

Vibe Coding: Survey of the birth of Vibe Coding

Introduction

Vibe coding is an emerging approach to software development where the programmer “fully give[s] in to the vibes” and lets AI generate code from natural language prompts ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). The term *vibe coding* was coined by computer scientist Andrej Karpathy in February 2025 to describe coding with the assistance of large language models (LLMs) such as GPT-4 or Anthropic’s Claude, to the point that one can “*forget that the code even exists*” ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). Instead of writing syntax and algorithms manually, the developer describes desired features or changes in plain English (even via voice), accepts the AI’s suggestions, runs the code, and iterates. Karpathy quips: “*I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.*” ([Will the future of software development run on vibes? | CediRates](#)) In essence, vibe coding turns programming into a high-level conversation with an AI agent, radically streamlining prototyping and “weekend projects” ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). This report dives into the workflows and tools enabling vibe coding, industry opinions (from excitement to skepticism), real-world examples of “vibe” development in action, and how this paradigm compares to both traditional coding and other AI-assisted methods.

Tools and Workflows

Vibe coding relies on advanced AI coding assistants and supportive tooling to translate natural-language “*vibes*” into working software. Key components of the vibe methodology include chat-based IDEs, powerful code-generating models, and even voice interfaces. Below are some of the prominent tools and how they support vibe-centric workflows:

- **Cursor (AI Code Editor):** A popular VS Code–derived IDE that integrates AI “pair programmers” and chat-based code generation. Cursor’s *Composer* chat allows users to type or speak instructions and get code written or modified accordingly ([ALL-VIBE-CODING-youtube.txt](#)). For example, one can ask Cursor to “*decrease the padding on the sidebar by half*” and accept its code change without ever manually searching the code ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). Cursor leverages powerful models like Anthropic’s Claude (v3.5/3.7 “Sonnet”) behind the scenes ([ALL-VIBE-CODING-youtube.txt](#)). It also introduced an *Agent* mode to autonomously analyze and edit code, and supports voice input via Whisper. One live coder explained their vibe workflow: “*we actually didn’t write any code ourselves – we just chatted with [a] chatbot*” to build a whole app using Cursor ([ALL-VIBE-CODING-youtube.txt](#)). By combining an editor and chat in one, Cursor exemplifies vibe coding’s “flow state” development environment.

- **Anthropic Claude (LLM):** Claude is a state-of-the-art AI model known for strong coding capabilities, and is often the engine behind vibe coding tools. Cursor’s latest versions are “powered by Claude 3.7 Sonnet”, which one developer called “*the best AI model that was just released*” ([ALL-VIBE-CODING-youtube.txt](#)). Claude can follow high-level instructions to generate entire project structures or complex functions. In vibe workflows, Claude (or similar models like GPT-4) handles the *heavy lifting* of writing code, while the human guides it. Claude’s extended context window and “extended thinking” features allow it to work on larger codebases, making it suitable for non-trivial projects ([Claude 3.7 Now Available! - Discussion - Cursor - Community Forum](#)). Many vibe coders credit Claude with massive productivity boosts – one early adopter said “*Sonnet 3.7 coded everything for me... I’ve never written a line of code [before]*”, yet they built a usable app in 20 minutes ([ALL-VIBE-CODING-youtube.txt](#)).
- **Replit Ghostwriter and Agent:** Replit, an online IDE, has embraced the vibe coding trend by integrating AI through its Ghostwriter (code completion/chat) and new “Replit Agent”. These allow users to “*quickly build software*” by describing what they want. In fact, Replit’s CEO Amjad Masad noted that “*75% of Replit customers never write a single line of code!!*” – indicating that a majority are effectively vibe coding already by using high-level prompts and letting the AI produce the code ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). Replit’s AI can scaffold entire projects from a prompt and even deploy them. This lowers the barrier so that non-programmers can create apps just by describing their ideas. Replit’s approach, along with Cursor, has made vibe coding “*increasingly accessible to non-programmers*” ([Will the future of software development run on vibes? | CediRates](#)).
- **GitHub Copilot:** Copilot (powered by OpenAI Codex/GPT) is an earlier AI coding assistant that autocompletes code inside your editor. While Copilot doesn’t quite enable full “*chat and run*” vibe coding, it introduced many developers to the idea of AI-generated code suggestions. Copilot excels at offering inline code while you type, whereas vibe coding usually involves more dialog with an AI. Copilot can be seen as a stepping stone: it boosts productivity (sometimes “*2× improvement... It’s amazing having an AI pair programmer*” as one dev praised ([Cursor - The AI Code Editor](#))) but generally expects the human to still drive the code writing. In vibe coding, by contrast, the AI often generates *every* line of code (subject to user prompts and approvals) ([Vibe coding - Wikipedia](#)). Many vibe coders still use Copilot for low-level assistance, but rely on chat agents like Cursor/Claude for bigger leaps in functionality.
- **Windsurf (AI IDE by Codeium):** Windsurf is a new AI-powered IDE often compared to Cursor. It provides a chat-centric coding experience through a feature called Cascade (equivalent to Cursor’s Composer) ([Windsurf vs. Cursor - which AI coding app is better?](#)). Windsurf’s UI is noted to be beginner-friendly and it automatically manages context (figuring out which files to modify based on your requests) ([Windsurf vs. Cursor - which AI coding app is better?](#)). Like Cursor, it has an agentic mode to execute multi-step tasks. Some reviews found “*Windsurf’s agent feature is a bit easier to use*” for guiding code generation step-by-step ([Windsurf vs. Cursor - which AI coding app is better?](#)). Windsurf is positioned as “*the first AI agent-powered IDE that keeps developers in the flow*” ([Getting Started with Vibe Coding: Build AI-Powered Apps ... - Medium](#)) – very much aligned with the vibe coding ethos of

“surrendering to the flow” of AI-driven development ([Will the future of software development run on vibes? | CediRates](#)). Essentially, Windsurf and Cursor are competing to be the go-to “AI coding app” for vibe-style programming ([Windsurf vs. Cursor - which AI coding app is better? - Prompt Warrior](#)), both being VS Code-like editors supercharged with chat, agents, and integration with models like Claude or Codeium’s own.

