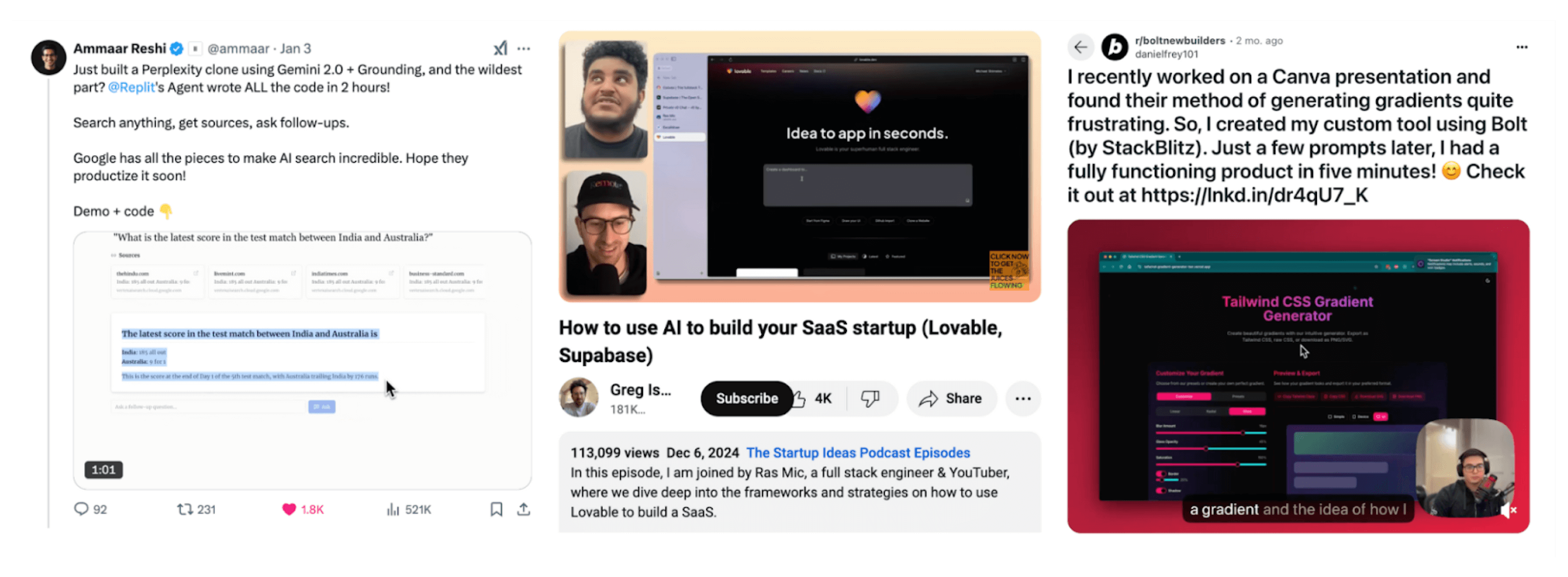
**From Prompt to Product: The Rise of AI-Powered Web App Builders**

Over the past few months, thousands of builders have created websites and web apps using tools like Bolt, Lovable, and v0. The results, shared across X, Reddit, and YouTube, seem nothing short of magic: people are building interactive, dynamic products — and even generating revenue – without coding.

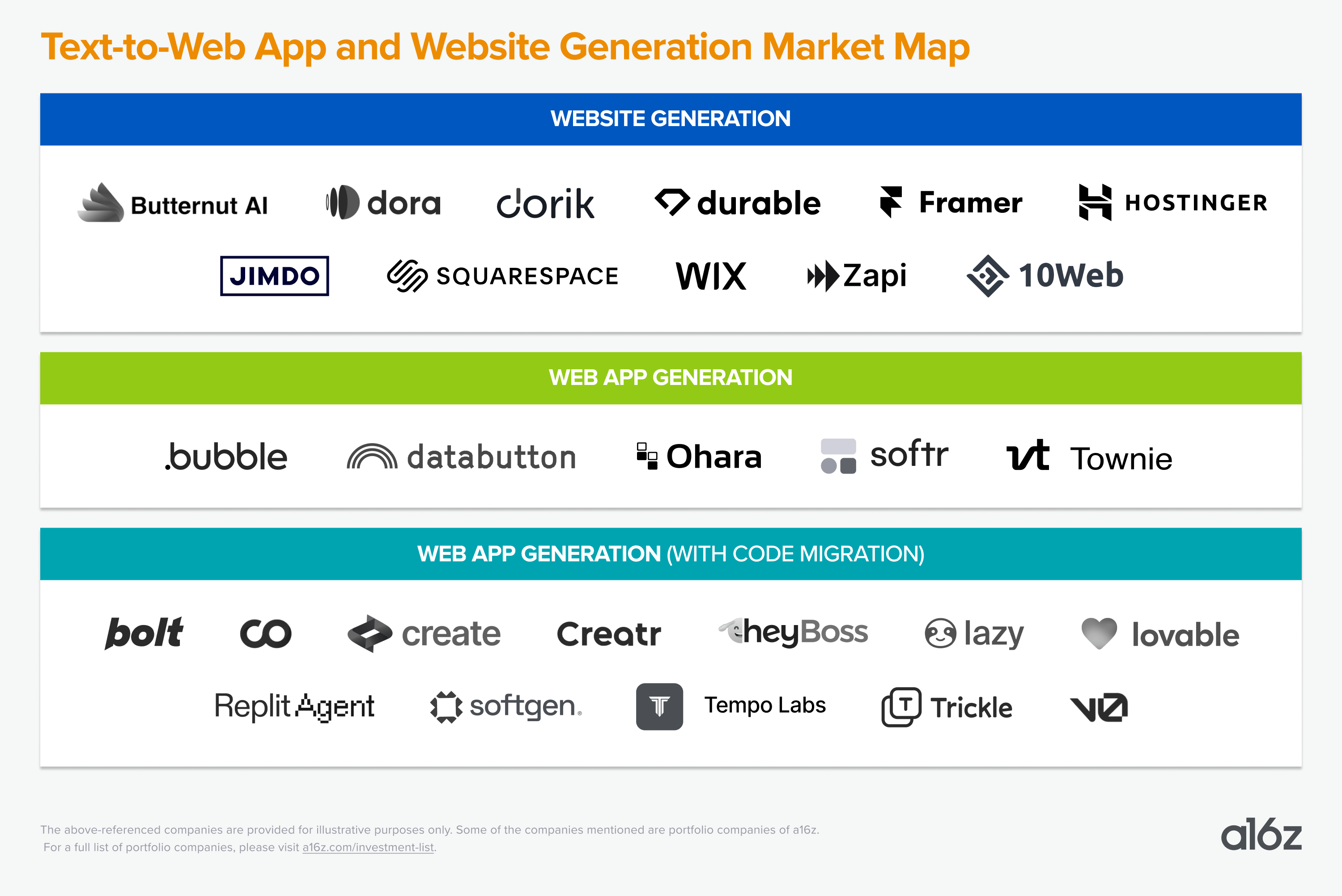
**[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-1-Three-Image-Collage.png)**

