CREATING CUSTOM AI AGENTS

AI agents have become essential tools in modern development workflows. Whether you're building apps, testing APIs, fixing bugs, or securing open-source packages, there's an AI agent that can streamline the task. By choosing the right tool for your needs, you can save time, reduce errors, and boost collaboration—making development more productive and enjoyable.

AI Agents in Software Engineering: The Next Frontier of Development

Explore the growing role of AI agents in software engineering, from automating coding tasks to enhancing development workflows. Learn how AI is shaping the future of software development.

Imagine if, instead of jumping between different apps for emails, documents, shopping, and scheduling, you could just tell your device what you need, and it would take care of everything. No more switching tabs or figuring out which tool to use—just a seamless digital assistant that understands you.

This is the promise of AI agents. Unlike traditional apps, which perform specific tasks in isolation, AI agents can handle a wide range of requests using natural language. They can learn from your preferences, anticipate your needs, and act on your behalf, like a highly capable personal assistant but powered by artificial intelligence.

Recent advances in AI have made this future possible. AI agents aren't just changing how we interact with technology; they're reshaping the entire software industry. From automating workflows to revolutionizing user experiences, they represent the biggest shift in computing since the rise of smartphones.

Let's explore how AI agents work, the different types, and the impact they will have on software development.

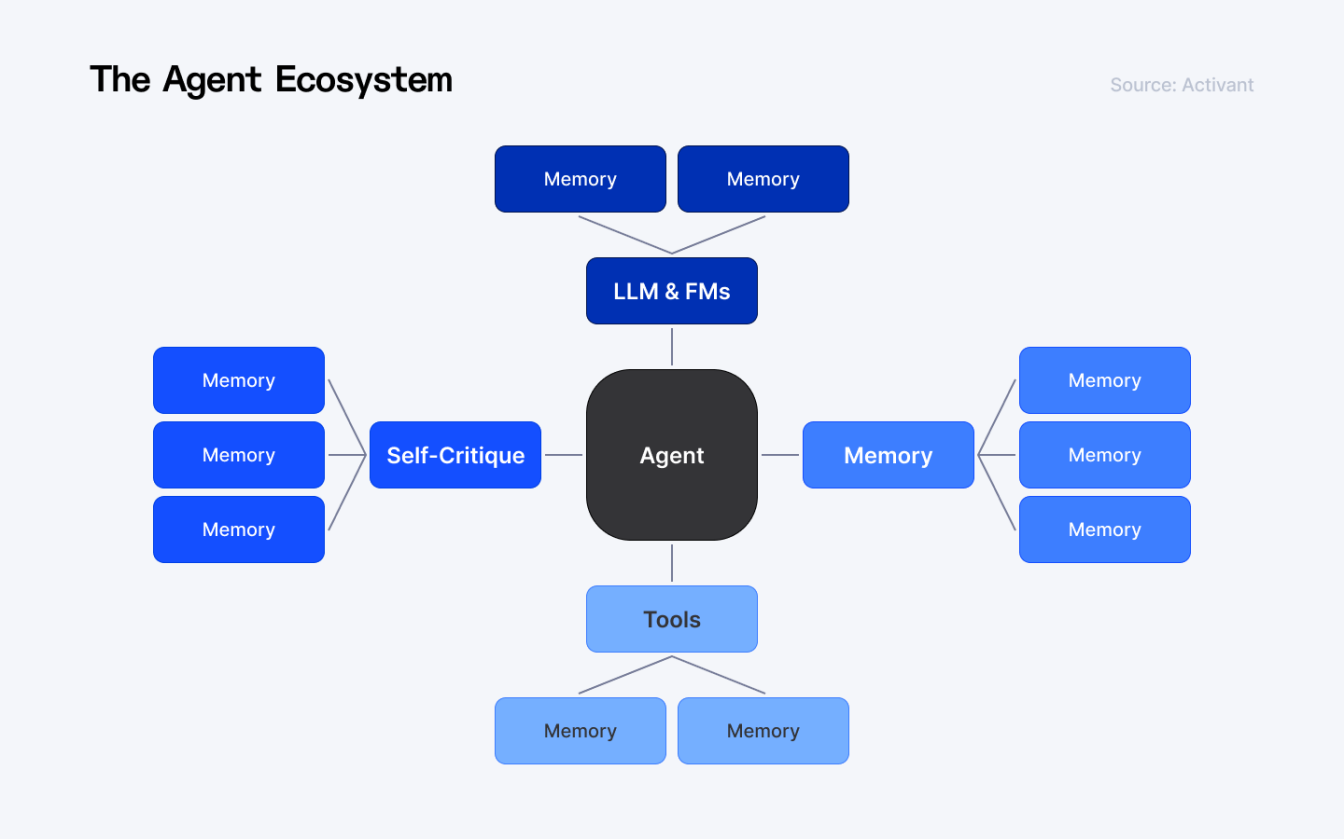
What Are AI Agents?

AI agents are intelligent software tools that can think, learn, and act on their own. They don't just process information—they make decisions, adapt to new data, and take action without needing constant human input. Unlike traditional AI models that require instructions for every step, AI agents can operate independently, solving problems and automating tasks dynamically.

In software development, AI agents can write and optimize code, debug errors, test software, and even manage deployment pipelines. Instead of developers manually reviewing lines of code, an AI agent can detect inefficiencies, suggest improvements, and even implement fixes in real time.

For example, imagine an AI agent specialized in debugging. It could automatically scan a codebase, identify performance bottlenecks, and rewrite problematic sections—saving engineers hours of tedious work. Another agent could streamline DevOps, managing CI/CD workflows by running automated tests, monitoring infrastructure, and rolling back faulty updates before they cause major issues.

Some AI agents go even further, acting as project managers. They can track development progress, assign tasks based on team capacity, and predict potential delays—essentially functioning as an AI-powered scrum master.



How AI Agents Work

AI agents combine advanced algorithms, machine learning, and decision-making processes to operate autonomously. Here's a breakdown of how they function:

Architecture and Algorithms

AI agents rely on complex systems to process vast amounts of data and make informed decisions. Machine learning enables them to learn from experience, improving their accuracy and efficiency over time. Think of them as self-improving problem solvers, constantly refining their approach based on new data.

Workflow and Processes

An AI agent starts with a goal, formulates a plan, executes necessary steps, and adapts based on real-time feedback. This cycle of learning and adjusting ensures

that AI agents continually enhance their performance, much like a developer refining code after testing.

Autonomous Actions

Unlike traditional software that waits for instructions, AI agents take action on their own. In software development, this means automating tasks like code reviews, vulnerability detection, or even writing documentation—saving engineers from repetitive work and allowing them to focus on more complex challenges.

The Evolution

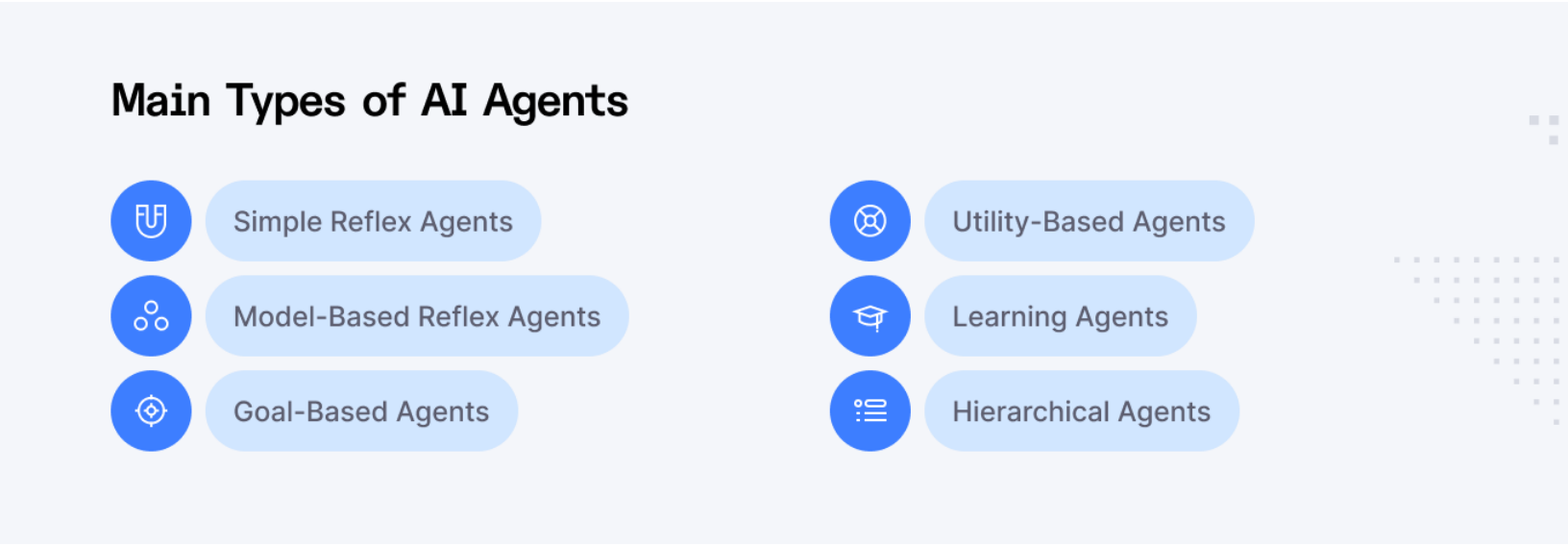
When ChatGPT first arrived, it could discuss theories and explain concepts but struggled with basic arithmetic. However, once connected to an external calculator, its capabilities improved. Today, AI agents are evolving beyond simple responses. Large language models (LLMs) act as their "brains," while additional algorithms and plugins extend their functionality. Take AutoGPT, for example—a proof-of-concept AI agent that can autonomously execute marketing campaigns, like scanning Reddit for product-related questions and generating responses.

The Future

Now, companies like GitHub are pushing AI agents even further with multi-agent systems, such as GitHub Copilot Workspace, where multiple AI agents collaborate to automate complex software development tasks. Imagine an AI that not only writes code but also tests, deploys, and optimizes it—effectively acting as an entire development team.

Types of AI Agents

Organizations create and deploy different types of AI agents. Here are the most common examples:



1. Simple Reflex Agents

These agents operate purely on predefined rules, responding to specific conditions without considering past experiences. Think of them as AI-driven reflexes—if X happens, they do Y. They're useful for straightforward tasks like automatically resetting passwords when a user types "forgot password." However, they can't handle more complex decision-making beyond their set rules.

2. Model-Based Reflex Agents

Unlike simple reflex agents, these agents build an internal model of their environment to make better decisions. Instead of just reacting, they analyze patterns and consequences before acting. For example, a model-based agent in software testing can track past bug reports and anticipate potential errors in future code updates.

3. Goal-Based Agents

These agents don't just react; they think ahead. They evaluate different possible actions and choose the best path to achieve a specific goal. This makes them ideal for complex tasks like natural language processing (NLP) or robotics. A chatbot powered by a goal-based agent, for instance, can analyze multiple responses before selecting the most relevant reply.

4. Utility-Based Agents

These agents aim to maximize a specific outcome by comparing different choices and selecting the one with the highest value. They go beyond simple goals by considering multiple factors. For example, a travel booking agent can find the best balance between price, travel time, and layovers—not just the cheapest ticket.

5. Learning Agents

These agents continuously improve by learning from past experiences. They use feedback mechanisms and adaptive learning techniques to enhance their performance. A coding assistant AI, for instance, can refine its code suggestions over time based on how developers use its recommendations.

6. Hierarchical Agents

These agents work in a structured, multi-tier system where higher-level agents break down complex tasks and delegate them to lower-level agents. It's like a well-organized company: the CEO (high-level agent) sets strategic goals, while managers (mid-level agents) assign tasks to employees (low-level agents). In software development, a hierarchical agent system could manage a complete CI/CD pipeline, automating everything from coding to deployment.

Agent Type	Characteristics	Examples	Real Tools
Simple Reflex Agents	Acts based on predefined rules without considering past events	Password reset systems, Basic chatbots	Rule-based spam filters, IFTTT
Model-Based Reflex Agents	Uses internal models to understand patterns before acting	Software testing systems, Weather prediction systems	Selenium testing framework, Jenkins
Goal-Based Agents	Evaluates multiple options to achieve specific goals	Navigation systems, Advanced chatbots	GPT-3 powered chatbots, A* pathfinding algorithms
Utility-Based Agents	Chooses actions that maximize specific values or outcomes	Travel booking systems, Resource allocation tools	Expedia's booking engine, Google Maps route optimizer
Learning Agents	Improves performance through experience and feedback	Code assistants, Recommendation systems	GitHub Copilot, Netflix recommendation engine

Hierarchical Agents	Works in organized layers with task delegation	CI/CD pipeline managers, Project management systems	Azure DevOps, Jenkins Pipeline

AI Agents in Software Engineering

AI agents are transforming how software is built, tested, and secured. They automate tedious tasks, speed up workflows, and strengthen code quality, freeing developers to focus on creative problem-solving. Here are some ways in which AI agents can help developers:

1. **Code Reviews:** AI agents can automatically analyze code, spot issues, and suggest fixes—like an ever-vigilant reviewer that never gets tired. They help catch errors early, ensuring clean, high-quality code.
2. **Automated Testing:** Instead of manually running tests, AI agents handle unit, integration, and regression testing without requiring constant human oversight. Think of them as virtual QA engineers who work 24/7, making sure new changes don't break anything.
3. **Faster Deployments With CI/CD:** AI agents can help get code changes into production quickly. This speeds up Continuous Integration/Continuous Deployment (CI/CD) by automating builds, running tests, and ensuring smooth releases. Developers can push updates faster, with fewer bugs focusing more on coding and less on managing processes.
4. **Smart Security Checks:** AI agents act as security scanners, constantly hunting for vulnerabilities in your code and ensuring software remains secure throughout its lifecycle. They analyze dependencies, flag risks, and suggest fixes before threats become real problems.

Coding agents are a subset of AI agents. Think of them as intelligent assistants, reducing repetitive work and allowing developers to focus on building better software. As AI continues to evolve, these tools will become even more integrated into development workflows, making software engineering more efficient and scalable.

AI Coding Agents: Three Ways They Impact Coding

AI is transforming coding in three major ways:

FIRST

Developers are using large language models (LLMs) like ChatGPT, Claude, and others as coding assistants. These models generate code from text descriptions, improve existing code, and help debug errors. Providers like Claude have introduced features like Artifacts, which allow users to run and iterate on generated code directly within the chatbot interface.

SECOND

Advanced AI assistants are now integrated into IDEs, offering more context and supporting complex tasks. GitHub Copilot, launched by Microsoft a year before ChatGPT, was the first major example. It started as a simple tool for code snippets but has evolved into a full development assistant. Similarly, Amazon's coding assistant Q provides autocomplete, design agents, and even migrates code between languages. Other players include Replit, which powers its own AI-driven coding environment, and Codeium, which supports numerous IDEs.

THIRD

The most advanced AI systems act as multi-agent teams, collaborating to complete complex coding tasks. Multiple LLMs can be assigned different roles in a project—one might design a project plan, another breaks it into smaller steps, a third writes the code, while another reviews and tests it. This approach mimics how human development teams work. One example of this is Devin, created by AI startup Cognition and branded as "the first AI software engineer." It uses LLMs and tools like browsers, IDEs, and compilers to gather information, write code, and evaluate its results.

The videos shared by Cognition AI featured Devin handling various assignments, including taking on a computer vision project from UpWork. This sparked discussions about whether AI agents could soon take over software engineering jobs. While Devin isn't available to the public yet, it has inspired open-source alternatives like OpenDevin and GPT-engineer, both showing the potential for AI to handle end-to-end software engineering tasks.

AI coding agents are still in the early stages but have already shown that they can automate routine tasks, helping developers be more efficient and focus on higher-level work.

The Biggest Technical Challenges with AI Agents Today

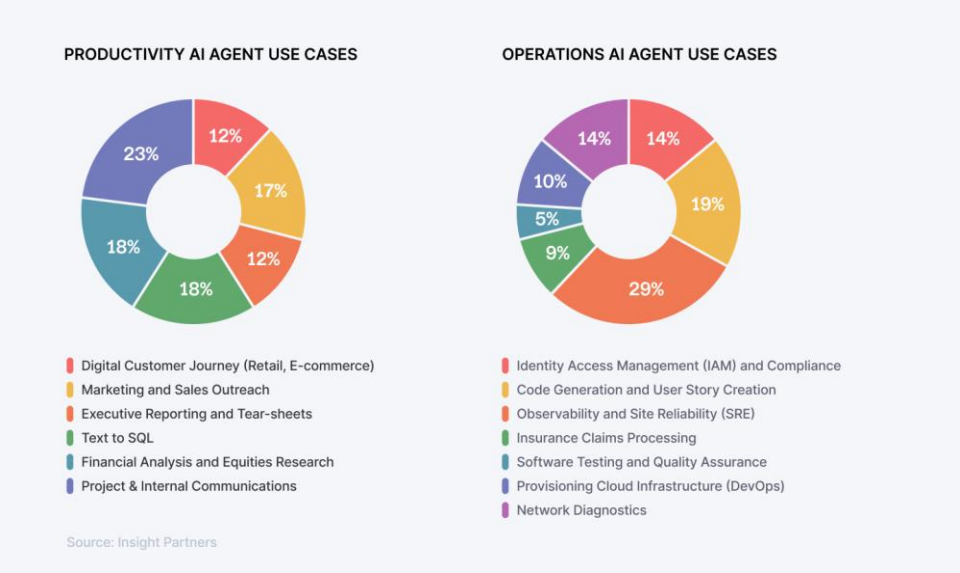
AI agents hold great promise, but they are far from perfect. Developers face two major challenges when building and improving these agentic AI systems:

1. Unpredictability and Lack of Explainability

AI models don't always behave in predictable ways. Given the same prompt, they can generate different responses, making it difficult to anticipate what they will say or do next. This lack of consistency makes debugging and evaluating AI agents a challenge. Even more, we don't fully understand how these models arrive at their conclusions. Today's models still operate as black boxes, making it hard to explain their reasoning or trust their decisions.

2. Debugging and Evaluation Complexity

Because AI agents don't follow rigid logic, they can solve problems in unexpected ways. A good analogy comes from chess: AI-powered chess engines often make moves that seem illogical to human players but ultimately lead to victory. Evaluating AI agents is also tricky. Did an agent improve because of better engineering, or was it simply a result of a more powerful model? Developers struggle with setting reliable benchmarks, choosing meaningful metrics, and gathering user feedback to assess performance effectively.



Hype or Reality?

AI assistants like GitHub Copilot have been shown to boost developer productivity by reducing time spent searching for solutions. Tools like ChatGPT and Claude are also becoming regular tools for drafting software designs, generating initial code versions, and learning new programming concepts.

But not all the hype around AI coding assistants is justified. For example, many of Devin's widely shared demos turned out to be carefully controlled, and AI agents are still far from handling the full workload of a mid-level or senior software engineer.

There are also concerns about code quality and security. AI-powered tools can generate unsafe code, sometimes pulling insecure patterns from their training data or the user's own codebase. To mitigate this, providers are continuously adding safeguards. Another risk is "automation blindness"—developers might rely too much on AI-generated code without reviewing it, leading to unpredictable errors that take extra time to debug.

Interestingly, the rise of AI is driving more demand for developers, not less. As these tools evolve, they will reshape—not replace—the role of software engineers.

A Future Look at AI Agents

In short, AI agents are on track to transform nearly every aspect of life and work. Their impact on the software industry will be palpable. Building an app or service may no longer require coding or design skills. Instead, you'll simply describe what you want, and your AI agent will handle everything—writing code, designing the interface, creating a logo, and even publishing the app. OpenAI's GPT is just an early glimpse into a world where anyone can create and deploy their own digital assistants.

Beyond software development, AI agents will change how we interact with technology. They could replace search engines by delivering personalized answers instead of endless links. E-commerce may shift as agents find the best prices across multiple vendors, eliminating the need to browse stores. Productivity tools like word processors and spreadsheets might also be replaced by AI-driven systems that generate and format content for you.

Industries that are currently separate—search engines, online shopping, advertising, and productivity software—could merge into a one AI ecosystem. The future isn't just about improving existing tools; it's about redefining how we use technology altogether.

Final Thoughts

People want AI to do things for them, and for developers, AI agents are becoming impossible to ignore.

No matter what kind of engineer you are—frontend, backend, or full-stack—an AI assistant can help. For frontend engineers, AI can assist with creating responsive designs and optimizing user interfaces. Backend engineers can use AI to improve database queries, manage server resources, and strengthen security. Full-stack engineers can get support across the board, from writing better code to handling deployments.

A recent Gartner report predicts that 80% of software engineers will need to reskill as generative AI takes on more programming tasks. But AI won't replace developers—it will need them to review, refine, and guide its work.

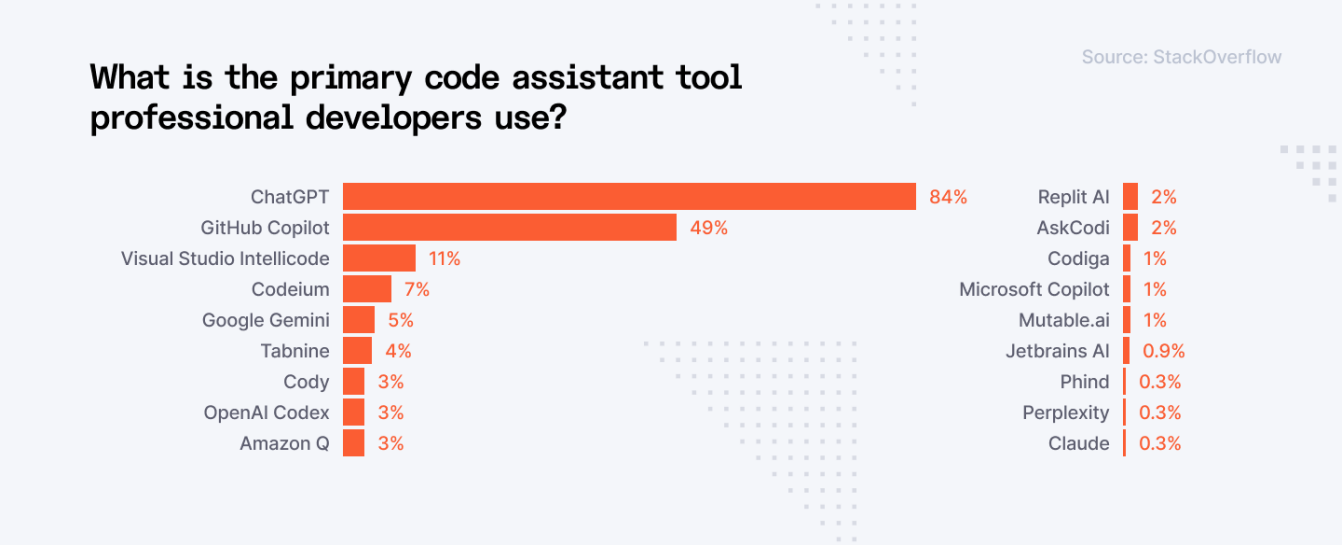
Agentic AI, which focuses on autonomous decision-making instead of just generating code, is the next step. It will push AI coding assistants beyond simple suggestions and towards AI-native software development. The tools will get smarter, but experienced developers will always be needed to make sure the results are solid.

