
FIELD REPORT NO. 04 · REDLINE ADVISORS

The box *score*

*Seven months of building with Claude, scored the way a
lifelong baseball fan keeps score.*

AUTHOR

Mick Hollison
Founder & CEO

PUBLISHED

October 2026
Park City, Utah

DATASET

211 days, 5 codebases
Mar 4 to Sep 30, 2026

SERIES

No. 01 Defying the
Algorithm
No. 02 Coding at the
Speed of Thought
No. 03 The Sound of
Sameness

My given name is Michael. My dad nicknamed me Mickey the day I was born, after his favorite ballplayer, Mickey Mantle, and it stuck. Anyone who knew me growing up still calls me Mickey. Everyone else knows me as Mick.

My dad is an even bigger stats guy than I am. He worked in computers for Western Electric and AT&T back when a computer filled an entire room and ran on punch cards. He had a high school education, and he taught himself Fortran and COBOL. His mind is built for calculating and measuring things, which is probably also why he is an expert craftsman, a gift I sadly did not inherit. Long before I wrote a line of code, he was the programmer in our family.

So I was nicknamed for a Yankee. But I was a kid from Atlanta, and my player was Dale Murphy, whom I got to watch nearly every night on TBS Superstation.

I played the game, too. I was the shortstop and captain of my high school team, though I was probably a more gifted center fielder and pitcher, and I kept playing in a very competitive men's league in Atlanta until my early thirties. I could handle just about any fastball. The curveball was my nemesis.

What stayed with me longest, though, was the numbers. No sport on earth treats statistics the way baseball does. I collected baseball cards as much for the backs as the fronts, and I can still tell you exactly how Ronald Acuña, Matt Olson and Michael Harris are hitting this season. I can also tell you that my Braves won fourteen straight division titles, from 1991 to 2005, still the longest run in major league history. Every great stat has an asterisk, and that one's is the strike-shortened 1994 season, which crowned no division champions, so the streak steps right over it.

Somewhere along the way I became a person who keeps score. Of ballgames. Of sales revenue, which is the best scoreboard there is. And lately, of commits in a repository and tokens exchanged with an AI.

In the spring I published a field report called *Coding at the Speed of Thought*. It covered forty-five days, from March 4 to April 20, in which I went from a former software executive who had not written code in thirty years to shipping three production codebases with Claude, Anthropic's coding agent, as my partner.

The most common response was some version of the same question, asked politely. *Sure. But will it last?*

It was a fair question. Forty-five days is a honeymoon. Anyone can sprint. I have watched plenty of executives fall in love with a tool in March and forget its password by June. I have been one of them.

So I did what I have always done. I kept score. This is the follow-up: two hundred and eleven days of data, measured the same way as the first report, with the method shown. It includes the parts that went well, the parts that did not, and one thing I nearly lost along the way.

The numbers

Seven months, the same scorebook. Each figure is set against the forty-five days in Field Report No. 02.

211 CALENDAR DAYS, VS. 45	5 CODEBASES, VS. 3
860 GIT COMMITS, VS. 213	75,203 NET LINES OF CODE STILL STANDING, VS. 25,747
4.0B+ TOKENS EXCHANGED WITH CLAUDE, VS. 1.44B	84+ ACTIVE HOURS WITH CLAUDE, VS. 21.6

860 git commits across five codebases: my firm's site, our internal CRM, two client sites, and one personal project. **75,203 net lines of code**, where *net* means the same thing it meant last time: lines I wrote with Claude that are still in the codebase today, not lines typed and later thrown away. **At least four billion tokens** of context exchanged. **At least eighty-four hours** of active, back-and-forth session time.

Two notes on method, because I would want them if I were reading this.

Active hours is strict on purpose. I count only the stretches where something happened in a session at least every five minutes. Thinking time, reading time, and the hours I spent staring at a page in a browser deciding whether it was right do not count. Eighty-four hours is about two working weeks, spread across seven months of running a consulting firm.

The words *at least* are honest, not modest. Claude Code keeps its session history on the machine where the work happened. The history for March and April no longer exists on either of my computers. More on that below, because it turned out to be the most useful lesson in this report. For those two months I use the figures I published at the time. For the gap between April 20 and early May I have nothing, so it counts as zero.