The ability to create a functional product simply by chatting with a bot has caught the attention of technical and non-technical users alike, leading to impressive growth for startups in this space. Based on publicly released metrics, Bolt [scaled to](https://www.youtube.com/watch?v=MJRgjb6BYck) a $20M revenue run rate, while Lovable [grew to](https://x.com/antonosika/status/1883622036872065217) $10M just two months after each company began monetizing.

There is a new, LLM-based app stack emerging. Before site generation solutions, if you asked a technical person “What’s your stack?”, their answer would be a list of libraries. But this new form of app creation — via text — has introduced a new level of abstraction. Instead of libraries, users who don’t often code will reach for a suite of pixel-based design generation, image generation, site generation, and hosting tools. The new abstraction for this fast growing group of users is pixels and natural language, compared to SDKs and frameworks for developers.

What products are builders using to make websites and web apps with AI? What are the main use cases, and what types of users are they attracting? Below, we dive into how these products work, their technical limitations, and what features are on the horizon.

**The current landscape for text-to-web app and website generators**

**[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-2-Market-Map.png)**

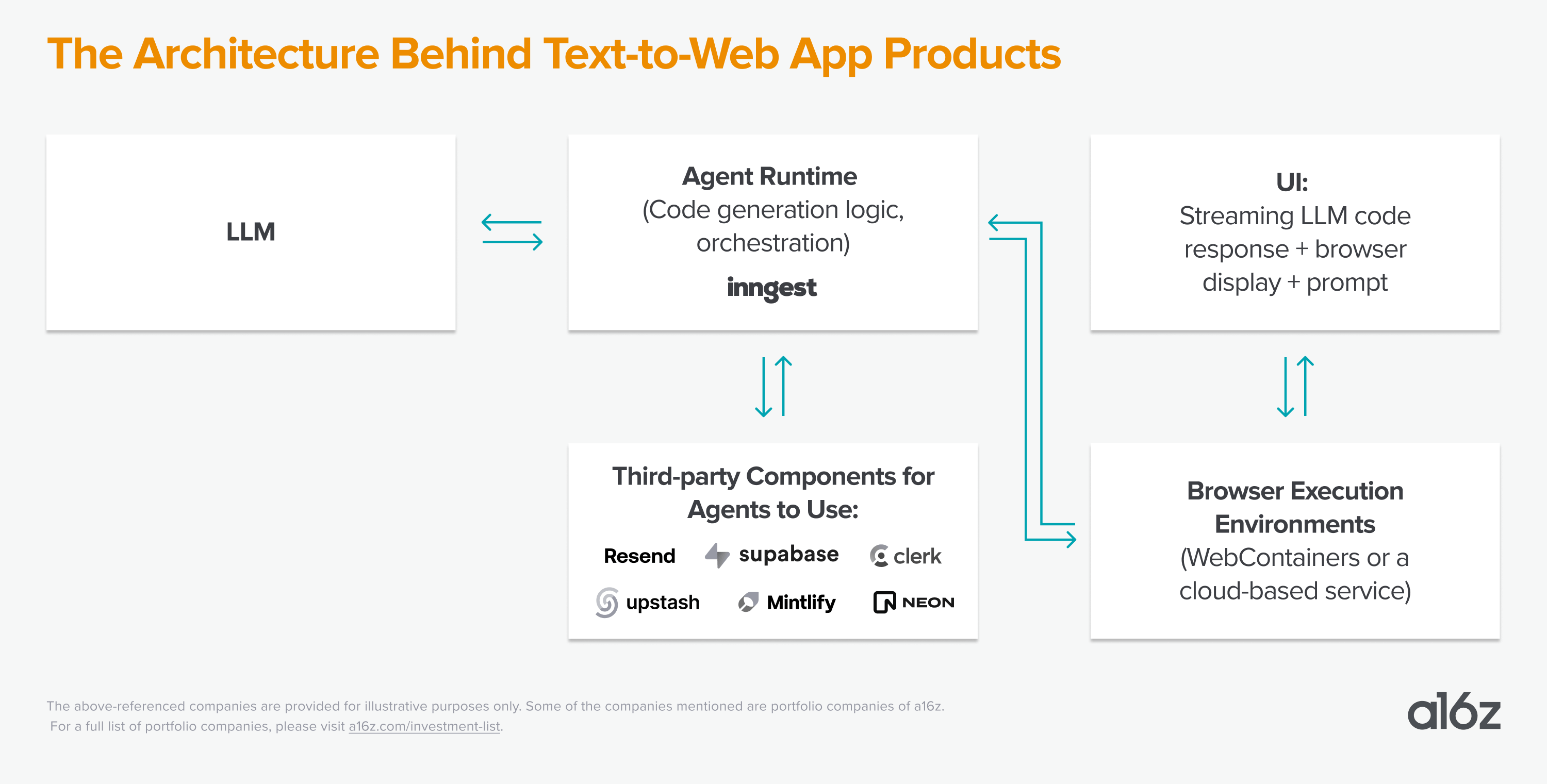
*Note: This market map is limited to companies that focus on turning text into a web app or a website. There’s a much broader range of ancillary products — for IDE-based code generation and editing, design iteration — that we’ll touch on below.*