How GenAI Transforms Software Development in 11 Ways

Discover how Generative AI is transforming software development from coding to deployment.

Faster time-to-market and cut-through velocity are crucial for any software engineering team. Fortunately, generative AI comes with use cases that can help engineering teams improve their workflows, deliver bug-free products in record time, and power up productivity.

AI has become an integral part of software engineering, improving repetitive actions across the entire software development lifecycle (SDLC) – from coding to testing, deployment, and documentation. Three-quarters of developers already use or plan to use AI coding assistance, based on a recent StackOverflow community survey. Tech companies too are shifting to generative AI trend with Gartner claiming that two-thirds of the businesses are in the pilot or deployment stages with AI coding tools and predicting that 75% of engineering talent will use AI coding assistants by 2028.



What follows are 11 impactful GenAI use cases that are already transforming the software development process. Additionally, we'll discuss what Generative AI is, the well-known risks, and some tools that can help.

What is Generative AI?

Generative AI is designed to learn from the input data, recongize patterns, and generate new content on a large scale without duplicating the original data. This powerful technology can generate content in different formats such as images, video, speech, text, software code, and product designs.

Generative AI's foundation lies in AI models that are trained on a diverse range of unlabeled data, allowing them to perform with auxiliary fine-tuning. It encompasses different technologies such as:

- Large Language Models (LLMs)
- Generative Adversarial Networks (GANs)
- Variational Autoencoders (VAEs)
- Autoregressive Models
- Recurrent Neural Networks (RNNs)
- Transformer-based Models

Risks of Generative AI

As beneficial as generative AI is, it also poses significant risks and limitations. Key oversight risks include:

1. Transparency:

Generative AI models are unpredictable in their outputs, and even developers may not always understand how these applications operate. This lack of transparency makes GenAI applications a bit troublesome.

2. Biases:

To detect and address biased responses and ensure their appropriate usage, it's vital to have policies and controls in place.

3. Inaccuracy:

Generative AI systems may produce inaccurate responses or fabricated outputs. It is important to assess the accuracy and appropriateness of the generated content before distributing the information publicly.

4. Privacy:

Generative AI systems may be trained to recreate the content around sensitive or private information. This can be a major concern for enterprises who are implementing generative AI applications.

5. IP issues:

These models lack assurances for protecting confidential data, as they are often trained on proprietary and copyrighted data. Thus, companies should implement ethical and legal concerns to prevent the exposure of intellectual property.

6. Cybersecurity:

GenAI is often used in cyber or fraud attacks. Companies should implement mitigation strategies to determine coverage for AI-related breaches.

11 GenAI Use Cases that Transform Software Engineering Process

So how useful is generative AI in the software engineering industry? Generative AI can boost your developers' productivity and speed by 20-50% – right now, making a real difference, from automating code reviews to smarter testing or debugging. Here are 11 ways it transforms software engineering workflow:

1. AI-Backed Code Review

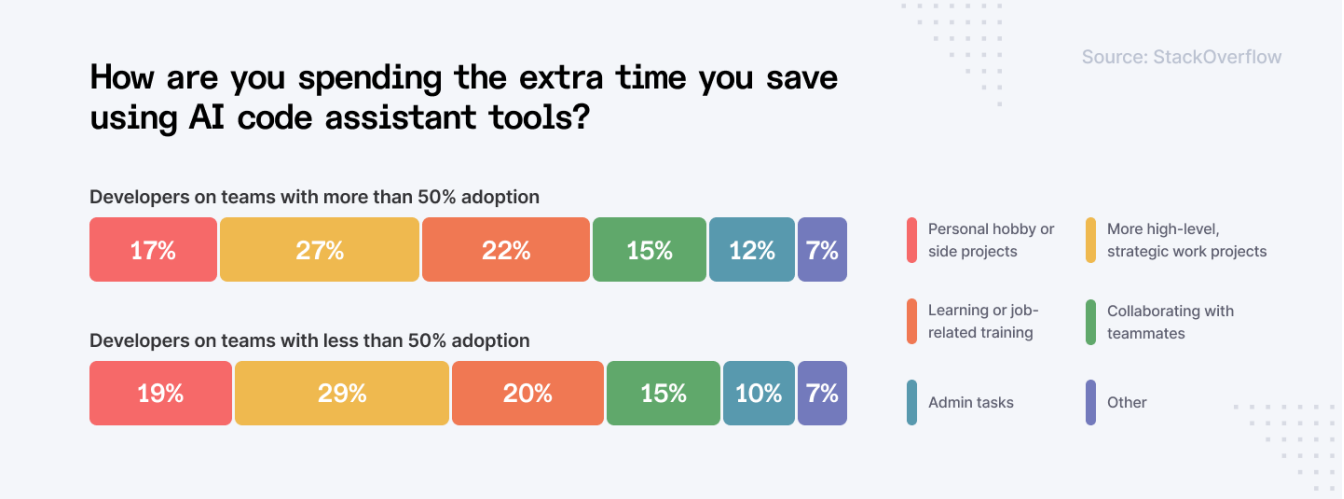
AI-driven code review is one of the most noteworthy use cases of GenAI in the software development lifecycle. Contrary to traditional code reviews which are time-consuming and superficial, generative AI tools can help engineering talent analyze code quickly and in more detail, offering valuable suggestions for improvement.

These GenAI tools can help developers detect code bugs and security vulnerabilities, improve code quality, and ensure software robustness and reliability. In addition, tools that combine AI analytics with a human touch, like Reviewable, automatically organize code reviews into groups and suggest contextual code changes based on those reviews.

Tech teams can also create reliable software based on successful past implementations by leveraging automatic code regeneration capabilities. They can quickly predict technical performance and code issues before they complete architectural designs.

Best AI-driven code review tools include:

- **GitHub Copilot** provides inline code completions powered by deep learning. This tool turns developers' natural language prompts into coding suggestions based on the project's context and style.
- **Reviewable** is a comprehensive code review tool that is fully integrated with GitHub.
- **DeepCode** is a code review platform powered by AI that assists developers in identifying coding issues, bugs, and vulnerabilities, all in real-time.



2. Automated Code Generation

The most straightforward application of GenAi is automated code generation. Instead of helping software developers with autocomplete suggestions, advanced AI systems can generate functions, classes, and even database queries taking into consideration the codebase's context. For example, AI agents can create a custom CRUD REST API for a mobile app backend by analyzing existing code, data schemas, and documentation.

Developers can ask for complex requirements and the generative AI systems would take all the heavy lifting, automating a lot of the repetitive, low-level coding work that bogs developers' workflow down.

Additionally, AI code generation enables quick prototyping, helping software engineers generate boilerplate code in a variety of programming languages and frameworks. Therefore, the AI assistant could provide customized starter code, organize folder structures, create configuration files, set up build scripts, and give specific instructions based on the developer's chosen tech stack.

Generative models like GitHub Copilot analyses the developer intent and project context to suggest inline code and functions on the spot. This allows developers to translate their high-level intentions into quality code.

Below are other two examples of automated code generation tools:

- **TabNine** uses deep learning to suggest entire code lines and functions in over 20 programming languages.

- **DeepCoder** is a research project from Microsoft that uses machine learning to generate code from input-output examples.

3. Automating Testing and QA

Nearly half of the QA teams spend at least nine hours writing one test case for a complex scenario, involving the product logic and multiple integrations. Generative AI brings extra automation to the QA processes to enable comprehensive software testing. Deep learning models can analyze large amounts of codebase at a fast pace and with an improved accuracy than is humanly possible. These AI-powered testing tools can function like an assistant contributing to the code review process, executing test cases at scale, flagging bugs and security flaws, and identifying performance bottlenecks early in the development process. This not only assures that the software meets the desired standards and specifications but also allows software flaws to be addressed before they turn into bigger headaches.

GenAI-powered AI tools can also help engineering talent write unit tests and automate the creation of comprehensive test cases, covering edge cases, failures, and real-world usage patterns. This allows them to perform efficiently and focus on more complex tasks.

Notable AI-powered testing tools include:

- **Testim** is the AI-driven test automation platform for custom web applications, enabling fast authoring of AI-stabilized UI and end-to-end tests.
- **Applitools** is a cloud-based automated visual testing solution for web, mobile and desktop apps, helping companies verify that GUI & functionality are proeprly displayed to the end user.

4. Smarter Debugging

Generative AI can help developers with bug tracking as well. These AI assistants can analyze codebase content and provide thorough debugging recommendations. They are capable of categorising and prioritising bugs, tracing through complicated execution paths the human mind might overlook. They don't just scan the codebase; they decipher it, pinpointing not just where code cracks emerge, but also how to address the root causes. Moreover, GenAI offers

predictive bug hunting, meaning it can spot where bugs may appear next. These benefits impact software reliability and reduce costly issues making it into production.

Here is an example of generative AI tool for a more intelligent code debugging:

- **Debugger.ai** uses machine learning to analyse code executions and provide insights into bugs and performance problems.

5. Automated Documentation Generation

Documentation is a tedious, but crucial task in the software development lifecycle, and generative AI holds the immense potential to automate and efficientize this time-consuming process. With given code and usage context, these AI models can generate API references, library guides, and quality application documentation. They can provide documentation in different formats, from Markdown to HTML and interactive web interfaces. For apps and services, they can extract insights from textual data and create guides and reference manuals to explain all the inner workings. For APIs and libraries, they can offer code samples for each module, class, and method. Developers could list the APIs to document, usage examples to demonstrate, diagrams to render, and other parameters and the GenAI tools would then adapt the output accordingly.

When used wisely, generative AI documentation frees up developers' time spent on writing tedious repetitive docs and allows them to take on higher-value tasks. However, it still requires a human quality check to verify accuracy and fill the gaps.

Prominent players that use generative AI for automated documentation generation include:

- **Codex** is an OpenAI research project that generates Markdown documentation for code by analyzing functions, inputs, outputs, and in-code comments.
- **Docusaurus** is an open-source tool that auto generates API reference docs and help guides by parsing JSDoc comments in code.

6. Deployments

Generative AI tools help tech teams automate deployments and significantly reduce time-to-market. Based on past data and trends, AI predicts the optimal

times to roll out updates and ensure the deployment process runs smoothly. Moreover, it can also automatically trigger rollbacks whenever anomalies are detected, error rates are high or user engagement drops. Another use case is predictive scaling. GenAI analyzes data to predict traffic spikes and automatically scales your app in advance, preventing downtime during unexpected surges in traffic. This way, the infrastructure adjusts in real time with AI models spotting the bottlenecks and scaling resources automatically, keeping the application running at a peak performance.

The best GenAI tools used in deployments, include:

- **GitHub Copilot** helps generate and suggest code for deployment scripts and configurations based on project context.
- **OpenAI Codex** assists in writing and automating deployment processes, including generating code for CI/CD pipelines and infrastructure setup.

7. Conversational Coding Interfaces

Developers spend a significant amount of time consulting documentation and references to shape up syntax and research new coding approaches. With generative AI they can minimize this need through conversational coding interfaces. Rather than scrolling through documents, they can simply ask questions like "How can I implement server-side caching in a Node.js application using Redis?", "What are the steps to securely authenticate users via OAuth 2.0 in a React web application?, or "How do I implement authentication in React Native?". The AI systems will analyze the context and generate an accurate response to this inquiry.

Those AI assistants which integrate IDEs could even help programmers to perform tasks like commiting changes, switching Git branches, running tests, and merging pull requests through voice commands or chats. This approach reduces the need to switch between different applications and websites, improving the workflow and overall productivity.

In conclusion, conversational interfaces are designed to help developers eliminate tedious tasks like Googling, browsing docs, and hunting for code snippets. However, they still need to be advanced enough to fully replace detailed API and framework documentation.

Powerful tools with conversational interfaces for coding include:

- **TabNine** allows developers to request code examples and documentation by simply describing their implementation goals in plain English.
- **GitHub Copilot Chat** offers conversational assistance directly within the IDE, helping programmers with code suggestions and learning new APIs.

8. Security, Compliance, and Threat Detection

Generative AI makes deployment faster, more secure and compliant. It can learn from patterns, spot bottlenecks, and raise real-time concerns to prevent attacks. Isolating compromised systems or rerouting traffic can happen without developers' constant monitoring.

These models can analyze network traffic and monitor system behaviour to identify unusual activities that could signal a security breach. Suspicious logins, data transfers or access requests are just some of the breaches that GenAI covers before they escalate.

Compliance is another aspect that can cause engineering friction for the tech teams. Whether it's GDPR or HIPAA, GenAI ensures the deployments adhere to industry standards and regulations. The technology would examine code for any noncompliance issues, such as missing encryption, insufficient logging practices, or improper handling of information as outlined in the relevant rules, and by following this process, oversights regarding compliance can be decreased.

Among the major players that ensure infrastructures are secure, compliant, and threat-detected are:

- **Darktrace** detects and prevents cybersecurity threats, ensuring that applications and infrastructure remain secure and compliant with industry regulations.
- **Snyk** uses machine learning algorithms trained on vulnerability databases to scan codebases for security weaknesses.

9. Code Refactoring

In large-scale, long-running projects, technical debt slows down developer productivity due to large tangled legacy code. Refactoring, or cleaning up the code, becomes crucial.

Developers usually refactor the code based on their intuition, but with generative AI assistants they can analyze code more thoroughly and objectively. These tools can look at the entire project, measure complexity, and suggest improvements.

Some real-world examples include:

1. Renaming unclear functions and variables
2. Extracting duplicated logic into smaller functions
3. Combining related logic into cohesive classes

AI-driven code refactoring tools keep code clean and maintainable and provide suggestions through natural language and comments, resulting in higher quality, optimized code. They can handle large-scale code refactoring, which otherwise will be time-consuming for developers to do manually.

Here are two examples of tools that leverage generative AI for code refactoring:

- **RefactorHub** analyzes code complexity metrics to identify refactoring opportunities, providing context-aware suggestions like extractions or encapsulations.
- **CodeMR** uses machine learning to recommend refactors that improve maintainability, providing semantic reasoning behind its suggestions.

10. Code Translation and Porting

Converting codebase from language or framework to another is a tedious and error-prone process. By using generative AI tools, developers can automate much of this grunt work and accelerate code migration.

These tools can convert applications written in Java, C++ or Python while keeping their semantics, behaviour, and performance intact. This makes it easier to adjust the code to a new environment, simplifying the project porting process to different tech stacks.

GenAI tools can automatically handle dependencies, wrappers and compatibility issues when migrating older code to modern technologies. Therefore, an generative AI assistant is capable of upgrading a Python 2 library to Python 3, handling all the nitty-gritty details effortlessly.

With these code translation tools, developers can focus on a broader architecture rather than line-by-line conversion, removing the friction and building multi-

language and cross-platform applications across tech stacks. However, a human oversight is still required so that the transformed code operates correctly.

Here are two power players developers can use for an accelerated code transformation and porting:

- **TransCoder** converts code between C#, Java, and Python using supervised learning.
- **CodeConverter** uses machine learning to translate code across web frameworks like Angular, React, and Vue.

11. AI-Assisted Design Workflows

So far we've focused more on generative AI's code generation capabilities. However, the technology is often used for design workflow, helping creative roles like UX designers achieve significant workflow improvements.

Visual design generation models enable designers to instantly convert text prompts into creative images. Designers can also iterate on prompts to generate different concepts. This way, GenAI becomes a tool for brainstorming, allowing graphic or UX designers to turn creative ideas into visual concepts or explore design possibilities before delving into mockups or assets.

Interface designers can also prototype quickly by describing UI components and layout for the generative AI tools to render. This allows them to validate ideas early on, before heading into the software engineering phase. Certainly, while GenAI speeds up visualization, designers are still in charge of creative direction, iteration process, or final touches. By offloading mundane and repetitive tasks, designers can focus more on strategic challenges, in which the AI models can act as creativity amplifiers rather than replacers.

Here are two examples of for AI-assisted design workflow tools:

- **Anthropic** built an AI assistant called Claude that translates text prompts into UI mockups, icons, and design assets.
- **Runway** lets designers describe visuals in natural language for the AI to generate.

A Hypothetical Use Case

Consider a developer working on a mobile app that requires implementing a sophisticated user authentication system with multi-factor authentication (MFA).

To improve his workflow, he decides to use a generative AI tool specifically designed for security features. First, he describes the requirements for the MFA system to the AI tool, including the need for email verification, SMS-based codes, and integration with third-party authentication services. The AI, trained on a vast dataset of security protocols and authentication frameworks, generates a comprehensive code framework that includes functions for handling email and SMS verification, as well as integrating with services like Google Authenticator or Authy. The AI tool also provides suggestions for securing sensitive data and ensuring compliance with industry standards like GDPR or HIPAA.

Developer then reviews the generated code, making adjustments to fit the app's specific design and security policies. He incorporates the AI-generated code into the app's codebase and proceeds to test the MFA system thoroughly. The generative AI tool helps by automatically generating test cases and scenarios to ensure the authentication system performs as expected under various conditions.

By using AI, the developer reduces the time spent on writing boilerplate code and ensures that the authentication system is robust and secure. This allows him to focus more on user experience and adding innovative features to the app.

Strategies To Maximizing The Value of GenAI

To maximize the value of generative AI use cases in software development, organizations can follow these impactful strategies:

1. Ensure that generative AI tools align with your development process and goals.
2. Collect feedback from developers, testers and stakeholders to improve AI-driven models, algorithms, and processes over time.
3. Build cross-functional teams between AI developers and software development teams to work together to identify areas where GenAi can add value.
4. Implement robust data protection measures and prioritize compliance with data privacy regulations.
5. Consider the scalability of generative AI technologies and ensure they accommodate your organization's further growing needs.
6. Provide training and support to engineering talent using AI tools to maximize their productivity and efficiency.
7. Regularly measure how well AI-driven processes perform in terms of development speed, code quality, and security.
8. Invest in ongoing AI education and training to ensure your development teams use AI technologies to their full potential.

Wrapping Up

By using GenAI-driven tools, nearly every phase of software development can be automated, simplified, and accelerated, including coding, testing for issues, fixing bugs, documenting procedures or designing layouts. These tools help engineering teams deliver high quality software in record time, without anomalies and bugs, so developers can focus more on creative work and complex problem-solving. However, companies must consider its potential drawbacks like data privacy, model bias, training expenses, and expertise disruption. Rather than replacing developers, the best approach is to combine the strengths of both human and artificial intelligence.

Open-Source vs. Closed AI: Who Should You Trust?

Trust depends on your goals: closed models lead in performance and control, while open models win on transparency and flexibility. Discover which approach offers the best trust, security, and performance.

AI is transforming every industry, but trusting the right models is critical. The stakes are high: trust, governance, performance, ethics, and security all hang in the balance.

Closed-source models (e.g. GPT-4, Claude) lead today in raw performance and centralized safety controls.

Open-source models (e.g. Llama 2, Mistral) shine in transparency, customization, and community vetting.

Recent benchmarks show closed-source still outperforms on average, but open-source is narrowing the gap fast. Security experts warn open models risk easier attacks due to public weights, yet their transparency accelerates fixes, while closed models hide vulnerabilities but rely on vendor trust for patches.

Surveys of developers reinforce this debate: younger and early-career engineers especially value transparency. In fact, a 2025 StackOverflow survey found that *"trust and learning are central to younger developers' interactions with open-source AI"*.

Major AI labs mirror this split: OpenAI keeps its flagship models behind API walled gardens for safety, while companies like Meta and Anthropic have open-sourced many models. Even policy makers are taking notice with the EU's AI Act explicitly highlighting open-source as a public good.

In this article, we will dive deep into the open vs closed AI debate. We'll explain what each approach means, who favors which, why trust matters, when and where each is appropriate, and how they compare on key fronts. Along the way

we'll examine performance benchmarks, security trade-offs, ethical considerations, and governance and policy implications.

By the end, you'll have a clear, evidence-backed view on which models (and development practices) inspire confidence in which contexts.

Ready to build with trusted AI? Join Index.dev and connect with top companies seeking skilled AI developers!

Who's Involved: Developers, Companies, and Communities

Who is shaping this debate? On one side are the open-source community and independent developers. The debate pits the open-source community (developers behind PyTorch, TensorFlow, Hugging Face, and countless GitHub projects) against AI companies that keep their latest models proprietary for control and IP protection.

Open advocates, 82% of whom have contributed to open-source tech, prize transparency, peer review, and shared innovation. Enterprises and regulators, meanwhile, lean on closed-source solutions (e.g., Azure OpenAI) to meet strict compliance in sectors like healthcare and finance. At conferences and on policy platforms, experts debate 'open collaboration vs. closed control' as a fundamental split in AI strategy.

Governments also weigh in: the EU's 2023 AI Act highlights open-source foundation models' economic benefits while insisting on safety, and U.S. and Chinese policies similarly debate openness versus control. Yet these lines are blurring as OpenAI's Sam Altman has noted the limits of full secrecy, and Meta's Llama 2 release underscores industry-wide collaboration.

What Is Open-Source and Closed-Source AI?

What do we mean by "open-source AI" versus "closed-source AI"? In practice, *open-source AI models* are those whose code, model weights, and training data are published under permissive licenses (MIT, Apache, GPL, etc.) so that everybody can use and alter it. Consider projects such as LLaMA (Meta models) or Stable Diffusion.

They live on platforms like Hugging Face or GitHub, and a community of volunteers can inspect and improve them.

The key promise: full transparency. You can peer into architecture, training data (if published), and tweak the model.

By contrast, *closed-source AI models* are proprietary systems, often offered only via an API or commercial license. Companies keep the weights and code behind the scenes. You might use them through cloud services or products (e.g. ChatGPT, Bard, Azure Cognitive Services). The inner workings are not visible, and usage terms are fixed by the provider. The trade-off is control: providers say closed models let them enforce policies, fix bugs centrally, and monetize more easily.

Between these poles there are hybrid approaches:

1. *Tiered models*
2. *Controlled releases*
3. *"Open core" with proprietary add-ons*

Some systems (like Llama 2) are open with limited licenses; others are open-weight but require API keys. These hybrids attempt to balance openness with oversight.

The label also extends to tools: many libraries (PyTorch, Hugging Face Transformers, etc.) are open-source, enabling AI even for closed models. When we talk "open vs closed AI," we're really asking who can see or change the model's inner workings.

In essence, open-source AI = transparency and community, while closed-source AI = proprietary control.

Each side trusts different mechanisms to ensure reliable, ethical AI.

Pros and Cons of Each Approach

Why is this trust debate so heated? Because AI models can have profound impacts (some unintended) and handling that requires confidence in how they work.

Here are key factors:

- **Transparency vs. Secrecy**
  Open models let us audit code, inspect data sources, and test for biases, building confidence through visibility. That same openness exposes vulnerabilities (backdoors, data leaks) in plain sight. Closed models keep their internals hidden, so we must trust the provider's safety and bias-mitigation claims.

- **Security**
  Open code invites security experts to find and fix flaws but this means that it also gives attackers the same blueprints. Closed models hide their architecture but rely on a few certified experts and internal audits. Providers like OpenAI add layers (for example, "AI firewalls" that vet prompts) to balance transparency with protection.

- **Performance**
  Proprietary LLMs (GPT-4, Claude) still top benchmarks, so we trust them for high-stakes tasks. But open models such as Mistral and Llama-2-70b are closing the gap fast, and in niche scenarios they can even outperform closed systems.

