PREDICT AP

Your Builders Are Already Building

A CRE Leader's Guide to Build vs. Buy in the AI Era

WHY THIS GUIDE

Whether you are a multifamily manager trying to improve leasing operations, a REIT working to deliver investor reports faster, a third-party manager trying to standardize workflows across clients, or a retail owner trying to take the pain out of monthly CAM reconciliations, technology is no longer a back-office consideration in commercial real estate.

It is central to how the work gets done.
The form that technology takes, however, is in flux.

AI has changed the build vs. buy conversation. It has made starting easier. It has made demos more impressive. It has given people close to the work new power to automate, analyze, and improve their own workflows. It has not, however, eliminated the cost of ownership.

For real estate companies of all kinds, the build vs. buy debate shapes operational efficiency, data security, competitive positioning, and long-term cost structure. Get it right and the benefits compound. Get it wrong and it quietly erodes margins, creates organizational risk, and leaves teams buried in workarounds that no one has time to fix.

This guide is about making that distinction practical.

Who this is for: CRE leaders who own budget, risk, operations, finance, innovation, or technology strategy. That includes CFOs, COOs, CIOs, CTOs, heads of finance, innovation leaders, operators, and the tech-forward builders who see friction first and want to move the business forward.

The Moment We're In

AI has moved fast enough that keeping up can feel less like a strategy and more like a sprint. The question is no longer simply whether technology can help. The question is when to build the tools your business needs, when to buy them, and how to govern both.

AI access is spreading faster than most governance models can absorb. In CRE operations, that can mean analysts automating reports with tools IT has never reviewed, property teams summarizing policies and vendor communications, finance teams testing document extraction, and ad hoc solutions quietly becoming part of the operating fabric.

IBM's 2025 Cost of a Data Breach research shows why the governance layer matters. IBM reported that 13% of organizations experienced breaches of AI models or applications, and that 97% of those organizations lacked proper AI access controls. That is the statistic to keep in mind when an internal tool connects to tenant data, investor distributions, lender covenants, or general ledger activity.

The vendor landscape has also matured. Purpose-built platforms for lease abstraction, investor reporting, compliance workflows, AP automation, portfolio analytics, and other core real estate operations now carry years of compounding expertise. Edge cases have been handled, audit trails built, integrations hardened, and security programs matured. A capable internal team starting from scratch should be honest about what it would take to replicate that learning curve.

Most organizations are making their build or buy decisions without a clear framework. In the AI era, the cost of that gap is rising quickly.

What's Inside

THE GOLD RUSH: WHY EVERYONE IS BUILDING NOW

There is a particular kind of energy that moves through an industry when a new technology arrives.

You can hear it in conference sessions, boardrooms, Slack channels, and hallway conversations. Someone shares a screen, pastes something into an AI tool, and gets back a result that would have taken a team weeks to produce two years ago. Then someone says it:

Why are we paying for that?
We could just build it.

Sometimes they are right. The mistake is assuming they are always right.

The story of business technology democratization has played out in waves. First came spreadsheets, which gave anyone who could navigate a grid the ability to build valuation models, rent trackers, forecasting tools, and reporting templates. That created enormous productivity. It also spawned a generation of fragile, undocumented infrastructure that organizations are still untangling today.

Then came low-code and no-code platforms, promising automation without developers. They delivered real efficiency gains, but they also created workflows that touched sensitive data without proper governance and broke quietly when upstream systems changed.

Generative AI is the next wave, but it is different in speed and reach. The translation layer that once sat between a business problem and a technical solution has been compressed. People can now describe what they want in plain language and get something back almost instantly.

The distance between having a problem and having a prototype has collapsed.

Microsoft and LinkedIn's 2024 Work Trend Index found that 75% of knowledge workers use generative AI at work and that 78% of AI users bring their own AI tools to work. That is the reality behind the gold rush.

IBM's 2025 research adds the other side of the story. Shadow AI was a factor in 20 percent of breaches IBM studied, adding roughly $670,000 to average breach costs, and 63 percent of breached organizations either did not have AI governance policies or were still developing them. The lesson is not that builders are the problem. The lesson is that builder energy without access controls, approved tools, and governance becomes expensive very quickly.

That capability collapse is genuinely new. It means this wave is moving faster, reaching further, and creating more unsanctioned infrastructure more quickly than anything that came before it.

This is the gold rush. Like every gold rush before it, it contains both genuine opportunity and the seeds of serious trouble.

Embrace the Builders

Every organization has builders.

They are the analysts that built the reporting template that everyone on the team now uses. The property managers that have figured out how to stitch together data from two systems in a way that saves hours every month-end. The asset managers that automated the weekly variance report because they were tired of doing it by hand.

They are the people who see a broken process and care enough to fix it. Those people are the gold nuggets of the gold rush. They understand the business. They are frustrated by friction. They want the operation to be better.

This is especially important in CRE. The people closest to the work know where the process breaks. They know which reports no one trusts, which workflows require manual cleanup, which approvals stall, which exports need reformatting, and which repetitive tasks are quietly burning hours every week.

Suppressing this instinct does not make the problems go away. It makes the problems invisible and the people who want to solve them frustrated. If companies only respond with hard stops, two things happen. They discourage the exact people who are trying to improve the business, and they push the work underground.

Problem solvers are going to solve problems. The question is whether they are doing it in the open, with the right support and guardrails, or in the dark, without visibility.

That is why the CIO and CTO role is adapting from gatekeeper to shepherd. This is not anti-IT. It is the opposite. AI makes technology leadership more important because building now happens everywhere. The technology leader's job becomes setting the operating model that lets builders move quickly without putting the enterprise at risk. The goal is not to argue that building is bad or buying is good. The goal is to know where building belongs.

Shepherding the Builders

Your teams are already building, daily, at every level of the organization, with or without you.

That's not a reason to panic. It's a reason to lead differently.

Shepherding builders does not mean opening the gates and hoping for the best. It means giving builders a path that is clear enough to use and safe enough to trust.

Where should you build? Where should you buy? Which experiments should be encouraged? Which workflows need controls? When does a useful prototype become unsupported infrastructure? When does internal innovation create value, and when does it quietly create risk?

This is the shepherding role: not saying no to builders, but helping the organization learn quickly while protecting the systems, data, and decisions that matter.

The role of IT does not shrink in this model. It becomes more strategic. The modern CIO or CTO is the shepherd that sets standards, enables builders, protects the enterprise, deciding what should graduate, and knows when the right answer is to buy a proven core platform.