There are two core product distinctions here:

1. **Static website vs. application generation:** We define static websites as web pages that are primarily intended to convey information and have basic interactivity (e.g. submitting a contact form or booking an appointment). Applications are more dynamic — users can perform more complex actions. Static websites are easier to generate, and can be template-based. Dynamic applications are more stateful. This type of application requires third-party tools like database and authentication systems to support use cases like CRM, form submission, and user login. Think of the difference between a website for a small brick-and-mortar business (you might be able to view information and schedule a consultation) and a web app like YouTube (you can like, comment, or even upload your own content).
2. **Enabling code export:**All of the products that generate web apps, and some of those that make websites, are creating code. But some allow you to export this code to edit in another application, while others don’t expose the code and instead require you to deploy either: (1) through their platform; or (2) through an integration partner. For developers, the ability to download, view and edit code is important. Most aren’t shipping products they’ve generated in a text-to-web app builder without building on or editing it in their IDE. For non-technical users, the ability to export code may be less important.

**How do text-to-web app products work?**

There are a number of elements involved in generating websites using LLMs. Most of the products in this category follow a similar (highly simplified) architectural diagram:

**[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-3-Architecture-1.png)**

At its core, the LLM generates code examples based on user input from the UI. The generation is then run through middleware logic that tracks files, code changes, and third-party API calls. This is often a long-running system process — we have seen agent runtimes like [Inngest](https://www.inngest.com/" \t "_blank) adopted as part of the stack.  
  
One of the most critical parts of this system is the third-party component calls. Like a human developer, a coding agent needs to utilize existing packaged libraries for sending emails, storing stateful data in a database, caching data, or adding authentication components. But unlike a human, who refers to documentation to discover new components, a coding agent refers to [llms.txt](https://mintlify.com/blog/simplifying-docs-with-llms-txt) to discover ways to use tools.

**Why did these products take off now?**

Models trained on a vast amount of high-quality coding and reasoning data made it possible to generate executable code. Given their widespread usage in web development, JavaScript and TypeScript dominate the distribution of publicly available application code. As a result, models are particularly well-equipped to generate application code for the web.

The maturity of the web frameworks also made it possible for agents to create new elements much faster. Browser runtimes are a natural lightweight “environment” for site generation products, as opposed to heavy distributed apps that require more compute and state management between services. Over the past few years, the business logic of applications has also moved further to the client side, making it even easier to run everything in a browser.