- **Voice Interfaces (SuperWhisper):** A distinctive aspect of Karpathy’s vibe coding was using voice input to *literally* speak programs into existence. He used an enhanced speech-to-text tool called *SuperWhisper* to dictate prompts to Cursor’s Composer ([ALL-VIBE-CODING-youtube.txt](#)) ([ALL-VIBE-CODING-youtube.txt](#)). This means instead of typing at all, you describe what you want out loud (e.g. “create a user login form with two-factor auth”), and the AI writes the code while you watch. SuperWhisper (built on OpenAI’s Whisper) can transcribe natural language accurately in real-time. One vibe coder noted, “*I barely even touched the keyboard – I just say stuff [and] run stuff*” thanks to voice control ([ALL-VIBE-CODING-youtube.txt](#)). This hands-free approach has been likened to “*coding at the speed of thought*”. It makes the development process feel more like commanding a smart assistant than traditional typing. Voice-enabled vibe coding is still experimental, but it points toward a future where “*speaking things into existence*” becomes commonplace in programming.

([Will the future of software development run on vibes? | CediRates](#)) Developers describe vibe coding as “surrendering to the flow” – here symbolized by riding a wave of binary code. In vibe coding, one rides on AI generation, guiding it loosely, rather than painstakingly writing every line. ([Will the future of software development run on vibes? | CediRates](#)) ([Will the future of software development run on vibes? | CediRates](#))

- **Other Tools (“Vibe Writing” and more):** The “vibe” methodology is spreading beyond code into other creative domains. For instance, *vibe writing* applies the same principle to content creation – you outline an idea or just speak a stream of thoughts, and AI (like Claude or ChatGPT) generates the article or copy. Writers have begun asking for a “Cursor for writing” to practice “*vibe writing, just speaking stream of consciousness thoughts and [letting] AI clean it up*” ([Posts with replies by benyo \(@mattbenyo\) / X](#)). Early adopters report using Claude with modular prompt chains (MCP servers) to churn out blog posts from a simple thesis and a few follow-up prompts ([Sid Bharath \(@Siddharth87\) / X](#)). The concept of *vibe design* is even being discussed (e.g. describing a UX and having an AI draft the interface). All these tools share a common workflow: the human provides high-level guidance or goals, and the AI does the detailed generation. The human then tests or reviews the output and iterates by giving more instructions or corrections (often just by feeding error messages back into the prompt, as vibe coders commonly do ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#))). This iterative prompt-execute loop is at the heart of vibe coding/workflows. It shifts the human role to that of a *product or idea director* and the AI into the role of an ever-willing, ultra-fast implementer.

Developer Opinions and Reactions

The rise of vibe coding has elicited a mix of enthusiasm, curiosity, and concern within the developer community. Below we consolidate some insightful opinions from recent podcasts and online discussions:

Excitement about Productivity and Accessibility: Many developers are amazed at how vibe coding can accelerate development and open programming to new people. Andrej Karpathy himself finds it “*quite amusing*” that he can build a web app by essentially not coding at all, just by iteratively prompting an AI ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). Experienced programmers like Karpathy enjoy “*embrac[ing] exponentials*” – harnessing the exponentially improving capabilities of LLMs to boost their output ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). From an industry angle, Y Combinator partners noted that vibe coding might be “*the dominant way to code*” going forward, warning that if engineers aren’t leveraging AI assistance, “*you might just be left behind.*” ([ALL-VIBE-CODING-youtube.txt](#)) In a YC podcast, several startup founders reported dramatic gains: one said in the last year coding went from a 10× speed-up to “*100× speed-up... I’m no longer an engineer, I’m a product person.*” ([ALL-VIBE-CODING-youtube.txt](#)) This sentiment that coding is shifting toward higher-level product thinking was echoed repeatedly. Another founder quipped, “*I don’t write code much; I just think and review,*” since adopting AI-assisted workflows ([ALL-VIBE-CODING-youtube.txt](#)). These practitioners argue that tools making “everyone a 10x engineer” let them focus on creativity and user needs rather than low-level details ([ALL-VIBE-CODING-youtube.txt](#)). Tech executives also see huge potential: Replit’s CEO declared “*vibe coding is already here*”, citing how the majority of users on his platform build apps without writing code ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). And in the New York Times, journalist Kevin Roose – not a professional coder – gushed that with AI help, “*just having an idea can be enough*” to create useful software ([Vibe coding - Wikipedia](#)). Many are thrilled that vibe coding might democratize software creation, empowering “amateur programmers” to realize their ideas without years of training ([Vibe coding - Wikipedia](#)) ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). As one observer put it, “*everyone deserves the ability to automate tedious tasks... You shouldn’t need a CS degree... vibe coding shaves that initial barrier down to almost flat.*” ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). For seasoned devs, vibe coding is also an exciting way to prototype rapidly and experiment. Developer Simon Willison notes he’s built over **80** small projects using vibe coding, learning new capabilities of LLMs each time ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). He argues that “*this weird new shape of programming has so much to offer the world,*” encouraging developers to try it and build intuition about what AI can or cannot do ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). Overall, the optimistic camp sees vibe coding as a **fun, productivity-supercharging innovation** that can make developers an order of magnitude more productive and make programming more inclusive.

