck trying to integrate with a vendor’s platform or API, Hupp said, “In the past, we would have spent a day or two with an engineer dedicated to the problem. Now the team figured out a solution in minutes.”

I heard a very similar story when I recently spoke with Aman Khan of [Arize AI](#) about his team’s uses of genAI, and he described a similar scenario where one of their large SaaS customers wanted to track how many of their team were using the Arize product; basically, they wanted a custom dashboard. While his team was debating some different approaches, considering putting an engineer on it for a week, Khan said, “I literally just went to Replit, prompted it with “build me a usage dashboard,” and it built me a Streamlit-based dashboard for that specific company and hosted it at a URL with authentication. I was like, “OMG, while the team was talking about this problem, I literally just built the prototype.”

The other big category of value that I heard from my interviews with pro coders is the ability to take on projects that were formerly too ambitious or even impossible. [Axel Rivera](#) is a pro iOS developer who delivers SaaS B2B apps in the finance sector. He knows Apple's programming language Swift very well, and he's a regular user of their integrated development environment or IDE, which is called XCode. But even for Rivera there are many areas of Swift / iOS coding he has yet to explore but he has found a friend in CodeGen. As he said, "There are projects or ideas that a couple of years ago I wouldn't have touched. Now, I can get a crash course on some obscure thing like a new video encoder, and I get a fast prototype with clear implementation examples. It is extremely satisfying that I can do things I wouldn't have considered before."

How and Why Does GenAI Coding Work So Well?

All LLMs can feel a bit mysterious. We're told they are *next word prediction machines*. But that seems an inadequate descriptor for this surprising and even spooky experience we get when vibe coding, or vibe writing or vibe music making, etc. At its most fundamental level, LLMs are indeed stochastic parrots. Trained on all the world's published knowledge, their outputs mirror all that they have seen before in their training. The surprising quality of our LLM interactions comes from the fact that deep multi-billion parameter neural networks are exquisitely sensitive to small variations in natural language inputs; i.e., to our prompts. As anyone using these tools for writing emails or making pitch decks knows, being very detailed and precise in your prompting (inputs) makes an enormous difference in the quality of output. Prompt engineering became a cottage industry just for these reasons, and that's also what has spurred the recent interest in meta prompting, the trend to have LLMs write those highly nuanced prompt inputs on our behalf.

However, in the arena of stochastically parroting *code*, as many research studies have shown, code generation systems deliver higher quality output more consistently than standard plain language interactions. The reasons for this are three-fold:

1. Code is by its nature more structured, offering a more constrained data space for LLMs to train on.
2. The outputs of codegen systems are immediately testable; code either works or doesn't.
3. There are clear evaluation benchmarks (good, better, best) for code quality that permit easier fine-tuning and model refinement, both human-in-the-loop and automated fine tuning.

As a non-coder myself, I've been shocked by the speed and quality of code that I get despite my rudimentary level prompting. For example, when I needed to access a list of public company 10k filings, I panicked because I found the byzantine EDGAR search interface intractable. But, asking Claude for help, it identified a well documented open source code project called [sec-api](#) for accessing SEC filings. I asked Claude to read the api documentation and then I literally said, "Write a script so that I can connect to and download a 10k filing for a company." In seconds, I had the following python script (see below), which I really didn't even know how to use at first. Claude walked me through how to open Terminal on my Mac and execute that script. I hit one error, but when I pasted my error from Terminal into Claude, it had a workaround. Within minutes, I had a bunch of 10k filings and went on my way merrily analyzing.

My actual Claude generated code:

```

import requests
import os

def download_sec_filing(api_key, sec_url, output_filename):
    """
    ...Download an SEC filing as PDF using the SEC-API.io service...
    ...Parameters:
    ...api_key (str): Your SEC-API.io API key
    ...sec_url (str): The SEC.gov URL of the filing
    ...output_filename (str): Name of the PDF file to save
    ..."""
    ...# API endpoint
    ...base_url = "https://api.sec-api.io/filing-reader"
    ...# Build parameters
    ...params = {
    ...    'token': api_key,
    ...    'url': sec_url
    ...}
    ...try:
    ...    # Make the GET request
    ...    response = requests.get(base_url, params=params)
    ...    # Check if request was successful
    ...    response.raise_for_status()
    ...    # Check if the response is a PDF (based on content type)
    ...    if response.headers.get('content-type') == 'application/pdf':
    ...        # Save the PDF file
    ...        with open(output_filename, 'wb') as f:
    ...            f.write(response.content)
    ...        print(f"Successfully downloaded: {output_filename}")
    ...    else:
    ...        print("Error: Response was not a PDF file")
    ...except requests.exceptions.RequestException as e:
    ...    print(f"Error downloading file: {e}")

```

If I can do this, then just about anyone can.

Should We, Just Because We Can?

