MAY 29, 2025

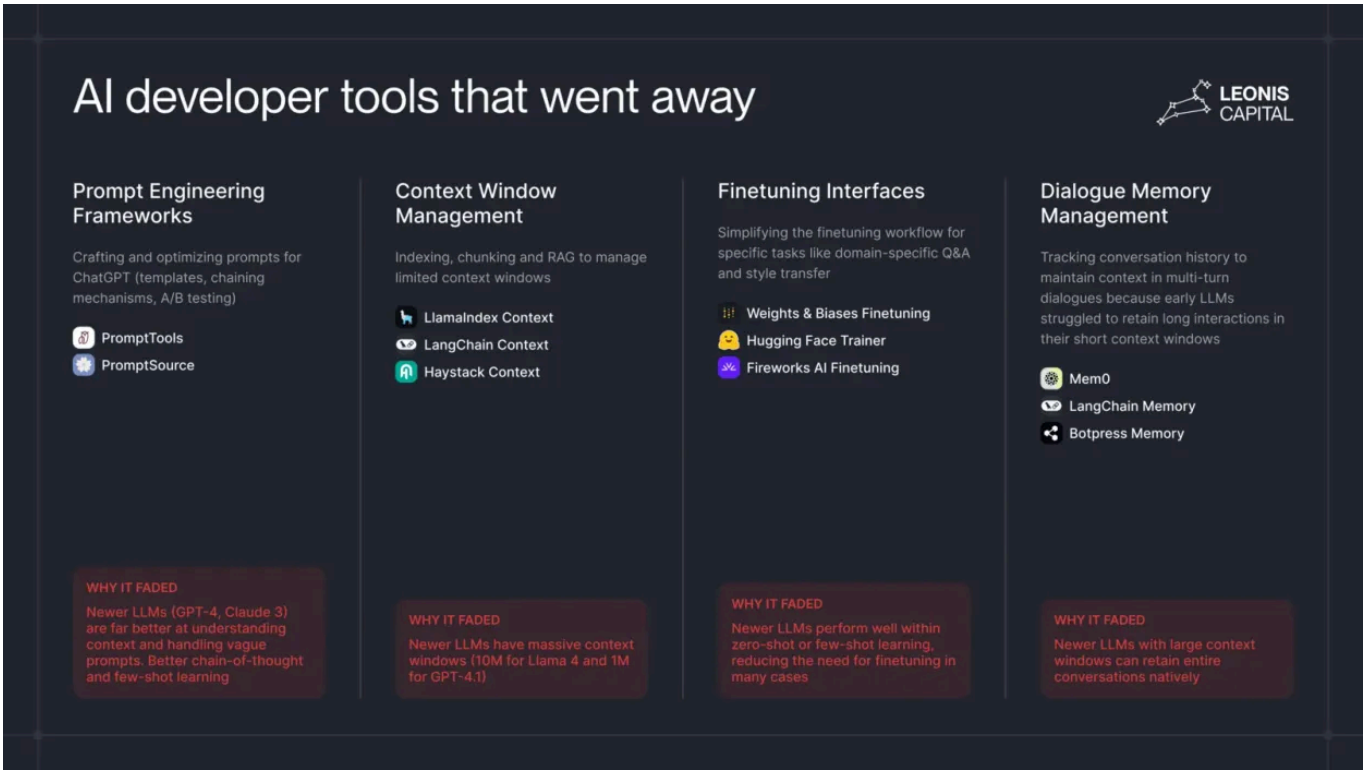
Goldmines or Graveyards? The Divergent Fate of AI Developer Tools

Our latest insight decks on technological and market trends.



By Jenny Xiao, Liang Wu, and Jay Zhao

Leonis Capital · leoniscap.com/research/goldmines-or-graveyards-the-divergent-fate-of-ai-developer-tools/



The $3B Paradox

When OpenAI announced that it was acquiring Windsurf, one of the leading developer tools for $3 billion in April 2025, it sent shockwaves through the AI developer tools ecosystem. Earlier, OpenAI had reportedly tried to acquire Cursor, the leading developer tool company by user count and ARR. The headlines poured in about the developer tools space and amongst investor circles, **developer tools were quickly seen as the "shovels" in the AI “gold rush.”**

The Windsurf acquisition highlighted a fundamental paradox in AI developer tools: while some command astronomical valuations and achieve remarkable success, many more die quietly as foundation models advance past their capabilities. Tools that seemed essential one day became irrelevant the next, yet new ones continue to emerge with sky-high valuations.