Cautious Optimism and Best Practices: Some experienced voices advocate a balanced view – leveraging the “vibes” for speed, but with discipline. They differentiate between *responsible* AI-assisted coding and careless vibe coding. Willison emphasizes that “*using LLMs for code responsibly is not vibe coding*” ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). In his words, “*If an LLM wrote every line of your code, but you’ve reviewed, tested, and understood it all, that’s not vibe coding... that’s using an LLM as a typing assistant.*” ([Vibe coding - Wikipedia](#)) In other words, blindly accepting AI output without scrutiny is the hallmark of vibe coding, whereas professionals should still rigorously review AI contributions. Many devs recommend **keeping vibe coding to low-stakes projects** or initial drafts. “*Projects should be low stakes... If your code could cause harm or needs strong security, vibe coding might pose risks,*” Willison advises ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). The YC community discussions similarly noted that while prototypes can be vibed into existence, production systems require engineers to maintain understanding and control ([Vibe coding - Wikipedia](#)). There’s a growing notion that the ideal workflow is a “*vibe then refine*” approach: use vibe coding for rapid scaffolding and idea exploration, then switch to traditional engineering rigor to polish and harden the result. As one Reddit commenter put it, “*AI is your tool, don’t make it the other way around.*” ([Hot take: Vibe Coding is NOT the future : r/ChatGPTCoding - Reddit](#)) – a reminder that the human should ultimately stay in the driver’s seat.

Skepticism and Criticisms: Not everyone is on board the vibe train. Some seasoned developers express discomfort or outright disdain for the concept. A viral Reddit thread titled “*Why ‘Vibe Coding’ Makes Me Want to Throw Up*” captured a visceral negative reaction, accusing the trend of encouraging sloppy, cargo-cult programming without understanding (the title itself indicates the emotional pushback) ([Why 'Vibe Coding' Makes Me Want to Throw Up? : r/programming](#)). More substantively, engineers worry that vibe coding can lead to codebases that nobody truly understands. A senior Microsoft engineer told Business Insider that vibe coding is “*a little overhyped*” and that while “*LLMs are great for one-off tasks,*” they are “*not good at maintaining or extending projects... [the AI] gets lost in the requirements and [can] generate a lot of nonsense.*” ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) This reflects a common experience: AI can spew working code for the first 80% of a feature, but when you need to adjust or debug something subtle, the model may flounder. Even tech investors like Andreessen Horowitz’s Andrew Chen shared frustration after trying vibe coding – he found “*you can get the first 75% [of an app] trivially... then try to make changes and iterate, and it’s... enormously frustrating.*” ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) Chen still calls the approach “brilliant,” but highlights the difficulty of fine-tuning AI-generated code beyond the happy path. Other critics note that vibe coding could **erode fundamental skills**. Cambridge researcher Harry Law warned that the “*ease of use is a double-edged sword – beginners can make fast progress, but it might prevent them from learning about system architecture or performance.*” ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) New coders who rely solely on AI might skip learning why the code works, leading to knowledge gaps. There’s also concern about **technical debt and bugs**: AI might introduce subtle errors that a vibe coder doesn’t notice in the frenzy of “accepting all” changes. Without careful review or tests, those errors accumulate. “*Security vulnerabilities may also slip through without proper code review,*” Law noted ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)).

Essentially, the skeptics argue that *code quality, maintainability, and long-term understanding can suffer* if one is always “coding on vibes.” Even Karpathy acknowledged these drawbacks in his original post: the code “*grows beyond my usual comprehension*” and sometimes the AI hits a wall with a bug it can’t fix ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) – at which point the vibe coder either manually debugs (negating the approach) or resorts to hacks like asking for random changes until something works ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). This is hardly a systematic or desirable engineering practice for mission-critical software. In sum, more critical voices view vibe coding as **useful in moderation** but “reckless” if taken as a primary approach for serious development ([Will the future of software development run on vibes? - Tech News](#)). They urge developers not to treat AI as a magic black box – or risk becoming dependent on solutions they don’t understand.

Success Stories and Use Cases

Despite the cautions, there have been numerous success stories that showcase what vibe coding (and writing) can achieve when used appropriately. These examples illustrate the power and current limitations of the approach:

- **Non-Programmers Building Apps:** A compelling story came from *The New York Times* tech columnist Kevin Roose, who experimented with vibe coding despite not being a coder by trade. Using tools like GPT-4, Roose managed to create several small custom applications by simply describing his ideas ([Vibe coding - Wikipedia](#)). He built a “LunchBox Buddy” app that analyzes the contents of his fridge and suggests lunch recipes, and even a prototype e-commerce site (where the AI amusingly invented fake product reviews) ([Vibe coding - Wikipedia](#)). Roose dubbed these personal utilities “*software for one*” – niche apps tailored to an individual’s needs, which previously would have been too costly or tedious to develop. With vibe coding, he found he could go “*from idea to a working program in a few prompts*”, something unimaginable a few years ago. Roose’s projects were not flawless (he encountered errors and AI hallucinations, like those fake reviews ([Vibe coding - Wikipedia](#))), but they *did* work. His takeaway was that vibe coding is *fantastic for hobby projects or personal tools*, even if it’s not yet reliable for “essential tasks” or commercial-grade apps ([Vibe coding - Wikipedia](#)). It demonstrates how someone with zero coding experience can leverage AI to create useful software quickly – a clear win for accessibility.
- **Startup Rapid Prototyping:** Within startup circles, vibe coding is becoming an asset for rapid prototyping and iteration. Misbah Syed, founder of Menlo Park Lab (a generative AI startup), told *Business Insider* that he is “all in” on vibe coding ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). He uses it to build features for their products like *Brainy Docs* (an app that converts PDFs into explainer videos with slides). Syed will simply describe a new feature or improvement, let the AI implement it, and if the AI makes a mistake, he feeds the error back into the prompt to get a fix ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). This tight loop often resolves issues in a few tries. “*If you have an idea, you’re only a few prompts away from a product*,” Syed said, encapsulating