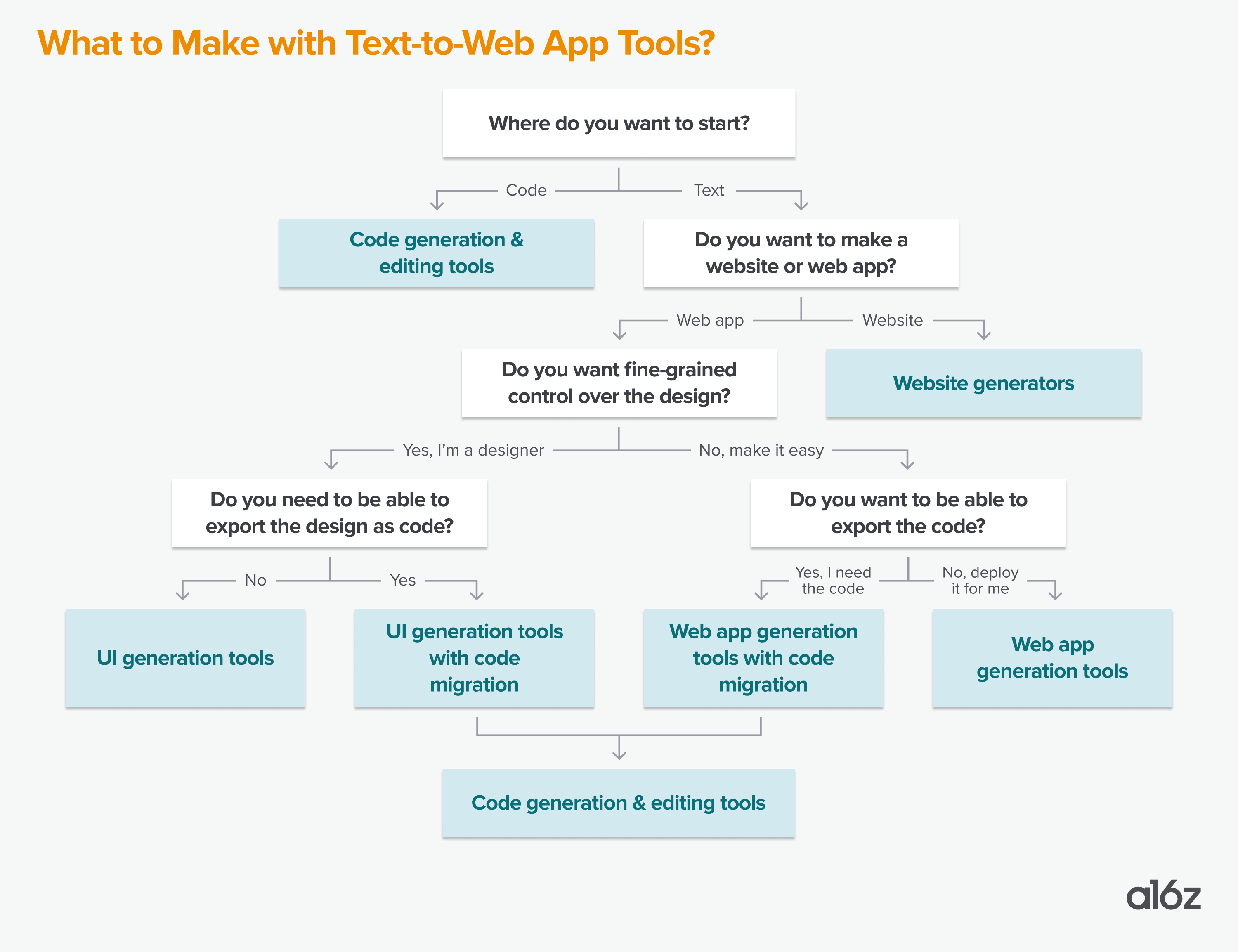
As a result, rendering and testing generated applications has become significantly easier. For example, a Vite + React application can be generated and displayed to the end user in real time. In many ways, site generators have evolved into dynamic, less deterministic scaffolders that consistently deliver unexpected but useful results, helping users get closer to their desired app.

Lastly, well-made libraries and SDKs from companies like [Resend](https://resend.com/), [Clerk](https://clerk.com/), [Supabase](https://supabase.com/" \t "_blank), [Upstash](https://upstash.com/" \t "_blank) now support long tail integration needs. The error rate is high when asking a LLM to generate an authentication system ,or an email-sending API from scratch. However, if we reframe the task as selecting and integrating existing libraries, rather than building everything from the ground up, the solution becomes much faster and more reliable.

**How do users decide which tools to use?**

We’ve scoured thousands of posts about text-to-web app tools and talked to dozens of customers, ranging from highly technical developers to everyday consumers.

We made a simple flow chart to illustrate what we’ve found — beginning with the question of where the user is looking to start.

**[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-4-Flow-Chart.png)**

If you’re a technical user who wants to start with code, you’ll probably just go to an AI-powered code generation and editing tool like Cursor or Windsurf. But if you want to start with a text prompt, you’ll probably explore another set of tools for website or web app generation.