In our last two essays, we highlighted our framework to value AI companies, both foundation models and applications. We focused our analysis largely on understanding displacement risk, the risk that a company gets displaced by its competitors and/or foundation model companies. As the developer tool space has increasingly gained mainstream attention and investment, many are wondering what actually determines which dev tools make it and which get crushed. Why do some raise at sky-high multiples while others get wiped out by the next model release?

In this essay, we apply our displacement risk framework to help distinguish the substance from hype in this increasingly crowded space. Specifically, we'll define what a developer tool looks like in the AI era, look at how the category's evolved, and unpack why the space might be overvalued. Then we'll dig into displacement risk to understand who is at risk, why it matters, and how to actually build defensibility. We'll tackle the big question of exit paths: Do you build to sell to a foundation model? Finally, we'll look forward to how vertical MCPs impact the developer tools landscape.

Developer Tools in the AI Era

Developer tools have always served one fundamental mission: maximizing developer productivity. Historically, these DevTools formed the essential backbone of the software ecosystem. Examples include React frameworks streamlining UI development, PostgreSQL databases abstracting data complexity, and Git repositories safeguarding code evolution. Their value proposition was straightforward: to enhance developer productivity through the standardization and abstraction of complex processes. And their defensibility was built around their ability to cultivate developer loyalty and build an extensive ecosystem of tools and plugins.

With AI transforming software development, a new category of developer tools has emerged to help build, deploy, and maintain AI-powered applications and systems. These AI developer tools encompass both AI-enhanced versions of traditional development infrastructure (like AI-powered code completion and debugging) and entirely new categories of tooling purpose-built for AI development such as agent development frameworks like LangChain for chaining AI operations, vector databases for embedding storage, observability platforms for monitoring model performance and costs, and browser automation tools that enable AI to interact with web interfaces. These tools address the unique challenges of building with AI, from managing non-deterministic outputs and tracking token costs to orchestrating complex multi-step AI workflows and ensuring reliable performance of intelligent systems.

While AI 10x the productivity boost for developers, ironically, it also threatens the defensibility of AI developer tools. Unlike traditional tools that own their entire technology stack, AI developer tools typically build on top of foundation models they neither own nor control. This architectural dependency creates vulnerability to upstream API changes, pricing shifts, and capability improvements. Traditional DevTools built moats through ecosystem lock-in and switching costs; AI tools must fight commoditization through either deep workflow integration that captures unique user data or vertical specialization in domains where general-purpose LLMs falter.

Assessing Developer Tool Displacement Risk

Developer tools startups command some of the highest valuations in the startup world today. In the last round, Cursor was reportedly valued at $9B, a 30X multiple off of its $300M ARR. Many of the hot companies we're seeing are raising their seed rounds at high valuations, anywhere between $50M to $100M rounds, often based entirely on singular metrics like the number of their GitHub stars. Despite the capital and excitement that have gone into this space, developer tools serve as a compelling example of how AI applications can sometimes be overvalued if we factor in displacement risk.

In our previous piece on valuing AI application companies, we introduced a valuation framework where the key variable to understand is displacement risk. We adapted the standard ARR framework with a displacement risk factor, or D factor, to reflect the unique risks AI startups face:

```
Valuation = ARR x Valuation Multiple x (1-D)
```

The D-factor, ranging from 0 to 1, quantifies the probability that a company's core value proposition becomes commoditized. This applies differently across company types but captures the same fundamental dynamic: either you maintain clear superiority in your domain, or your value rapidly approaches zero and crosses the “Zero Value Threshold.” Given the rapid pace of innovation in AI and the threat of encroachment by foundation model companies, most AI applications are closer to a displacement factor of 1 than their founders care to admit.