the core benefit of vibe coding ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). Such speed is incredibly valuable for startups trying to find product-market fit; they can test ideas without sinking weeks into engineering. Another example: a founder in YC's Winter 2025 batch reported that because he can generate code 3× faster with AI, he's "*far less attached to [his] code now*" and is willing to scrap or refactor without hesitation ([ALL-VIBE-CODING-youtube.txt](#)). This indicates AI generation can make code more disposable – enabling a "*fail fast*" mentality since rewriting is cheap when an LLM does most of the work. Y Combinator even found that 25% of startups in that batch had codebases that were 95% AI-generated ([Vibe coding - Wikipedia](#)). That statistic ([Vibe coding - Wikipedia](#)) is striking – it suggests a substantial share of new companies are managing to build their initial products almost entirely with AI producing the code. These early adopters clearly find vibe coding good enough to launch with (though one assumes they still have engineers in the loop for review). It's a real-world validation of the viability of AI-driven development.

- **Personal Automation and Scripting:** On an individual level, developers have shared many anecdotes of how vibe-style prompting helps them automate daily tasks or create "weekend projects" quickly. Simon Willison, for example, built dozens of small tools (web apps, data scrapers, etc.) via vibe coding ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). In one case, he "vibe coded" a plugin to search a library of conference transcripts – something he accomplished in hours by iteratively prompting an LLM to write the code and querying errors, whereas it might have taken days traditionally. There are also success stories of **vibe writing** in content creation: e.g. a tech blogger "vibe wrote" an entire article series using an AI writing assistant, only lightly editing the AI's output. They described the experience as having an infinite first-draft assistant – you pour in ideas and get a coherent draft back, which dramatically cuts down writing time. Similarly, game designers have toyed with "*vibe programming*" game logic: describing game mechanics to an AI and getting working prototype code for simple games. While these small successes don't make headlines, they show that thousands of developers and creators are integrating vibe techniques into their workflows to great effect.
- **Educational and Learning Uses:** Interestingly, vibe coding is also finding success in educational contexts. Some coding instructors use vibe coding to help students visualize projects without getting bogged down in boilerplate. For instance, a student can describe a simple app (like a to-do list) to ChatGPT or Cursor and get a runnable template, which they can then tweak. This *immediate feedback loop* helps learners see results and stay motivated. One podcast guest noted a case of university students "*vibe coding for a year*" for their assignments ([ALL-VIBE-CODING-youtube.txt](#)) – essentially using AI to do the heavy lifting. While this raises academic honesty questions, it also points to how the next generation of developers might naturally lean on AI. Some educators suggest that if guided properly, vibe coding can teach concepts faster by letting students experiment (though others worry it could become a crutch).

It's important to mention that **not all attempts are success stories** – there are plenty of failures where vibe coding produced a tangle of bugs or an off-track implementation. Yet, even those failures often happen much faster (and with less effort) than a failed traditional coding attempt, which is why many

hackers don't mind trying. Overall, the success stories show that for **prototyping, personal projects, and iterative product development**, vibe coding is already delivering real value in terms of speed and flexibility. From journalists automating their chores, to founders compressing development cycles, to amateurs creating software out of thin air, the “describe it, and let AI build it” approach has proven its worth in a short time.

Criticisms and Limitations

Vibe coding, for all its promise, comes with a number of caveats and limitations that experts and practitioners have pointed out. Understanding these pitfalls is crucial to using the approach wisely.

Lack of Code Understanding: The most frequently voiced concern is that vibe coding can produce code that the user doesn't truly understand. By bypassing the act of writing code, developers may end up “glossing over” the implementation details. As one Reddit user bluntly put it, “*you have to remember, AI is your tool – don't make it the other way around*” ([Hot take: Vibe Coding is NOT the future : r/ChatGPTCoding - Reddit](#)). If a person accepts AI-generated code without reading it, they might not notice subtle bugs or design flaws. Karpathy joked that in vibe coding he stops reading diffs and just hits accept ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). This is fine for a toy project, but in a real system it could be dangerous. The code might function initially, but with no one understanding how the pieces truly work together, future changes or debugging become a nightmare. A Microsoft engineer warned that LLMs “*get lost in the requirements*” for complex projects and can “*generate a lot of nonsense content*” if used to extend a codebase ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). Without human oversight, these systems might introduce errors early that only surface much later.

Brittle Iteration and Debugging: Vibe coding tends to work great up to a point – often described as the “80%” mark of a feature – and then can hit a wall. Andrew Chen described the scenario: the AI gets you a mostly working prototype “*trivially*”, but “*then try to make changes and iterate*” and things fall apart ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). This brittleness comes from the fact that the AI doesn't truly *understand* the code's intent; it's predicting likely completions. When the user's requests get into edge cases or require refactoring, the AI might introduce regressions or conflicting changes. Karpathy mentioned that “*sometimes the LLMs can't fix a bug*”, so his strategy in vibe mode was to “*work around it or ask for random changes until it goes away*” ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). Clearly, *randomly poking the code until the error disappears* is not a reliable debugging method! It highlights that without traditional debugging and logical analysis, one can end up in AI-induced rabbit holes. Vibe coding currently lacks robust support for complex debugging – the AI can try to fix issues if you paste error messages (a common trick), but if the fix requires a conceptual leap or architectural change, the model may not get it. In such cases, the human has to step back in and solve it the old-fashioned way. This indicates that vibe coding works best for relatively self-contained tasks or well-trodden implementations, and struggles with novel or intricate problems.

