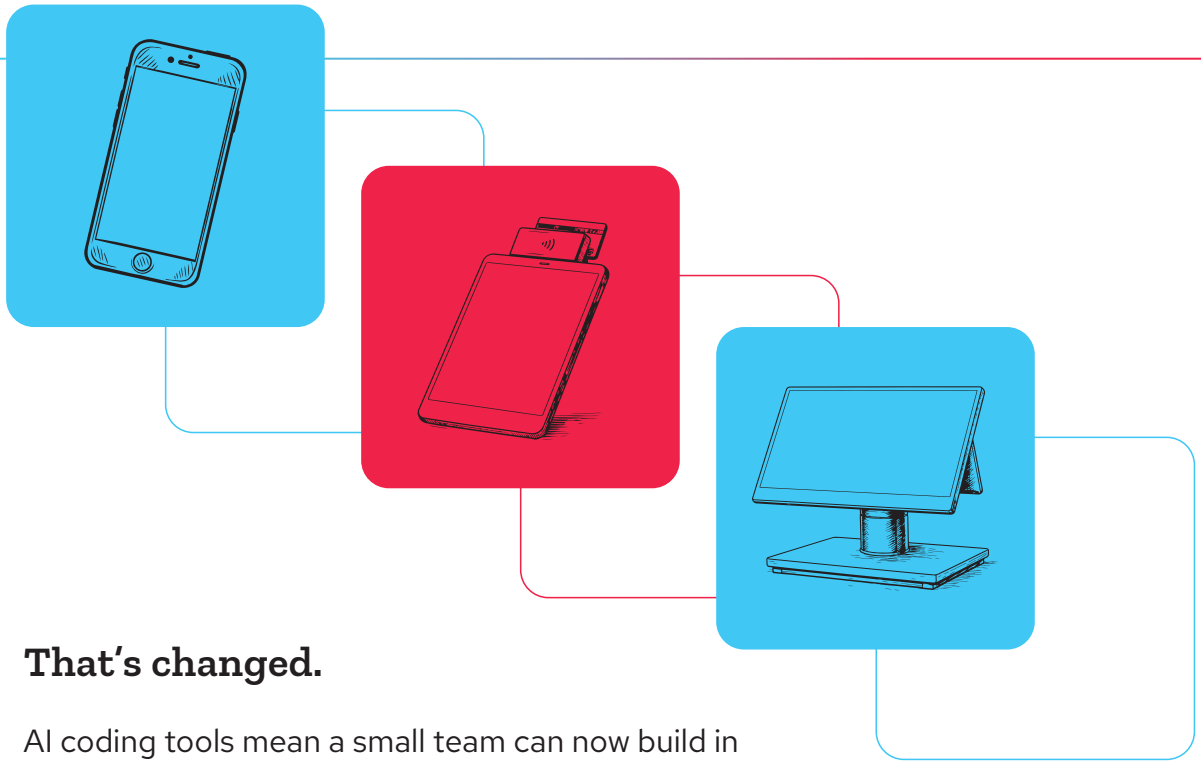
Build, Buy, *or* Both?

A Restaurant Leader's Guide to Technology Decisions in the Age of AI

READ TIME
12 MINUTES

INTRODUCTION

A few years ago, building your own restaurant technology was a real undertaking. You needed a serious engineering team, a real budget, and months of runway. Most brands took one look at the cost and bought something off the shelf instead.



That's changed.

AI coding tools mean a small team can now build in weeks what used to take months. A technically curious engineer can describe what they want and have a working prototype within days.

The entry cost for custom software has dropped dramatically, and for the first time, 'build it ourselves' is a question most restaurant brands can seriously consider.

But here's what the tools don't change: what happens after you ship. Building got cheaper. Owning what you built did not.

This guide is for restaurant leaders navigating the buying decision and seeking clarity in today's environment. It covers what custom software actually costs over time, where buying makes more sense than it might look, and what a smart combination of both looks like in practice.

01

Building Got Cheap. Owning Didn't.

The numbers on AI and software development are hard to ignore right now:

72%

of developers use AI coding tools every day

41%

of all code being written today is AI-generated

Senior developers report getting 81% more done with AI assistance. Pull requests are up 20% across engineering teams. Non-technical people are shipping functional prototypes in days.

For restaurant brands, this means something concrete: a CTO with a team of two or three engineers can now seriously consider building a custom loyalty platform, a voice AI ordering system, or a kitchen analytics dashboard. That wasn't realistic a few years ago unless you had the engineering headcount of a Starbucks or McDonald's.

So 'build' has become a real option for more brands. That's good. The trap is assuming that because building got easier, owning what you build got easier, too.

Building got faster. The maintenance burden didn't go away; it got *bigger*.

The build is the easy part to budget. What comes after it is where most brands get surprised.

02

When Building Makes Sense

Building isn't always the wrong call. There are times when it creates a real, lasting advantage. The key is being honest about your own situation before you commit.

Build

You need something that doesn't exist.