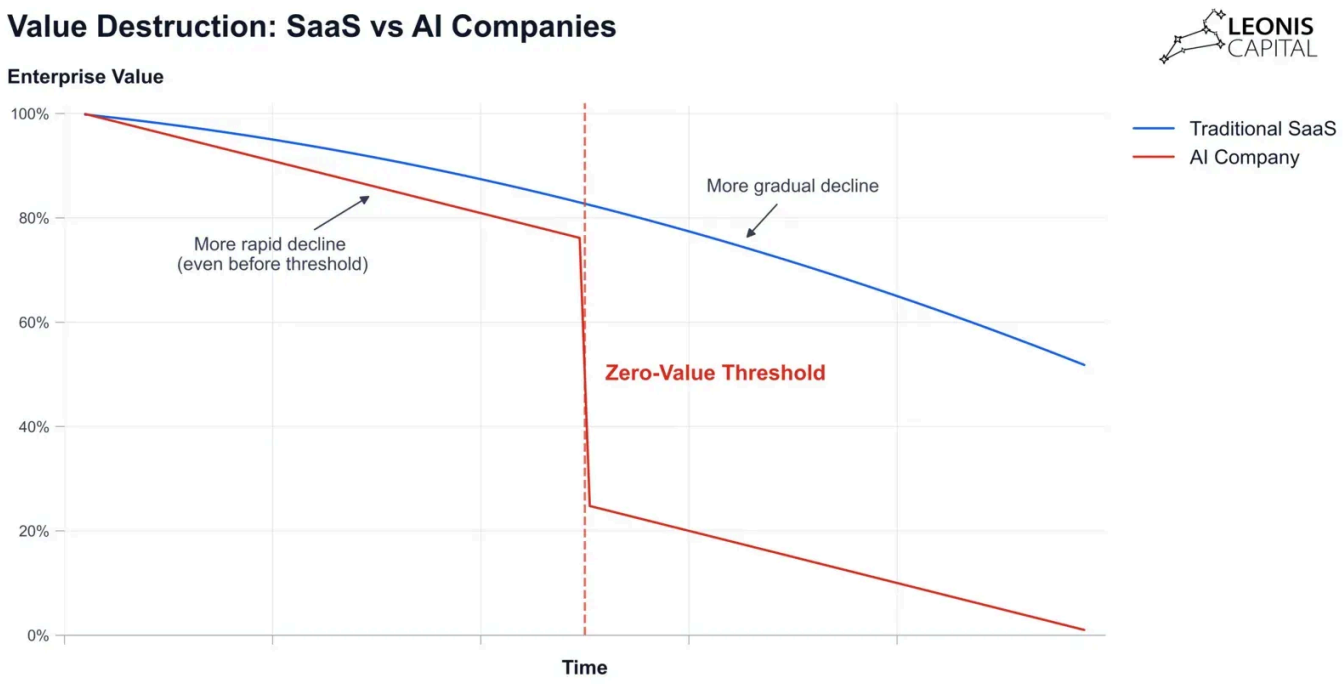


Figure 1: The value destruction curve from the "Zero or Hero" framework

While many developer tools have created lasting value, many others face displacement risk factors (D = 0.7-0.9) that are among the highest of any AI segment. This highlights the challenges of maintaining value in a rapidly evolving AI landscape.

Structural Vulnerabilities That Contribute to Displacement Risk

Many AI developer tools struggle to build sustainable moats due to structural vulnerabilities that threaten their long-term viability. There are at least four major categories of vulnerabilities that contribute to the displacement risk in the AI developer tools segment.

- **API Proximity:** Many developer tools sit dangerously close to the foundation model APIs they're built upon, making them extraordinarily vulnerable to upstream absorption. Every valuable feature is just one API update away from becoming obsolete. When OpenAI releases function calling or Anthropic improves code generation, entire categories of developer tools can become redundant overnight.
- **Minimal Switching Costs:** Traditional developer tools built strong moats through specific coding languages, frameworks, and data lock-in. For example, imagine the effort required to migrate a Vercel-hosted app to AWS or move off of Supabase. **The current vintage of AI developer tools doesn't lock in users in the same way.** Users of AI developer tools face minimal switching costs as developers can effortlessly migrate between tools with nearly identical capabilities, which can create a race-to-zero pricing dynamic.