From the Field: Freedom Within a Framework

The tension between empowering builders and protecting the organization is not new to AI.

Writing in The Wall Street Journal, technology researcher Joe Peppard argued that tech has become so central to business performance that tech knowledge and decision-making need to move closer to the work. The point for this guide is not that IT matters less. It is that IT matters more.

Give builders room to solve real problems: approved tools, sandboxes, data rules, security standards, architectural guidance, and clear escalation paths. Let them build where the work is close, the risk is bounded, and the learning is valuable. Speed from the AI era makes the builder more valuable, and the framework more important.

WHEN SHOULD YOU BUILD?

The question today is not “can we build it?” It’s whether the organization should own the system after it works. Who will maintain it? Validate it? Monitor it? Who will ensure it respects permissions? Take responsibility when the output is wrong?

AI lowered the cost of starting. It did not lower the cost of owning.

A Framework Before You Build

Most real estate organizations are making the build vs. buy decision by feel, or sometimes simply to check a box (“we use AI”). Instead, ask yourself the following questions:

1. Is this problem the right place to put internal energy and investment?
2. If we build it, who maintains it — and what happens when that person leaves?
3. Does solving this problem here solve it for everyone, or just for one team?
4. Is there another solution that creates more value across the organization?
5. What are the risks if the tool fails, and who is responsible for managing them?

Sometimes the answer is yes, clearly and obviously. Internal builders are often the best people to solve narrow, high-specificity problems that vendors will never prioritize because the market for that exact problem is one organization.

Sometimes the honest answer is that a vendor has already solved it, or that the problem is significant enough that it deserves more comprehensive resources.

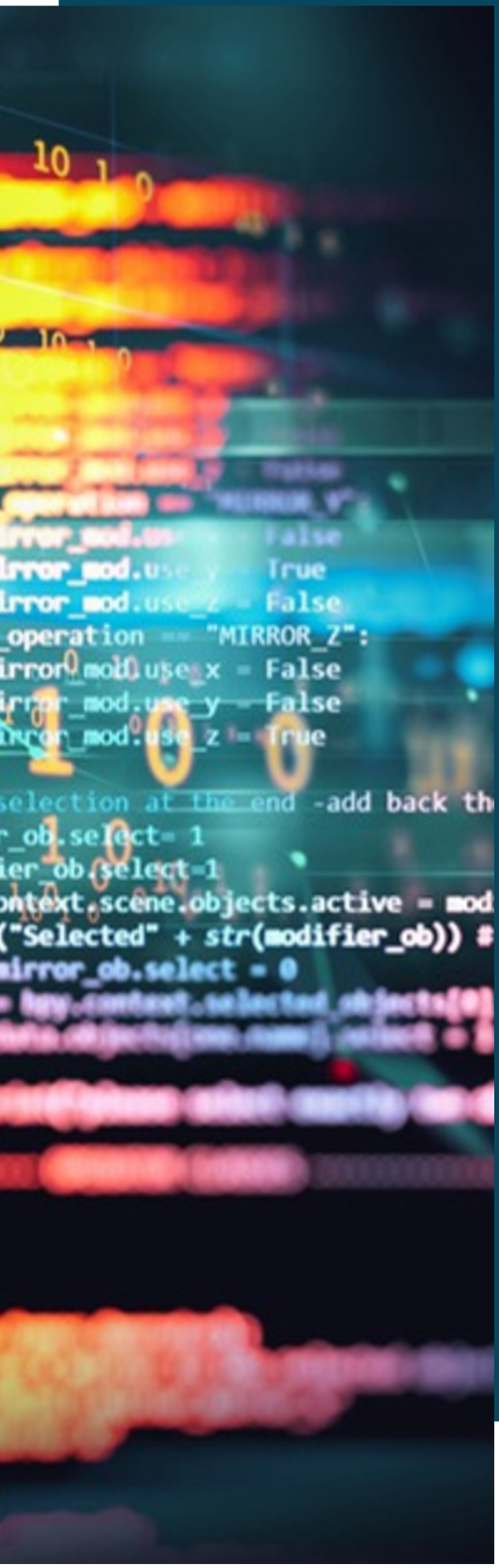
Problem Type is a Diagnostic

Before choosing a lane, ask what kind of problem you are dealing with. The categories below are not another framework to memorize. They are signals that point back to whether it's a quick win, a proprietary edge, or a solved core.

Contained-data problems are often quick wins. The right answer is mostly visible in the data in front of the model. Examples include summarizing a document, drafting a routine email, extracting fields from a clean form, classifying simple tickets, or generating a first-pass narrative from supplied numbers. These are good candidates for internal experimentation when the stakes are low and a human reviews the output.

Workflow and productivity problems can be quick wins or a proprietary edge. The goal is to help a team move faster in a workflow that is specific to how the organization operates. Examples include board-report formatting, variance commentary drafts, data cleanup utilities, policy summaries, onboarding assistants, and portfolio-specific dashboards. These are strong build candidates when the data is controlled, the risk is understood, and there is a clear owner.

Knowledge-based production problems often point toward the solved core. The right answer is not fully contained in the data in front of the model. It lives in organizational memory, accounting history, definitions, exceptions, policies, permissions, and judgment. Lease abstraction at scale, investor reporting infrastructure, compliance workflows, AP automation, payment controls, and general-ledger-impacting processes usually require more than a prototype. They may still be build candidates if they are truly strategic or if no viable solution exists, but they should be funded and governed like products.



Core vs. Context, Mapped to the Lanes

The build vs. buy question is an investment decision. A capability can be operationally important and still not be strategic to own.

Core means tied to how the company wins. If a capability differentiates the business, creates a strategic advantage, or reflects a proprietary operating model, building may be justified. Context supports the business. It may be critical, but it is not the reason the company wins. In those cases, buying is often the more disciplined allocation of time, capital, and attention.

For a real estate owner, operator, REIT, or property manager, the core business is asset performance, tenant experience, portfolio strategy, investor returns, operating discipline, and market judgment. The core business is not usually maintaining document pipelines, permissioning systems, validation layers, model monitoring infrastructure, or integrations for common workflows that vendors already support well.

Just because something uses AI does not make it core. That is the trap. AI can make every automation feel strategic. The discipline is deciding where the organization should place its bets: build where the upside is immediate and the blast radius is small, build where the capability or data advantage is proprietary enough to justify long-term ownership, build where the market has not produced a viable answer, and buy where a well-made solution already solves the problem economically.