Quality, Security, and Maintainability: Professional developers worry that code generated through vibes might be inefficient or insecure. Since vibe coding emphasizes speed and “getting it working”

over careful design, it may ignore best practices. For example, an AI might write a SQL query that is correct but not optimized, or use a library with known vulnerabilities simply because it was present in training data. Without the developer scrutinizing the output, these issues slip by. “*Security vulnerabilities may also slip through without proper code review,*” cautioned researcher Harry Law ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). Maintainability is another issue: code that “magically” appears can be a black box to the team that owns it. If an AI wrote a thousand-line component and none of the developers fully read it, updating that component six months later could be daunting. Simon Willison’s rule for AI code is he won’t commit anything he couldn’t explain to a colleague ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) – a good principle that vibe coding might violate in practice. Furthermore, reliance on AI means **dependency on external systems** – if the API or model changes (or if you lose access), will you be able to continue development? Vibe-coded projects might also inherit any biases or licensing issues from the training data (e.g. an AI might inadvertently reproduce copyrighted code, which the user then unwittingly includes). These are concerns that traditional coding doesn’t have in the same way.

Not a Silver Bullet for All Problems: Vibe coding excels at producing common boilerplate and well-known patterns, but it’s not as good at truly novel algorithms or highly domain-specific logic. If your problem requires creative new solutions, an AI might not have seen anything similar and could flounder. Developers note that LLMs sometimes fail to correctly implement complex mathematics or innovative techniques, requiring the human to intervene with actual coding. There’s also the matter of tools integration – while AI can write code, setting up infrastructure, configuring services, or dealing with legacy systems may require steps that aren’t easily done via a text prompt. In current vibe coding setups, tasks like deploying to cloud, managing database migrations, etc., still need either human work or specialized AI plugins. So there are practical limits to how far an idea-to-code prompt can go in a real production pipeline.

Cultural and Team Challenges: On the human side, some engineers resist vibe coding because it feels like *cheating* or devalues their hard-earned skills. Team dynamics could suffer if part of the team is vibe coding and others are manually coding – there might be disparities in code style or understanding. Also, code reviews become interesting when an AI wrote the code; reviewers might be more critical or unsure how to give feedback (“tell the AI to do better”?). Some organizations have policies against committing AI-generated code without review, which essentially forbids true vibe coding (since vibe coding by definition skips thorough review). Additionally, using AI tools may raise compliance issues (e.g. can we send this proprietary code as context to a third-party LLM service?). These are limitations not of the technology per se, but of the ecosystem and environment in which software is built.

In summary, vibe coding in 2025 is **not a panacea**. It has clear weaknesses around code correctness, maintainability, and reliability. As one engineer said, “*Vibe coding your way to a production codebase is clearly risky*” ([Vibe coding - Wikipedia](#)). It shines for quick hacks and drafts, but falls short for robust software engineering needs unless paired with traditional practices. Even Karpathy only advocates it for “*throwaway weekend projects*” or when you’re intentionally exploring what the latest AI can do ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). The current limitations highlight why human developers and solid software engineering fundamentals aren’t obsolete – the “*AI magic*”* still needs a lot of human judgment to yield durable results.

Cultural Adoption and Community Impact

The concept of “vibes” in coding has rapidly infiltrated tech culture, sparking both serious discussion and lighthearted meme-making. In just a few months since Karpathy’s coinage, *vibe coding* has become a buzzword in Silicon Valley and beyond ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). Here’s how it’s shaping discourse and community practices:

Mainstream Media Attention: Unusually for a programming technique, vibe coding has been featured in major media outlets. The New York Times, The Guardian, Ars Technica, and others ran pieces explaining the trend ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). Merriam-Webster even listed “*vibe coding*” in their dictionary as a trending slang term by March 2025 ([Vibe coding - Wikipedia](#)). This level of coverage indicates that vibe coding isn’t just a niche hacker idea; it’s being presented to the general public as a glimpse of how AI is changing everyday work. Journalists like Kevin Roose wrote in an accessible way about building apps with natural language, likely inspiring more non-coders to give it a try ([Vibe coding - Wikipedia](#)). Such articles also often mention the caveats, helping set public expectations (e.g. the Times noted the AI-produced apps are “prone to errors” and best for personal use ([Vibe coding - Wikipedia](#))). Overall, having vibe coding in mainstream conversation is contributing to a broader cultural narrative of *AI as a creative collaborator* rather than just a back-end tool.