- **Feature Commoditization Velocity:** The development cycle from "innovative feature" to "standard offering" has shrunk from years to weeks in the AI era. The GitHub Copilot advantage lasted barely six months before competitors and upstarts released viable alternatives. This acceleration occurs because very few developer tools today solve significant technical barriers; most are just UIs on top of widespread technology, which can be easily replicated.
- **Dependence on External Infrastructure:** Unlike traditional tools that control their own stack, many AI developer tools rely on third-party compute (e.g., AWS GPUs) and/or foundation models (e.g., OpenAI's GPT models). When providers change pricing or their APIs, the tool's value proposition weakens immediately. When the infrastructure part of the stack pushes an innovation, that innovation is also accessible to a developer tool's competitors.

So why does overvaluation frequently occur in this space?

The market consistently confuses adoption with defensibility, mistaking high usage rates for staying power. Developer tools often showcase capabilities that impress technically but don't translate to sustainable competitive advantages that can fight the displacement factors above. For example, as it relates to minimal switching costs, there is a persistent myth that developers as a user segment can remain loyal to a product or tool. Developers love to try new products, they are often the first to tinker, and in an industry where it's easy to jump between a variety of different tools, it **becomes increasingly challenging to claim a moat based on lock-in effects, let alone developer loyalty**. Lofty valuations often assume stable, captive user bases when, in reality, developers are historically among the most willing to switch tools when better alternatives emerge. Similarly, when it comes to feature commoditization velocity, we've seen features like AI-powered code completion, pull request summarization, and automated documentation that were once premium offerings become standard capabilities across multiple tools within months of their introduction.

More fundamentally, as the cost of generating high-quality code decreases, the willingness to pay for developer tools will also diminish. When everyone can be a "10x engineer," even those who are great at vibe coding, through AI assistance, the core value proposition of developer tools boosting developer productivity will be an increasingly tougher sell. Companies will opt to build directly with AI tools that offer functionality beyond coding assistance, like the growing feature set by the leading foundation model companies.

Investors who invest in this space need to factor in the displacement risk factors and understand more deeply where the displacement risk is real and where it may not be.

The AI Developer Tool Graveyard

To understand which tools will survive, we must first examine those that didn't. Here's a sampling of popular LLM developer tools that have faded as foundation models advanced:

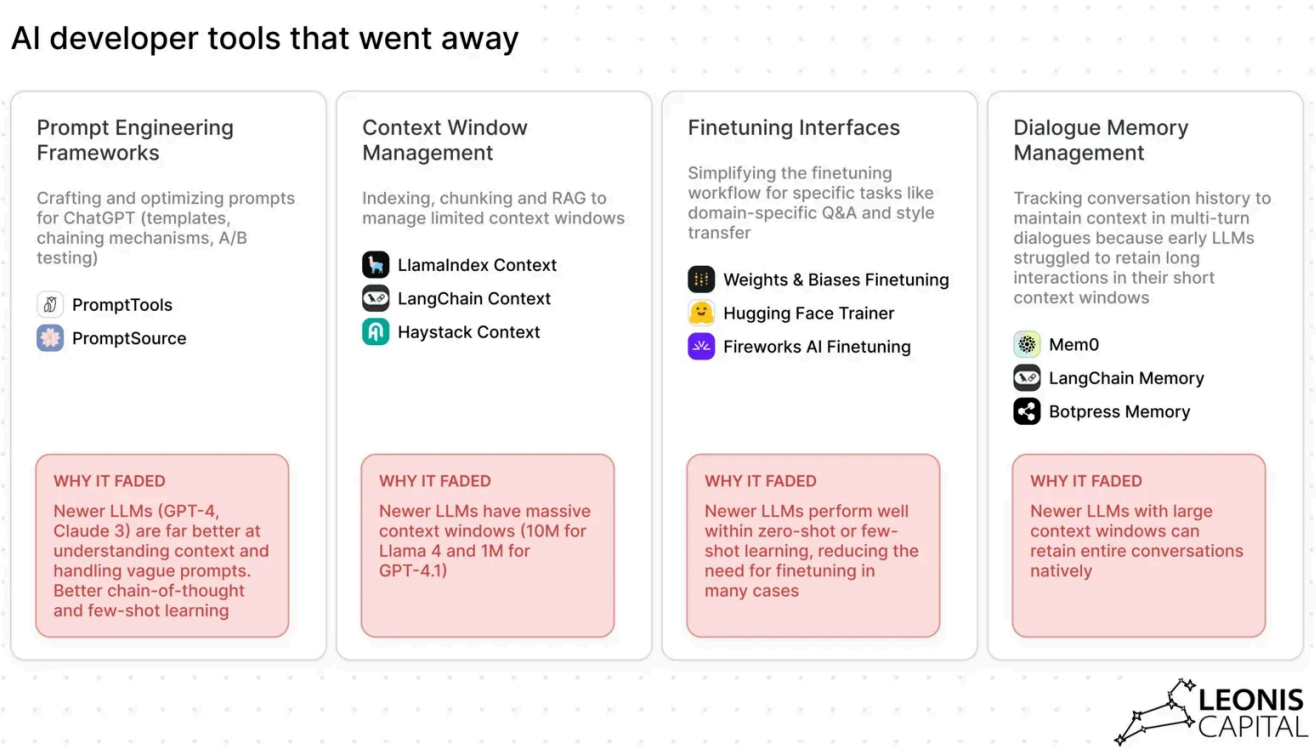


Figure 2: Popular LLM dev tools and features that faded as foundation models advanced

Each advancement in foundation models, from larger context windows to enhanced reasoning, has instantly rendered entire categories of developer tools obsolete. Developer tools that try to solve existing LLM limitations are racing against time. When OpenAI or Anthropic releases their next major update, will your tool still be necessary? Most cannot answer this question confidently, and many more refuse to confront this brutal reality entirely.

Building Towards Defensibility

Despite these displacement risks, some developer tools have successfully built sustainable businesses by implementing strategies that mitigate these vulnerabilities. After analyzing dozens of AI developer tools in our investing work, a clear pattern emerged that we've distilled into a single principle: **"Build on the model, don't band-aid it."**

The winners in the developer tool space are those that grow stronger as foundation models improve. These include AI-native coding and debugging platforms that become more powerful as models gain advanced reasoning capabilities, and enterprise governance tools that become more critical as AI usage scales across organizations and regulatory requirements tighten. The losers merely patch current model limitations, offering short-term utility, but are unable to sustain value as models evolve. Examples include basic guardrail systems that become obsolete as models develop better built-in safety mechanisms, and simple context management tools that lose relevance as models natively handle longer contexts and better memory.

Three principles separate winners from losers in this space.

- **Synergy vs. Band-Aids:** Tools that create synergy, leverage each new model release as a feature upgrade to their underlying platform. When GPT-5 arrives with better reasoning, these tools automatically become more valuable because they're designed to harness whatever capabilities the model provides. Band-Aid tools, by contrast, focus on the gaps of the existing models and build their entire business around them. While solving targeted problems is important, Band-Aid tools fail to appreciate what problems can be solved by the foundation model companies themselves. They're constantly playing defense, fixing problems that foundation models are actively working to solve.
- **Integration Depth:** Successful tools embed into existing pipelines like CI/CD systems, IDEs, Git, etc, and add value regardless of the model's raw quality. They become essential parts of the development workflow that would be difficult to remove even if better alternatives emerged. Tools that sit outside the core stack vanish the moment the foundation model natively fixes the problem that the developer tool was calling their core feature.
- **Roadmap Alignment:** Successful developer tool roadmaps track closely with the underlying foundation model's roadmap. They anticipate where models are heading and build infrastructure that will be more valuable as models advance. Whereas failing developer tool roadmaps only track user complaints without a broader vision, and as those complaints evaporate with every model update, so does the developer tool.

While these principles apply broadly to all developer tools and can help build defensibility against displacement, they are insufficient as horizontal developer tools compete in a crowded market. To build a true moat and lower displacement risk, AI developer tools should turn to vertical specialization and focus deeply on specific industries where domain expertise, compliance requirements, and specialized workflows **create natural moats that resist commoditization and protect against displacement**.

Exiting to Foundation Models or Staying Independent?

The looming displacement risks for developer tools raise a blunt, unavoidable question: What realistic exit paths exist when foundation model companies dominate?

OpenAI's $3 billion acquisition of Windsurf grabbed headlines, but not all tools face the same acquisition prospects. The reality is that foundation model companies will mostly **build rather than buy**, especially when it comes to anything they consider core strategic innovation or mission-critical infrastructure. For example, anything related to model architectures, training processes, or alignment.