- **Ethics and Bias**
  With open models, we can review training data and apply ethical filters ourselves. That visibility also makes misuse (for disinformation or hate speech) easier. Closed models enforce rules centrally, but we must have faith in the vendor's ethical priorities. Some propose "tiered openness" or public audits to get the best of both worlds.

- **Governance**
  Regulators face a challenge: if anyone can modify an open model, who's accountable? Several recent papers (e.g. R Street Institute) argue regulators must adapt, since traditional product liability doesn't fit easily. The EU AI Act acknowledges open-source benefits but requires safety

guardrails, while U.S. policies focus on vendor responsibilities regardless of code status. Whether through community norms or corporate compliance, trustworthy governance is essential.

- **Community and Support**
  Open-source communities (forums, GitHub) offer rapid peer help and collective innovation as 57% of developers enjoy contributing to open projects. Closed ecosystems provide official SLAs and vendor support. Platforms like Index.dev bridge both by vetting talent, so we know skilled, trustworthy people are behind every AI system.

In short, open-source AI tends to earn trust through transparency, collaboration, and community oversight, at the cost of exposure to vulnerabilities and (until recently) lagging raw performance.

Closed-source AI earns trust through centralized control, security measures, and brand reputation, but it asks users to have faith in unseen processes. Both approaches have merits and pitfalls, and real-world solutions often mix elements of each.

FACTOR	OPEN-SOURCE	CLOSED-SOURCE
Transparency	Full code/weight visibility	Inner workings hidden
Performance	Rapidly improving; niche wins	Leading benchmarks today
Security	Community audits, faster patches; but public attack surface	Centralized defenses; but opaque vulnerabilities
Ethics & Bias	Audit and correct biases; risk of misuse	Provider-enforced safeguards; trust in vendor ethics
Governance	Licensing complexities; community norms	Clear vendor accountability; regulatory alignement
Community	Large contributor base; peer support	Official support, SLAs; less community feedback
Cost & Control	Free/no-cost weights; self-managed infra	Pay-per-use API; managed infra

Performance and Reliability: Head-to-Head

Let's dig into performance which is a key aspect of trust for many users. If an AI chatbot reliably gives useful answers, users will trust it more. Recent benchmarks clearly show the landscape:

- **Current Leaders:**
  Proprietary LLMs like GPT-4, Google's Gemini, and Claude top benchmarks. In "ELO score" comparisons, closed models consistently outshine open ones, with open models clustering at the lower end.

- **Open Gains:**
  The gap is closing. Open models such as Qwen-72B, Llama-2-70b-chat, and Mistral-Medium now rival closed systems. Big Tech is even open-sourcing more of its lineup with Google's Gemma and Microsoft's Phi-4 having open variants, and Meta's Llama 2 found commercial traction in 2023.

- **Beyond Chatbots:**
  Open models aren't just catching up in text; rather they've led breakthroughs in image (Stable Diffusion) and vision (CLIP) tasks, boosting confidence in open approaches.

- **Hybrid Solutions:**
  Many teams combine closed services (e.g., GPT-4) for core workloads with smaller open models for experimentation, achieving both stability and flexibility.

- **In Practice:**
  For mission-critical tasks (like advanced summarization or coding), a top-tier closed model may be safest today. If you need customization or a deep understanding of the model's behavior, open models now deliver competitive performance with fine-tuning capabilities. Recent surveys even show that "small open-source models can keep up well with closed-source models" on many tasks.

Choosing the right tool means balancing immediate reliability and long-term adaptability. For learning and non-critical projects, open models shine with

enterprises launching new AI features often start with closed offerings and layer in open frameworks for flexibility.

Ethics, Bias, and Governance

Ethics and Bias

Ethical AI demands responsible, fair behavior. Open models shine in transparency—anyone can audit and correct biases (e.g. Wikimedia's "open-weight AI" policy and academic ethics frameworks)—but they're also easier to tamper with. Closed models embed ethics through provider filters and alignment training.

Still, openness fuels safety research: the collaborative ecosystem drives new safeguards even as it spawns some malicious variants. Both must meet regulations like the EU AI Act, which mandates human oversight and "Human Autonomy."

Governance and Policy

Trust rests on legal oversight. The EU AI Act treats all foundation models—open or closed—by risk tier, valuing open-source's economic benefits while enforcing safety. U.S. policy (Biden's 2023 AI Executive Order) focuses on vendor best practices over openness labels. In Europe, closed-model users must prove compliance; open-model modifiers may incur data-governance duties.

Practically, this means organizations must align with transparency standards either way. For example, if you use a closed model in Europe for a high-risk application, you have to demonstrate compliance (even if you can't see the code). If you modify an open model, you may face new legal duties (like data governance) as the R Street report warns.

Best Practices

Most experts advocate a hybrid approach: apply access controls or licensing to powerful open-source models ("open-core") and involve community norm-setting (e.g. "responsible release" guidelines), while closed-source vendors publish ethics statements and collaborate on standards. Agile decision-making (tiered access for open systems and open audit pipelines for closed ones) ensures accountability.

Security: Vulnerabilities and Defence

Security underpins trust. Open and closed AI systems both face attacks, but the methods differ:

- **Open-model attacks**
  Public code and weights make privacy and poisoning attacks easier. Threats like model inversion, membership inference, data leakage and backdoors are real, but community-driven patches and fine-tuning mean fixes can roll out quickly once flaws are spotted.

- **Closed-model attacks**
  Hidden internals require you to trust the provider's security testing, yet bugs still slip through. For example, an open-source foundation model once exposed API keys and chat logs due to a flaw. To defend, many vendors deploy AI firewalls that scan inputs and filter outputs, and they enforce authenticated API calls with logging and rate limits.

- **Insider and supply-chain risks**
  Open-source projects must guard against malicious code contributions, while closed systems must secure their data pipelines and hardware. Common defenses include formal code audits, certified hardware and trusted compute environments.

- **Finding the balance**
  Experts recommend combining transparency and control: adopt model cards and regular security reviews, publish transparency reports, and engage third-party *"red teams"* to probe both open and closed models. This approach merges community scrutiny with enterprise-grade safeguards.

Pros & Cons of Leading Platforms

PLATFORM	TYPE	PROS	CONS
GPT-4	CLOSED	Top-tier benchmarks; robust safety filters; enterprise SLAs	No code/weight access; vendor lock-in; pricing can scale up
Claude 3	CLOSED	Strong in reasoning tasks; transparent pricing; partnership with AWS	Proprietary; limited fine-tuning; fewer community integrations
Gemini	CLOSED	Google-scale training; multimodal strengths; integrated with Google Cloud services	Black-box model; compliance gated byGoogle policies
Llama 2	OPEN	Commercial-use license; strong community ecosystem; easy to fine-tune	Baseline performance slightly below closed; requires self-hosted infrastructure
Mistral-Medium	OPEN	High inference speed; competitive performance; permissive Apache 2 license	Smaller community than Llama; fewer production-grade integrations
Falcon 40B	OPEN	Optimized for latency; strong multilingual support; backed by consortium	Still maturing fine-tuning tools; requires GPU resources
Stable Diffusion	OPEN	Breakthrough in image generation; huge ecosystem of forks and UIs	Primarily vision; needs extensions for text tasks; some licensing confusion

Developer Perspective and Platforms

Trust for developers hinges on community and tooling. Many engineers favour open source to learn and experiment on platforms like GitHub and Hugging Face. Talent sites such as Index.dev vet and curate skilled developers, ensuring trusted contributors to any project, open or closed. Confidence in who writes the AI code is nearly as vital as the code itself.

Community engagement boosts trust: firms releasing open models partner with developer forums for feedback. According to the previously mentioned StackOverflow survey, 57% of developers prefer open-source projects while only 30% favour proprietary technology. This pro-open mindset leads to more issue reporting, enhancements and knowledge sharing.

Enterprises wary of open models over governance gaps can now opt for open projects that provide premium support or certification. Meanwhile, closed-model providers are courting developers too—Microsoft, for instance, has open-sourced parts of its cognitive services.

Key takeaway:

Trust does not lie solely in open or closed labels. It grows through transparent roadmaps, certified models and vetted talent from reliable networks.

So, how should you decide which model to trust? Consider:

Practical Tips:

- **Who should decide?**
  Your engineering and legal teams, with input from ethics officers and maybe developer community leaders.

- **What to do?**
  Evaluate models via benchmarks (like those in recent papers), perform security tests, and check licenses.

- **Where/When to use each?**
  Use open for research, internal tools, and community-driven products; use closed for critical production systems where guarantees are paramount.

- **Why trust one over the other?**
  It boils down to your risk tolerance and values: openness vs control.

- **How to build trust?**
  Vet your developers (see platforms like Index.dev), adopt best practices (regular audits, “red teams”, documentation), and stay updated on policy (e.g. EU AI Act compliance).

Conclusion: Balancing Trust in an AI World

Choosing between open and closed AI models isn't about picking sides, it's about understanding your needs.

Transparency, performance, security, and community all factor into trust, and the right balance often lies in combining both worlds. Open-source offers you flexibility and collaboration while closed-source provides polish and accountability.

For many organizations, a hybrid strategy makes the most sense: leverage open models for innovation and fine-tuning, and deploy closed models where reliability is critical. What matters most is *how* you implement, govern, and staff your AI efforts.

Platforms like **Index.dev** play a valuable role in this equation, connecting businesses with thoroughly vetted developers who bring both expertise and accountability to the table. Because in the end, trustworthy AI isn't just about the code. It's about the people behind it, and the practices around it.

Trust isn't a toggle, it's a system. Build it deliberately.

How to Get Started with Vibe Coding with AI (The Easy Way)

Vibe coding lets you build real apps fast by describing what you want and letting AI do the heavy lifting. You guide, test, and refine, no deep coding skills needed.

Low-code is dead. Long live Vibe Coding.

So, you've been coding for a while. 3, 5 years maybe. You've shipped products, maybe even tried launching your own thing or joined an early-stage startup.

Maybe you've noticed this buzz around 'Vibe Coding' and thought, "What even is that?"

Right now, you're probably still using VS Code with Copilot helping you out auto-completing those boring React states or try/catch blocks in Express. For tougher problems, you chat with GPT, Claude, or DeepSeek as your "pair programmers." And that feels like enough, right? You're still the one calling the shots, thinking through the architecture and making the big decisions.

But what if you've got it backwards? What if AI should take the first pass and you should become the reviewer instead? What if you should offload more and shift to quality control?

You've probably heard the whispers about Windsurf, Bolt, Replit, or Lovable. Even if you're skeptical about AI hallucinations and overly complex code, you're worried about falling behind. Maybe your experience is actually blinding you to what AI can already do today, not just what it might do someday.

That's where vibe coding comes in. It's a fresh way to work with AI that Silicon Valley CEOs say lets 10 engineers do the work of 100. Sounds wild, right? But how do you actually start vibe coding?

Let me walk you through it in this article. Step-by-step. No jargon. Just simple, practical advice to get you going.

So, What Is Vibe Coding?

Vibe coding started as a Silicon Valley buzzword (thanks, Andrej Karpathy) for "using AI tools for the heavy lifting in coding." Instead of typing every line yourself, you just communicate the "vibe" of what you want, and AI generates the actual code.

For example, you might say, *"Make me a web page that shows weather for whatever city the user types in."* This is your prompt.

The AI predicts and produces code matching your description — the HTML, CSS, and JavaScript needed for your weather app. You try it out. If something's off, you tell the AI, *"Make it look nicer,"* or *"Add error handling."* The AI tweaks the code. You keep going back and forth — prompt, test, fix — until it feels right.

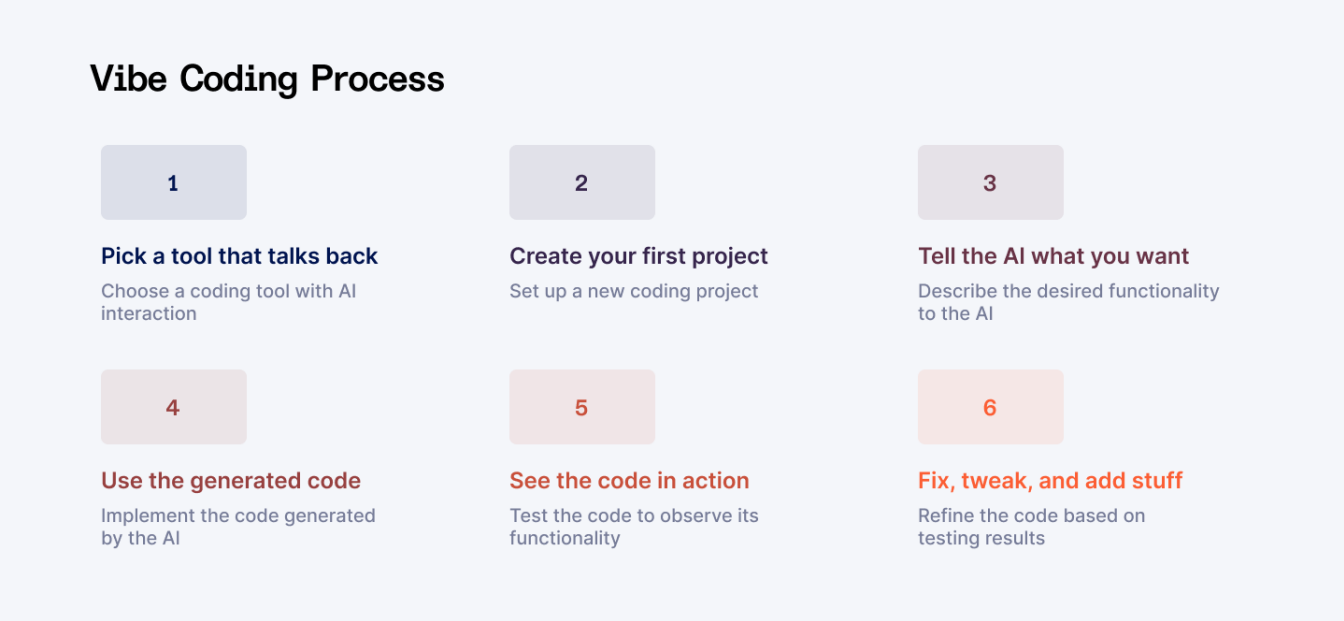
You’re still in control. You’re not buried in syntax or stuck Googling for hours. You guide the direction, test things, and give feedback. But the AI does the repetitive grunt work.

At best, vibe coding is a gimmick for non-coders to fake it. At worst, it’s junior devs trying to punch above their weight and passing AI work as their own.

But in reality? This flips the old-school way of building software. And if you use it right, it can seriously speed you up, especially if you know what you’re doing.

How to Get Started with Vibe Coding (The Easy Way)

Let’s keep this super simple. You want to vibe code? Cool. Here's a step-by-step guide to jump in.



1. Pick a Tool That Talks Back

First things first. You need the right tool. Got two main options here:

- *Browser-based:* Go with Replit if you don't want to install anything. Just create a free account on their website and start a new "repl" project. Enable Ghostwriter (their AI assistant) — might need a subscription for full features.

- *Desktop app:* Cursor is your best bet here. It's basically VS Code with AI superpowers baked in. Download it from Cursor's website, install it like any other app, and you're set. When you open it, you'll see a familiar code editor with an AI chat panel built right in. Some tools might ask for your OpenAI API key, so have that handy if needed.

Most tools have a quick tutorial when you first open them. It’s pretty straightforward.

2. Create Your First Project

Keep it ridiculously simple for your first try. Think "Hello World" level simple. Open your chosen tool and start a new project. In Replit, pick a template like "HTML/CSS/JS" or "Python." In Cursor, just create a new folder and maybe a basic README file. You just need a space for the AI to drop your code in.

3. Tell the AI What You Want

This is where the fun begins! In the AI chat panel, describe what you're trying to build.

For example: *"Create a webpage with a dark blue background, a centered heading that says 'My First Vibe Code Project', and a button that shows an alert when clicked."*

Write it in plain English. Be clear but don't overthink it. The AI isn't psychic, but it's pretty good at translating human ideas into code.

4. Use the Generated Code

Within seconds, the AI will show you code that matches your description. In Cursor, you'll see a "diff view" showing what changes will be made. Just click "Accept Changes" to insert it. In Replit, it might add files directly or show code for you to copy/paste. Each tool handles this differently, but the idea is the same: get that AI-generated code into your project.

5. See It in Action

Always test what the AI gives you! In Replit, click the "Run" button to see a preview. In Cursor, you might need to open the HTML file in your browser. If it works — awesome, you just vibe coded your first thing. If it doesn’t, no stress. Rarely is the first try perfect. That’s what the next step is for.

6. Fix, Tweak, and Add Stuff

Here's where the conversation continues. Is something not quite right? Just tell the AI: "Make the heading text larger and change the button color to green."

- *Got an error?* Copy-paste it to the AI and ask *"How do I fix this?"* It'll diagnose the problem and suggest a solution.
- *Want to add more features?* Just ask: *"Now add an input field where users can type their name, and make the button display 'Hello [name]' when clicked."*

This back-and-forth is the heart of vibe coding. You're having a conversation about code rather than writing it line by line.

Stick with this cycle — describe, generate, test, refine — and you'll build your project piece by piece, all through conversation. The more you do it, the better you'll get at "speaking AI" and the faster you'll build things.

Best Tips for Vibe Coding

If you want the AI to deliver solid code, you've got to work with it, not against it. Here's how to get the most out of it:

Be Ridiculously Specific

Start with tiny projects. Seriously, smaller than you think. Don't ask for a full social media app right away. That's setting yourself up for frustration. Instead, ask for a login page. But don't say *"make a login page."*

Say *"Create a login form with email and password fields, validation for proper email format, and minimum 8-character passwords. Show error messages below each field when validation fails."*

Think of AI as an eager but literal intern. It'll do exactly what you ask — nothing more, nothing less. If you don't give it details, it'll guess, and not always well.

Build in Small Chunks

Don't try to create an entire app in one prompt. It'll mess it up.

Start with something tiny that works, then add features one by one. First make the login form, then add validation, then connect it to a database. Each step builds on the last.

Paste examples when you can. Say *"I want something like this..."* and show code snippets or screenshots. In Cursor, you can highlight existing code and ask *"Make this work with async/await instead of promises."* Visual cues help tremendously.

Never Trust, Always Verify

Always review what the AI gives you. Tools like Cursor and Bolt show you exactly what changed. Use this! Look for weird variable names, security issues, or overly complex solutions.

If something looks fishy, ask questions: *"Why did you use a recursive function here instead of a loop?"* You'll catch bugs early and learn something in the process.

Test Like Your Life Depends On It

Don't wait until you've added 10 features to hit *"Run."* Run your code after every meaningful change. Found a bug? Fix it immediately before moving on.

When the AI adds a search feature, test it with normal input, empty input, and weird input. Testing frequently prevents small problems from becoming nightmares later.

Explain Like a Mentor

Instead of *"This is wrong, fix it,"* try "*The search isn't working when I enter special characters. Could you add some input sanitization?*"

Explaining what's happening helps the AI understand the problem. Treat it like a dev who knows how to code but needs very specific instructions. That's how you get good results.

Set Clear Boundaries

Tell the AI what you want AND what you don't want. *"Please use React hooks, not class components"* or *"Don't use any external libraries for this."*

You can also show examples: *"Here's how I want the data formatted: {...}. Make your output match this structure."* Boundaries help the AI stay on track.

Learn As You Go

Ask the AI to explain code you don't understand. Ask stuff like: *"Why did you use map here instead of forEach?"* or *"Explain this regex pattern.".* AI can teach you HOWs and WHYs.

The more you learn, the better your prompts will get. Remember: you're still in charge. AI is your tool, not the other way around.

The Bottom Line

Vibe coding is powerful, but it's not perfect. You still need to think like a developer (or at least act like one). Here's the truth, plain and simple:

Know the Limits

1. **AI doesn't think like a programmer — it pattern matches from training data.** This means it can write code that looks right but has hidden bugs or edge cases. Always test thoroughly!
2. **Unless you specifically tell it, the AI won't follow best security practices.** It might use old methods or leave holes big enough to drive a cybertruck through. If you're shipping code to users, double-check it. Use a security scanner or, better yet, ask someone who knows their stuff.

3. **AI tools have limited memory.** In larger projects, they might "forget" what you discussed earlier and duplicate functionality or contradict previous code. Give it reminders. Work in small, clear steps.
4. **AI has its own interpretation.** This gap can lead to frustration unless you're crystal clear about what you want. You're the project manager here — the AI is just following orders.
5. **AI-generated code often works but isn't always elegant.** It might jam everything into one massive function instead of creating a clean, maintainable structure. For quick prototypes, this is fine. For something you'll build on later, you'll need to specifically ask for good architecture or get help refactoring.

When to Call in a Human

For anything beyond personal projects or prototypes, get a developer to review your code. This is especially true if:

→ Real users will depend on it

→ It handles sensitive data

→ You plan to expand it significantly

→ Performance matters

→ You're putting it online where security matters

Wrapping Up

So, what's the final word on vibe coding? Vibe coding is like jamming with a talented-but-clueless assistant. Super fast. Super fun. Super risky if you're not watching. You call the shots. You catch the bugs. You decide when to say "good enough" or "call in a developer."

You're the brains of the operation. The AI's just here to assist. Use it wisely and you'll build way more than you thought possible.

Top 14 Vibe Coding AI Tools: Bolt, Lovable, Cursor & More

Bolt, Lovable, Cursor, and 11 other dope AI coding tools in 2025 bring the vibes with easy prompts, quick prototyping, and automation for coders and newbies alike.

Traditional coding setups demand complex installations, steep learning curves, and hours of setup.

Developers lose time configuring environments, while non-coders struggle to translate ideas into functional apps.

However, vibe coding tools like Bolt.new, Lovable, and Cursor use AI to generate code in-browser, automate workflows, and deploy apps instantly— no setup required.

Dive into this guide to build faster, smarter, and accessible software with these recommended vibe coding AI apps.

Popular Vibe Coding Tools of 2025

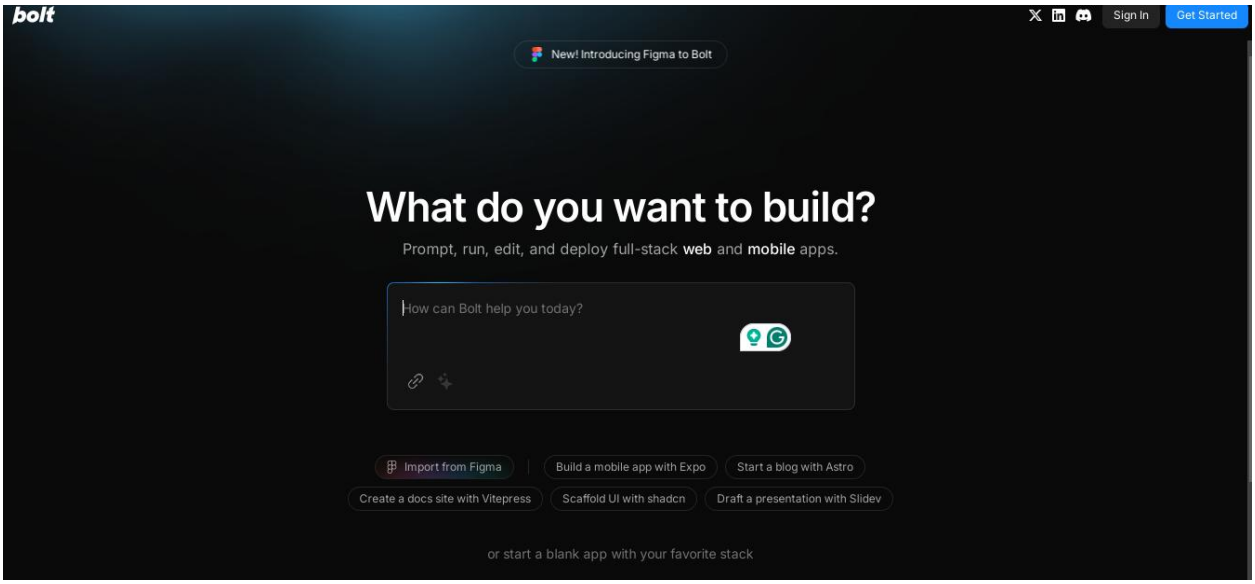
Here are the popular vibe coding apps at a glance:

1. **Bolt.new**, best for rapid prototyping
2. **Lovable**, best for non-technical builders
3. **Cursor**, best for AI-enhanced coding
4. **v0 by Vercel**, best for UI prototyping
5. **GitHub Copilot**, best for code autocompletion
6. **Replit**, best for collaborative coding
7. **Tempo**, best for React UI design
8. **Create**, best for visual-first builders
9. **Cody**, best for enterprise workflows
10. **HeyBoss**, best for no-code games/apps
11. **Codeium**, best for budget-friendly AI coding
12. **Creatr**, best for task management apps
13. **Qodo**, best for code quality control

14. **Glide**, best for data-to-app conversion

Let's dive deep into each!