Social Media and Developer Forums: On Twitter/X, Reddit, and YouTube, vibe coding has become a hot topic. Karpathy’s original tweet garnered nearly 4 million views ([ALL-VIBE-CODING-youtube.txt](#)) and kicked off lively threads of developers sharing their experiences. Some responded with humor – e.g. one user joked vibe coding is like “*a random walk through the space of app hallucinations*” if you fully give up control ([Andrej Karpathy on X: "@mattshumer Haha so it's like vibe coding ..."](#)). Others shared short demo videos of themselves “vibe coding” small games or websites, often captioned with astonishment at how quickly things came together. The r/ChatGPTCoding subreddit is filled with tips and debates about vibe coding (with some purists pushing back, as noted). YouTube content creators jumped on the trend, producing tutorials titled “Vibe Coding 101” or live-coding sessions where they attempt projects by only talking to the AI. One popular YouTuber streamed building a front-end without typing, dubbing it “*full-on vibe coding*” and narrating the triumphs and struggles in real time ([ALL-VIBE-CODING-youtube.txt](#)) ([ALL-VIBE-CODING-youtube.txt](#)). These videos demystify the process – viewers can see both the “wow, it wrote that whole component!” moments and the “hmm, the AI is stuck, let’s rephrase” moments. This transparency in community content helps newcomers pick up effective strategies and also realize it’s not *pure magic* (you still need to guide and sometimes wrestle with the AI).

Terminology and Spin-offs: The very word “vibe” has caught on as a prefix for AI-assisted creative processes. We now see discussions about **vibe writing**, **vibe designing**, even “*vibe translating*” (using AI to translate with style rather than literal accuracy) ([Posts with replies by benyo \(@mattbenyo\) / X](#)). This meme-ification of the term indicates a cultural moment – much like “cyberpunk” or “metaverse” became part of the lexicon. It’s worth noting that *vibe* in this context implies a sort of intuition, flow, or feel-based working style, as opposed to analytical precision. Developers speak of “*vibing with the AI*” to solve a problem, almost as if collaborating with a colleague. A LinkedIn post introduced “*vibe*

writing” as “*like vibe coding but for tech-savvy writers*”, stressing letting ideas flow and the AI handle the wording ([Vibe Writing. You put your ideas into AI. You don't worry ... - Threads](#)). On Hacker News and other forums, some have started using the term “*vibeware*” to describe software produced by vibe coding – often to discuss how maintainable or good it is. There’s a tongue-in-cheek element to this (implying the software is held together by vibes), but it’s entering the jargon. Culturally, this reflects a blending of internet colloquialism (“good vibes”) with technical practice, showing that programming conversations are not immune to the fun and fast-paced nature of internet culture.

Community Projects and Challenges: Embracing the vibe trend, communities have started challenges like “*Vibe Code Weekend*” where participants attempt to build something solely via AI prompting and then share results. These hackathons encourage learning by doing in a low-pressure environment (since everyone expects some messy code – it’s part of the fun). In educational communities, some instructors have organized sessions on vibe coding to introduce students to AI tools. There is also an influx of open source projects labeled as built with vibe coding. For example, Simon Willison open-sourced many of his small “*80 experiments*” ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)), inviting others to see how an AI-written codebase looks. This has sparked constructive discussion in issue trackers and pull requests about how to improve or refactor AI-generated code – a new kind of collaborative learning.

Industry Adoption and Buzz: On the industry side, the fact that Y Combinator’s survey found *one-quarter of new startups* were heavily using AI in development created quite a buzz ([Vibe coding - Wikipedia](#)). It signaled to many that vibe coding (or AI-assisted coding generally) might be the *new normal* for scrappy startups. VC firms have started asking founders how they leverage AI in product development – it’s becoming a point of interest, if not expectation. Business Insider called vibe coding “*Silicon Valley’s latest buzzword*” and indeed, tech meetups and conference talks are now peppered with the phrase ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)). Some engineering teams at larger companies are experimentally adopting vibe coding for internal tooling or prototype features, to evaluate productivity gains. However, culturally, larger organizations are slower to adapt due to the earlier mentioned risks. Still, even seeing formal acknowledgement – like internal blog posts or lunch-and-learns about vibe coding – indicates it has a foothold in the developer zeitgeist.

In the wider online culture, vibe coding has also sparked philosophical debates about the nature of coding and creativity. Is coding still an art if you’re mostly curating AI output? Are we heading towards a world where “*the hottest new programming language is English*” ([Vibe coding - Wikipedia](#)) as Karpathy mused? These discussions, sometimes heated, show that vibe coding sits at the intersection of technological change and cultural shift. It challenges the long-held perception of programming as a meticulous, logical endeavor by introducing a more improvisational style. How the culture ultimately assimilates this – whether it’s seen as a welcome evolution, a passing fad, or a dangerous dilution of software craftsmanship – is an ongoing conversation. For now, the term “*vibe coding*” has firmly planted itself in the lexicon, symbolic of this moment where AI’s role in creativity dramatically expanded. The developer community’s mix of fascination, humor, and critique around it is a healthy sign of grappling with that change.

Comparison to Traditional Programming Methods

Vibe coding represents a significant shift from traditional programming, and it's worth comparing the two approaches, as well as positioning vibe coding among other AI-assisted development methods:

Role of Natural Language vs. Code: The most obvious difference is that traditional programming requires writing code in a formal language (Java, Python, etc.), whereas vibe coding uses *natural language (English)* to instruct the computer. In a classic workflow, a developer must know how to implement a feature (data structures, APIs, syntax). In vibe coding, the developer just needs to express *what* they want, and the AI figures out the *how*. This echoes Karpathy's 2023 tongue-in-cheek claim that "*the hottest new programming language is English*" ([Vibe coding - Wikipedia](#)). Traditional coding is akin to manually crafting a machine – you handle each nut and bolt. Vibe coding is like describing the machine to a very skilled builder and letting them handle the assembly. The upside of the vibe approach is **dramatic speed and lower skill threshold** – you skip a lot of the rote work. The downside is **loss of fine-grained control** – the AI's implementation might not be the exact way you would do it, and it might not be optimal. Classic programming lets you optimize and tailor every detail; vibe coding often yields a more generic solution (which might be fine or might need later tweaking). In essence, vibe coding trades some precision for convenience, whereas traditional coding maximizes precision at the cost of time and required expertise.