A very hot topic in the CodeGen space at the moment is centering around the risks of letting novices into the world of professional coding, and the risks of creating inflated expectations for end-users who might not know that there be dragons in the back of the cave; i.e., there may be lurking bugs or generic clunky inefficient code that “works,” but is non-standard practice and very likely to be a resource hog at the best case, or in the worst case, a security hole.

One particular vulnerability identified in April of this year is “SlopSquatting.” Yes, you heard that right. SlopSquatting gets its name from two sources. “Slop” is the term for genAI generated pablum that is now passing for “good enough” email, marketing, sales and customer services copywriting. For example, if the word “delve” is used, as in, “let’s delve into the future of AI in this podcast”, it likely came from an LLM. Certain words are more favored by these systems than others. And “squatting” comes from the Web 1 era, when there was a gold rush on domain names, and people would literally buy up likely high-value names and sit on them. This new slopsquatting vulnerability comes from malicious hackers who know that codeGen LLMs will occasionally hallucinate the incorrectly spelled names of software libraries. These hackers then create and post their malware to open repositories like Github and cleverly name their poison packages with the same names that the LLMs hallucinate. Unwitting developers pull that malware into their projects.

Who Should We Be Watching?

The world of CodeGen startups is roughly cut into three big groups. The first is general purpose multi-modal models targeting broad consumers and developers, such as Claude, Gemini, chatGPT, etc. This is where many non-programmers like myself have discovered the “joy of coding” by creating all kinds of simple scripts and very basic apps.

Then there is a middle niche of prosumer tools (like Bolt, Replit or Lovable) which target creative and innovation professionals who want to go from idea to working app really quickly. The output from these systems come wrapped with hosting capabilities and their models are trained and tuned on modern web design palettes.

Finally, the third segment would be true professional grade coding tools (such as Cursor, Poolside, Windsurf, GitHub Copilot, etc.) that may be accessible to novices like myself, but are really tuned for software teams collaborating on big systems. These tools can automate much of the drudge work of dev teams like spawning cloud instances of platforms, creating commits and PRs (“pull requests”) into code repositories like GitHub. And they are fine tuned for identifying and fixing bugs in code as well, many of them automating such processes.

Below is my short list of the most well-funded and well known CodeGen tools in all three categories, and a bit about each of them.

Consumer Offerings

Name	Known For	Investors
Claude	Most folks (today) are giving Claude the edge for homebrew coding. Outputs are rendered in what Anthropic calls ‘artifacts’ which are like mini built-in web browsers for previewing.	Anthropic
chatGPT	Sports almost the same functionality as Claude but also offers a wider set of models to coding and code reasoning such as its highly regarded “o4-mini-high”	OpenAI
Gemini	This is probably the one to watch as Google begins to bring its full deep learning expertise to the Gemini suite. They have many CodeGen developer offerings already as well.	Google

Prosumer Offerings

Name	Known For	Investors
Lovable	It has certainly the most endearing name of them all, and it is loved by coders and non-coders for its lovely design touches and React-style code output.	Creandrum, byFounders, Hummingbird

Replit	This is very similar in capability to Lovable, but has a Ghostwriter AI assistant. Also distinguished by its built-in database editor, user authentication, and the ability to deploy apps directly from the browser.	Craft Ventures, Andreessen Horowitz, Coatue Management
Bolt	Again, very similar to the Lovable and Replit, but also has WebContainers, technology that runs dev environments directly in the browser, making it more cost-effective.	Emergence, Google Ventures, Madrona, Mantis

Developer Offerings

Name	Known For	Investors
GitHub CoPilot	Its inline code-completions across many dev environments (IDEs), helping developers rapidly write common code snippets. An Accenture study pegged the CoPilot productivity enhancement at 55%.	Microsoft which owns Github, and is powered by OpenAI's GPT.
Cursor	This is an AI-powered coding tool made by AnySphere. They have one of the largest user bases and are the most well funded. An assistant for engineers. Cursor's nearest competitor is Devin by Cognition Labs.	Thrive, Andreessen Horowitz, OpenAI
Poolside	They are developing engineering grade CodeGen. They are differentiated by their proprietary coding models that continuously learn how developers write code.	Bain Capital Ventures, Felicis, Redpoint Ventures
Windsurf	Acquired for \$3 billion by OpenAI in May 2025, this coding tool had almost the same traction as Devin and Cursor but a lower valuation. It will now be integrated into the chatGPT suite.	Kleiner Perkins, General Catalyst, Greenoaks, Founders Fund

How You Can Start To Test These Tools

A good thing to do when starting with any of these tools is to first ask it for its own capabilities. You'll get a refreshingly sales-lingo free output that lays out in clear concise terms the precise features and limitations.

Here is Google Gemini's response to my query (prompt) "tell me about your coding capabilities" on May 21st, 2025.