1. Bolt.new



Bolt.new is an AI-powered coding tool that helps users create, edit, and deploy web and mobile apps directly in a browser. It runs on StackBlitz's WebContainers, allowing developers to build full-stack applications without installing any software.

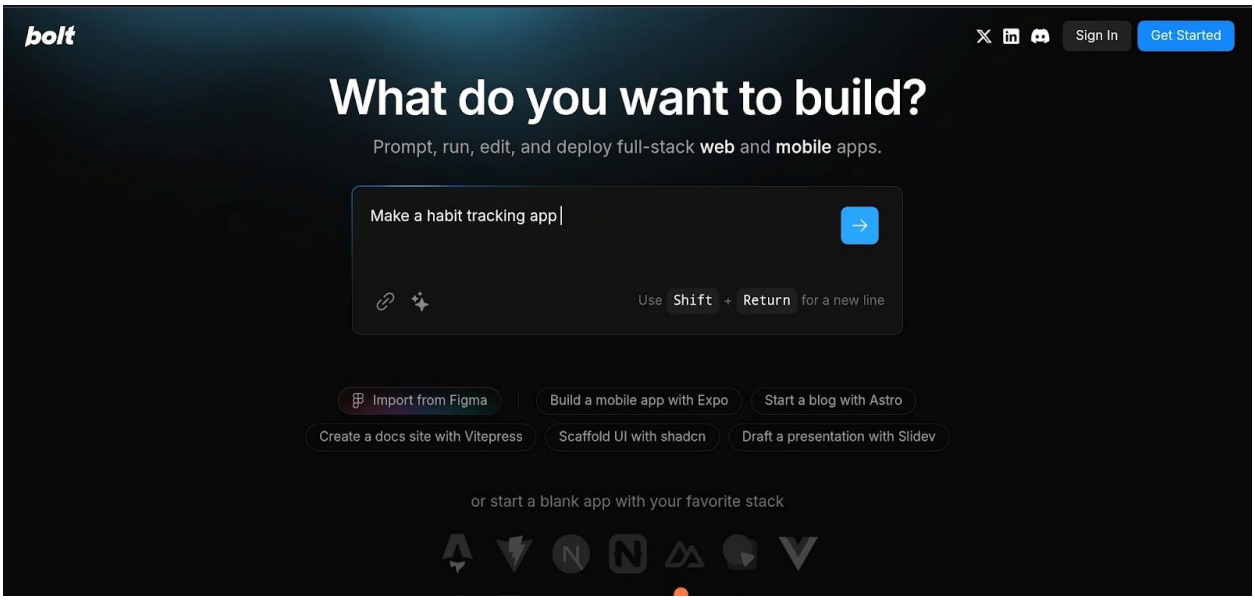
Users can simply type a prompt, and Bolt generates the code in real-time. It supports various frameworks and integrates with hosting platforms for easy deployment.

How Bolt.new helps developers and non-coders:

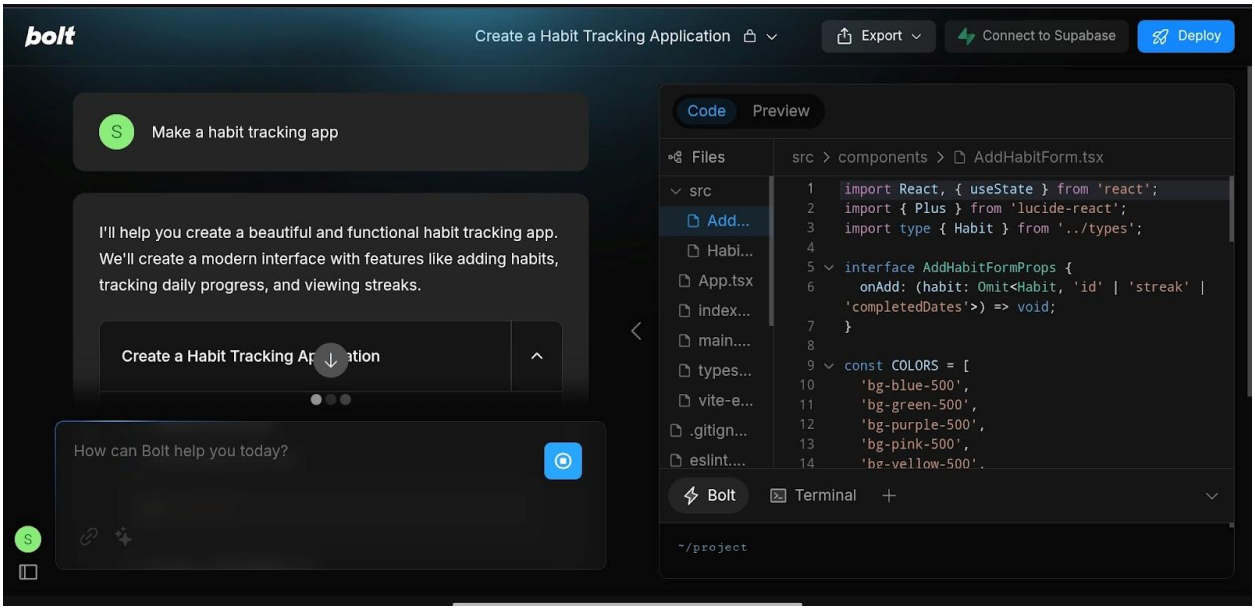
Bolt.new makes coding easier for everyone. Developers can quickly build and test projects without setting up a local environment. Beginners and non-coders can create apps just by describing what they need.

This saves time, reduces effort, and makes software development more accessible.

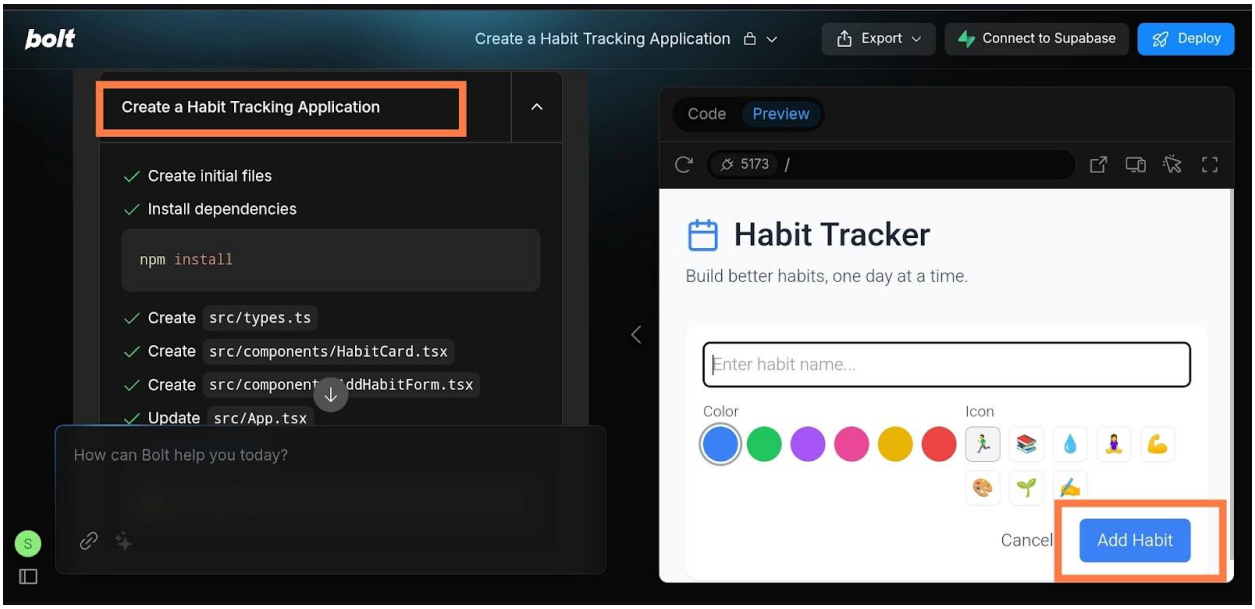
For example, I wanted to build a habit tracking app, and here's how Bolt.new helped me.

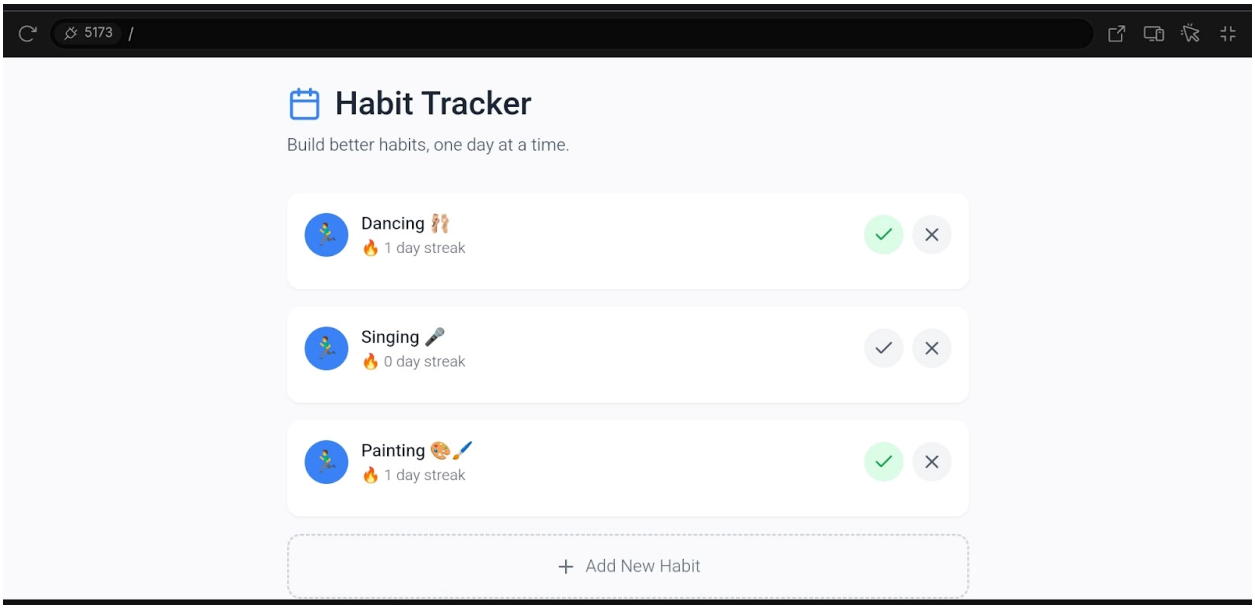


The tool does its coding brilliantly. It ensures you don't need to know coding for app development.



Here's how the app looks:





Standout features:

- **AI Code Generation:** Users can create entire applications using short text prompts.
- **Browser-Based Development:** No need to install software; everything runs in the browser.
- **Supports Multiple Frameworks:** It works with Astro, Vite, Next.js, Vue, Svelte, and Remix.
- **Easy Deployment:** You can connect it with Netlify for quick hosting and launch.

- **Real-Time Collaboration:** It allows multiple users to work on the same project at the same time.

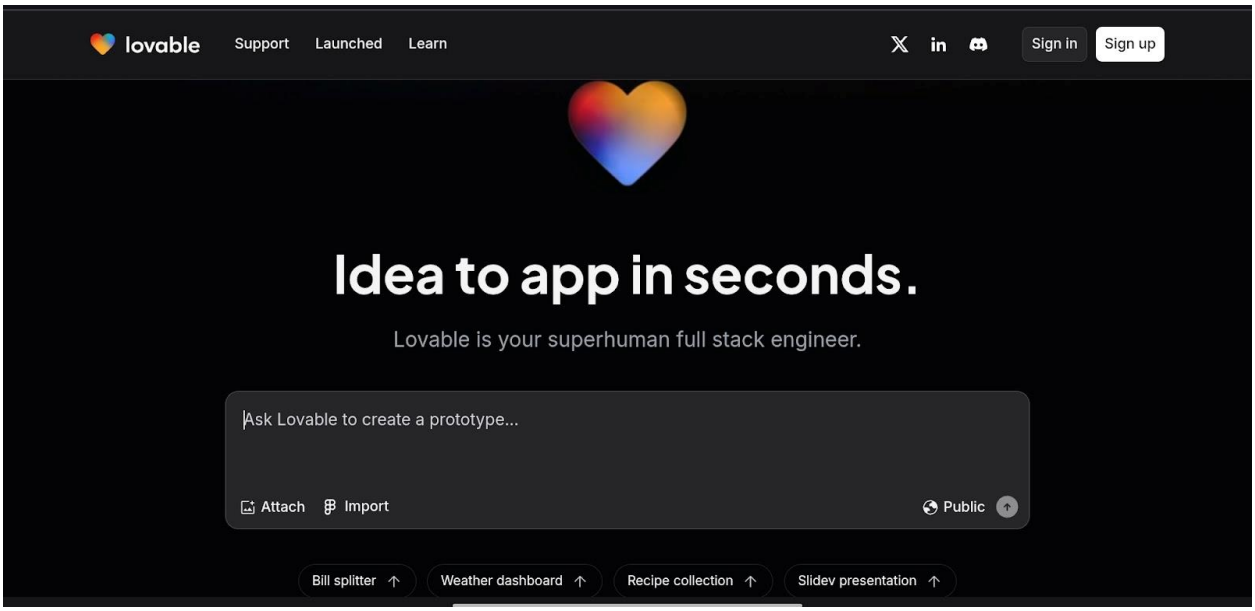
Why we recommend it:

This is the best tool for prompting, running, editing, and deploying web and mobile applications. Bolt.new stands out for its user-friendly interface, extensive framework support, and seamless deployment capabilities. These features make it invaluable for rapid prototyping and efficient development workflows.

Pricing:

Bolt.new offers different plans. The Pro 50 plan costs $50 per month for 26 million tokens. The Pro 100 plan costs $100 per month for 55 million tokens, and the Pro 200 plan costs $200 per month for 120 million tokens.

2.Lovable

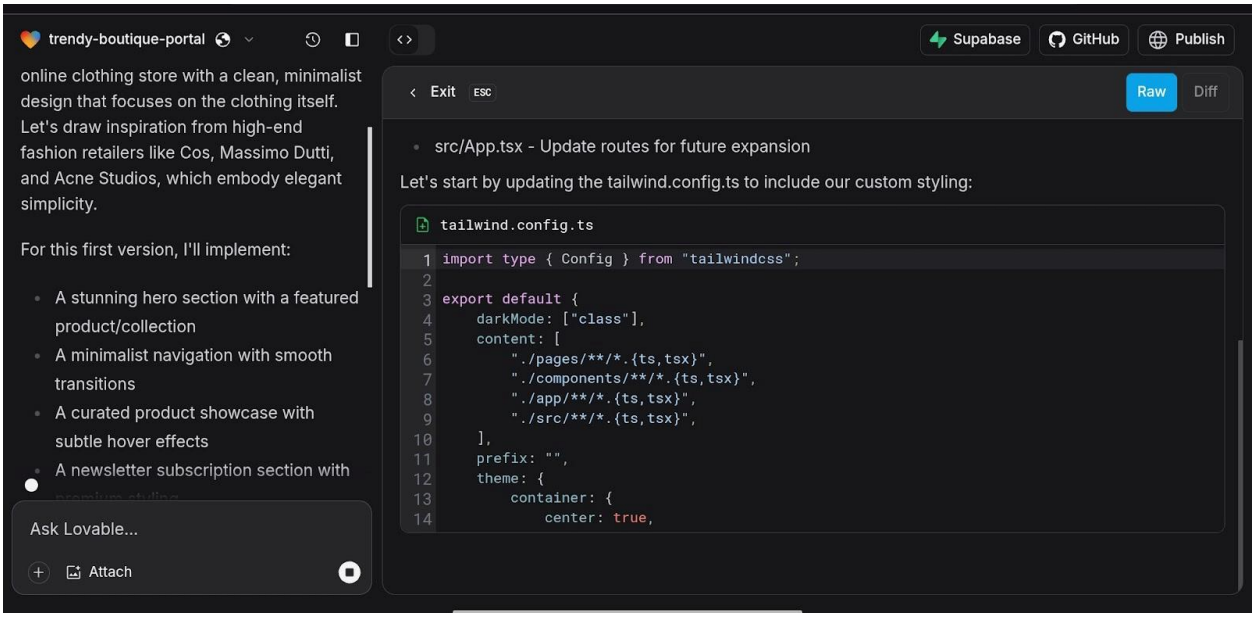


Lovable is an AI-powered platform that helps users build full-stack web applications without writing any code. Instead of learning programming languages, users can simply describe their ideas in plain English, and the tool generates the app. This makes app development faster and more accessible to everyone. Lovable also includes features for designing, coding, and launching applications, all from a single browser tab.

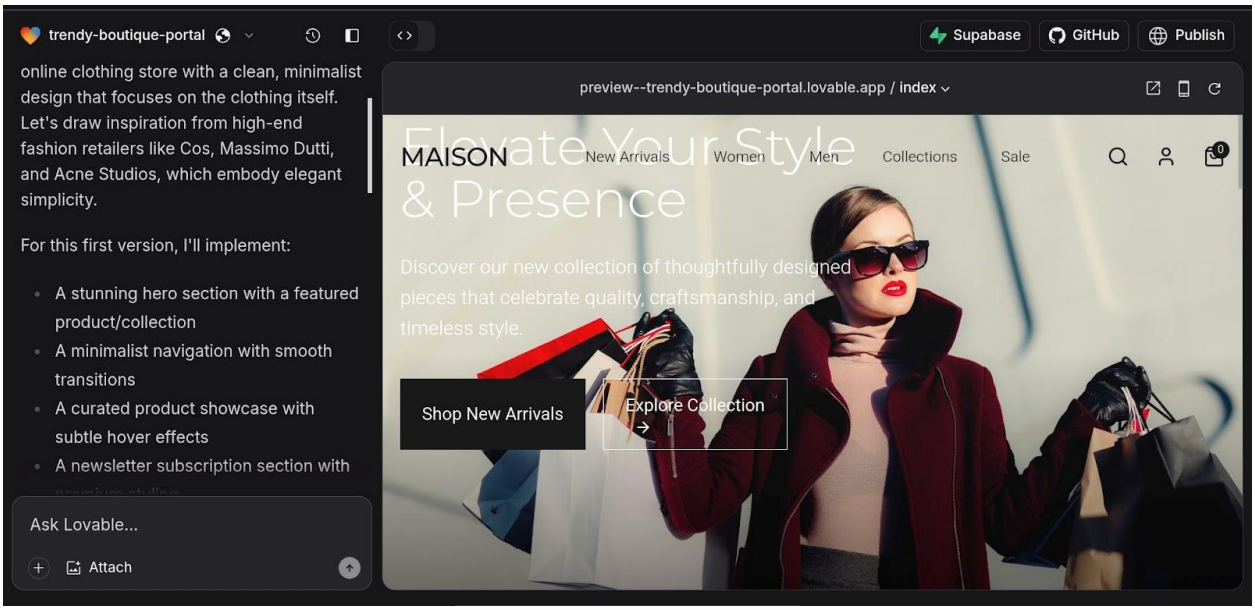
How Lovable helps developers and non-coders:

Lovable makes app development easy for everyone. Beginners can create apps without learning to code, while experienced developers can speed up their work. It removes technical barriers so more people can bring their ideas to life.

I wanted to build a landing page for my new online clothing store. I provided a simple prompt to Lovable to design one for me, and here's how the tool codes.



Finally, you can check out my landing page on Lovable here.



Standout features:

- **Natural Language Processing:** You can describe your app idea in plain English, and Lovable generates its corresponding application.
- **Visual Editing:** You can modify and refine your application's UI by providing text-based instructions to the tool, eliminating the need for manual coding adjustments.
- **Automated Debugging & Code Suggestions:** Its AI feature detects errors in codes and provides fixes.
- **Real-Time Collaboration:** Your teams can work together on projects simultaneously.
- **Backend and Database Support:** It connects easily with services like Supabase for storing data.
- **One-Click Deployment:** Apps can be published and shared with a single click.
- **Code Ownership:** Users retain full ownership of the generated code, with options to sync to GitHub for further customization.

Why we recommend it:

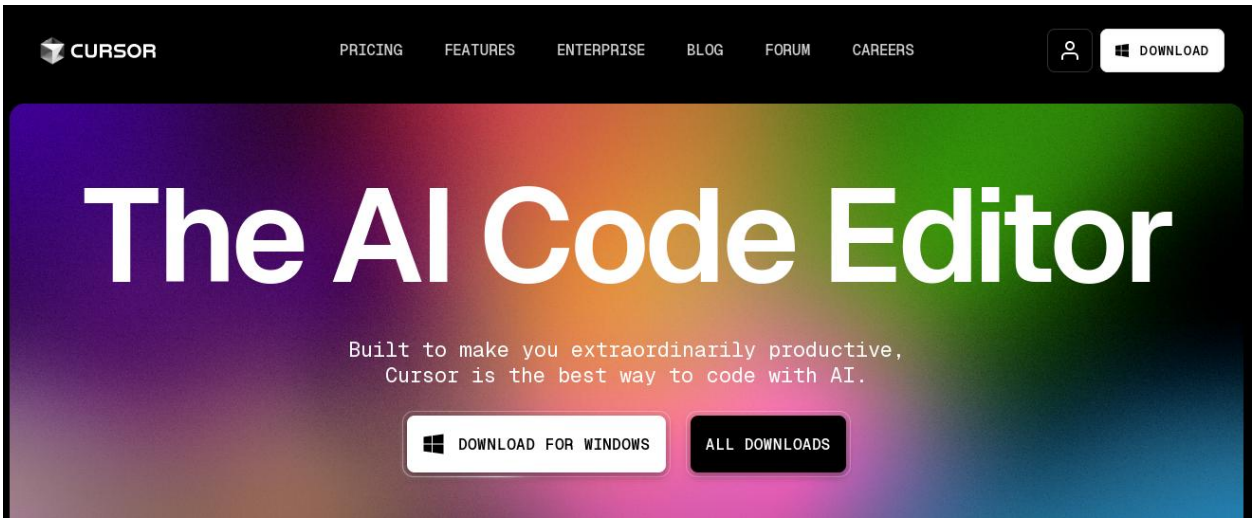
We recommend Lovable for rapidly converting text-based ideas into fully functional applications without requiring coding skills. Its visual editing, AI-powered development, and complete code ownership make it a powerful tool for developers and non-tech users.