The principal rule: Build the quick wins. Build the proprietary edge. Buy the solved core.

The Heart of the Framework

Consider these three build lanes:

1. Build the quick wins
Quick wins are small, bounded tools that help one person or one team move faster. They are close to the work, easy to test, easy to stop, and low enough risk that the organization can learn without creating a permanent burden.
For CRE, quick wins often include: Board-report formatting helpers, internal variance commentary drafts, data cleanup utilities, leasing follow-up drafts, team productivity tools that do not touch sensitive systems. The upside should be immediate, the blast radius should be small, and the cost of being wrong should be manageable.

2. Build the proprietary edge
The proprietary edge is where your organization wins differently. These are capabilities tied to your investment strategy, operating model, portfolio structure, decision-making process, or a proprietary data asset that changes the economics of the problem. They may be complex. They may require integrations, permissions, audit trails, and real engineering. That is not a reason to avoid them. It is a reason to fund and govern them properly.

For CRE, proprietary edge builds may include: A proprietary asset selection or risk model, portfolio strategy tools based on internal investment theses, unique tenant experience capability, reporting layers that reflect how leadership actually makes decisions. The test is whether owning the capability creates strategic advantage and whether leadership is willing to fund ownership, not just development.

3. Build where the market has not produced a viable answer
Sometimes the right product does not exist. Sometimes available products are immature, too generic, too expensive for the value delivered, or misaligned with the workflow. In those cases, building may be the disciplined choice, provided the organization is honest about maintenance, security, access, testing, and long-term ownership.
The test is whether there is a viable economic solution in the market. If there is not, and the problem is worth solving, building may be necessary.

Gut Checks Before You Commit

These tests do not replace the build lanes. They sharpen them.

Data readiness. Data readiness can change the build decision. A capability that usually points toward buying may become a defensible build if the organization already owns a deep, consistent, high-quality structured data asset. Lease abstraction is the clearest example. If a firm has spent years producing standardized lease abstracts with reliable fields and strong quality control, building on top of that proprietary layer may be defensible. If the starting point is thousands of raw, highly variable lease documents, the team may solve a useful subset quickly, but the edge cases will appear later in definitions, clauses, amendments, exclusions, recoveries, and review workflows.

Shortcut dependency. Internal builders can use assumptions vendors cannot. They can assume a specific chart of accounts, a known approval path, a consistent property structure, a single ERP instance, or a local business rule. That can be a real advantage for quick wins and proprietary edge work. Those shortcuts should be named, not hidden. Every local assumption is a debt marker: useful today, brittle tomorrow if the workflow expands.

Market scan. A build decision is not disciplined unless it includes a serious look at what the market has already solved. Before committing meaningful internal capital, scan the market: what exists, how mature is it, what does it cost, what gaps remain, and are those gaps important enough to justify ownership? The full buying discipline comes later in this guide.

If the build does not fit one of the three lanes, and if these gut checks raise concerns, the decision should start to move toward buying or toward a hybrid path.

Authority Raises Risk

The more a tool can see, decide, change, approve, or report externally, the more it needs enterprise controls and the more likely it belongs in the solved core or in a deliberately funded proprietary build.

Type of Build	Examples	Typical Build Posture	Watch Outs
Personal Productivity Tool	Email drafts, meeting summaries, formatting help	Usually Build	Do not expose sensitive data
Internal Knowledge Chatbot	Policy assistant, onboarding assistant, SOP search	Often Build or Hybrid	Permissions, source freshness, hallucinations
Report Builder	Board packs, variance summaries, asset dashboards	Often Build or Hybrid	Metric definitions, data consistency, stale data
Document Scraper	Pull fields from invoices, leases, contracts	Sometimes Build	Extraction isn't understanding, downstream use matters
Read-Only Agent	Research agent, reconciliation helper, exception finder	Higher Risk	Data access, source reliability, overreach
Decision Engine	Code invoices, support GL entries, payment approvals	Usually Buy	Accuracy, determinism, accountability, controls

When to Graduate a Build

There is a moment in the life of almost every internal build when the problem outgrows what an individual builder should own. Recognizing that moment and acting on it is not failure. It is the mark of a builder who understands their role clearly.

The signals are recognizable:

- The tool is now used by people who were never part of its design.
- The outputs feed reports or systems that carry organizational accountability.
- The maintenance consumes more time than the tool saves.
- The edge cases are multiplying faster than the builder can handle them.
- The data has become more sensitive than it was when the project started.
- The tool now needs access controls, audit trails, monitoring, or writeback.
- The original builder has become the de facto owner, maintainer, trainer, and support desk.

At that point, the right move is not to build faster or extend the tool further. The right move is to surface the problem to the people equipped to own it at the scale it has become: IT, leadership, risk, finance, a consultant, or a vendor.

A well-functioning organization should want this outcome. An internal builder identifies a real problem, proves that it is worth solving, and hands it off to a more robust implementation when the proof of concept has done its job. Building something that demonstrates a problem is worth solving is a contribution. Knowing when to hand it to the appropriate support team is what makes it lasting.

WHAT TO WATCH WHEN BUILDING

Most demos prove capability. Production tests accountability.

You build a proof of concept in a weekend. It handles 80% of cases beautifully. Leadership is impressed. You get the green light. What you demonstrated is that the model reasons well when the answer is in the room. What you did not demonstrate is whether your real problem is that kind of problem.

That hidden work often starts as review and exception management. There are workflows where the right answer is not contained in the data in front of you. It lives in organizational memory, years of accumulated judgment, exceptions, precedents, and the gap between how the business works on paper and how it works in practice.

A transaction may match the usual pattern, but this quarter the business restructured and the normal treatment is now wrong unless you know why the change happened. A multifamily investor may ask about per unit rehab cost. A general model might interpret that as unit-level economics: appliances, fixtures, finishes. The investor may mean per door cost across the entire project, the standard way the industry aggregates rehab budgets. Same words. Different meaning. A purpose-built vertical system is more likely to know the difference because the domain knowledge is baked into the product.

In CRE, accuracy is not only pulling the right number. It is pulling the right number under the right definition for the right decision.

Nondeterminism and Consistency

There is a failure mode embedded in AI-powered internal tools that most builders do not discover until they have already caused a problem. It is called nondeterminism — give the same input to an AI system twice, and you may get two different outputs.

That behavior can be useful. But it's dangerous when the task requires consistency. A report can have a different narrative paragraph. It cannot have a different NOI. An invoice cannot be coded to three different GL accounts under the same facts. An occupancy calculation cannot change definitions depending on who asks the question.