OpenAI's decision to acquire Windsurf highlights a strategic calculus directly tied to its own displacement risk factors:

- **Time-to-market advantage:** OpenAI gained instant AI-coding platform capabilities, addressed competitive threats from Google and Anthropic, and achieved market dominance, effectively addressing potential displacement by immediately acquiring capabilities rather than slowly building them.
- **Product validation shortcut:** Facing "Feature Commoditization Velocity" themselves, OpenAI acquired a proven developer experience platform and enterprise integrations rather than risking an unproven internal build that might still lag specialized tools.
- **Talent and focus preservation:** Windsurf's team expertise in developer UX and enterprise compliance differentiates them from other developer tools that suffer "API Proximity" risks. By acquiring Windsurf, OpenAI gets its specialized talents and is able to maintain focus on core model R&D.
- **Data advantage:** Perhaps most valuable, acquiring Windsurf created a direct data pipeline from developer interactions back to OpenAI's model training, transforming a potential displacement threat into a competitive reinforcement mechanism. This data advantage puts OpenAI in a good position to build the next-generation coding agents.

Not all developer tools will (or can) choose Windsurf's path, however. Cursor notably made a different strategic move, rejecting OpenAI's acquisition overtures and instead raising another round at a $9 billion valuation. But this decision also significantly increases its displacement risk: Cursor now competes directly with the combined force of Windsurf and OpenAI. The company is betting that its product strength, growth, and distribution can sustain independence in a more aggressive competitive environment. While this path is riskier with OpenAI entering the space, Cursor's scale and capabilities may still carry it through.

Whether acquisition or independence is the superior strategy remains to be seen. We're watching a real-time experiment unfold: Windsurf integrated into OpenAI's stack versus Cursor building alone. This matchup will test whether tight foundation model integration or independent execution creates more durable value in AI developer tools.

Still, neither strategy matters if startups don't first survive. The core challenge is mitigating displacement risk by building defensible products that make both acquisition and independence viable outcomes. Without that, there's no path to any exit.

Looking Ahead: The Case for Vertical MCPs

One of the biggest opportunities emerging in vertical AI developer tools is building vertical Model Context Protocols (MCPs), developer tools designed specifically to help AI agents interact with vertical software.

Vertical MCPs aren't for human developers but for AI agents to access and operate complex, domain-specific applications like EHRs, financial systems, or legal platforms. They expose software capabilities in a standardized, safe, and reliable way, translating AI instructions into precise, compliant actions. In other words, vertical MCPs are like Postman or Zapier, but designed for AI agents working within vertical domains. They are the infrastructure that lets AI agents discover what a vertical tool can do, parse commands accurately, maintain context, handle errors, and safely execute domain-specific tasks.

Unlike traditional developer tools focused on developer productivity, vertical MCPs prioritize trust, compliance, and tight integration to prevent costly errors in high-stakes workflows. They solve problems too complex or regulated for foundation models to handle alone.

The key buyers of vertical MCPs will also look different. These tools are sold primarily to vertical SaaS providers who want their software to be AI-agent compatible from day one or vertical enterprises that want AI agents to operate the SaaS platforms they use. It's less about winning over generalist developers who want to build faster and more efficiently, and more about embedding an essential layer of AI interoperability that protects compliance and enhances agent-driven automation.