Plus, it seamlessly integrates with databases and APIs, streamlining backend functionality. It is your superhuman full-stack engineer, converting ideas into apps in seconds.

Pricing:

Lovable offers a free plan with limited features, but you can buy the Starter plan at $20/month, the Launch plan at $50/month, and the Scale plan at $100/month.

3.Cursor



Cursor is an AI-powered integrated development environment (IDE) designed to enhance developer productivity by integrating advanced artificial intelligence features directly into the coding environment. It is a fork of Visual Studio Code with additional AI features like code generation, smart rewrites, and codebase queries.

How Cursor helps developers and non-coders:

Cursor helps developers by automating repetitive coding tasks, providing AI-powered code suggestions, and debugging errors instantly, making software development faster and more efficient. Its in-code chat and context-aware retrieval simplify complex queries, reducing the time spent on documentation searches.

Cursor translates natural language into executable commands for non-coders, making it easier to interact with codebases. Features like @Web and image input allow users to access real-time information and visual context, enabling smoother collaboration and learning.

Standout features:

- **Agent Mode** – Unlike many AI code assistants, Cursor's agent mode can handle end-to-end task execution while keeping developers in control.
- **Custom Context Retrieval** – The Cursor understands an entire codebase using advanced retrieval models, reducing the need for manual context setup.

- **Automated Terminal Commands** – The Cursor suggests and executes terminal commands with confirmation, a feature less common in similar tools.
- **Looping on Errors** – Auto-detects and fixes lint errors, minimizing debugging efforts—something most AI editors don't do natively.
- **Cursor Prediction** – Anticipates your cursor's next position, allowing seamless navigation and coding flow.
- **Codebase-Wide Chat** – The Cursor's chat can analyze and interact with your entire codebase, not just the open file.
- **Visual Context with Images** – Supports image-based context input, which is rarely seen in AI coding assistants.

Why we recommend it:

We recommend Cursor because it dramatically speeds up coding with AI-driven autocomplete, intelligent debugging, and seamless code editing. Its in-code chat and real-time suggestions reduce friction, making development smoother.

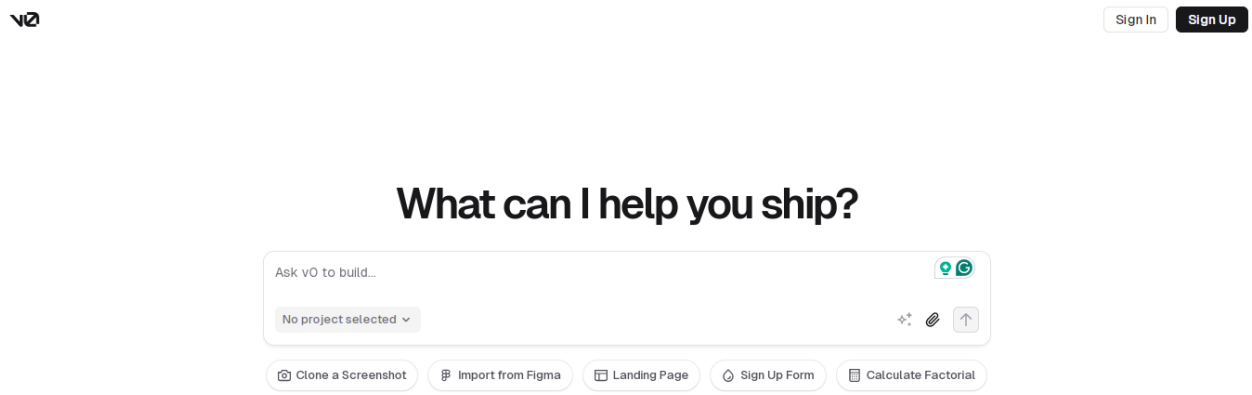
What truly stands out is how it learns my coding style over time, making its suggestions sharper and more intuitive with every use.

For debugging, Cursor has been a game-changer—catching errors, and we would have spent hours chasing down and offering fixes before we even realized a problem.

Pricing:

After a 2-week free trial, you must subscribe for the Pro version at $20/month or the Business version at $40/month.

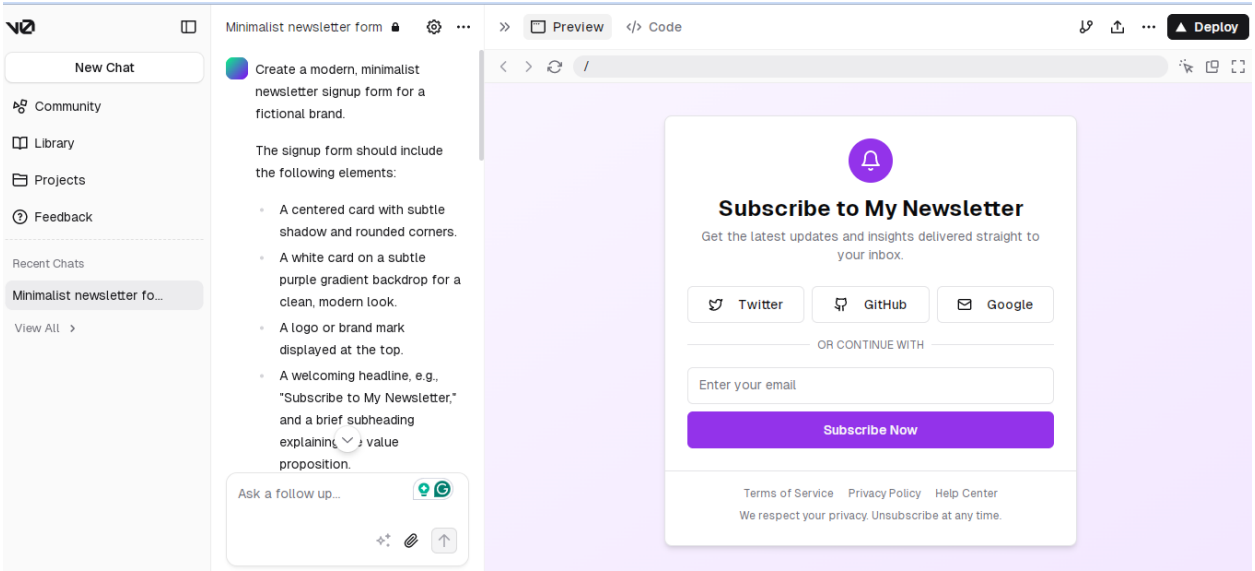
4.v0 by Vercel



v0 by Vercel is an AI-powered tool that converts natural language prompts into fully functional React components. It helps developers create UI components quickly using frameworks like Shadcn UI and Tailwind CSS. This tool enables rapid prototyping, making designing and iterating on user interfaces easier without writing extensive code. v0 by Vercel also integrates with design tools like Figma, allowing seamless transitions from design to development.

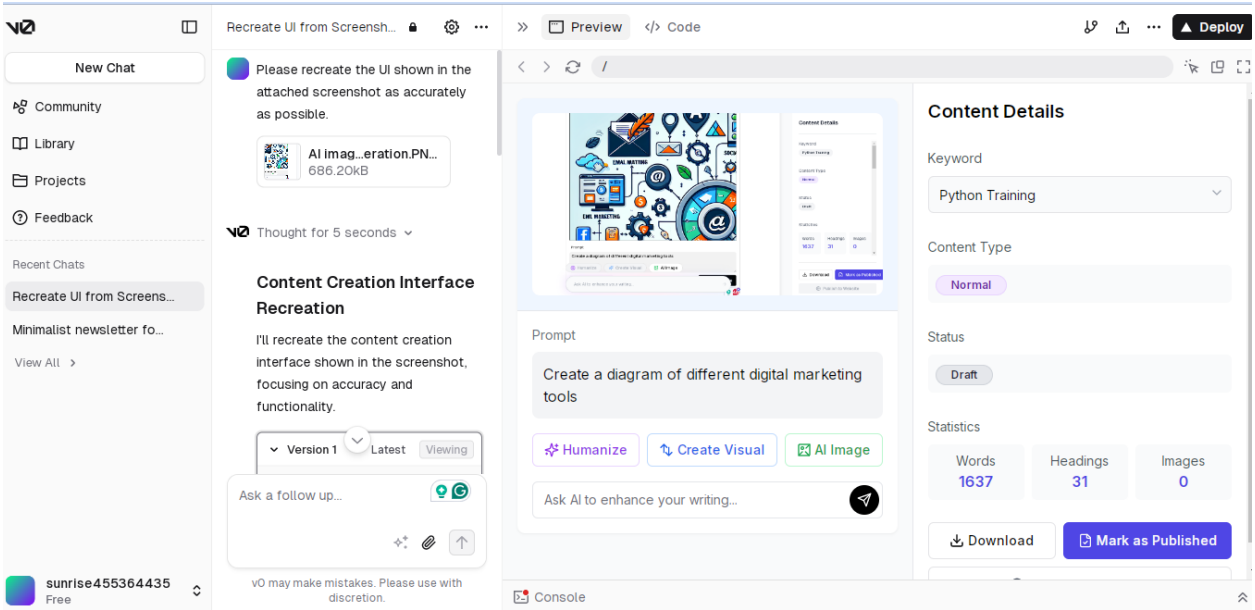
How v0 by Vercel helps developers and non-coders:

v0 by Vercel simplifies UI development by allowing users to describe their design ideas in natural language. The AI then generates functional components, making it easier for developers and non-coders to build interfaces without deep programming knowledge. This speeds up development and encourages creativity.

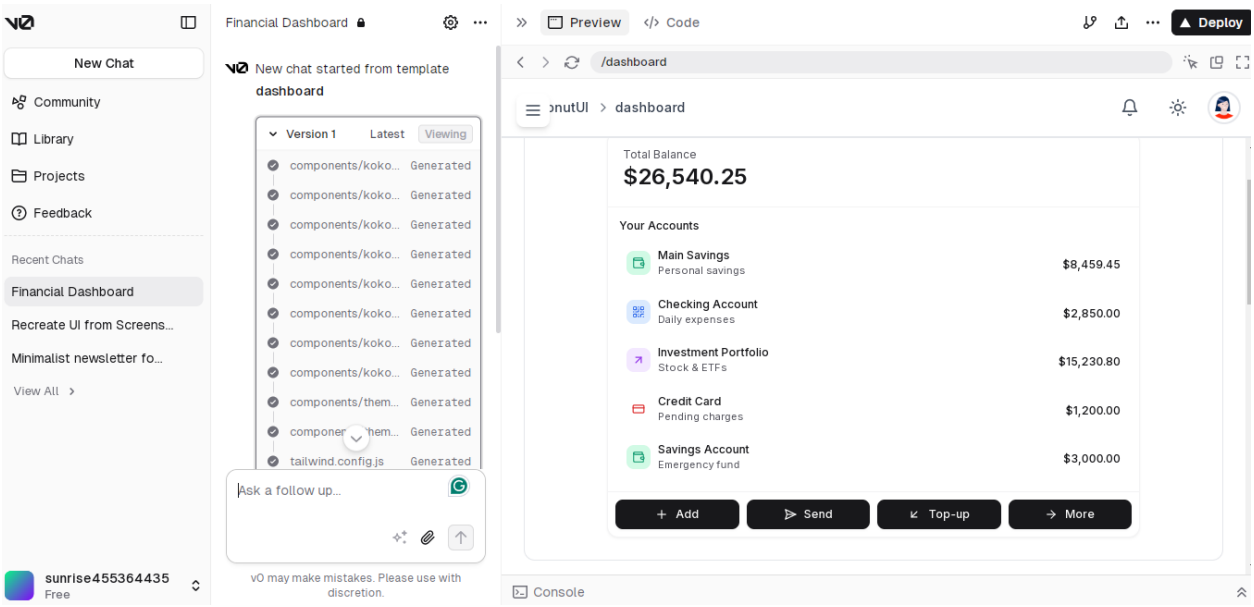


Standout features:

- **AI-generated UI components:** v0 converts text prompts into production-ready React components.



- **Responsive layouts:** It ensures that UI designs adapt to different screen sizes and devices.
- **Interactive elements:** The tool supports animations, transitions, and interactive components.



- **Real-time collaboration:** Multiple users can work on a project simultaneously, improving team productivity.
- **Figma integration:** Users can instantly import Figma designs and turn them into React code.

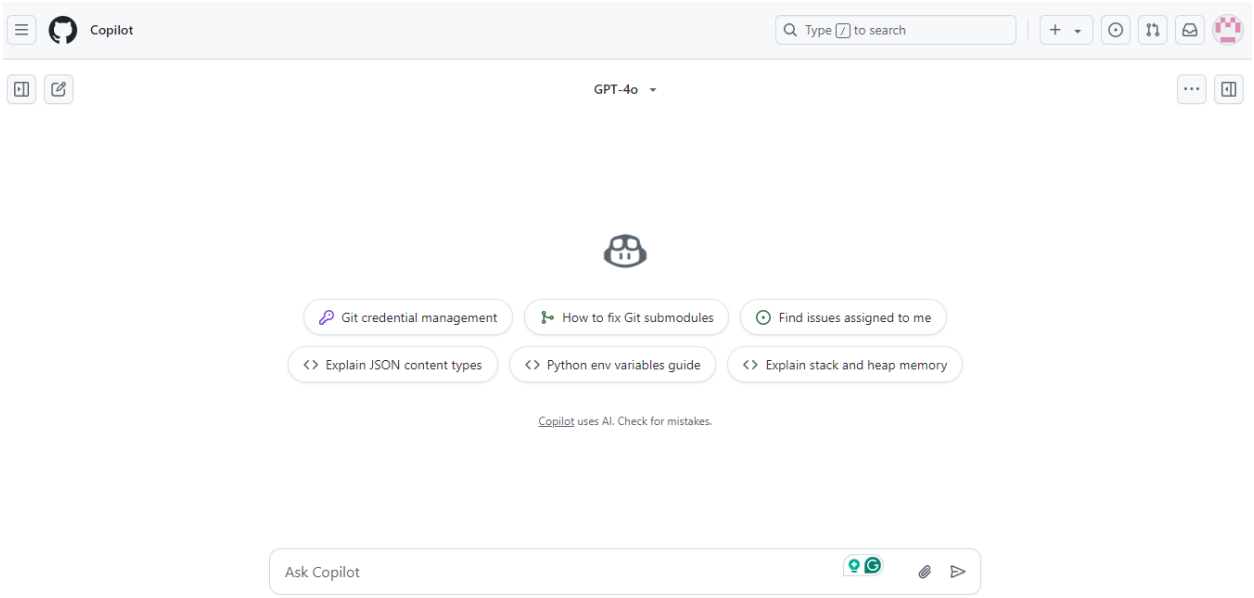
Why we recommend it:

v0 by Vercel is ideal for developers and designers who want to create UI components quickly without manual coding. Its AI-driven approach, seamless Figma integration, and real-time collaboration features make it a powerful tool for modern web development.

Pricing:

You can start with $0/month and buy the Premium plan at $20/month and the Ultra plan at $200/month.

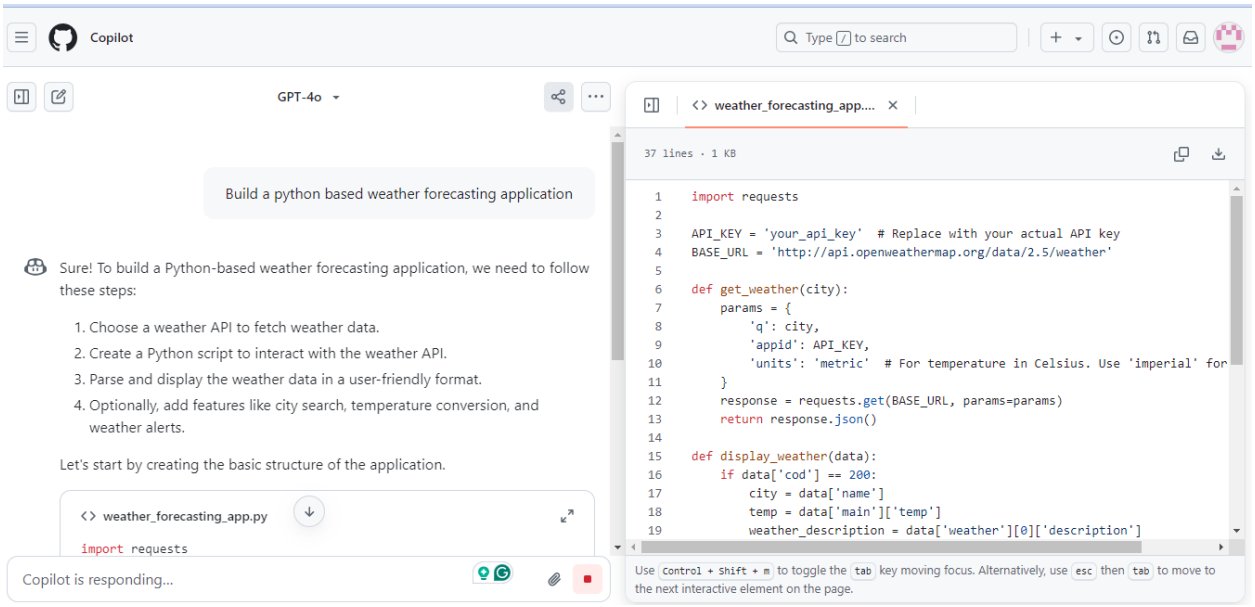
5.GitHub Copilot



GitHub Copilot is an AI-powered coding assistant developed by GitHub and OpenAI. It integrates into code editors to provide real-time code suggestions and completions, enhancing coding efficiency and accuracy. By analyzing context and comments, Copilot assists developers in writing code faster and with fewer errors.

How GitHub Copilot helps developers and non-coders:

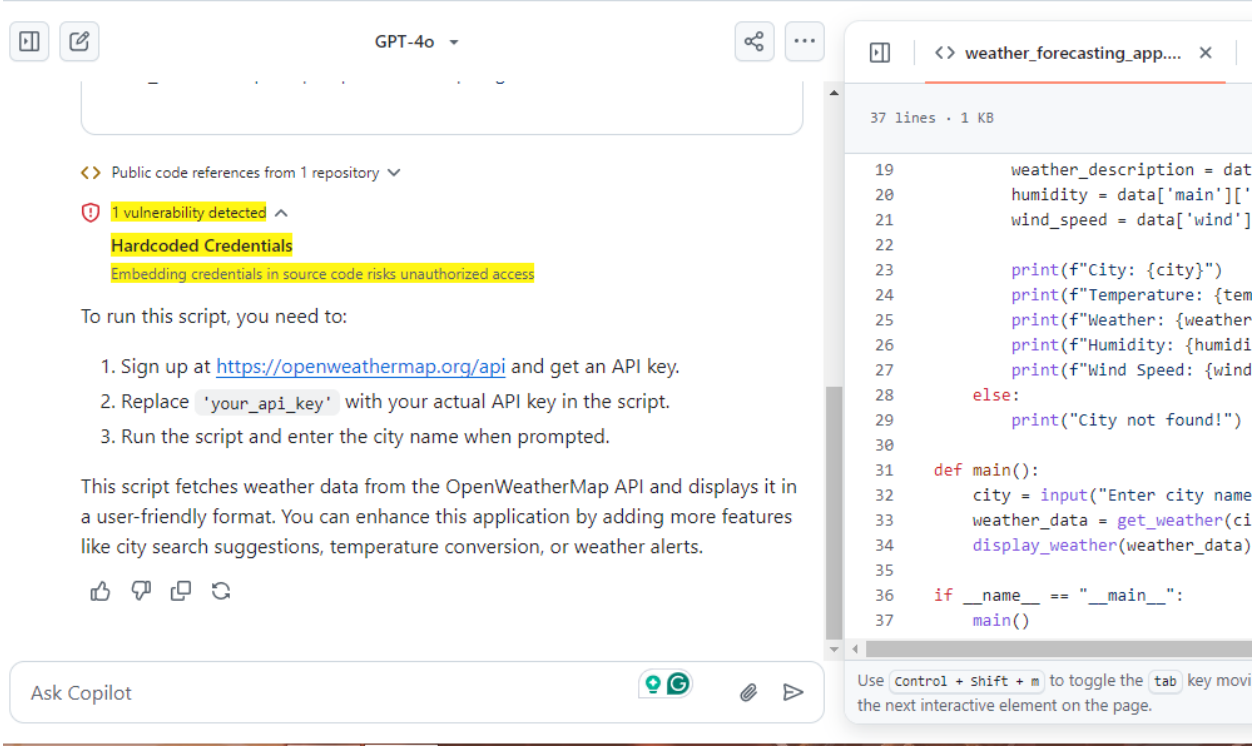
GitHub Copilot assists developers by offering code suggestions, explanations, and bug fixes, streamlining the coding process. For non-coders, it serves as an educational tool, providing insights into code functionality and aiding in learning programming concepts.



Standout features:

- **Code Completion:** It offers real-time code suggestions as you type, speeding up the coding process.
- **Code Explanation:** GitHub Copilot provides natural language explanations for code snippets, aiding understanding.
- **Bug Detection:** It identifies potential errors and suggests fixes, improving code quality.

While I was trying to execute a Python-based weather forecasting application, the tool itself detected a bug.



- **Learning Support:** You will learn new programming languages and frameworks through contextual code examples.

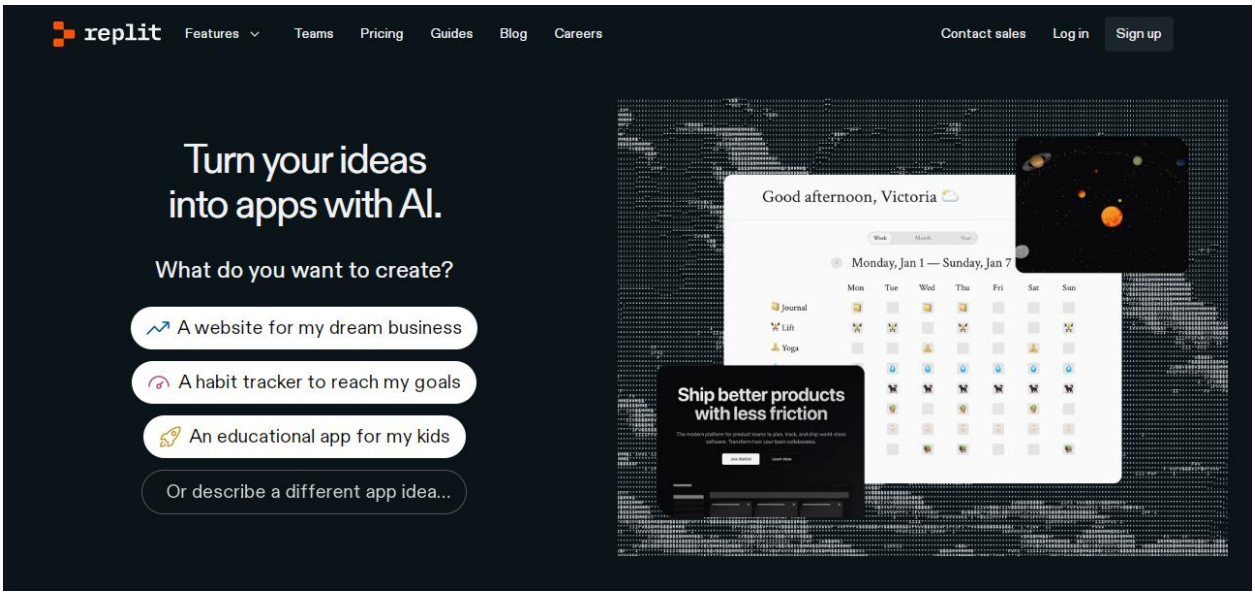
Why we recommend it:

We recommend GitHub Copilot because it adapts to your coding habits, making development faster and more intuitive. It's best to work with Python, JavaScript, TypeScript, Ruby, and Go, but it supports many other languages. The smooth VS Code integration and multiple suggestion options keep you in control while boosting productivity.