Development Workflow and Iteration: Traditional programming follows a cycle of writing code, running/tests, and debugging, with the developer deeply involved at each step. Vibe coding compresses this cycle – you issue a command, the AI writes code, and you run it immediately, often in one interface. The iteration loops are driven by high-level prompts ("fix this error" or "add a login feature") rather than low-level code edits. This can feel more like an interactive dialogue (hence why it's often done in chat form) than the edit-compile-run loop of traditional coding. One could compare it to pair programming: *in vibe coding, the AI is the partner writing the code*, whereas traditionally another human might help or one would do it solo. Interestingly, vibe coding's tight prompt-run loop is somewhat analogous to interpreted languages and REPLs – quick feedback – but taken to an extreme where the "writing" step is fully automated. In contrast, other AI-assisted methods like using Copilot in an IDE still involve the developer writing and understanding most code; Copilot just offers suggestions. ChatGPT usage for coding (outside of vibe methodology) often involves getting a snippet, integrating it, testing it – still somewhat manual. Vibe coding is more *end-to-end* AI generation. So compared to those, vibe coding is a more *holistic* integration of AI into the dev cycle. It's worth noting that when vibe coding, the *testing and debugging* steps remain crucial – perhaps even more so. A responsible vibe coder will test the AI's output frequently, whereas a traditional coder might mentally reason about the code before running it. So testing becomes the main way to verify correctness, since the coder might not have written or even read all the code themselves.

Quality and Reliability: Traditional coding, when done by an experienced programmer, tends to produce reliable, well-structured code (one hopes). Code can be code-reviewed, and the developer can enforce standards as they write. In vibe coding, initial quality depends on the AI's training and the

prompts given. Often the AI writes code that *works* but might not be idiomatic or the cleanest. It might use outdated patterns it learned from older codebases, etc. Over time, we expect AI to produce better and better quality code (some say it's already on par with average StackOverflow solutions). But as of now, a careful dev can usually outdo the AI in elegance and efficiency for complex tasks. That said, for many straightforward tasks, the AI's code quality is sufficient or even excellent (because it's seen the problem many times in its training). A key point of divergence is **consistency** – a human coder can maintain a consistent style and architecture vision; an AI might produce a different style of solution depending on the prompt phrasing or even on each invocation. Traditional programming also allows more intentional design up front – you might sketch a system architecture and then implement step by step. Vibe coding is more opportunistic: you prompt for pieces and maybe refactor after the fact. This can lead to a less coherent architecture if you're not careful. In practice, many vibe coders mitigate this by guiding the AI with additional context or instructions on style (“use functional components”, “follow REST principles”, etc.). It's an emerging art to get AI to produce code not just that works, but that aligns with a desired design.

Learning Curve and Skills Required: Traditionally, becoming a programmer involves learning syntax, algorithms, frameworks, debugging skills, etc., often over years. Vibe coding, in theory, lowers the initial learning curve dramatically – you can *be productive with just natural language and basic logical thinking*. As Simon Willison noted, it “*shaves the initial barrier [to coding] down to almost flat*” for newcomers ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). However, to *excel* with vibe coding, one must develop new skills: prompt engineering, reading AI outputs critically, and knowing when to intervene. It's less about memorizing syntax and more about communicating with the AI and steering it. There's also a skill in decomposing a problem into prompts. In that sense, vibe coding might be easier to start with (you can create something without knowing loops or data types), but it has its own learning curve to master effectively. Traditional coding expertise doesn't become obsolete either – it often underpins successful vibe coding. For example, if you know how to code, you can prompt the AI in a way that avoids pitfalls (e.g., specifying edge cases). If you don't know coding at all, you might accept broken logic because it “looks right.” Thus, many see vibe coding and traditional coding as complementary: experienced coders use vibe coding as a force multiplier, while novices use it as training wheels (with the caveat that they should eventually learn the fundamentals that the AI is handling for them).

Other AI-assisted Methods: Apart from vibe coding, we have things like **Copilot-style autocompletion**, **AI code review assistants**, **stack overflow chatbots**, etc. Vibe coding differs in that it implies the AI is *generating whole blocks or entire programs*, not just finishing your line of code. One could say vibe coding is a subset of AI-assisted programming where the assistance level is maxed out. There's also the concept of **AGI Agents** (like AutoGPT, etc.) that attempt to recursively plan and code with minimal human input. Vibe coding is more interactive and human-guided than that – the human is still in the loop every step, giving feedback and new instructions. Compared to *no-code platforms* (like Bubble or Wix), vibe coding is more flexible – you're still ultimately creating code under the hood, just not by hand. No-code tools constrain you to certain templates or components, whereas vibe coding (with a capable AI) can produce arbitrary code to meet your requests. In a way, vibe coding sits between no-code and traditional coding: you don't write the code, but you're not

limited to a pre-defined palette either – you can ask for anything code can do, and if the AI understands, it will attempt it.