If every vendor in a category offers roughly the same thing, buying one puts you at parity. If what you need genuinely isn't available, or if building it creates data or workflows that are hard for competitors to replicate, building can be justified.

Build

You have the team to keep it running.

Building is not a project with an end date. It's an ongoing commitment. Before you start, answer honestly: do we have, or can we hire and keep, the engineers needed to maintain this system for the next three to five years? If that's uncertain, the case for building has a hole.

Build

The data advantage is real.

Some of the most defensible restaurant technology builds aren't about the application itself; they're about the underlying data layer. Proprietary guest models, operational data that no vendor can access, integrations that give you a unified view nobody else has. Building at the data layer can make sense even when building the application on top of it doesn't.

Build

A commercial platform genuinely can't serve your operation.

Some restaurant formats are unusual enough, franchise configurations, menu complexity, and service models, that existing platforms handle them badly. When the customization required to make a commercial platform work approaches the cost of building it, the calculus shifts.

03

The Real Cost of Building: What Happens After You Ship

Here's the number most build/buy analyses leave out: annual maintenance on custom software runs 15–25% of the original build cost, every year, indefinitely.

Build a platform for $300,000. You're committing to $45,000–$75,000 per year just to keep it running. That's before adding a single new feature, before any model retraining, before your AI vendor changes their API, and something breaks.

Over the full life cycle of a software system, maintenance typically accounts for 2 to 4 times the cost of the original build. Between 50% and 80% of what you spend on software over its lifetime is maintenance, not development. *The build is the down payment. Maintenance is the mortgage.*

A $300K platform maintained for five years doesn't cost $300K. It costs $600K to $1.2M. Most technology budgets don't reflect this.

AI-generated code makes maintenance *harder*, not easier.

45%

of AI-generated code contains security vulnerabilities that need ongoing patching

There's a common assumption that if AI makes building easier, it probably makes maintaining easier, too. The data doesn't support that.

- AI-generated code is 1.7x more likely to have significant logic errors than code written by hand
- It's 2.74x more likely to have security vulnerabilities
- Developers already spend 42% of their time on maintenance and technical debt — that number grows as AI-generated code accumulates
- When AI-generated code isn't actively managed, maintenance costs reach 4x normal levels by year two
- 75% of technology leaders are projected to face serious technical debt by 2026 — most of it from moving fast with AI tools

The engineering team at CodeRabbit put it plainly: '2025 was the year of AI speed. 2026 will be the year of AI quality.' The speed arrived first. The cleanup is happening now.

AI can help you build faster.
It can also help you *build into a corner* faster.

The People Problem

Custom software isn't just maintained by tools. It's maintained by people. When the engineer who built your custom ordering system leaves, their replacement needs weeks or months to understand the codebase well enough to make changes without breaking anything.

Every departure is a hidden maintenance cost. Every new hire is an onboarding cost. For restaurant brands without a deep engineering bench, this creates a dependency that's hard to see on paper and painful to discover in production.

The restaurant industry spent a decade building its own tech and calling it a competitive advantage. For most brands, it's now a liability. It's expensive to update, falling behind platform vendors, and losing the confidence of their franchisee base.

One analyst predicts that before the end of 2026, a restaurant chain with more than 200 locations will announce the dismantling of its owned technology stack. The stated reason is that their AI strategy can't run on systems built in 2017.

04

The Buy Trap: Buying Has Its Own Problems

None of this means buying is automatically the right answer. There are traps on the buy side that restaurants regularly run into.

Too Many Vendors

Most restaurant technology stacks have grown into collections of point solutions: POS, online ordering, loyalty, kitchen display, labor management, delivery, analytics, and AI. Each vendor has its own contract, support team, and integration requirements.

Buying five vendors doesn't cost five subscriptions. It costs five subscriptions plus the engineering time to connect them, the operational risk when one changes its API, and the management overhead of five renewal cycles.

Prices Go Up

Major software vendors spent 2025 and 2026 buying up competitors, killing lower-priced tiers, and pushing customers toward enterprise contracts. AI feature surcharges have pushed SaaS costs up by an average of 34% since 2024. When your operations depend on a single vendor, your leverage at renewal is weaker than you think.

> **Most companies underestimate what AI software actually costs. A majority are off by more than 10%. Nearly a quarter are off by 50% or more. The real costs show up after launch: maintenance, integration work, model updates, and compliance, not in the original contract.**

AI Features Bolted Onto Old Systems

Many established restaurant technology vendors are adding AI features to systems that weren't designed for them. The demos look great. In production, these bolt-ons often can't access the real-time data they need to do anything useful.

Before buying AI capabilities from any vendor, the question to ask is architectural: was this platform built to handle real-time AI workloads, or is AI a layer sitting on top of a transaction system that wasn't designed for it?

05

When Buying Makes Sense

For most of the restaurant technology stack, buying from a well-designed commercial platform is the smarter use of engineering resources and capital. Here's why:

Buy

Buy what's ready-made, build what sets you apart.