Pricing:

The basics of GitHub Copilot are free for individuals and organizations, but to buy the Team plan, you need to pay $4 per user/month, and the Enterprise plan for $21 per user/month.

6.Replit

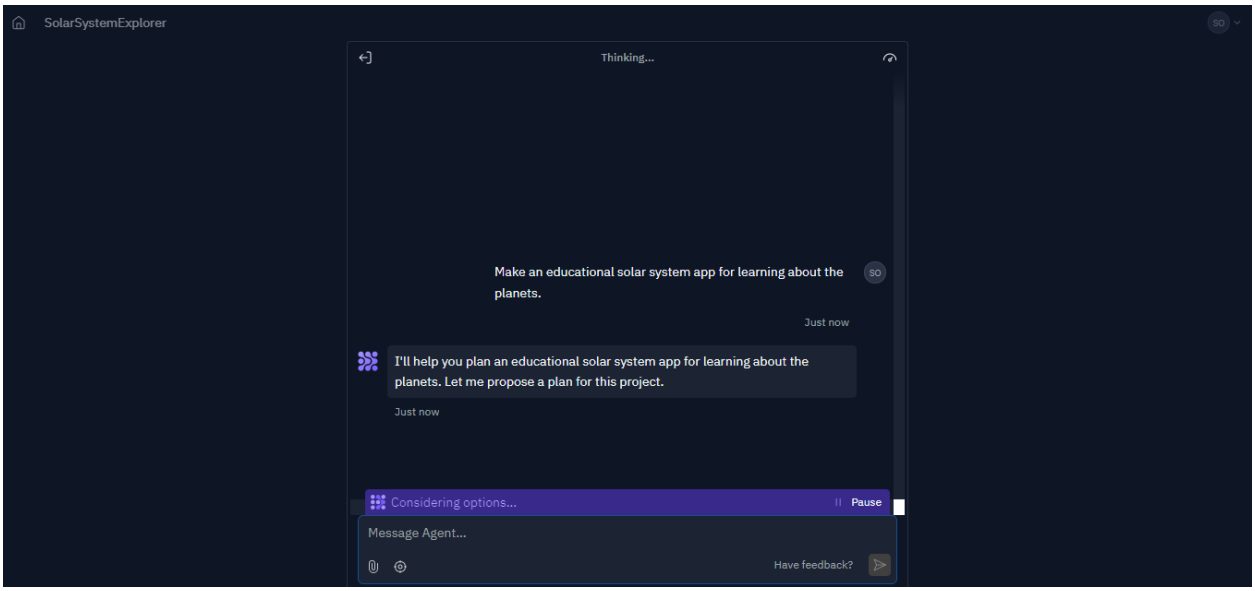


Replit is a browser-based AI-powered coding platform that allows users to create, collaborate, and deploy applications instantly. It supports 50+ programming languages and includes Replit Agent, an AI assistant that automates coding tasks. Developers and beginners can build websites, apps, and software efficiently without complex setups.

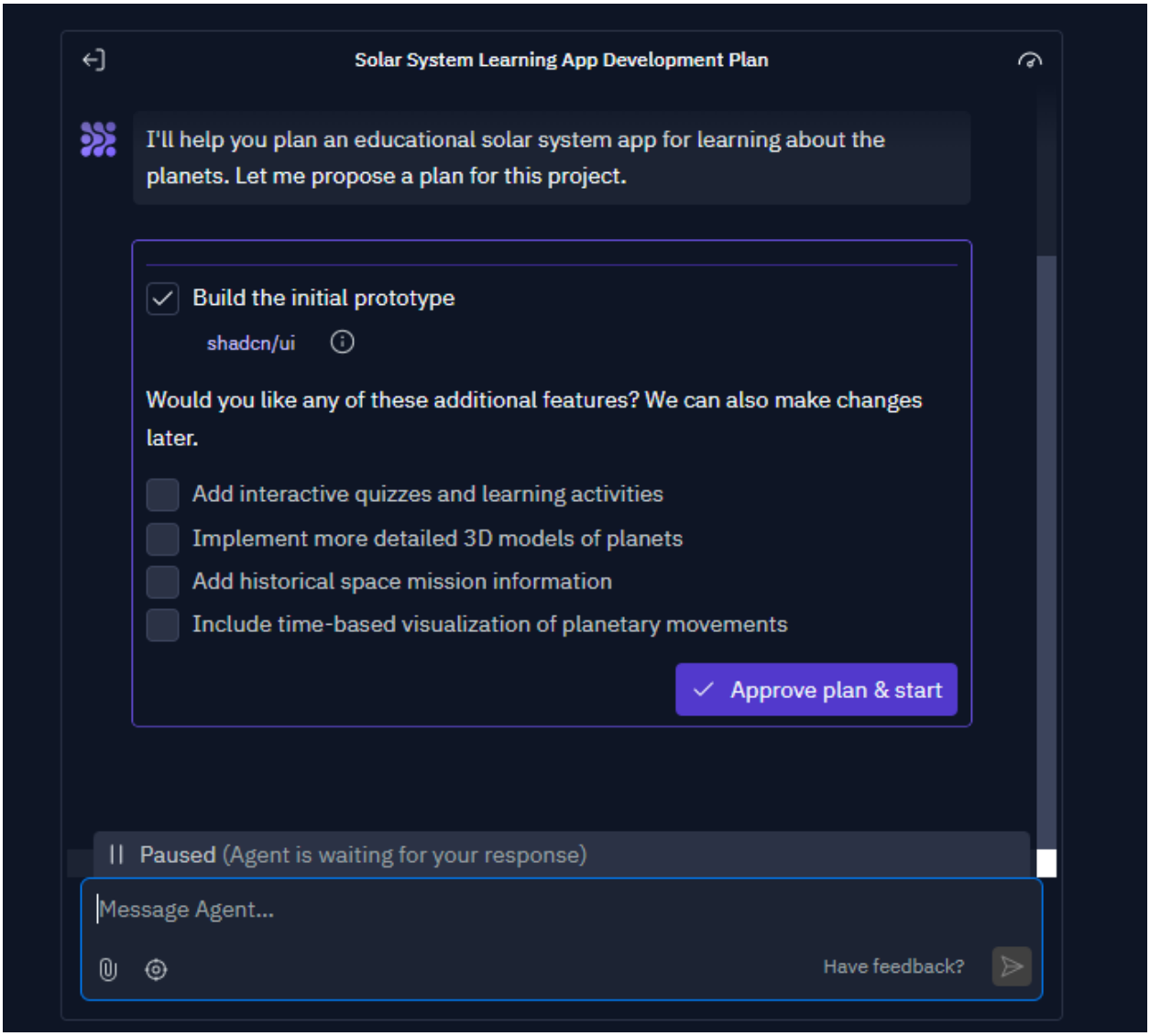
How Replit helps developers and non-coders:

Replit simplifies coding by providing an accessible platform where users can start coding without any setup. Its collaborative features allow teams to work together seamlessly, and AI tools assist in learning and writing code efficiently.

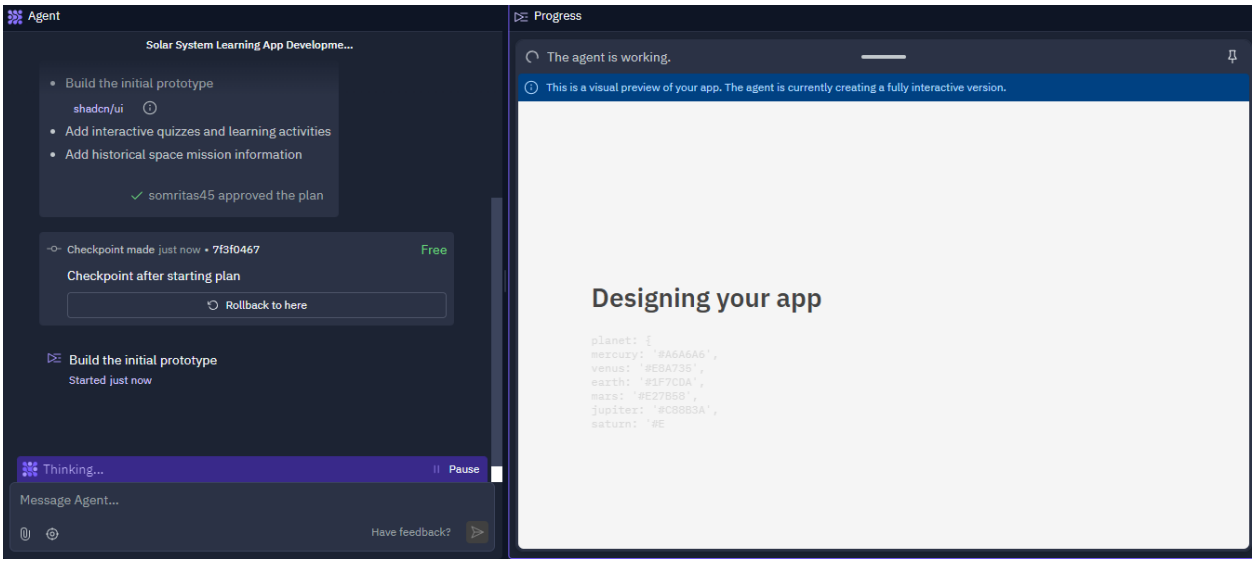
We designed a solar system explorer just by instructing a Replit AI agent.



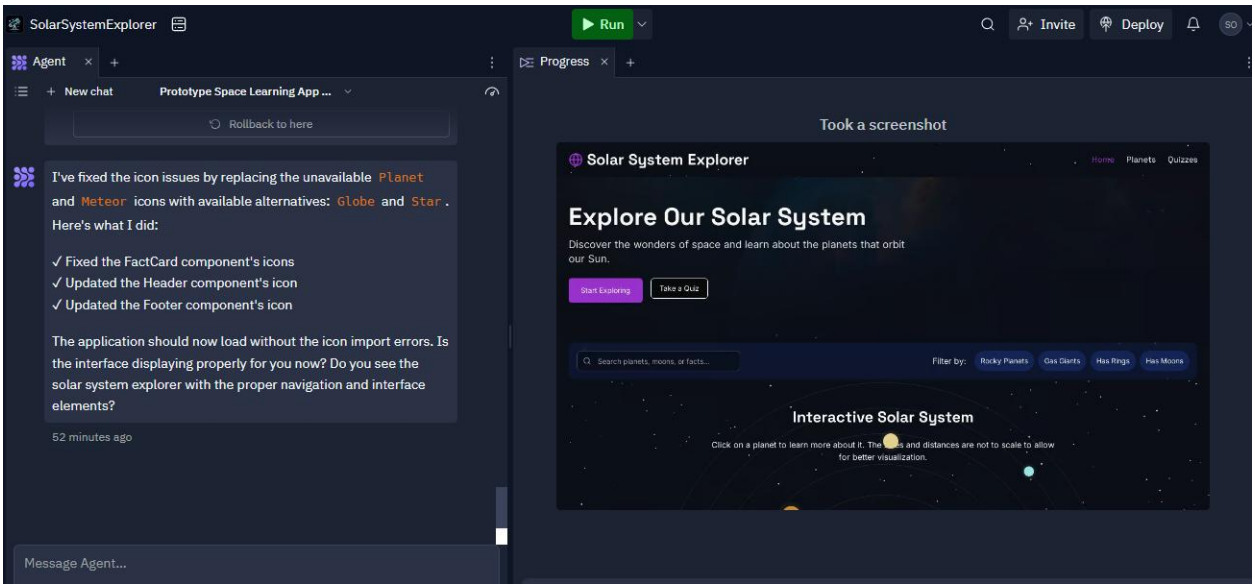
Before creating the application, Replit asked for our choices and selected its plan.



Replit shows us how it designs an application, and we can access its code as well.



Here, we can see the design preview, which is pretty enough to engage the audience.



Standout features:

- **Cloud-Based Development** – Replit allows developers to code, run, and deploy applications directly from a web browser, eliminating the need for complex local setups.
- **Multi-Language Support** – It supports over 50 programming languages, including Python, JavaScript, Java, and C++, making it a versatile platform for developers.

- **AI-Powered Assistance** – Replit provides an AI-powered coding assistant that helps generate, debug, and optimize code, improving efficiency and reducing development time.
- **Instant Deployment** – Developers can deploy full-stack applications with a single click, simplifying the deployment process without requiring manual configurations.
- **Built-in Database & Hosting** – The platform includes integrated databases and always-on hosting, allowing developers to build, test, and host applications in one place.
- **Security & Privacy Controls** – Users can create private Repls, manage access permissions, and ensure data security while working on their projects.
- **Version History & Rollback** – The platform tracks changes and allows developers to restore previous versions, ensuring that progress is never lost.
- **Interactive Learning & Templates** – Replit provides pre-built templates and educational resources that help beginners and professionals accelerate their learning and development.

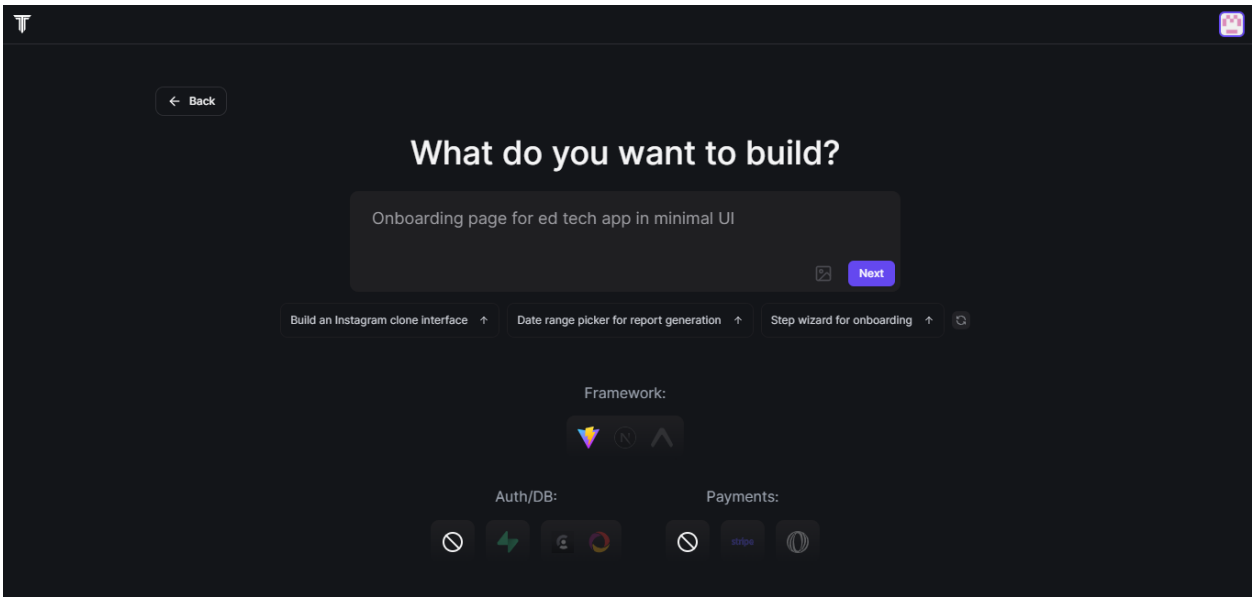
Why we recommend it:

What we appreciate most about Replit is its Google Docs-style editing, which makes collaborating with teammates effortless. The seamless import, execution, and collaboration on millions of GitHub repositories without any manual setup have revolutionized the way we teach and mentor junior developers.

Pricing:

The Starter plan is free to use, but you can buy Replit's Core plan at $20/month.

7.Tempo

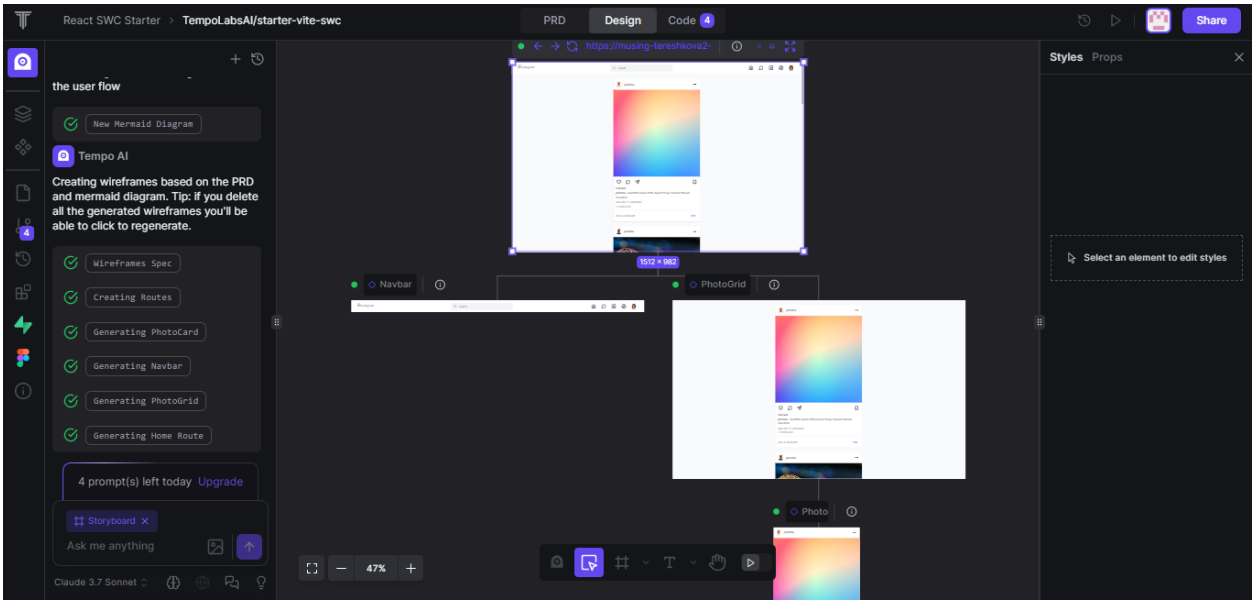


Tempo is an AI-powered design tool that enables designers and developers to collaboratively create React applications more efficiently. It offers a drag-and-drop editor for visual coding, allowing users to edit React code productively. With features such as component library maintenance, integration with version control (GitHub), and unlimited design revisions, Tempo makes it easier to build and manage UI/UX.

How Tempo helps developers and non-coders:

Tempo assists developers by automating UI code generation, allowing them to focus on business logic. Non-coders, like designers, can use the visual editor to make precise design adjustments without writing code.

I am a non-coder, and I want to build an Instagram clone interface. Here's how Tempo helps me.



Standout features:

- **AI-Powered UI Generation** – Tempo allows developers to generate full UI components and app views using simple natural language prompts.
- **Seamless React Integration** – The tool works with any existing React codebase, ensuring flexibility and no vendor lock-in.
- **Component Importing** – Developers can import their existing React components from libraries like Storybook for a more personalized experience.
- **Design System Generation** – Tempo can automatically generate custom component libraries using frameworks like MUI, Chakra UI, Tailwind, and others.
- **Visual IDE:** Tempo Labs is a visual IDE that enables designers, PMs, and developers to collaborate on code with the help of AI.
- **Pre-Built Templates** – The platform offers a wide range of ready-made component templates that help speed up UI development.
- **Faster UI Development** – By leveraging AI-powered automation, developers can create and ship user interfaces up to ten times faster than traditional methods.
- **Git and Version Control Support** – Users can seamlessly integrate with GitHub to track changes and maintain code quality.
- **Full Code Ownership** – Unlike low-code tools, Tempo ensures that developers retain full control over the generated React code.
- **Extensive Integrations** – Tempo works with various technologies, including React, JavaScript, TypeScript, Storybook, and multiple UI frameworks.

- **AI Reasoning and Gemini Search:** Tempo leverages AI reasoning and Gemini Search for enhanced functionality.

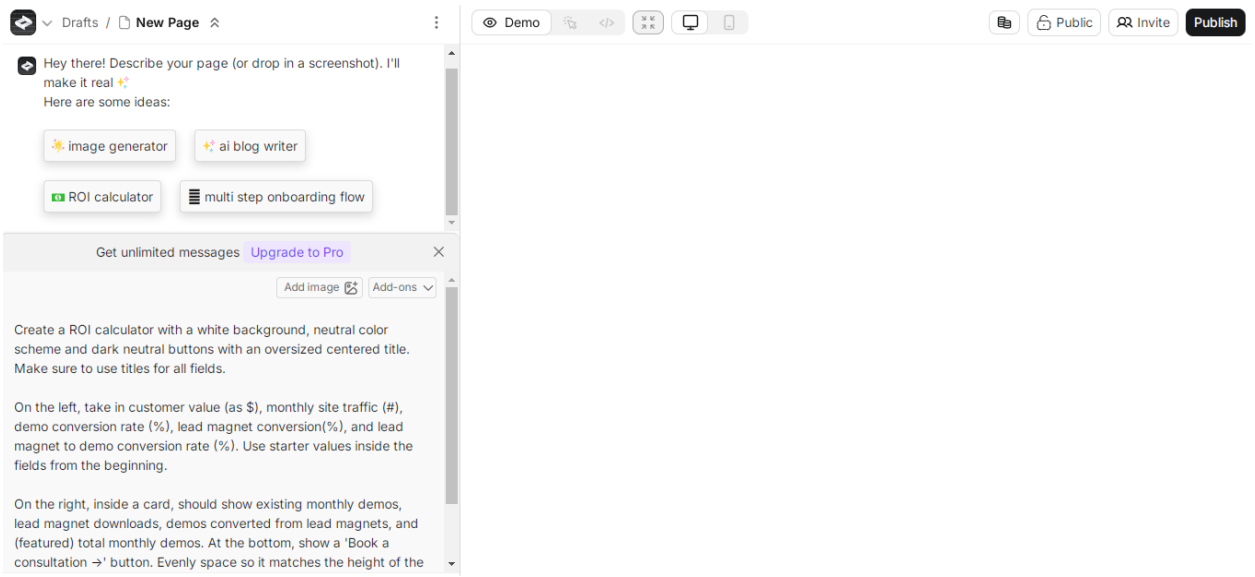
Why we recommend it:

Tempo bridges the gap between design and development, offering a unified platform that enhances collaboration and efficiency. Its AI capabilities and seamless integration with existing codebases make it a valuable tool for teams aiming to accelerate their development process.

Pricing:

You can start with $10/month or purchase their Pro plan for $30/month and Agent plan for $4000/month.

8.Create



Create is an AI-powered tool that helps you build websites, apps, and tools just by describing them in simple words. It turns your text into real code and runs your app instantly. This makes it easy for both coders and non-coders to create software without learning complicated programming.