This is the distinction that should guide AI architecture: Use nondeterministic models where variation creates value, deterministic controls where consistency creates trust.

The Three Invoice Problem

Imagine building an internal invoice-coding assistant. They paste in an invoice and the model returns a plausible coding suggestion. It looks good. They paste in a second invoice. It also looks good. Then the third invoice goes wrong.

Jimmy Astle has seen this pattern play out repeatedly.

Astle has spent the last several years building AI-powered agents for cybersecurity operations. These are systems that ingest a continuous stream of security alerts and reason through them the way a skilled analyst would, following structured workflows, checking systems, drawing connections.

What happened is not a bug. It is the nature of the system. LLMs generate likely answers. They do not automatically know the approved answer. Without validation, source checks, deterministic rules, and human review, the wrong output can look exactly like the right one.

That quality is tolerable in a writing assistant. In an operational workflow where outputs feed the general ledger, it is a specific category of risk.

It's what Astle calls the 80/20 problem: AI tools can typically handle the straightforward cases reliably, but the remaining 20% — the unusual invoice formats, the edge cases — are exactly where the highest-stakes errors occur. And unlike a spreadsheet formula, which fails consistently and visibly in ways that can be traced and corrected, an AI-powered tool can be wrong intermittently, in ways that look correct until they don't.

"These models are sycophantic. They sound so convincing, even when they're wrong"
-Jimmy Astle

Guardrails Are Not Optional

The answer to nondeterminism is to build guardrails into every AI tool before it goes into production. The principle is simple: do not trust any tool, automated or otherwise, without an independent check to verify.

A guardrail can flag an anomaly. It cannot always decide what the anomaly means. For any AI tool whose outputs feed investor reporting, lender compliance, AP processing, payment workflows, or general ledger entries, human review is not an optional enhancement. It is part of the control structure.

A Lesson

A real estate investment firm needed to run performance attribution on its portfolio. The issue was a daily calculation that broke down returns by property, by strategy, by period — it was manual, time-consuming, and error-prone. Someone on the team built a custom tool to automate it. The tool worked, producing outputs the team needed daily, and freeing up meaningful time that had previously gone to a process nobody enjoyed.

2 years later, it stopped rolling. The outputs, however, did not stop. What the tool produced after it stopped rolling was not nothing — it was stale data, carried forward, still formatted correctly, still landing in the right place, still being read and trusted and used to inform decisions.

Nobody knew what they didn't know. What makes these failures particularly dangerous is precisely what makes the tools valuable in the first place — they became the fabric of daily operations. **The takeaway: invisible infrastructure fails invisibly.**

The Ongoing Maintenance Trap

The most persistent misconception in internal tool-building is that the cost of building a tool is the cost of building it. It is not. The cost of building a tool is the cost of everything that comes after.

Development time is the visible cost: the hours spent designing, prompting, coding, testing, and deploying. What rarely gets counted is the maintenance burden that begins the moment the tool goes into use: fixing unexpected behavior, updating the tool when a connected system changes, answering questions from colleagues, retraining users, investigating exceptions, revisiting permissions, and eventually sunsetting the tool when it can no longer be sustained.

A practical total-cost-of-ownership view should include:

- Build time and opportunity cost
- Model, API, hosting, and token costs
- Security review and access-control design
- Integration build and integration maintenance
- Testing, backtesting, and validation
- Documentation and training/user support
- Monitoring and incident response
- Change management when the business changes
- Retirement or migration when the tool is no longer viable

For AI-powered tools, the burden is more dynamic than traditional spreadsheet macros. Models change. McKinsey has estimated that technical debt can amount to 20 to 40 percent of the value of an organization's technology estate before depreciation. That is the enterprise version of a simple truth: shortcuts compound.

The honest cost-effectiveness question is not "is building this cheaper than buying a vendor solution?"

The honest question is: Is building this, maintaining it, updating it, securing it, monitoring it, supporting it, and eventually sunsetting it cheaper than buying a vendor solution, and is the person who will do all of that work actually available to do it?

Every Internal Build Needs an Exit Plan

The initial build, Astle says, was the easy part. What followed was what he calls "care and feeding" of AI-created creatures, and as he describes it, it's the part that the enthusiasm around AI consistently undersells.

When you update a single instruction inside an AI agent, he explains, you can break everything the agent has learned to do correctly. The fix for one edge case becomes the source of a new failure somewhere else. That means any serious internal build requires backtesting: a process of running proposed changes against the full archive of historical cases to verify that the new behavior works without degrading the old. Building and maintaining a backtesting infrastructure is its own engineering project, invisible to the people using the tool, requiring ongoing attention from people who understand both the underlying models and the business logic the models are supposed to apply.

Then there is model deprecation. When a model provider updates or discontinues a model version, the tools built on top of it do not receive a graceful transition. They simply start behaving differently — or stop working. In cybersecurity, where Astle's team monitors their agents continuously, they can detect and respond to that kind of shift. In organizations where an internal tool is running quietly in the background, trusted because it has always worked, the change may go undetected until it has already affected outputs that people are depending on.

These are not hypothetical risks. They are

the predictable lifecycle of any AI-powered internal build, and they require someone to own them, continuously, for as long as the tool is in use.

That is not a technology failure. It is an organizational one.

A healthy build program should include sunset rules.

Before a tool becomes widely used, answer: Who owns this tool? Who reviews it? How often is it tested? What would cause us to shut it down? What happens when the builder leaves? Where is documentation stored? How do users know whether it is approved? Is there a vendor or enterprise platform that should replace it if usage grows?

Every internal build needs an owner, and every owner needs an exit plan. Internal tools often survive too long because nobody decides when they should die.

WHEN BUYING IS THE ANSWER

Buying requires vendor evaluation, procurement cycles, implementation planning, and organizational alignment. It can take months before anything changes in daily operations. The path of least short-term friction often points toward building.

That is precisely why a framework for recognizing when to buy matters.

There are two moments to watch. The first is before a build begins, when leadership should ask whether a mature market solution already exists. The second is after a prototype succeeds, when the organization must decide whether to graduate the build into a supported internal product, replace it, or retire it. These signals apply to both moments.

When a viable market solution already exists
If the market already offers a well-made, economically sensible solution, the burden of proof shifts. Building may still be justified if the capability is proprietary, underserved, or strategically important enough to own. But the organization should be able to explain why internal ownership is worth more than vendor leverage.