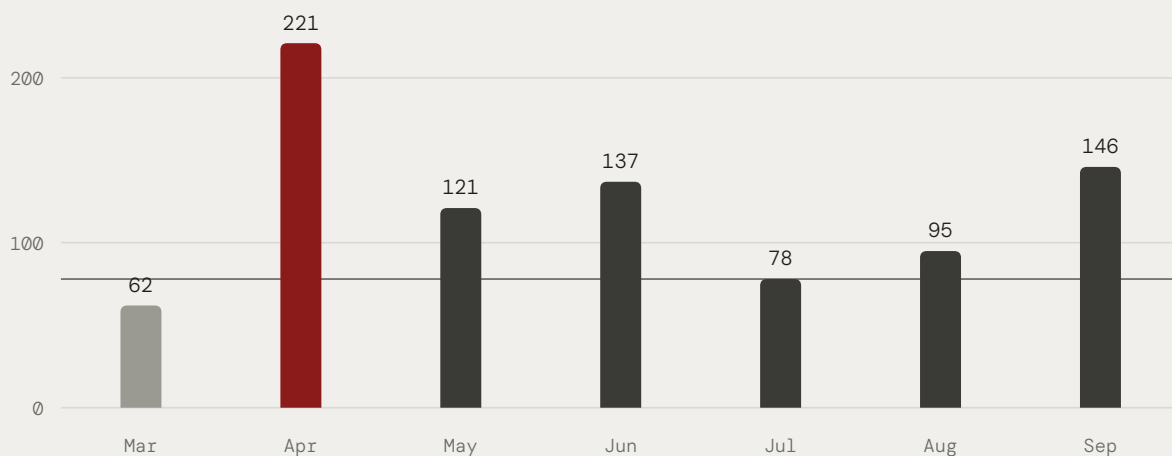
The pace held

The honeymoon became the habit.

Here is the number that answers the question people asked. During the forty-five days in the first report, I averaged a little over four commits a day. Over the five months since, I have averaged about four a day. Since the sprint ended, no full month has dropped below 78 commits.

COMMITTS PER MONTH, 2026

The sprint, then the season



■ Report 02's 45-day sprint — 78: every full month since
■ March: partial month

That does not mean every month looked the same. The sprint was concentrated: three codebases, all of them new, built in a burst. The months since have been spread across more work. A second client site joined the list. So did a website for the wedding of my oldest son, Connor, to Maria. It is the least commercial thing I have shipped this year and, given the client, easily the most important one to get right.

The pattern I see in the commit log is not a sprint followed by a fade. It is a sprint followed by a working rhythm. Some weeks are heavy and some are nearly empty, because some weeks are full of client work. The tool did not become a hobby. It became how I work.

03

The code that stayed

The fair critique of AI-written code is that it is disposable: fast to produce, quick to rot, rewritten before it earns its keep. I wanted to test that against my own work, so I asked a specific question of the code. Of the 25,747 lines I counted at the end of the first report, how many are still in the codebases today, untouched?

About 18,800. Roughly seventy-three percent of the spring code is still standing.

The rest were rewritten, and most of those rewrites were deliberate. Pages got redesigned. A scoring model got rebuilt when I learned it was rewarding the wrong things. A framework upgrade replaced pieces of plumbing. That is what healthy software does. But the core of what I built in those first six weeks is still the core of what runs today. It was not scaffolding. It was foundation.

04

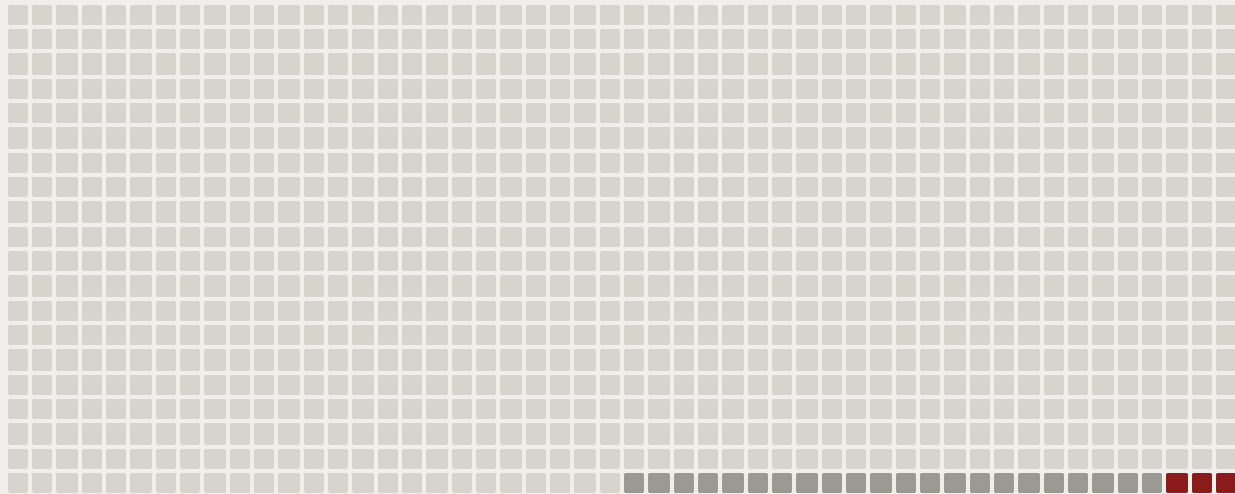
What four billion tokens actually means

Big token numbers sound impressive, and I put one on the cover of the last report. This time I looked underneath it, and what I found changed how I think about the number.