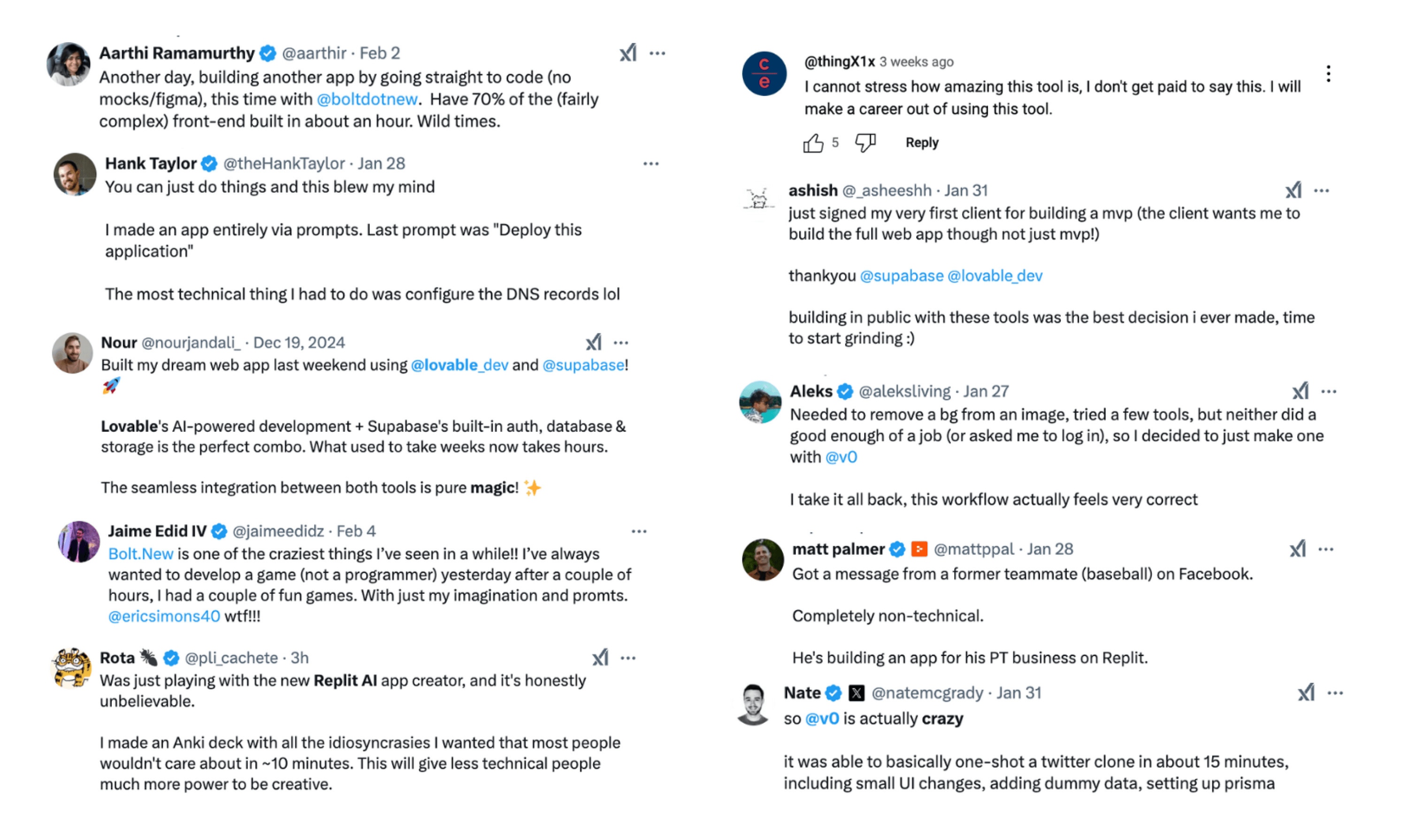
It’s worth mentioning that this chart includes a category of products we don’t explore in depth: design-first UI generators. Tools like Bolt and Lovable enable you to edit design with iterative text prompting, but don’t enable the finegrained control (down to moving elements) of something like Relume or Uizard.

Finally, many technical users end up using these products in combination. They might start by generating a basic web app in a product like Lovable, export the code to Cursor, and add components from a more design-oriented tool like [21st](https://21st.dev/). Alternatively, they might start in a design product like Magic Patterns and move the code to a tool like Bolt (example [here](https://x.com/donvito/status/1880543692131553751)).

**Do they work?**

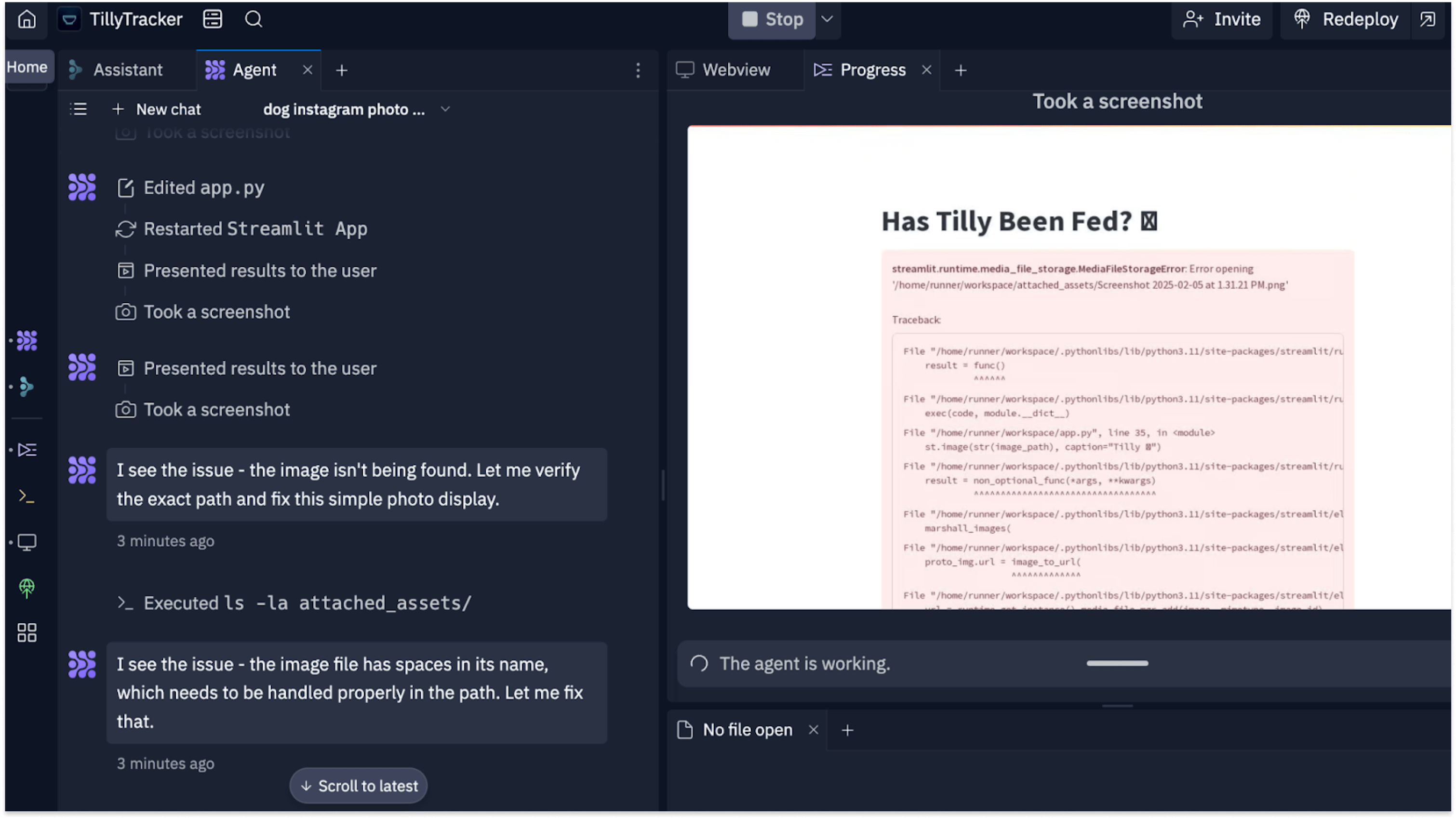
For users who have never been able to deploy an app before due to lack of coding skills, these products can feel like magic. And technical users appreciate the ability to get started quickly and simplify hours of development time into a few simple prompts.