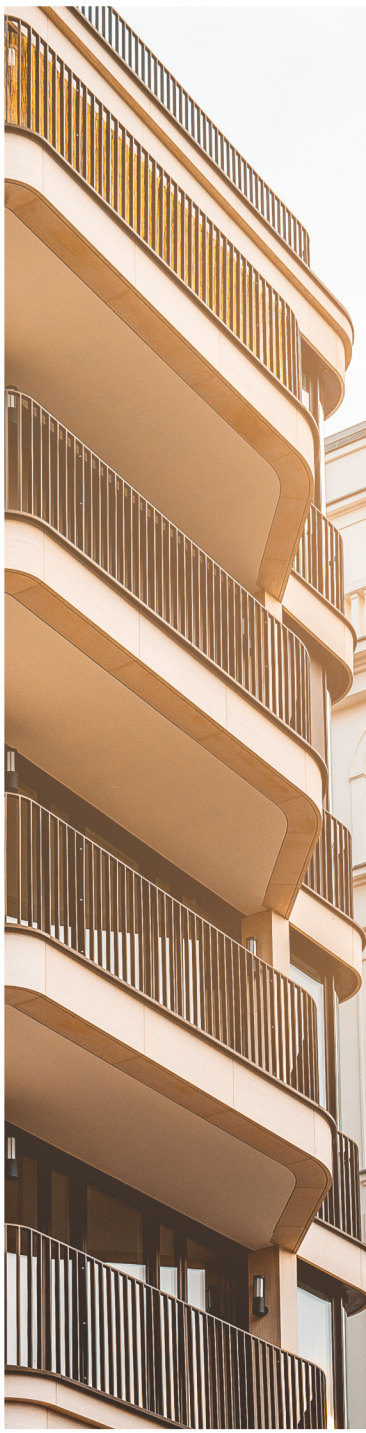
When the problem is bigger than one team
When a tool has accumulated upstream and downstream dependencies the builder no longer controls, it has outgrown its origins. It is now infrastructure, and infrastructure requires institutional ownership.

When compliance and accountability are in the picture
Investor reporting, lender covenant compliance, payment approvals, tenant data, and general ledger integrity require governance. Tools that touch these workflows need audit trails, access controls, documented logic, exception handling, and accountability.

When deterministic output matters
If the same input under the same facts must produce the same answer, the system needs deterministic controls. An LLM may assist, but it should not be the only authority.

When the maintenance math has turned
Add up the hours spent maintaining, fixing, updating, explaining, or manually checking an internally built tool. Multiply by the fully loaded cost of the people spending those hours. Add the cost of delayed work, missed improvements, and risk exposure. Then compare that to the cost of a vendor solution that solves the same problem. The math may already have answered the question.

When vendor scale is the advantage
When quality depends on the volume and diversity of data, a purpose-built vendor has the advantage. The question is not whether edge cases will arise. It's whether the vendor has already encountered them, solved them, and built that learning into the platform.



Buying the Solved Core

The solved core is where the market has already absorbed the hard parts. These are repeatable, high-volume, high-control functions that many organizations share and where vendors improve through scale, feedback, security investment, and domain expertise.

For CRE, the solved core often includes: Lease abstraction at scale, investor reporting infrastructure, compliance-heavy workflows, ERP integration and system-of-record writeback, invoice ingestion and coding, workflows that affect the general ledger, approvals, payments, or sensitive data.
Could a sophisticated organization build some of these capabilities? Yes. The better question is whether it should, rebuilding that learning curve internally is rarely the highest-value use of time.

What Vendors Bring to the Table

A vendor's accumulated experience is not just a convenience. It is the product.

You are buying years of intelligence identified, handled, and built into a system that creates a feedback loop for compounding accuracy. Specialist vendors encounter edge cases that any single organization's team may never see: unusual lease structures, complex invoicing scenarios, client-specific approval rules, data quality problems, utility allocation patterns, chart of accounts variations, vendor behavior, integration failures, and exception workflows.

Every problem solved across the vendor's client base can make the platform more robust. Every implementation sharpens the product. Every correction improves the model, rules, or roadmap. Over time, that creates a compounding intelligence advantage that client organizations can benefit from without having to fund, staff, and manage the entire program internally.

How to Buy Well: Separating Software from an Impressive Demo

Before committing significant internal capital to a build, look seriously at what is already in the market.

A good market scan asks what products exist, how mature they are, how they handle the messy parts, what they cost at real volume, what implementation requires, and where they fail. It should also ask whether the gaps that remain are truly strategic or simply preferences.

Every vendor looks good in a demo. It's how software is sold. A demo is a controlled environment, built around the vendor's strongest use cases, populated with cleaner data than most real organizations have, and guided by someone who knows exactly where the sharp edges are.

This is especially important to note in CRE because the data is complex, inconsistent, and highly variable. Leases have unusual structures. Invoices arrive in non-standard formats. Properties have complex ownership arrangements. Client definitions differ. Charts of accounts vary. Legacy systems export data in ways that only a few people understand.

The question is not whether edge cases exist. The question is whether the vendor has encountered them, solved them, and built those lessons into the platform.

The most valuable thing a buyer can do is run the platform against their own data, not the vendor's sample data.

Ask the vendor to show:

1. Realistic source documents, not clean demo files
2. Edge cases from your actual workflow
3. What happens when the model is unsure
4. How exceptions are routed
5. What audit trail is preserved
6. How corrections improve future outputs
7. Which integrations are native and which require custom work
8. What the implementation team needs from you
9. What success looks like after 30, 60, and 90 days
10. What happens when business rules change

If the vendor is unwilling to run a proof of concept against real data before the contract is signed, that reluctance is itself a data point.

Red Flags & Green Flags

The questions that matter most are often not on a standard RFP. Ask the vendor to describe the last significant bug or data quality issue that affected clients in production. A vendor worth buying should be able to pass the accountability test and talk honestly about failure, limitation, and the unglamorous realities of production use.

Red flags tend to cluster around a few patterns: Reluctance to test on your real data, inability to speak specifically about exceptions, overstated automation claims without clear exception handling, weak answers on permissions, audit trails, and data separation, a roadmap focused on features but not reliability, security, and data quality.

Green flags are their mirror. A vendor who proactively surfaces the limitations of their platform for your specific use case before you ask is demonstrating a level of honesty that tends to predict a better long-term relationship. References who describe not just what works but how the vendor responded when something did not work are valuable signals. A proof of concept process that the vendor treats as seriously as the sales process suggests a company that understands its product is being evaluated against reality, not against a presentation. And a roadmap that reflects visible investment in the problems that matter most in production, data quality, exception handling, audit trails, and reliability, is a more durable signal than a list of impressive features.