As a large language model, I have a range of coding capabilities developed through training on a massive dataset of text and code. Here's a breakdown of what I can do:

Understanding and Generating Code:

Multiple Programming Languages: I can understand and generate code in a wide variety of popular programming languages, including:

- Python
- JavaScript
- Java
- C++
- C#
- Go
- Ruby
- PHP
- Swift
- Kotlin
- SQL
- HTML/CSS
- And many more.

Code Comprehension: I can analyze and understand existing code, identify its purpose, and explain its functionality.

Code Generation: I can write new code based on your descriptions, requirements, or prompts. This can range from simple scripts and functions to more complex algorithms and boilerplate code.

Code Completion: I can suggest completions for partially written code, helping to speed up the development process.

Translation/Transpilation: I can assist in translating code from one programming language to another (though this often requires careful review and testing).

Coding-Related Tasks:

Debugging Assistance: I can help identify potential bugs in code by analyzing error messages and the code's logic. I can suggest possible causes and fixes.

Code Optimization: I can suggest ways to improve the efficiency or readability of code.

Algorithm Design: I can help you think through and outline algorithms for specific problems.

Explaining Concepts: I can explain programming concepts, syntax, and best practices.

Writing Documentation: I can help generate documentation for code, such as comments or README files.

Unit Test Generation: I can assist in creating unit tests for your code to ensure it functions as expected.

Regular Expressions: I can help you create and understand regular expressions.

API Usage: I can provide examples and explanations for how to use various APIs.

Version Control (Conceptual): I understand concepts related to version control systems like Git, although I don't directly interact with repositories.

How I Can Help You:

Learning: If you're learning to code, I can provide explanations, examples, and help you work through exercises.

Prototyping: I can quickly generate boilerplate or initial versions of code to get you started.

Problem-Solving: You can describe a coding problem, and I can suggest approaches or code snippets.

Brainstorming: I can help you explore different ways to implement a feature or solve a problem.

Important Limitations:

No Execution Environment: I don't have an actual coding environment to run or test the code I generate. You'll always need to test the code in your own environment.

Potential for Errors: While I strive for accuracy, the code I generate might contain bugs or not be perfectly optimal. It's crucial to review and test it thoroughly.

Context is Key: The quality of my output heavily depends on the clarity and detail of your prompts. The more specific you are, the better I can assist.

Security Concerns: Be cautious when using AI-generated code for security-sensitive applications. Always review it for potential vulnerabilities.

Complex Systems: For very large, complex, or highly specialized systems, my abilities might be more limited to assisting with specific parts rather than generating the entire system.

Keeping Up-to-Date: The world of programming is constantly evolving. While I'm trained on a vast amount of data, there might be very recent libraries or frameworks I'm less familiar with.

In essence, I can be a powerful coding assistant, helping you learn, write, understand, and debug code more efficiently. However, I am a tool, and human oversight, testing, and critical thinking remain essential in the software development process.

Where Is It All Headed?

Perusing the above list of CodeGen offerings, nobody has a crystal ball, and certainly much consolidation is expected. Windsurf won't be the last one snatched up. But despite the uncertainties, it does appear that these startup investments are dividing up between professional and consumer plays, with the pro offerings getting the lion's share of the capital.

Here are some final open-ended questions I believe worth pondering about all of this.

Are the VCs betting on companies that aim to completely replace traditional developer workflows? The most well funded firms (Anysphere, Poolside, Sourcegraph, Devin) are all clearly reframing software engineering as something way more comprehensive than code completion assistants or copilots. What then will be the definition of a software developer in another year or two?

Is the new AI arms race in the code arena moving to the model level? Poolside is being funded like an LLM-scale foundation model company. Windsurf's \$3 billion acquisition by OpenAI suggests more consolidation to come. It appears that domain-specific models for code (distinct from general LLMs like Claude) are worth competing for. More capital may flow into vertical foundation model plays with code-specific modalities for things like security, or governance, or code quality.

Are the agentic codegen platforms the new arena of competition? Massive valuations (for Anysphere, Cognition Labs, Magic) despite relatively early stage success suggest that VCs believe the agentic interface is the new IDE. Perhaps the playbook is no longer about better interfaces; it's about becoming the Slack/Figma/GitHub of autonomous user intent capture and codegen.

Is there an under-appreciated opportunity for investors in the much larger consumer space? Lovable and Bolt receive comparatively little capital, despite targeting a larger latent market; i.e., non-developers who want to create software. Perhaps we'll see a consumer-grade dev tool for "citizen developers"—filling the gap between Replit and Lovable. The education sector seems like interesting terrain for this. A Khan Academy-type offering just for getting students from idea to app in seconds.

What does a software developer or development team look like in two to three years? When the things we are most capable of can be done far better by AI agent, and CodeGen tools are able to quality-control and debug their work, software development may truly get democratized. That will certainly change the role of anyone who trained to be a coder, tester, or software development manager. But the changes in how software gets created will be just one component of the emerging AI-driven future of work.