Efficiency vs. Control Trade-off: To summarize the comparison: vibe coding dramatically increases efficiency and approachability of coding by leveraging natural language and AI’s vast knowledge. In exchange, it sacrifices some control, predictability, and at times, code quality. Traditional programming is slower and harder to master, but yields exacting control and deep understanding. Other AI-assisted methods (like using Copilot or ChatGPT as a helper) lie somewhere in between – they speed you up but still require you to write a good chunk of code and make decisions. It’s telling that many developers using vibe coding end up doing a hybrid: they vibe code the boilerplate and core, then manually clean up or optimize parts of it. Or they use traditional coding but switch to vibe mode when they need to quickly generate repetitive code (like dozens of API endpoints). This suggests that, at present, vibe coding is **not a wholesale replacement** for traditional methods, but a powerful addition to the programmer’s toolbox. As AI models get better and tools more integrated, the line might further blur – we may reach a point where what starts as vibe coding seamlessly transitions into maintainable code with AI explaining as it goes, etc. For now, developers often consciously decide when to vibe and when to code by hand, based on the task at hand.

Conclusion

Vibe coding has rapidly gone from a quirky notion in a tweet to a bona fide movement in software development. In essence, it flips the traditional script: instead of humans writing code and maybe asking AI for small suggestions, the AI writes the code and the human guides and verifies. This “describe it, don’t write it” approach has proven to be *incredibly empowering*. It has enabled non-coders to create simple applications just by having a good idea and communicating it clearly. It has made seasoned developers dramatically more productive for certain kinds of tasks – allowing them to focus on the big picture and let the AI handle tedious details. The burgeoning ecosystem of vibe-oriented tools (Cursor, Windsurf, Replit, etc.) and their fast adoption rate in the community signal that this approach addresses a real desire for faster and easier development.

However, as our analysis shows, vibe coding is not a silver bullet, nor is it “programming utopia” just yet. It comes with serious trade-offs around code quality, understandability, and the reliability of outcomes. Essentially, it asks developers to *trust but verify* – trust the AI to do the grunt work, but verify that the results make sense and are safe. For throwaway projects and prototypes, the cost of failure is low and vibe coding’s benefits far outweigh its flaws. But for production software, many experts urge caution: at the end of the day, someone needs to know what the code does. As Simon Willison put it, if you can’t explain the code that was produced, then you’re not really off the hook as a responsible engineer ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)).

In comparing vibe coding to traditional methods, it’s clear that each has its place. Traditional programming isn’t going away – it remains essential for building robust, large-scale systems and for pushing the frontiers of what software can do. But vibe coding is expanding the realm of who can participate in programming and how fast they can iterate. Perhaps the future lies in a **hybrid paradigm**: where AI does the heavy lifting under human supervision, and developers oscillate between

high-level prompting and low-level tweaking. We might also see improved AI tools that address some limitations – for example, an AI that can *explain* its generated code, helping the user learn and trust it, or guardrails that enforce security and style guidelines automatically on AI outputs.

Culturally, the advent of vibe coding is forcing us to reevaluate the definition of programming. Is the core skill of a programmer the ability to write syntax, or the ability to solve problems and define behavior? If it's the latter, then vibe coding is just a natural evolution, allowing programmers to work at a higher level of abstraction (much like compilers and high-level languages did in the past). There's an oft-cited parallel: in the early days, programmers wrote assembly by hand until higher-level languages came and people worried that new programmers wouldn't understand the machine. Yet, we largely moved on, and programming flourished. Vibe coding could be a similar inflection point – today's concerns about lost understanding might be mitigated by tomorrow's best practices and AI improvements.

In closing, *vibe coding* encapsulates the exciting synergy – and tension – between human creativity and artificial intelligence. It has turned coding into a cooperative game of sorts: you bring the ideas and vibes, the AI brings the code. Used wisely, it can feel like having a superpower, turning hours of grinding into minutes of chatting. Used recklessly, it can lead to brittle or baffling results. The best outcomes so far seem to come when developers respect both the *power* and the *limits* of the vibe approach. As the tools mature and the community shares more learnings, we can expect vibe coding to become more reliable and integrate more smoothly with traditional workflows. For now, it remains one of the most intriguing developments in software engineering – a “vibe check” on how ready we are to partner with AI in building the future. As one commentator optimistically remarked: “*Why not let it rip for low-stakes projects? The speed is an order of magnitude faster... For the rest, keep your engineering hat on.*” ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)). That balanced mindset will likely serve us well as we navigate the era of coding on vibes.

Sources:

- Karpathy, A. on vibe coding ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) (original definition and workflow)
- Willison, S. – *Not all AI-assisted programming is vibe coding (but vibe coding rocks)* ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#)) ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#))
- Ars Technica – *Will the future of software development run on vibes?* ([Will the future of software development run on vibes? | CediRates](#)) ([Will the future of software development run on vibes? | CediRates](#))
- Business Insider – “*Silicon Valley's next act: bringing 'vibe coding' to the world*” ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#))

- YC Podcast (Winter 2025) – Partner and founder quotes on vibe coding ([ALL-VIBE-CODING-youtube.txt](#)) ([ALL-VIBE-CODING-youtube.txt](#)) ([ALL-VIBE-CODING-youtube.txt](#))
- Roose, K. – “*Not a Coder? With A.I., Just Having an Idea Can Be Enough*” (NYTimes) ([Vibe coding - Wikipedia](#)) ([Vibe coding - Wikipedia](#))
- Reddit and Twitter discussions (community sentiment and humor) ([Andrej Karpathy on X: "@mattshumer Haha so it's like vibe coding ..."](#)) ([Posts with replies by benyo \(@mattbenyo\) / X](#))
- Cursor and Windsurf documentation (tools features) ([ALL-VIBE-CODING-youtube.txt](#)) ([Windsurf vs. Cursor - which AI coding app is better?](#))
- Various developer accounts and interviews on vibe coding experiences ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) ([Silicon Valley's Next Act: Bringing 'Vibe Coding' to the World - Business Insider](#)) ([Not all AI-assisted programming is vibe coding \(but vibe coding rocks\)](#))).