A great vendor does not pretend production is easy. A great vendor shows you how they manage the difficulty.

MAKING THE BUY DECISION STICK

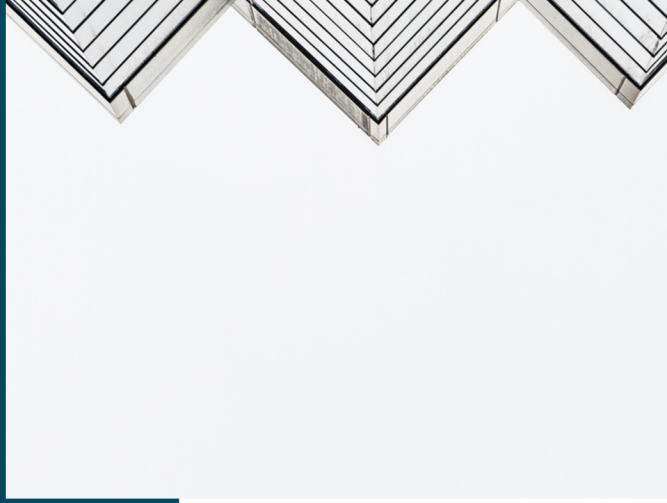
A technology buying decision made well and implemented poorly produces outcomes that are hard to distinguish from a bad buying decision.

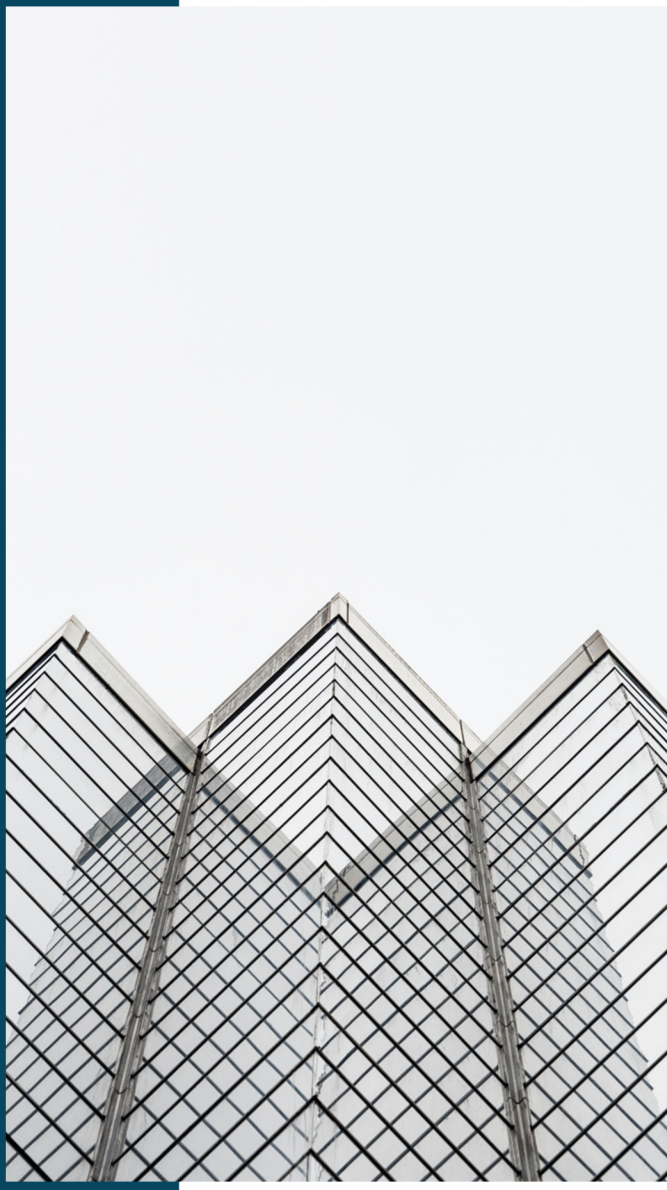
The platform underperforms. Adoption is lower than expected. The vendor gets blamed. Evaluation usually involves a small group of decision-makers with high context and strong motivation, but the implementation affects a larger group of people who may have had no involvement in the selection. Closing that gap requires communication before implementation begins, not after problems surface.

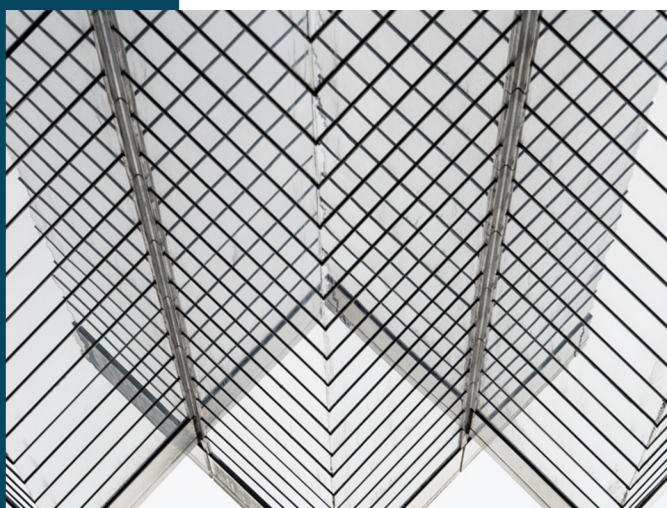
The implementation often deserves as much scrutiny as the product.

Implementation risk tends to concentrate in predictable areas: the mitigation is more time, more specificity, and more organizational involvement earlier than feels necessary.

That's why implementation should include the people whose workflows will change most. They know where the process breaks. They know which edge cases matter. They know what will be trusted and what will not. A strong product meets users where the work already happens. A tool that creates another place to check, reconcile, explain, or manually correct may be technically impressive and still fail operationally. Adoption is trust. And trust comes from accuracy, consistency, speed, fit with workflow, and confidence that exceptions will be handled.







LESSONS FROM OTHER INDUSTRIES

Commercial real estate is not the first industry to wrestle with build vs. buy. Technology, financial services, insurance, healthcare, and logistics have all learned similar lessons.

When Joe Peppard revisited his Wall Street Journal argument after receiving pushback from technology leaders and practitioners, the most useful lesson was not whether technology should be centralized or decentralized. It was that most organizations need a hybrid model: local speed with central standards, business ownership with technical guardrails, and experimentation with accountability. Peppard noted that while many leaders agree the answer is some version of hybrid, “nobody described what this would actually look like in practice.”