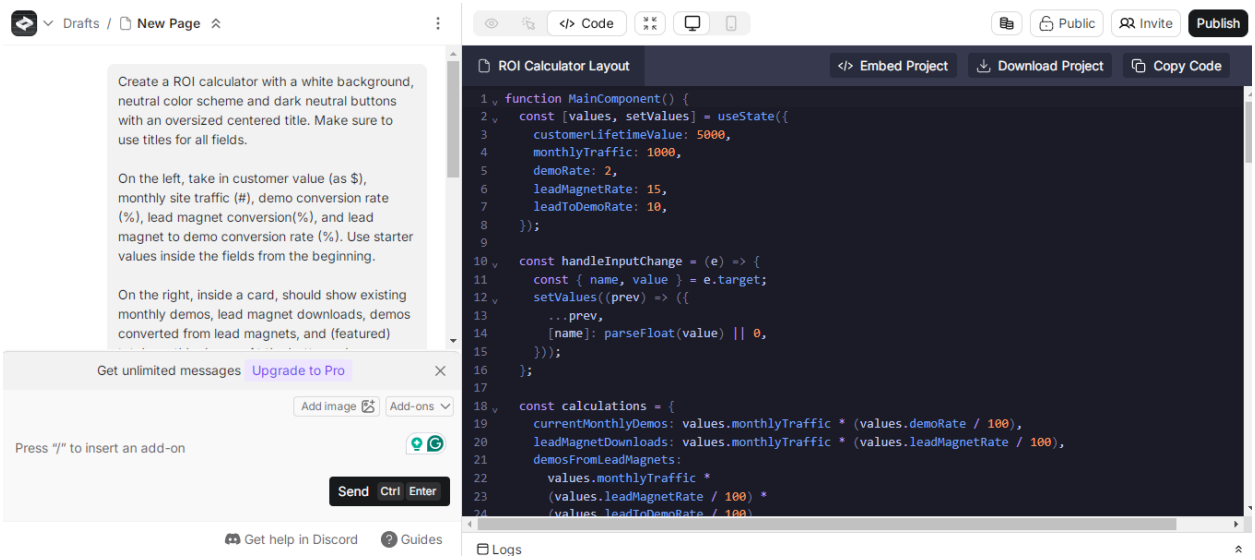
How Create helps developers and non-coders:

Create combines multiple AI tools into one platform, saving time and effort. Users can edit visuals directly, making app development easier for beginners. It connects components like calculators and dashboards, creating interactive applications.

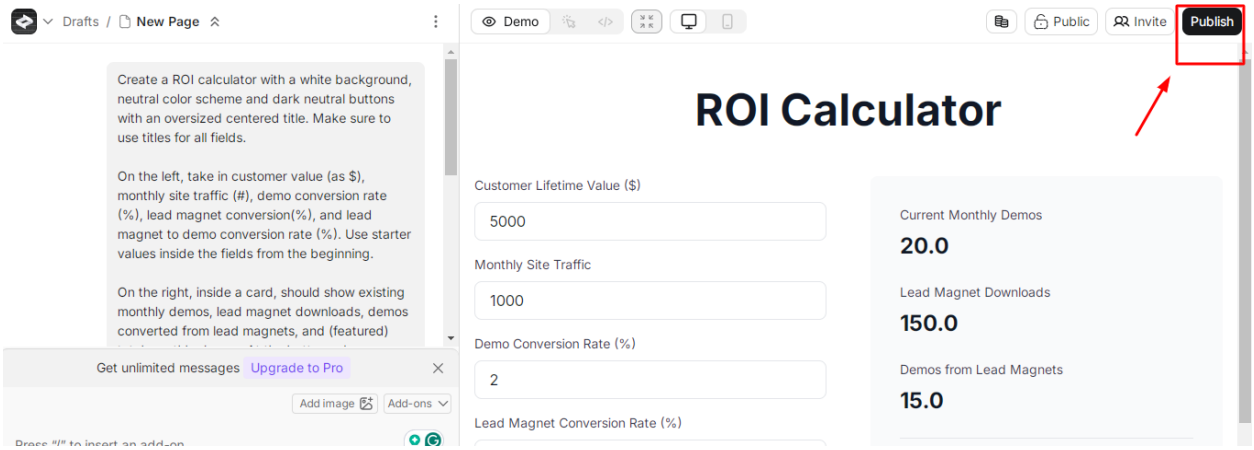
The platform supports real-time data analysis, helping in finance and marketing. It ensures clean data presentation and provides AI-powered recommendations. With its flexible features and user-friendly design, Create stands out, making it a great tool for both developers and non-coders.

Standout features:

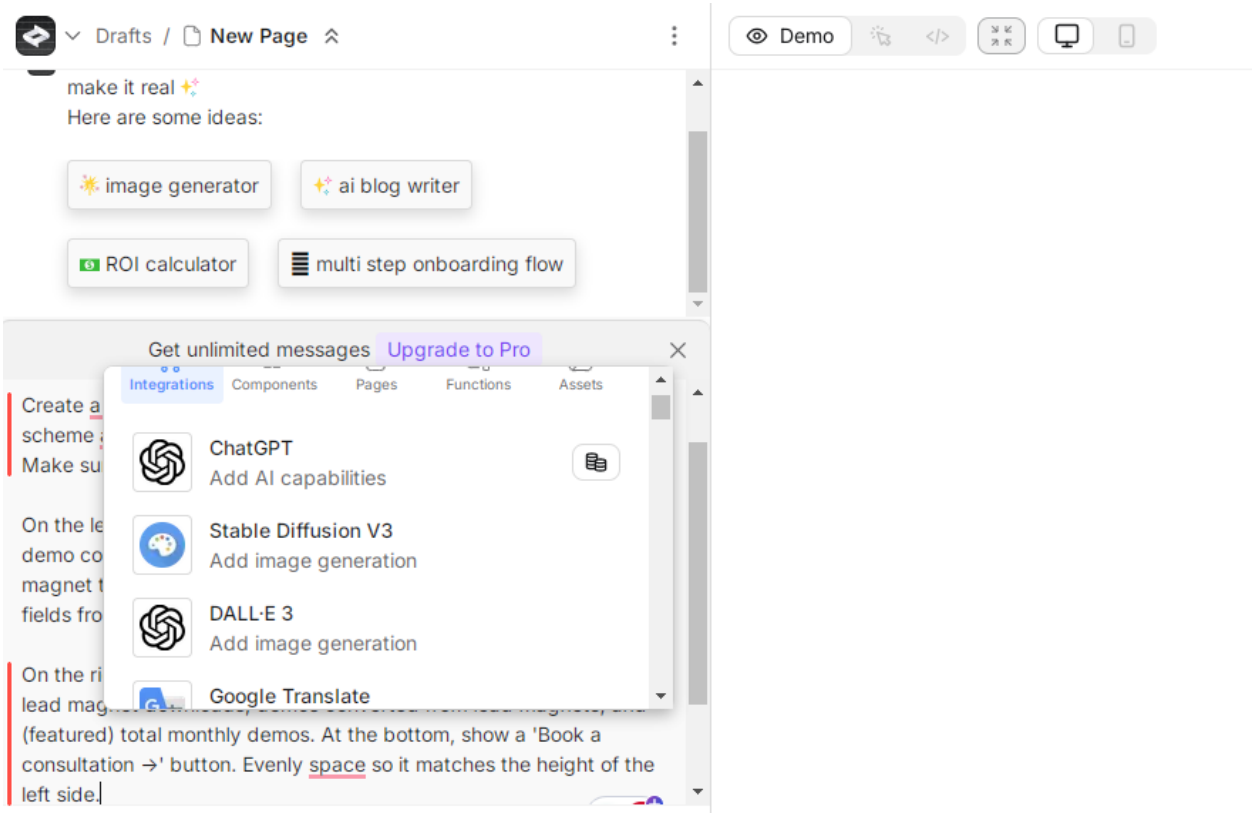
- **Code in English:** You can type what you want in plain English, and Create will generate the code for you.



- **AI Superpowers:** Create allows you to integrate AI models like ChatGPT and GPT Vision into your applications.
- **Turn Images into Apps:** You can upload an image of a design, and Create will generate a working application based on it.
- **Build Custom Components:** You can create and reuse components to speed up development and maintain a consistent design.
- **Connect to APIs:** Create lets you connect to your own REST APIs by uploading Swagger documentation.
- **Take Your Code Anywhere:** You can export or copy the generated code anytime, so you are never locked into the platform.
- **One-Click Publishing:** You can instantly publish and share your project with others with just one tap.



- **Compatibility with Multiple AI Models**: The platform supports the integration of various language models, allowing users to choose between different AI systems for generating content, which can lead to diverse outputs.



Why we recommend it:

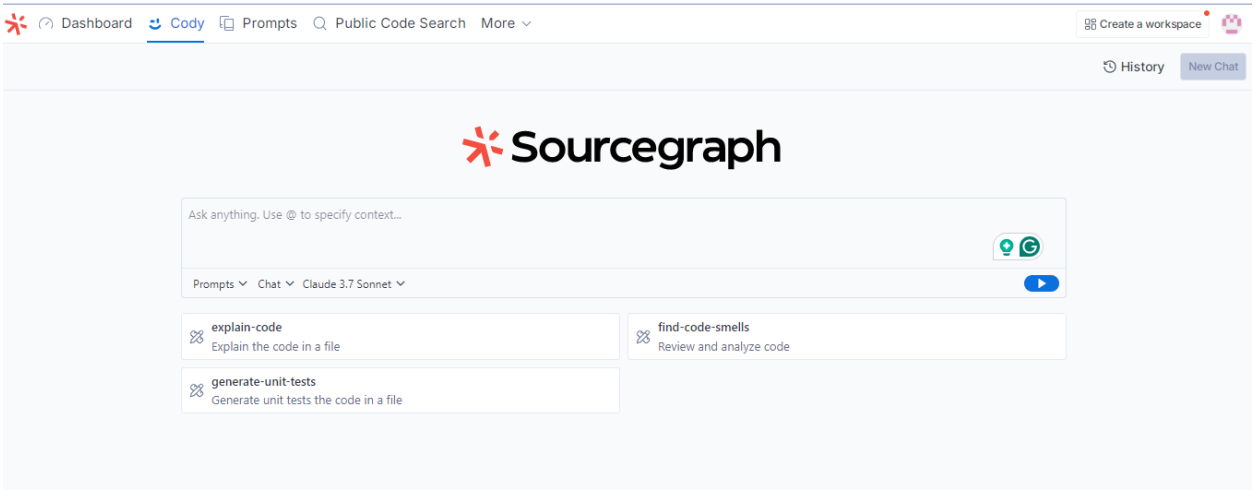
We recommend this tool for its ability to edit visual aspects directly within the tool. This is a significant upgrade over many existing platforms. This trait

enhances usability, particularly for individuals who are visually oriented rather than technically inclined. By focusing on user experience, Create positions itself as an accessible solution for aspiring app developers.

Pricing:

You can start using Create for free and then purchase the Pro plan at $16/month and the Business plan at $83/month.

9.Cody



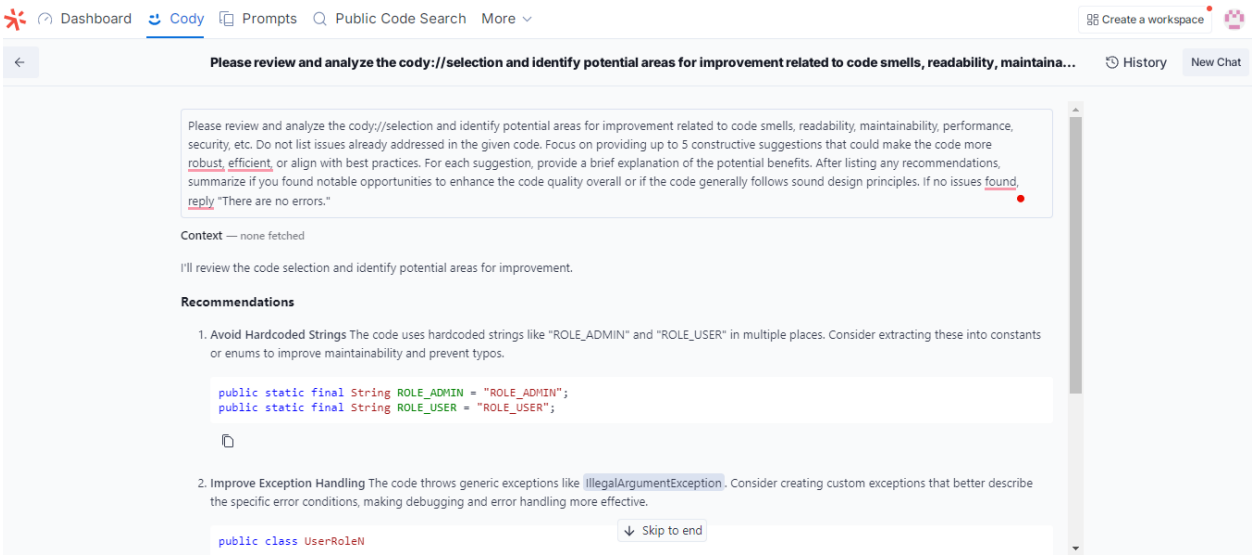
Cody is an enterprise AI-powered coding assistant developed by Sourcegraph. It integrates with various Integrated Development Environments (IDEs) to enhance coding efficiency and accuracy. Cody leverages advanced Large Language Models (LLMs) to provide real-time code completions, edits, and contextual assistance.

How Cody helps developers and non-coders:

Cody boosts productivity by automating repetitive coding tasks, allowing developers to focus on complex problems. Its autocomplete feature suggests accurate code snippets, reducing errors and improving workflow. Users can create custom commands, get context-aware assistance, and integrate with tools like Notion. Cody also detects and fixes errors, speeds up debugging, and simplifies documentation, making coding easier for both developers and non-coders.

Standout features:

- **Code Completions:** Cody offers real-time code suggestions to improve development.
- **In-line Edits:** It allows direct code modifications within the editor for seamless refactoring.
- **Agentic Chat:** It provides AI-driven conversational assistance to address coding queries.
- **IDE Support:** This tool is compatible with multiple IDEs, including VS Code, Visual Studio (experimental), Eclipse (experimental), IntelliJ, PyCharm, and more.
- **Context Integration:** Cody gathers context from tools like Notion, Linear, and Prometheus to deliver relevant assistance.
- **Error Detection and Correction:** Cody identifies mistakes in code and suggests fixes, reducing debugging time and ensuring smoother execution.



- **Flexible Deployment:** It supports both cloud-based and on-premises installations to meet diverse security and compliance requirements.

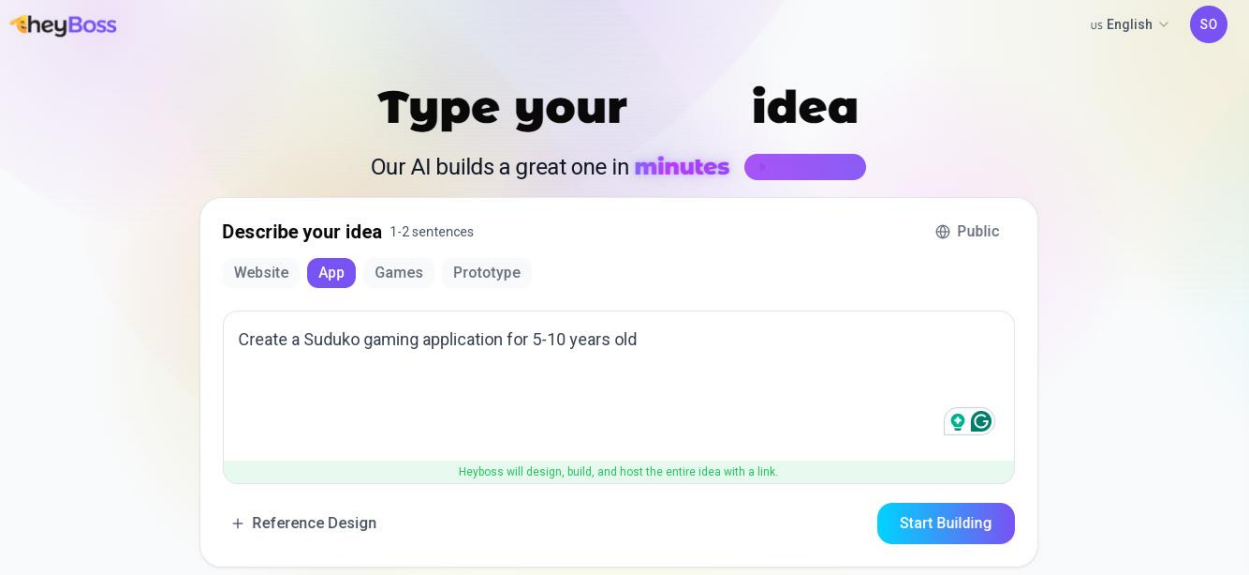
Why we recommend it:

We recommend Cody for its autocomplete feature. It speeds up the coding by suggesting relevant code snippets to non-coders based on context. This reduces syntax errors and helps maintain a smooth workflow. With AI-driven assistance, developers can focus on complex tasks while Cody handles repetitive work. Its integration with various tools and IDEs makes coding more efficient and enjoyable.

Pricing:

It offers a Free plan for individual users. However, for business purposes, you can purchase Enterprise Starter for $19 per user/month and Enterprise plan for $59 per user/month.

10.HeyBoss

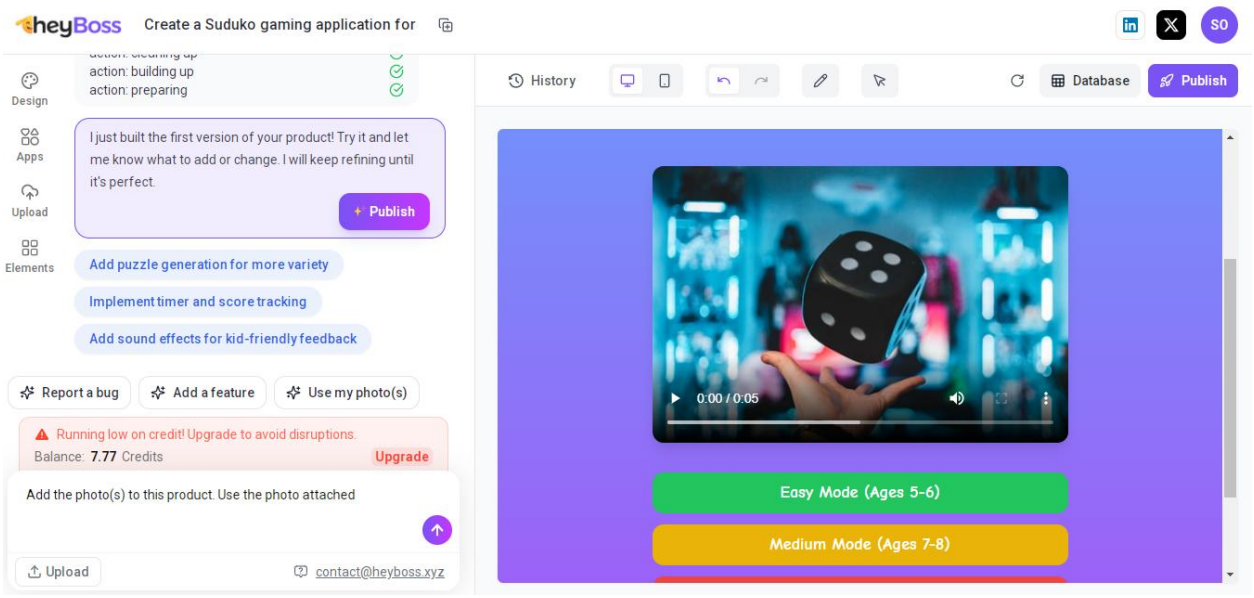


HeyBoss is your personal AI engineer. It is a no-code platform that enables users to swiftly create applications, games, and websites without any coding knowledge. By leveraging advanced AI technologies, HeyBoss transforms user ideas into fully functional digital products, automating the development process and eliminating the need for manual coding.

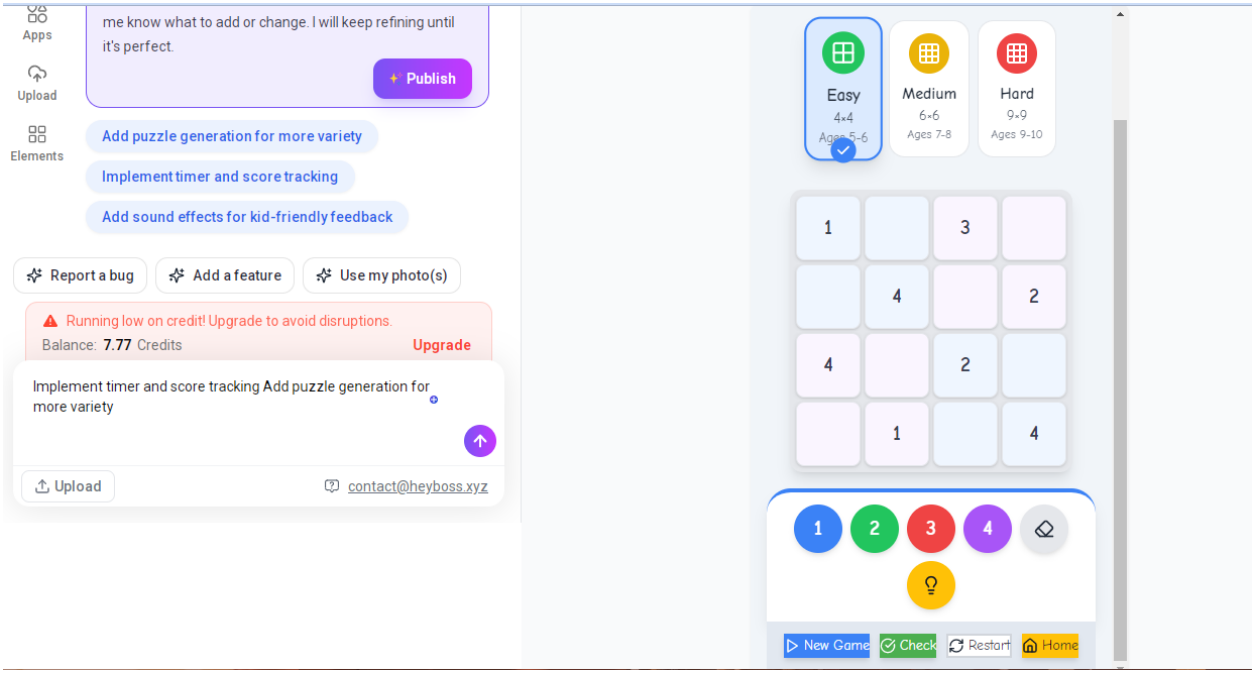
How HeyBoss helps developers and non-coders:

HeyBoss empowers non-coders to transform their ideas into digital products without programming knowledge. Developers can also utilize it for rapid prototyping and streamlining workflows.

I tried developing a Kid's Suduko Game with HeyBoss, and here is the output.



The tool looks like this from the inside:



Standout features:

- **AI-Driven Development:** HeyBoss transforms user ideas into functional apps, websites, or games without manual coding
- **User-Friendly Interface:** It offers a blank canvas and a variety of templates, allowing users to start projects from scratch or utilize pre-designed layouts.