It’s not unusual to see people singing the praises of these products:

**[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-5-Positive-Feedback.png)**

However, there’s a limit to what they can reliably generate today — it’s comparable to working with a junior developer who knows the basics but can easily get stuck in a “doom loop” of bugs, especially as a project gets more meaty or complex.

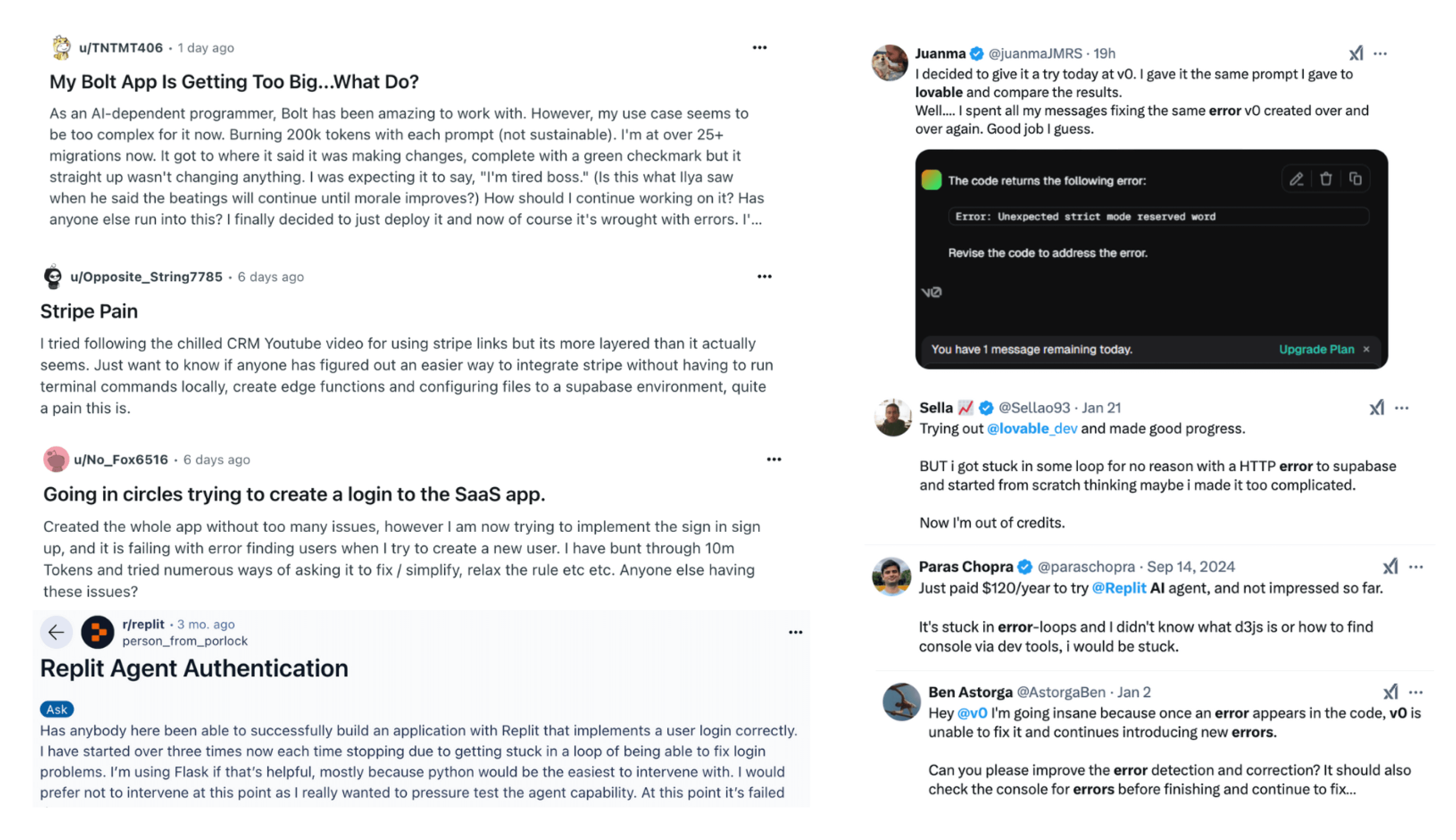
Something simple like uploading a photo for the top of your website might work perfectly, or it may trigger a cascade of errors that takes you 45 minutes to debug. The great thing about these products is that you can point out errors and ask to generate a fix, but it may take a while and require multiple approaches!