Your POS processes transactions. Your kitchen display routes orders. Your loyalty program tracks points. Every brand in your competitive set has all of this. Building these things doesn't make the food better or the service faster; it just ties up engineering capacity on work that isn't unique to you.

Buy

The system needs to be bulletproof.

A custom-built system owned by a small internal team can't match the reliability of a commercial platform with a dedicated operations organization and around-the-clock support. For anything that touches every transaction in every location, the cost of an outage usually exceeds the cost of the vendor relationship.

Buy

The platform improves faster than you can.

A commercial platform serving dozens of enterprise restaurant brands is continuously improving, with new features, model updates, security patches, and integrations. Your internal build improves at whatever pace your engineering team can manage alongside every other priority.

Buy

The right integrations already exist.

The most underappreciated value in a commercial platform is the integration library it comes with. Every vendor connection that already exists is weeks or months of engineering work you don't have to do or maintain.

06

The Both Answer

The best restaurant technology strategies aren't purely build or buy. It's a deliberate approach: buy the platform layer that handles infrastructure, build the differentiation layer on top of it.

AI is where the decision gets expensive the fastest, right now.

One major AI provider reportedly increased its revenue outlook from under $10B to $30–$45B in a matter of months; driven partly by repricing, not just new customers.

Since it isn't all demand, *it's a warning*: the economics underneath today's AI strategy can change quickly.

The equation has become: **Usage x Restaurant-Scale Volume x Increasing Vendor Costs.**

> **The "both" answer for AI isn't about choosing edge over cloud. It's about knowing *which* workloads belong *where* — and having a platform smart enough to make that call.**

What Runs Where

RUN AT THE EDGE

Real-time, high-frequency decisions that use local data:

- Drive-thru voice AI
- Kitchen vision models
- Order sequencing
- In-store orchestration

Faster response. Local resilience. No compounding per-call cloud cost.

RUN IN THE CLOUD

Compute-intensive intelligence that draws on data across locations:

- Complex reasoning
- Personalization models
- Enterprise analytics
- Model training & improvement

Greater compute power and access to enterprise-wide data.

The goal isn't to avoid the cloud. It's to use it where it creates the most value.

The Intelligent Commerce Platform is the orchestration layer that makes this decision in real time. It's neither an edge-first nor a cloud-first system. It routes each workload to where it runs best, which means restaurants get the performance of local processing when it matters and the capability of cloud AI where it's worth paying for.

This architecture works because Qu originally built it for a different problem: connectivity resilience. Seven-plus years ago, we built the In-store Cloud. That same foundation now gives restaurants a smarter way to deploy AI, without sacrificing performance or creating unpredictable infrastructure costs.

When every AI interaction carries a cost, cloud-only architecture leaves restaurants with fewer levers to pull. Running each workload where it performs best creates a new one.

07

10 Questions to Ask Before You Decide

These are the questions that surface what a technology decision actually costs and whether you're set up to make it work.

IF YOU'RE THINKING ABOUT BUILDING

01 What does this cost over 5 years, not just year one?

Build cost plus 15–25% annually for maintenance, security, model updates, and talent. Run the number before you commit.

02 Do we have the team to maintain this — not just build it?

Building is a sprint. Owning is a multi-year organizational commitment.

03 What happens when the engineer who built it leaves?

Custom codebases carry talent concentration risk that compounds over time.

04 How much of this will be AI-generated code, and what's the plan for managing it?

AI-generated code introduces more errors and vulnerabilities, and drives maintenance costs to 4x normal levels by year two if left unmanaged.

05 Does this create real competitive advantage, or are we rebuilding something a vendor already does?

An honest answer to this one changes the whole case.

IF YOU'RE THINKING ABOUT BUYING

06 Was this platform built for AI workloads, or is AI bolted on?

Ask about the underlying architecture. Bolt-ons perform in demos and underperform in production.

07 What does this actually cost, including integration work and price increases at renewal?

Most vendors raised prices 15–25% in the past year. Model that into your TCO, not just the current contract.

08 What's our exit path if this relationship stops working?

Mission-critical dependency without a realistic exit weakens your position at every renewal.

09 What data do we own, and what stays with the vendor if we leave?

Get data portability established in the contract. Don't try to negotiate it during a transition.

IF YOU'RE THINKING ABOUT BOTH

10 Is this platform actually designed to be extended?

Open APIs, documented integration points, and a developer ecosystem. Or a closed system that handles what it handles today and requires the vendor to build anything new.

The decision hasn't changed at its core: build, buy, or both is really a question of where you're putting your engineering time, capital, and organizational focus. AI has made one part of that decision, the initial build, faster and cheaper. It hasn't made the rest of it easier.

The restaurant brands that end up in the best position aren't the ones that defaulted to build because it felt like control, or defaulted to buy because it felt like simplicity. They're the ones that asked the right questions before committing and picked a platform that was actually designed to support the strategy they chose.

WANT TO SEE HOW QU'S INTELLIGENT COMMERCE PLATFORM IS BUILT FOR THE 'BOTH' APPROACH?

qubeyond.com/demo