More than ninety-seven percent of those tokens are Claude *rereading* context it has already seen: the code, the conversation so far, the rules of the project. Only about three tokens in every thousand are Claude writing something new. Every time I ask a question in a long session, the entire working context comes along with it, and Claude reads it again before it answers.

EVERY 1,000 TOKENS, BY JOB

Three of them are writing



- 3 Claude writing something new
- 22 Saving context to reuse
- 975 Rereading context it already has

Most of it is memory, not writing.

That is not waste. It is what makes the work coherent. A partner who remembers the whole project every time you speak is a different kind of partner from one you have to re-brief. But it tells you where the cost of this work really sits. It sits in context, not in output. That matters for anyone thinking about the consumption pricing I wrote about last time. You will pay to keep your partner informed, and that cost is worth managing on purpose.

For a sense of scale: Anthropic publishes the typical daily usage of a developer on Claude Code. On the days I worked, my usage ran a little over three times that average, and above where nine in ten developers land. I am not a developer. That is rather the point.

— 05

The second act is operating

Building is the fun part. Owning is the job.

In the first report I made an argument that annoyed some people in the software industry. I said a small firm could now build the custom tools it used to rent, and own them outright. The best objection I heard was this: *building is easy, wait until you have to maintain it.*

Seven months in, I can answer that with data instead of opinion.

The work did change character. The spring was almost all building. By late summer, a growing share of my time went to operating what I had built. I ran a full security review of the site and fixed what it found. I rebuilt how the site asks visitors for consent, so it treats people by region and fails safe when it cannot tell. When someone attempted an invoice-fraud scam through our contact form, I built a screen that reads what a submission says and stops that pattern without blocking real customers. I upgraded the whole site to a new major version of its framework. And I built secure online questionnaires inside our client vault, so a client can work through a scoping document with us instead of trading attachments by email.

The maintenance tax is real. By my rough count, about one commit in five over the seven months was maintenance: fixes, security, upgrades, hardening. In September, during a full audit, it was about one in three.

But here is what the objection misses. The maintenance was done the same way the building was. Same partner, same speed, same ownership. When a vendor's product needs a security fix, you wait for the vendor. When mine needs one, I describe the problem and it is fixed that afternoon. Owning your software used to mean owning a maintenance burden you could not carry. That is no longer the trade.

06

The factory worked

What I built for myself became what I build for clients.

The first report ended with a claim I could not yet prove. I said that each deliverable, done right once, becomes a template, and each template becomes a reusable unit of service.

It held up. The way I sized the RedLine site for real laptop screens became a written playbook, and it went to a client. The whole-site audit I ran in September became a portable playbook for client sites, framed so that my conclusions are treated as hypotheses to verify on their site, not rules to impose. The hard-won setup for contact forms and conversion tracking became a

playbook too. When Rocket Partners, the firm that built CodeVine, asked for help with their own site, I went from a standing start to eighty-five commits in September, largely because so much of the method already existed.

None of that is code you can see. It is the factory. It is the part of the first report I believed most and could show least, and it is now the part I would defend hardest.

— 07

The thing I could not build my way out of

Code turned out to be the fastball. Attention was the curveball.

One area of the last seven months did not go well, and leaving it out would make this report less honest.

RedLine's whole thesis is that the first surface that matters now is the AI answer a buyer reads, not a page of search results. So I tested the opposite on my own firm: a deliberately small Google Ads and LinkedIn Ads experiment, run properly, for most of a year. I spent a meaningful share of the spring and summer making it work. Analytics, ad conversion tracking, consent rules, the tracking tags each platform requires, the security settings each tag needs in order to load at all. Claude and I built every piece of it, and rebuilt some of it more than once as we learned what each platform actually required. The software side was solvable.

The problem was the ad platforms. To be clear, that is not Google or LinkedIn as a whole. My firm runs on Google's tools, and posting on LinkedIn has worked well for me and for my clients; even the simple "boost" button has drawn real attention. The campaign machinery was the problem. Defaults that quietly widened where my money went. Interfaces that changed under me or stopped rendering mid-setup. A verification step I could not complete on demand. And in the end, no lead I could trace to a single paid click. The experiment came out the way the thesis predicted. What I did not predict was how hard it would be to set up correctly, for a firm that builds its own software.

