Why do developers always choose Claude over other AIs?

People use AI programming tools every day. Claude Code is the primary choice. Some have tried alternatives—Gemini, Codex, other open-source models. But they always come back to Claude.

People use AI programming tools every day. Claude Code is the primary choice. Some have tried alternatives—Gemini, Codex, other open-source models. But they always return to Claude. Not out of loyalty, not because of marketing, but because the alternatives disappoint them in the same specific way.

Performance tests don't lie, but they don't tell you what you think.

When a new AI model tops programming performance tests, those tests are usually accurate. The model actually produces better code on individual problems. Higher accuracy on HumanEval. Cleaner solutions on LeetCode-style problems. The numbers are real.

Older performance tests like HumanEval work exactly like that – you're given a single function to write and you get scored based on whether it passes the tests. Newer performance tests like SWE-bench are more realistic. They give the model real GitHub issues from actual repositories and ask it to generate patches. That's closer to the real development process.

But even SWE-bench is still a controlled environment. Real-world programming involves much more. You have to manage the conversation with the user, decide which files to read and which to ignore. You're making targeted edits without breaking surrounding code. You encounter unexpected errors and decide whether to ask for help or try a different approach. You're focused on the task across more than 20 steps without distractions. This kind of interactive, robust workflow is difficult to capture in any performance test.

The gap between workflow processes and raw intelligence.

Anthropic seems to have trained Claude extensively in the programming process, not just the output. A workflow is a sequence of decisions that a truly competent developer makes when given a task within a real codebase.

To be clear – all major programming tools can read files, edit code, and run terminal commands. Codex, Antigravity, Gemini CLI – they all have these capabilities. The difference lies in the consistency with which the underlying model implements the workflow. Read the file correctly before making changes. Make targeted edits

instead of unnecessarily rewriting entire files. Know when to act, when to stop and ask questions to stay focused on the original task instead of getting distracted.

All of these tools can do it, but Claude does it more reliably. Other models produce excellent code—sometimes even better than Claude's code on a piecemeal basis. The gap isn't in any single output product, but in consistency across the entire task. They repeat more often, losing track of what they're doing midway. They make edits that break the surrounding context. They need more guidance to stay on track. Not always—but often enough to change your confidence in the tool's ability to work unsupervised.

The difference isn't pure intelligence, but process discipline. And training yourself in that is harder than most people realize.

What it really takes to be "good at programming"

Creating accurate code probably only accounts for about 40% of what an AI programming assistant needs to do well. The remaining 60% involves everything surrounding the source code:

1. Edit the file without breaking the surrounding code.
2. Read the correct file before making any changes.
3. Complete a multi-step task without getting lost halfway through.
4. Clearly communicate what it is doing and what it has found.
5. Know when to ask instead of assuming.
6. Focus on the task at hand instead of making unnecessary changes to unrelated files.

All major automated programming tools attempt to accomplish all of these things. The question is how many times they succeed at each step throughout a complete task. In my experience using Claude Code daily—building API endpoints, debugging production issues, refactoring components—it consistently achieves these things. Not perfectly, but consistently enough that you don't feel the need to track every single step.

With other tools, you have to intervene more. The code they generate is often good, but somewhere in the middle of a multi-file task, something goes wrong—a file gets partially overwritten, or the model goes off track and starts "improving" something you didn't ask for. That's the gap, not the capability. What matters is how many times the tool remains stable without you having to readjust it.

Why is Google having structural problems here?

Gemini writes very good code. The underlying model is clearly very capable. Just give it a clearly defined problem with specific specifications, and it will produce a good, sometimes excellent, solution.

The problem seems structural. Google is essentially a search and general-purpose company. Their models are optimized across a vast range of tasks—translation, summarization, multimodal comprehension, general conversation. Agentic software development, on the other hand, is a narrow, specific workflow requiring dedicated, focused training.

Training for agentic workflows means the model needs to successfully complete long sequences of tool calls, skillfully recover from mid-process errors, and maintain context across multiple steps without deviation. This requires reinforcement learning focused on that specific scenario, rather than simply extending the underlying

model.

Anthropic published research on agent autonomy showing that software engineering accounts for nearly 50% of their total agentic activity on their API. Half of their agentic use is programming. Then you have to train it, optimize tool usage, edit files, multi-step workflows – because that's what your paying users are actually doing. Google doesn't face the same pressure. Their model serves search, translation, multimodal tasks, and general chat. Programming is just one use case among dozens. The success or failure of Anthropic's model depends on its programmability.

The actual situation

Here's an honest assessment from my perspective as someone who uses these tools in their daily work:

Claude is the primary tool. Claude Code handles everything from structuring new features to debugging complex production issues. The workflow is reliable enough that you can trust it for tasks you don't want to personally oversee.

The Codex has improved significantly in agentic tasks. The gap has narrowed more than expected in the last few months. It's not as reliable as Claude yet, but it's worth watching.

Gemini is capable of performing individual tasks. It generates truly impressive code for many well-defined problems. However, as a system operating independently on multi-step tasks, it still struggles. Loops, crashes, the need for constant re-navigation – these are real, frequent errors that people encounter.

Many people have tried the "plan in one model, execute in another" approach. They use Gemini for architectural thinking, then switch to Claude for the actual work. In reality, this increases complexity without creating value. It's better to use Claude for the entire process.

What does this mean for the future?

The leading models will constantly change. A new model will top the charts. Developers will experiment with them. Some will switch, but most will return.

The gap will narrow. Google has the resources to address the process discipline issue if they decide it's a top priority. OpenAI is clearly taking agent-based workflows seriously with the Codex. The advantage Claude has today is not permanent.

But what Anthropic discovered—training for workflow, not just output—is a significant insight. Other labs will have to explicitly replicate that focus to close the gap. Large models alone won't do that. You can have the smartest model in the world, but that's meaningless if it can't edit one file without corrupting the next.

Standards will tell you one thing. Developers who use these tools daily will tell you something else. Generally, you should listen to the developers.