This guide is one attempt to give that hybrid model a practical shape.

Build the quick wins. Build the proprietary edge. Buy the solved core. Shepherd the builders.

Another way to describe the same principal rule is through the lens of systems of record, systems of differentiation, and systems of innovation. This is not a competing framework. It is the same decision logic from a different industry vocabulary. Systems of record require stability, security, integrations, permissions, auditability, and long-term support. These are often better bought because vendors have spent years hardening the infrastructure.

Systems of differentiation require more nuance. If the capability is genuinely proprietary and central to competitive advantage, building may be justified. Systems of innovation are where experimentation belongs. These are the new ideas, prototypes, and narrow workflows that help the organization learn. Many will not survive, and that is fine.

The mistake is treating a system of record like a science experiment.

The highest-performing technology organizations are not pure builders or pure buyers. They build quick wins, invest in proprietary edge, buy aggressively where the market has already solved the core, and create governance that lets builders work with the organization rather than around it.

Real estate should do the same.

For most CRE firms, the core product is managing assets, serving tenants, operating efficiently, and generating returns for investors. It is not building software that a dedicated vendor has already spent years perfecting.

WHAT CRE SHOULD TAKE AWAY

The best answer is often not a pure build or a pure buy.

"Build where only you can build. Buy where the market has already solved the hard parts." — David Stifter, Founder & CEO, PredictAP

Mature organizations buy the durable core, configure the workflow, and build the edge around it.

The durable core is the part of the problem many organizations share: security, permissions, audit trails, integrations, uptime, exception handling, source traceability, and long-term maintenance. When a purpose-built vendor has already absorbed those hard parts, buying creates leverage.

The workflow layer is where configuration matters. Every real estate organization has its own approval paths, reporting preferences, chart of accounts, property structures, and operating conventions. A strong vendor platform should support that variation without requiring the customer to rebuild the platform itself.

The edge is where internal builders shine. These are the reporting layers, decision-support tools, data cleanup helpers, portfolio-specific analyses, and internal workflows that reflect how one organization actually operates. They are close to the work, shaped by local context, and often too specific for the market to solve.

This hybrid path avoids the false choice. It lets vendors carry the solved core while builders focus their energy where the organization is genuinely different.

Seven Takeaways for CRE Leaders

The lessons for REITs, owner-operators, management companies, and investment firms are practical.

1. Find the builders
The builders are the gold nuggets. They show you where friction lives. They show you where the business wants to move faster. They show you where the roadmap is missing the lived reality of the operation.
Encourage them. Give them safe tools. Give them sandboxes. Give them simple rules. Capture the prompts, templates, and small workflows that work. Give them a path to raise their hands before a prototype becomes infrastructure.

2. Classify the build before you start
A personal productivity tool is not a decision engine. A read-only chatbot is not a writeback agent. A report commentary assistant is not an investor reporting system of record.
The risk should follow the authority of the tool.

3. Build in the three defensible lanes
Build the quick wins when the upside is immediate, the blast radius is small, and the cost of being wrong is manageable.
Build the proprietary edge when the capability is tied to how your company wins, when your proprietary data advantage changes the economics, and when leadership is willing to fund it like a product.
Build when the market has not produced a viable economic solution and the problem is still important enough to solve.

4. Buy the solved core
Buy the workflows where vendors have already absorbed the hard parts: lease abstraction at scale when you do not have a genuine structured-data edge, investor reporting infrastructure, compliance workflows, payment controls, role-based

permissions, ERP integration, AP automation, invoice coding, audit trails, and other processes where accuracy and trust compound through scale.
These are not places to recreate the market's learning curve unless owning the capability is truly strategic or the available market solutions are not viable for your use case.

5. Treat access and definitions as product requirements
If an AI tool connects to enterprise systems, it must respect enterprise permissions. If it reports metrics, it must use approved definitions. If it touches financial data, it must trace to source. If it acts, it must be auditable.
These are not technical details. They are the foundations of trust.

6. Remember that maintenance is forever
Every internal build needs an owner, a review cadence, documentation, testing, monitoring, and an exit plan. A tool that no one maintains is not an asset. It is future technical debt.

7. Make the vendor prove production, not the demo
Test on real data. Ask about failures. Ask about exceptions. Ask about permissions. Ask about drift. Ask about implementation. Ask how the product gets better over time. The demo should start the conversation. It should not end it.

The Final Principle
The build vs. buy decision is not a referendum on innovation. Building is not automatically bold. Buying is not automatically conservative. The smartest organizations do both, with discipline.

They encourage builders. They create safe lanes. They learn from prototypes. They graduate the best ideas. They buy purpose-built platforms for the workflows where scale, controls, integrations, and accumulated domain expertise matter most.
Build the quick wins. Build the proprietary edge. Buy the solved core. Shepherd the builders.

That single idea can save more time, money, and frustration than almost any technology decision a CRE firm will make in the years ahead.

About PredictAP

PredictAP is an AI-powered invoice ingestion and coding platform built exclusively for real estate.

Designed with the workflows of owners, operators, and property managers in mind, PredictAP replaces manual data entry, error-prone coding, and reliance on templates or outsourcing.

Its patented AI learns from each portfolio's historical data to deliver accurate, property-level coding that integrates directly into existing AP systems.

PredictAP helps real estate finance teams reduce costs, accelerate processing, and free staff from repetitive work, so they can focus on what matters most.

Learn more at www.predictap.com

EXECUTIVE QUESTIONS

Question	Why it matters
Is this core or context?	Core capabilities may justify investment. Context should usually be bought or kept lightweight.
Is the answer in the data, or in institutional knowledge?	Knowledge-based workflows require history, rules, exceptions, and feedback loops.
Does the tool touch sensitive data?	Tenant, vendor, investor, bank, lease, and financial data require controls.
Is the input data structured and trusted?	Builds on clean internal data are very different from builds that must interpret messy source documents.
Does the build rely on shortcuts or local assumptions?	Local shortcuts can accelerate a build, but they can also break when the tool expands.
Does it affect financial reporting or the GL?	Outputs must be consistent, traceable, and auditable.
Does it need deterministic output?	Same inputs and same facts should produce the same answer.
Does it respect user permissions?	The AI should not see more than the user can see.
Does it need writeback?	Write access requires approvals, rollback, logs, and monitoring.
Who owns it after launch?	Without ownership, the tool becomes technical debt.
How will it learn from corrections?	Learning must be validated and auditable.
Have we scanned the market first?	A build is more defensible when there is no viable economic solution to buy.
What is the path to graduate, buy, or retire it?	Every internal build needs an exit plan.