I have written about that separately, because it deserves its own space. The lesson for this report is narrower. AI removed the cost of building almost anything I control. It did nothing about the cost of renting attention from systems I do not control. If anything, the contrast

got sharper.

08

I almost lost the evidence

When I sat down to reproduce the numbers in the first report, I discovered something I should have known. The detailed history of my work with Claude lives in local files on whichever machine did the work, and I did most of the spring work on a different computer from the one I use today. Claude Code also cleans up old session history on a schedule by default.

The history from March and April is gone from both machines. Git remembered every commit and every line. But the record of *how* the work happened, every question asked, every correction made, every dead end abandoned, is gone for those first two months.

In the first report I argued that the most valuable thing a company's best people produce is not their output but their method, and that organizations which fail to capture it will watch it walk out the door. I meant it about engineers at large companies. It turns out I needed to hear it myself, about a two-computer, one-person firm.

So, three pieces of practical advice for anyone doing this work.

Decide where your history lives, on purpose. If you work on more than one machine, know which one holds the record. If the history matters to you, change the cleanup setting.

Write down your method while it is fresh. The playbooks in section 06 survived because they were written down. The session history did not, because I assumed it would always be there.

Count the same way every time. Baseball has an official scorer for a reason: a hit is a hit only if everyone counts it the same way. I could compare seven months to forty-five days only because the method could be reconstructed from data that survived. Write the method down next to the number.

The long season

The question was whether it would last. A forty-five-day sprint is a hot April, and anybody can hit .400 in April. The season is the test. Seven months in, the answer is yes. Not as a burst of enthusiasm, but as a way of working: steady, a little less glamorous than the first six weeks, and far more useful.

The surprise was not the volume. It was what the volume turned into. A firm with its own software, maintained the same afternoon it breaks. A set of methods that travel from my work to my clients' work. And a clearer view of where the hard problems in a small business actually are now. They are not in the software anymore. They are in attention, trust, and measurement, which happen to be the problems RedLine exists to solve.

Old dogs can absolutely learn this trick. Seven months later, this one is still learning it every day, and still keeping score.

SOURCES AND NOTES

1. **Window.** March 4 to September 30, 2026, counting both ends: 211 calendar days. Field Report No. 02 covered March 4 to April 20.
2. **Codebases and commits.** Five repositories: the RedLine Advisors site, the Signal Engine CRM, the CodeVine and Rocket Partners sites, and a family wedding site. Commits are those I authored on any branch still on GitHub, deduplicated. Counted the same way, the original window reproduces within about three percent of the published figure; the gap is consistent with branches deleted after being merged.
3. **Net lines still standing.** Every line in each repository today, attributed with *git blame*, keeping text lines I authored on or after March 4 and excluding package lock files. The same method applied to the site's April 20 snapshot gives 13,837 lines, matching No. 02's "roughly fourteen thousand." The seventy-three percent survival figure in section 03 is the lines authored March 4 to April 20 that remain today, set against the published 25,747.

4. **Tokens.** Input, output, cache-write and cache-read tokens from Claude Code's local session logs, each request counted once, summed across both machines. March and April use the figures published in No. 02; April 21 to May 4 is unlogged and counted as zero, so the total is a floor. The shares in section 04 are computed on the 2.56 billion tokens in the surviving logs; March and April were published only as a total.
5. **Active hours.** The sum of gaps of five minutes or less between consecutive session events. A ten-minute threshold adds roughly a fifth. Same floor caveat as tokens.
6. **Usage benchmark.** Anthropic's published Claude Code cost guidance gives the average usage per developer per active day across enterprise deployments, and a level that nine in ten users stay under. That level works out to roughly two and a third times the average. My usage is compared at list-price equivalent across my logged active days. I work on a subscription, so this is a measure of volume, not of what I paid.
7. **Maintenance share.** Commits whose messages describe fixes, security, upgrades, hardening, accessibility or dependency work. Classified by message, so treat it as an estimate.
8. **Paid advertising experiment.** Google Ads, November 2025 to September 2026, and LinkedIn Ads, June 26 to July 7, 2026. Figures come from each platform's own statements and exports. Until September 2026 the site's consent settings meant the platforms saw only visitors who opted in, so platform-reported conversions were undercounted by design.
9. **Baseball.** The Atlanta Braves won fourteen consecutive division titles across 1991 to 2005; no division champions were named in the strike-shortened 1994 season.

Disclosure. This report is about my own use of Claude, Anthropic's product. I pay for it and have no commercial relationship with Anthropic beyond being a customer.

RedLine Advisors · Park City, Utah · contact@redlinecs.ai · redlinecs.ai