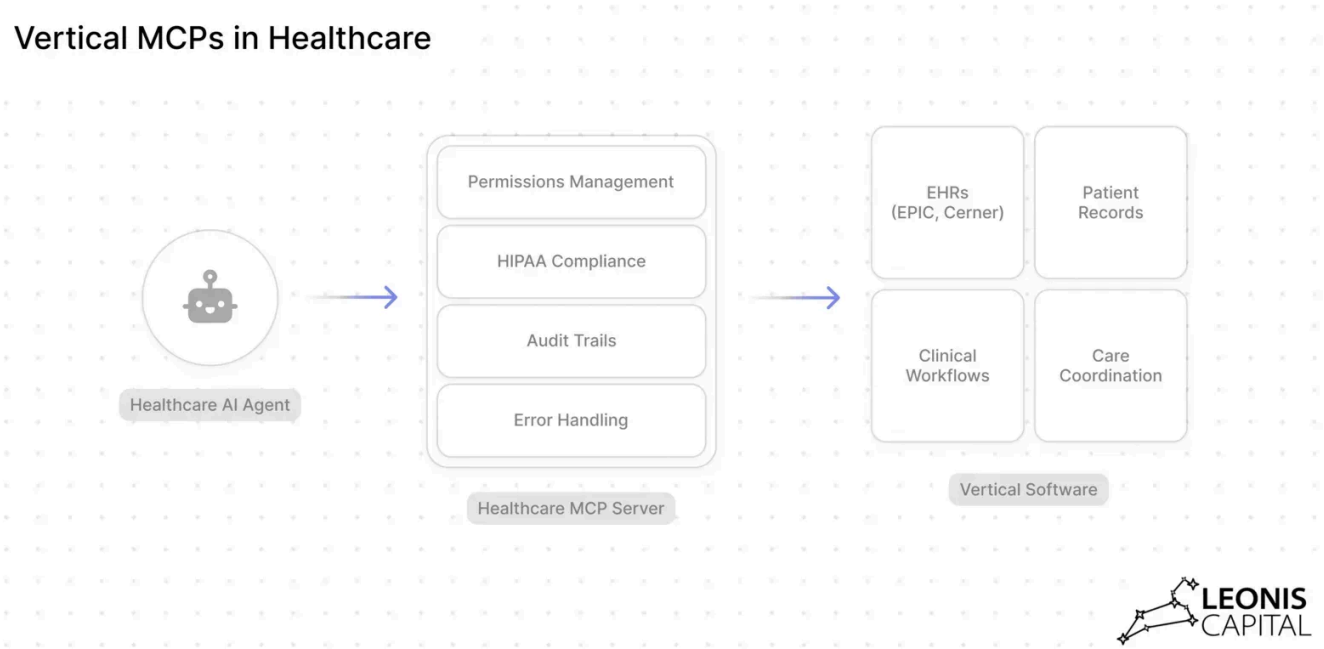


Figure 3: An example of a vertical MCP in healthcare

In the healthcare industry, this would be an MCP server built for Epic, one of the most widely used EHR systems. This could give AI agents structured, compliant access to core functionality like retrieving patient history, updating charts, or ordering tests. While Epic does offer APIs today, they're often read-only and are not ready for real-time agent use. A vertical MCP server would sit between the AI agent and the EHR, translating agent instructions into precise, permissioned write operations and returning structured responses or helpful error messages. This abstraction would eliminate the need for fragile prompt engineering and unlock safe, agent-driven automation of high-value workflows such as discharge planning or care coordination, without rebuilding the entire application stack.

This trend points to a broader shift in AI software development towards **MCP-first development**. Instead of bolting on AI capabilities after the fact, teams will design their products from day one to be usable **by both humans and AI agents** by building the MCP server alongside the GUI. This is a foundational shift in how vertical software is architected. In fact, there's even a scenario where vertical MCPs extract more value than the underlying SaaS platforms. For example, in healthcare, Epic EHR's outdated software frustrates users, creating an opportunity for an MCP and AI agents to take over by abstracting Epic's complexity, simplifying workflows, and commoditizing its backend into a mere data repository while the MCP captures the high-margin value of AI-driven automation. However, unpacking the full implications of this shift deserves its own deep dive.

Building for Future Paradigms

The AI developer tools landscape is defined by constant reinvention. Interfaces shift, abstractions collapse, and what felt defensible six months ago is suddenly legacy. In this environment, the core challenge isn't just building something useful, it's building for paradigms that haven't fully arrived yet. The most durable developer tools are the ones that anticipate where the interface between AI and software is going, not where it is today. They actively think about displacement risk and how to address it.

The best developer tools won't compete with foundation models; they'll **compound their power**. This means aligning with shifts like agent-first workflows, protocolized interaction (e.g., MCP), and the growing need for AI-native infrastructure that works out of the box, not just for humans, but also for machines.

Valuation alone tells you little. The more important question is: ***Does this tool gain relevance as the models improve?*** If the answer is no, you're fighting against gravity. If the answer is yes, you're building leverage.

The tools that matter in the long run are the ones that treat the rapid pace of AI progress not as a threat, but as an input to their product roadmap. That's where real defensibility and real upside live.