Appendix: Short AI Primer

This appendix is for readers who want a quick vocabulary check. The terms will evolve, but the concepts are useful when evaluating AI-enabled builds and vendors.

Public models, connected models, tuned systems, and purpose-built platforms
Public AI models are trained to be broadly useful. They can write, summarize, classify, reason, translate, draft code, and answer questions across a wide range of topics. That breadth is what makes them impressive in a demo.

But breadth is not the same as institutional knowledge.

A general model does not automatically know how your company defines occupancy, how your accounting team treats a specific vendor, which entity should receive a charge, which costs are recoverable, which approval path applies, or which exception became policy after a dispute last year.

That knowledge lives in systems, history, corrections, workflows, people, and precedent.

The model may be public. The operating knowledge is not.

Once you start connecting models to internal systems, teaching them company-specific patterns, capturing corrections, routing exceptions, and writing back into systems of record, you are no longer just using an LLM. You are building and maintaining a production system.

Deterministic vs. Nondeterministic
A deterministic system is designed to produce the same output when given the same input, rules, and data. A nondeterministic system, such as a large language model, may produce different wording, reasoning, or conclusions from the same or similar input. That distinction matters because some workflows benefit from variation, while others require repeatability.

Use nondeterministic models where variation creates value. Use deterministic controls where consistency creates trust.

The full implication of this distinction shows up later in the guide when we discuss financial data, auditability, and production controls.

Agents
An agent is an AI system that can pursue a task across steps. It may read information, call tools, ask follow-up questions, make recommendations, create records, update systems, or trigger workflows.

The risk of an agent depends less on whether it is called an agent and more on what authority it has. The more an agent can do, the more governance it needs.

Appendix: Short AI Primer

APIs, connectors, and MCP
APIs, connectors, and protocols such as MCP are part of the emerging architecture for connecting AI systems to enterprise tools and data. Anthropic describes the Model Context Protocol as an open standard for building secure, two-way connections between data sources and AI-powered tools.

That is important. Connections matter. But a connection is not a control.

A connector can make data reachable. It does not automatically make data trusted, current, correctly defined, permissioned, or safe to act on. APIs and MCP do not automatically solve user permissions, data lineage, metric definitions, audit trails, writeback safety, approval workflows, client separation, retry logic, rollback, prompt injection, or feedback loops.

The technical connection is only the first mile. The production system still needs governance around what the AI can access, what it can do, how answers are verified, and how corrections improve the system over time.

Token efficiency
One of the least understood costs in an internal AI build is token efficiency.
In a demo, token efficiency rarely matters. Someone pastes in a document, adds a prompt, gets a good answer, and the cost feels negligible. Production systems do not run one prompt. They run thousands or millions of prompts across users, workflows, documents, policies, examples, corrections, and integrations.

Every extra piece of context sent to a large language model has a cost. It can increase latency. It can increase spend. It can introduce irrelevant information that makes the answer worse. It can also increase the amount of sensitive business data moving through the system.

That is why production AI systems need context discipline.

A well-designed system does not send the entire policy manual, vendor history, chart of accounts, property list, prior invoice population, and accounting rules to the model every time. It retrieves only the relevant context. It caches stable instructions. It separates reusable knowledge from dynamic inputs. It uses smaller models where appropriate and more advanced models only when the task justifies the cost. It measures token usage by workflow, user, exception type, and outcome.

Large prompts can make prototypes look smart because they give the model a lot of information. At scale, large prompts can become expensive, slow, difficult to govern, and less accurate.

Token efficiency is not just a technical optimization. It is part of production readiness.

SELECTED SOURCES

These sources were used to ground the statistics and frameworks referenced.

1. Microsoft and LinkedIn, 2024 Work Trend Index Annual Report, “AI at Work Is Here. Now Comes the Hard Part,” including 75 percent knowledge-worker AI use and 78 percent bring-your-own-AI usage. https://www.microsoft.com/en-us/worklab/work-trend-index/ai-at-work-is-here-now-comes-the-hard-part
2. IBM, Cost of a Data Breach Report 2025, including findings on breaches of AI models or applications, lack of proper AI access controls, AI governance gaps, and the added cost of shadow AI exposure. https://www.ibm.com/reports/data-breach
3. IBM Newsroom, “13% Of Organizations Reported Breaches Of AI Models Or Applications, 97% Of Which Reported Lacking Proper AI Access Controls,” July 30, 2025. https://newsroom.ibm.com/2025-07-30-ibm-report-13-of-organizations-reported-breaches-of-ai-models-or-applications%2C-97-of-which-reported-lacking-proper-ai-access-controls
4. Joe Peppard, “It’s Time to Get Rid of the IT Department,” The Wall Street Journal, November 27, 2021. https://www.wsj.com/business/entrepreneurship/get-rid-of-the-it-department-11637605133
5. Joe Peppard, “Get Rid of the IT Department? Some People Think Otherwise,” The Wall Street Journal, September 18, 2022. https://www.wsj.com/articles/get-rid-of-it-department-people-think-otherwise-11663180383
6. McKinsey, The State of AI: Global Survey 2025. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai
7. McKinsey, “Tech debt: Reclaiming tech equity,” including CIO estimates that technical debt can represent 20 to 40 percent of technology estate value before depreciation. https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/tech-debt-reclaiming-tech-equity
8. Anthropic, “Introducing the Model Context Protocol,” describing MCP as an open standard for secure, two-way connections between data sources and AI-powered tools. https://www.anthropic.com/news/model-context-protocol
9. OpenAI, Prompt Caching guide, including cost and latency benefits from prompt caching. https://developers.openai.com/api/docs/guides/prompt-caching
10. OWASP, Top 10 for Large Language Model Applications, including prompt injection, sensitive information disclosure, and insecure plugin design. https://owasp.org/www-project-top-10-for-large-language-model-applications/
11. NIST, AI Risk Management Framework, focused on incorporating trustworthiness into the design, development, use, and evaluation of AI systems. https://www.nist.gov/itl/ai-risk-management-framework
12. Reuters coverage of Gartner’s agentic AI project risk forecast, useful context on agent experimentation and proof-of-concept risk. https://www.reuters.com/business/over-40-agentic-ai-projects-will-be-scrapped-by-2027-gartner-says-2025-06-25/

PREDICT AP