**[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-6-TillyTracker.png)**

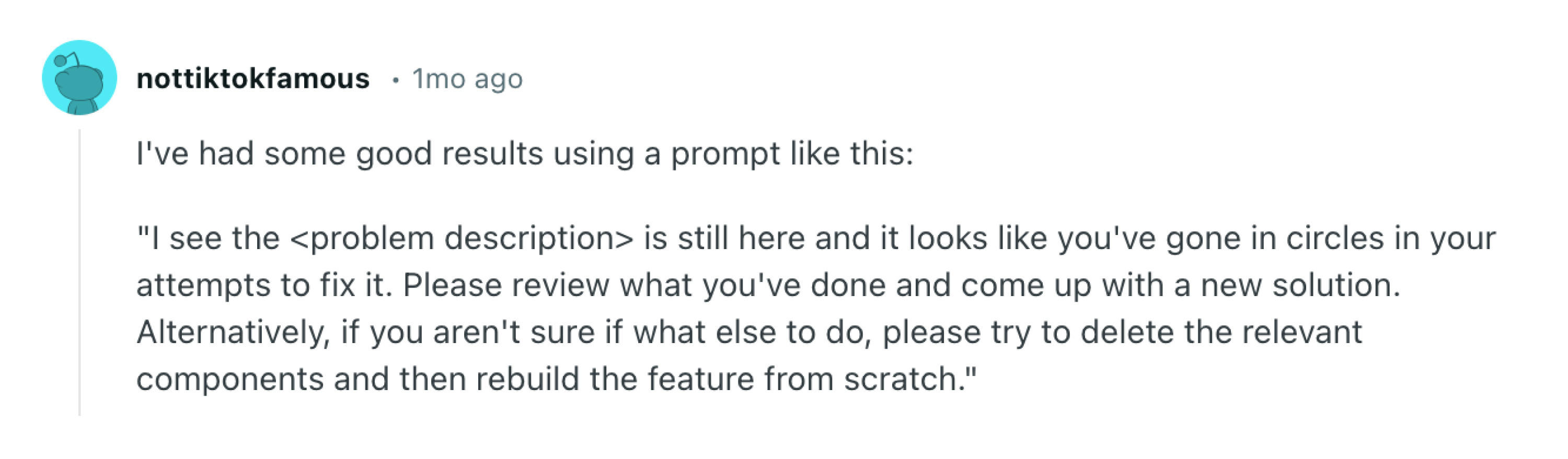
Why is debugging hard for site-generating agents? First, finding the right code to debug is a reasoning and search problem: how to locate the right segment of business logic requires the agent implementation to locate the right files / lines of code to debug. This doesn’t always happen deterministically, especially when a long context is fully loaded through a model.

Another reason debugging is difficult is code has deep dependencies. Changes in one part of the system can cascade unpredictably across multiple layers, and debugging (especially for an LLM agent) becomes even harder when these dependencies introduce asynchronous execution, race conditions or other side effects.

A few issues that commonly arise:

**[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-7-User-Feedback.png)**

1. **Integrations are difficult.**Especially for non-technical users, adding elements like a database, authentication, and payments can cause significant issues. There are many potential points of failure — from setting up an account on another service (e.g. Stripe, Supabase), to correctly inserting the API key, to troubleshooting errors, particularly if the other party’s process or documentation changes.
2. **Bugs persist.**It’s not uncommon to get stuck in a frustrating loop of never-ending bugs — particularly when a cheery AI assistant is repeatedly telling you that the error is solved (when it clearly isn’t). To get out of this “doom loop,” these tools sometimes need additional guidance on how to approach a problem or you simply have to instruct them to start from scratch.

[](https://d1lamhf6l6yk6d.cloudfront.net/uploads/2025/02/Inline-8-User-Feedback-Single.png)

1. **Code can get “too big” for the product.**Errors often occur when you’ve generated a lot of code that begins to exceed the product’s ability to manage it. Sometimes the product will directly tell you that your project is getting too large — but it may just start deprecating performance or even delete / re-write code in a way that doesn’t meet your objectives.

As a result, we find that these tools are mostly used to dramatically reduce the costs of prototyping — which is a huge pain point today. In our conversations with customers, we found that many are testing out these products, but not switching over if they plan to ship a website to a large audience and/or monetize it. It’s hard to rely on these tools working 100% of the time, and it’s often difficult to iterate on what you’ve created in a predictable way. A seemingly simple change may trigger an issue somewhere else in the codebase that is hard to debug.

However, as with other modalities of AI-generated content (e.g. images, video, music), we expect to see a steep improvement curve. The capabilities are still incredibly impressive, and they’re improving every day. We expect these tools will move up the value stack as they mature. And we’re excited to watch a new AI-native debugging flow emerge that tackles the unique challenges of this space.

**What are people building with text-to-web app tools?**

When we took a closer look at how people are leveraging these tools, we found different use cases based on user backgrounds. They generally fell into three categories: 1) consumers, 2) developers (both new and experienced), and 3) consultants.

**Consumers**

For the first time, everyday consumers are able to build truly interactive web apps — not just landing pages or template-based systems. We’ve seen enthusiastic adoption from people who can finally make things that used to require the ability to code.

What are they building? We’ve seen a long tail of highly personalized apps that cater to a user’s specific interests and needs. Today, these products are typically relatively simple — they don’t require things like user authentication or payment — and the main use cases are fun, fulfillment, or basic utility.

A few examples:

*A dad built a bedtime story creator using Bolt that writes a custom story based on your child’s information (e.g. name and age) and interests.*

*A v0 user built her own personal finance tracker that ingests data on her spending and surfaces insights.*

*A designer built a Flappy Bird x Frogger game to play on his iPhone using Replit.*

**Developers (both new and experienced)**

There’s a wide range of developers — from those who write code at work to users who understand technical concepts on a high level. We’ve found that for the developer or developer-adjacent audience, site generators are treated as a 10x better new scaffolder. Instead of running *npx create-react-app*, this group can now get further by generating their own customized starter templates, making the barrier to edit code much lower than before.

Much as image generators streamlined the creative process for artists, site generators solve the “blank canvas” problem for developers.

A few examples:

[*A Sudoku game*](https://x.com/imolivercom/status/1879944480264188404)*built with Bolt by a user who wants to play the game for free.*

*A professional grade*[*audio meter*](https://x.com/lucasmcgartland/status/1879648528911491323)*created in v0 by a startup founder for their product.*

*A tool to test and debug Three.js creations in the browser built with Bolt.*

**Consultants/Agencies**

There’s an interesting category of user between consumers and developers: consultants/agencies. These people typically aren’t engineers, but are hired to create websites or web apps for businesses — often solopreneurs and SMBs.

Imagine the person hired to make the website for your local restaurant or tax accountant. Pre-generative AI, they were likely using a no-code tool like Squarespace or Wix. Now, some of them are exploring text-to-web app tools as a way to (1) prototype and iterate more quickly, and (2) take on more complex projects.

Most consultants don’t publicly share their work and the tools they use to make it – but here are a few examples of the types of apps they might create:

*A website for a restaurant in Helsinki built with Bolt*.

*An ecommerce store built with Replit Agent.*

*A website + store for a coffee shop in Lagos.*

**What’s coming next?**

We’re witnessing near-daily advancements in this fast-moving space, with teams working around the clock to deliver the next breakthrough. A few ideas on how it might evolve:

1. **Diverging products between user personas.**Today, most products in this space are “everything to everyone” — they serve all users, from the most technical developers to everyday consumers. However, we think the ideal product likely differs for each user persona, or even use case. For example, the best text-to-web app tool for consumers might be mobile-first, have lots of “guardrails,” and prioritize integrations with platforms like Instagram and Shopify. The best developer-facing tool will look very different!
2. **Possibility to unlock a new segment upmarket.**We are already seeing less technical users generate their own productivity tools for internal use. These tools are often downloaded and deployed within a company’s firewall, meaning they are rarely visible to the public. However, given these tools are used at *work*, there’s an opportunity for site generation platforms to add more enterprise features and move upmarket. This is likely to intersect with the existing low-code internal tools market.
3. **Packaged integrations with common partners.**One of the most common complaints we hear is that it’s hard to add integrations. This is especially important if you want to build an app used by other people. You might need to accept payments, create user log-ins, save information in a database, or pull content from another website. We expect to see products build “one click” integrations with the most common tools their customers use, making it truly seamless, rather than requiring users to sign up for multiple services, pull API keys, and study docs. There will also be generation tools for specific components or essential tools for agents to use when generating an app — like this [v0 for email](https://new.email/) built by Resend.
4. **Pixel-level design control.**These products often start with a default design and aesthetic. You can edit it by prompting, but don’t have nearly the level of control as in a product like Figma —which is a problem for design-oriented consumers or enterprises. We’re eager to see these products provide more fine-grained design editing capabilities for those who want it, then convert these changes back to code.
5. **Pricing clarity and education.**Most products in this space have some kind of usage-based pricing model — you pay a monthly subscription, but it only includes a certain number of tokens (or credits). If you want more, you have to buy them or upgrade. But users don’t know how many tokens they’ll use when they get started, which makes it hard to estimate the cost of building their website. There’s also a lot of frustration around wasted tokens, and a mini-industry of courses/content creators that help new users figure out how to “use their tokens effectively.” We expect companies can bring these capabilities in-house and better guide users through the process of learning how to prompt efficiently.

**What might the text-to-web app space look like in the long run?**

It’s also worth asking if this capability (generating web apps from text prompts) will stay in standalone tools, or get integrated into the existing products where users are already doing their work.

The new Replit agent is an interesting example of this. And web app generation becomes even easier when packaged inside Replit’s existing product – spinning up authentication and databases is seamless within the ecosystem, you don’t need to go to a third party provider.

We can imagine that other platforms are interested in exploring this surface area and will have advantages in doing so for their existing customer base. For example: design tools like Figma could move into code, enabling designers to go from prompt to editable design interface to deployed website. Or AI coding tools could add browser interfaces with a friendlier consumer UI.

We’re also curious to see if the large LLM players decide to get into this space. As companies like OpenAI and Anthropic are increasingly focused on code, it may make sense for them to add interfaces for users to program and even deploy simple apps.

Today’s text-to-web app tools may continue to scale on their current paths, or we may see them double down on specific verticals that aren’t well-served by existing tools. For example, we can imagine a product in this space deciding to focus on SMBs and going deep into this workflow — making it easy for a small business owner to set up a custom domain, generate marketing materials, schedule appointments, or run search ads.