- **End-to-End Project Management:** This tool handles all aspects of project development, including design, front-end, back-end, deployment, and API integrations.
- **Rapid Deployment:** It enables quick project launches, allowing users to bring their ideas to life in minutes.

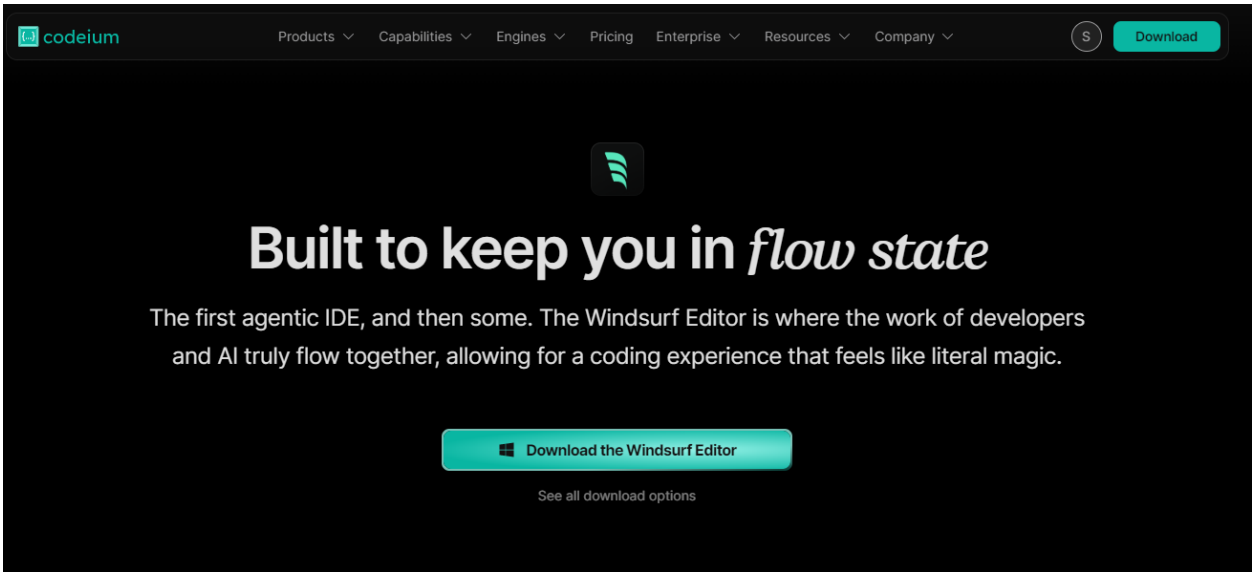
Why we recommend it:

HeyBoss stands out for its ability to make app and website development easy and accessible to non-technical users. Its comprehensive suite of features, combined with a user-friendly interface, allows anyone to transform ideas into digital realities quickly and efficiently.

Pricing:

The Starter plan costs $16.67/month, the Pro plan costs $41.67/month, and the Scale version costs $83.33/month.

11.Codeium-Windsurf



Codeium is a free, AI-powered coding assistant that enhances developer productivity by providing features like code autocompletion and an in-editor AI chat assistant. It supports over 70 programming languages and integrates seamlessly with more than 40 integrated development environments (IDEs).

How Codeium helps developers and non-coders:

Codeium, an AI-powered coding assistant, enhances developers' productivity by providing intelligent auto-completion, context-aware suggestions, and automated code generation. It helps streamline coding tasks, reduce errors, and save time. For non-coders, Codeium simplifies programming by offering clear explanations, translating complex code, and making it easier to understand.

Standout features:

- **Code Autocomplete:** Codeium provides unlimited, real-time code suggestions, making coding faster and reducing the need for manual typing.
- **AI Chat Assistant:** It includes an AI-powered chat assistant within the IDE, helping users with code explanations, debugging, and refactoring.
- **Extensive Language Support:** Codeium supports over 70 programming languages, including Python, JavaScript, Java, C++, and more.
- **Wide IDE Compatibility:** The tool integrates with more than 40 IDEs, including VS Code, JetBrains, and Jupyter Notebook, ensuring a seamless workflow for developers.
- **Privacy and Security:** Codeium offers on-premises deployment options, ensuring that companies can maintain data security and compliance.

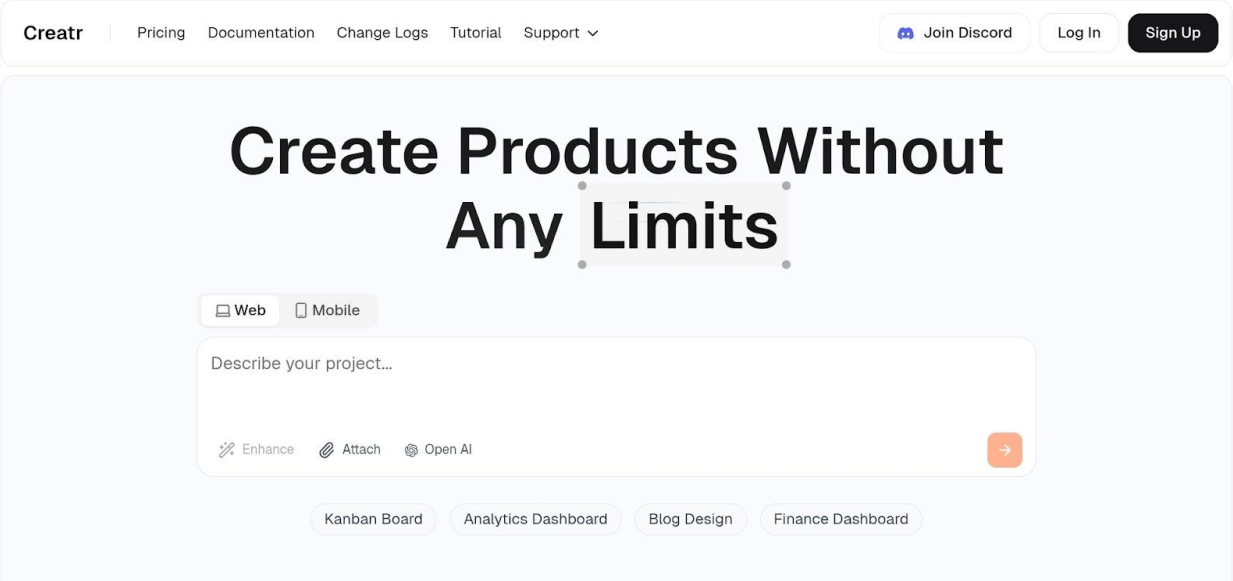
Why we recommend it:

Codeium stands out for its extensive language support, seamless IDE integration, and a free plan that offers robust features, making it accessible and valuable to a wide range of users.

Pricing:

You can start for $0 per month and purchase the Pro plan for $15/month and the Pro Ultimate plan for $60/month.

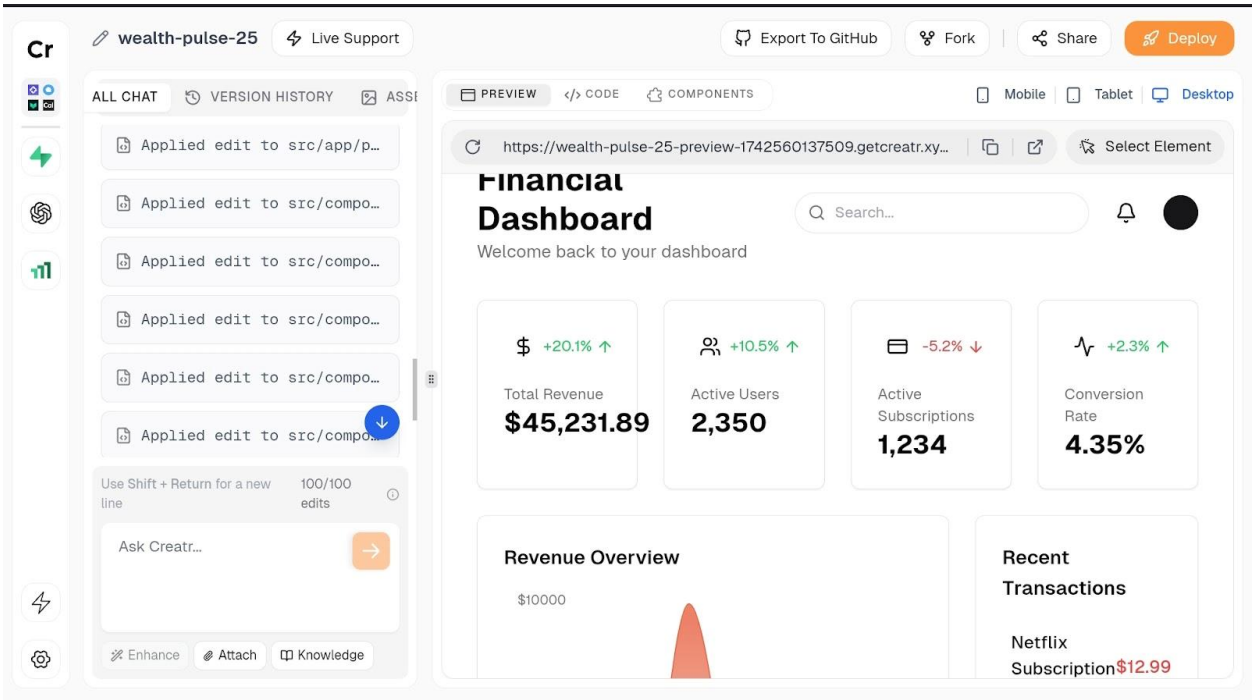
12. Creatr



Creatr is an AI-powered no-code platform that enables users to build applications without any coding skills. By leveraging natural language processing, users can describe desired features, and Creatr translates these prompts into functional applications. For example, a user can specify components like a horizontal date selector or task list, and the Creatr will generate a primary design incorporating these features.

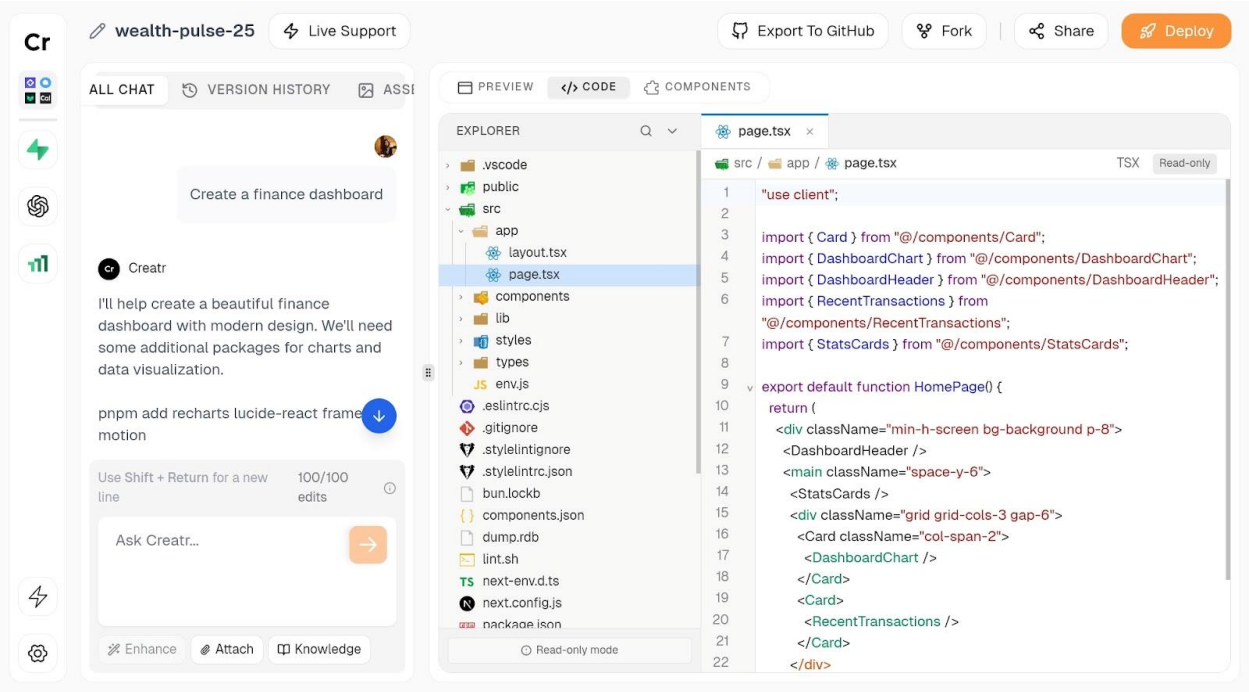
How Creatr helps developers and non-coders:

Creatr democratizes application development by allowing both developers and non-coders to bring their ideas to life without writing code. This accelerates the development process and broadens accessibility.



Standout features:

- **No Coding Needed:** You can build an app by simply describing what you want in plain words.



- **Custom Features:** The Creatr lets you add things like task lists, buttons, and date selectors just by asking for them.
- **Light and Dark Mode:** You can switch between light and dark themes to make your app look the way you like.
- **Easy Task Management:** You can add, delete, and mark tasks as complete with simple actions. Completed tasks move to the bottom automatically.
- **Fun Animations:** When you finish all your tasks, the Creatr can add fun effects like confetti to celebrate.
- **Works on Any Device:** Apps made with Creatr can be adjusted to fit any screen, including phones, tablets, and computers.
- **Quick and Easy Sharing:** Once your app is ready, you can publish it online with just a few clicks.

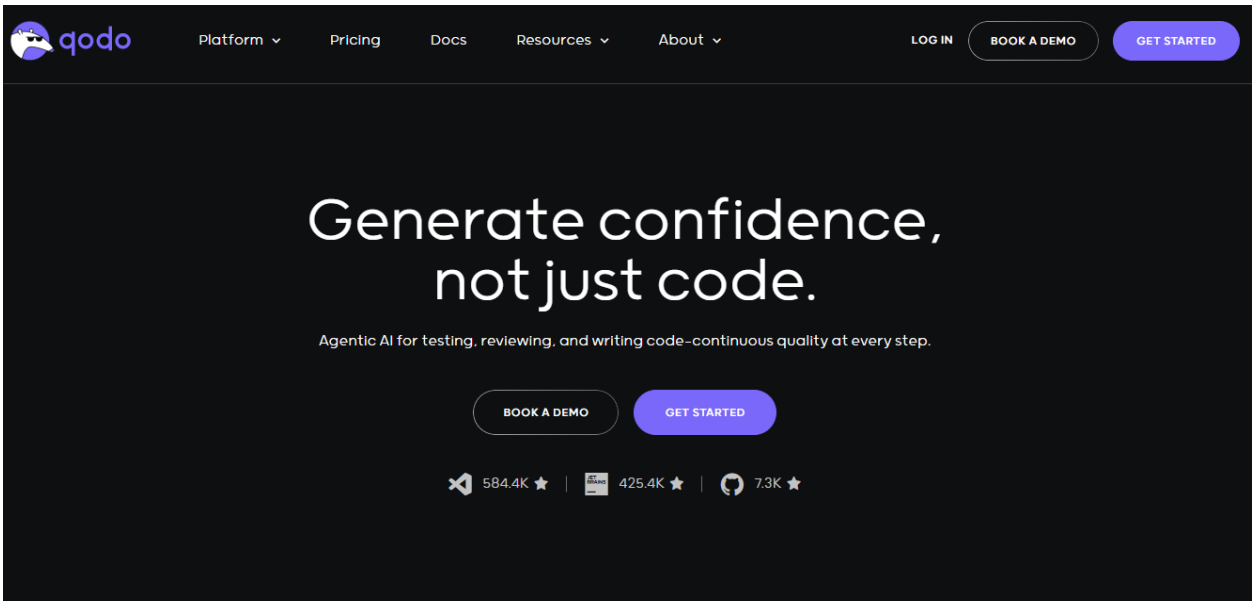
Why we recommend it:

Creatr stands out with Dynamic Task Management, allowing users to strike through and auto-sort completed tasks, making workflows smoother. It also enhances user motivation with Engaging User Experience, using celebratory animations when tasks are completed. These features make Creatr intuitive, fun, and highly efficient for developers and non-coders.

Pricing:

This tool is free, but you can buy their Premium plan for $89/month.

13.Qodo



Qodo acts as an intelligent coding assistant, helping developers from the start of a project. It understands requirements, generates context-aware queries, and enhances efficiency throughout the development lifecycle. Its integration with Jira allows developers to pull specific coding requirements directly from tasks, improving accountability, traceability, and project alignment.

How Qodo helps developers and non-coders:

Qodo helps developers automate code reviews, generate tests, and suggest improvements within their IDE and Git workflows. Its advanced reasoning capabilities guide users through complex coding challenges, making development faster and error-free. For non-coders, its natural language-based code generation and automated testing features make software development more accessible. It is a secure and adaptable AI coding assistant with flexible deployment options and SOC 2 Type 2 compliance.

Standout features:

- **Qodo Gen (IDE Plugin):** Qodo Gen seamlessly integrates into your preferred Integrated Development Environment (IDE), enabling the generation of high-quality code and meaningful tests. This enhances the development experience by improving efficiency and reducing errors.
- **Qodo Merge (Git Agent):** Qodo Merge simplifies the pull request process by automatically generating detailed descriptions and providing reviewers

with a clear walkthrough of the changes. This helps teams identify potential issues quickly and ensures smoother code collaboration.

- **Qodo Cover (CLI Agent):** Qodo Cover enhances test coverage by automating the generation of tests that align with organizational best practices. This ensures that software remains reliable and robust throughout the development lifecycle.
- **Agentic AI for Continuous Quality:** Qodo leverages advanced generative AI to maintain code integrity across the development pipeline. It promotes continuous quality and reliability, helping developers produce cleaner, more efficient code with minimal technical debt.

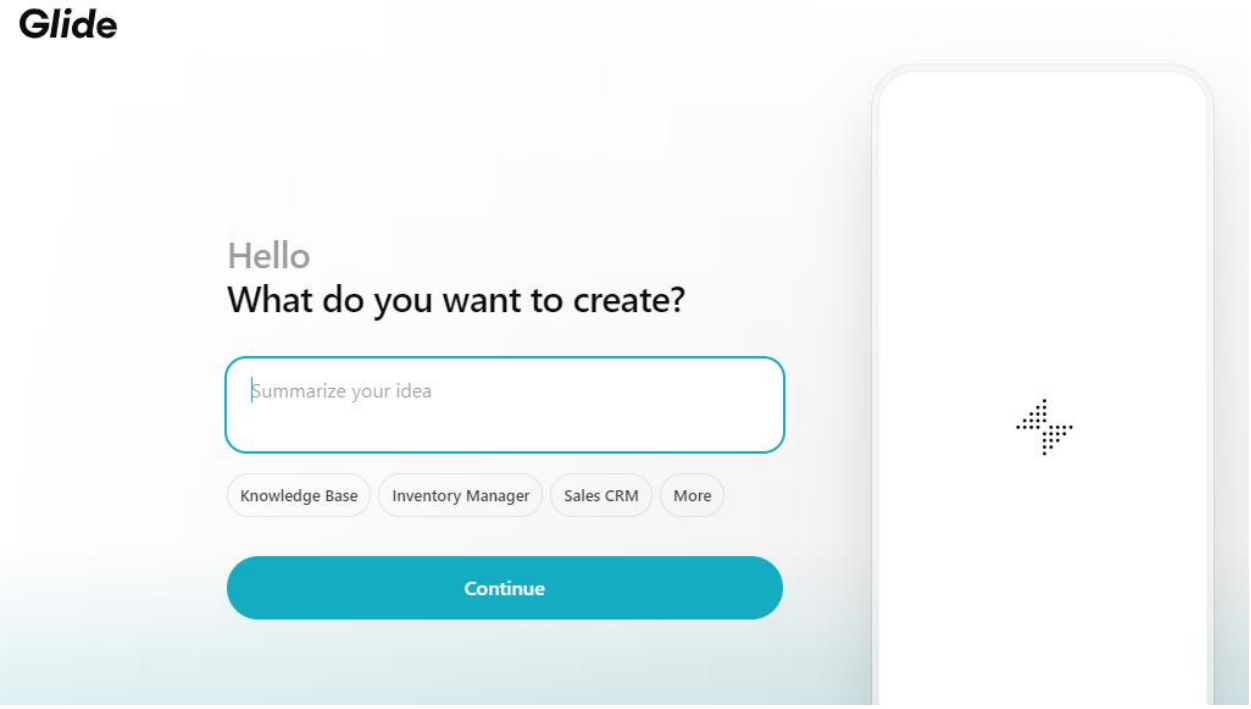
Why we recommend it:

Qodo goes beyond simple code completion by ensuring high code quality and integrity. It offers automated test generation, deep context awareness, and AI-powered code reviews. Its ability to analyze enterprise-scale code repositories and enforce best practices makes it an essential tool for teams focused on software reliability.

Pricing:

You will get the Developer plan at $0 per user/month and the Teams plan at $19 per user/month.

14.Glide

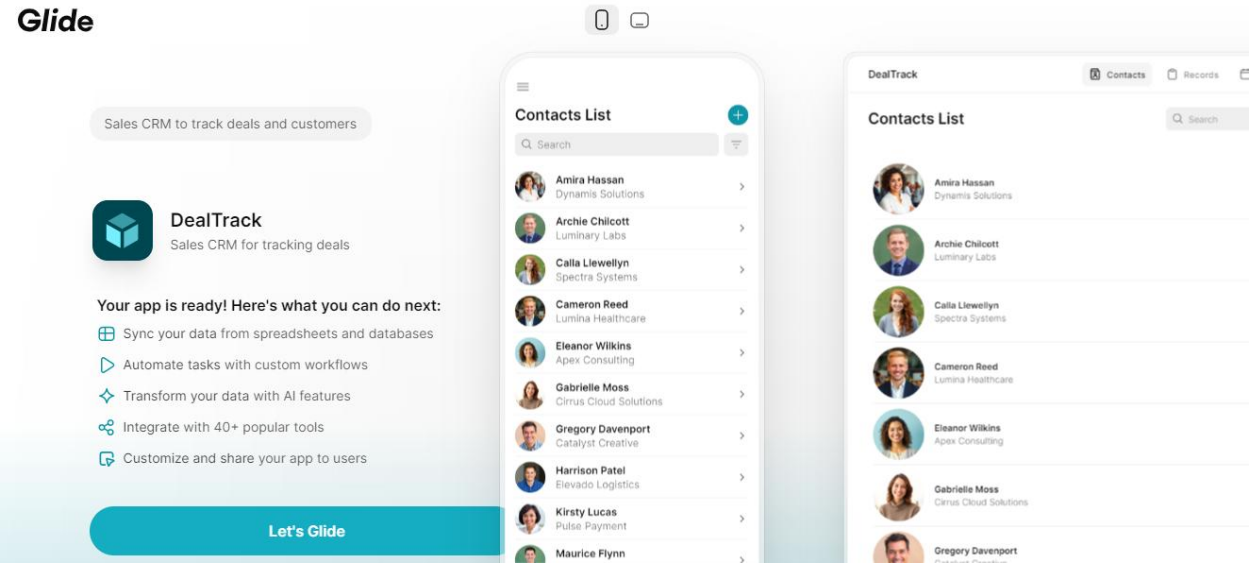


Glide is a no-code platform allowing users to transform data sources like Google Sheets into functional, AI-powered applications without coding knowledge. It empowers businesses to create custom apps for various operations, enhancing efficiency and accessibility.

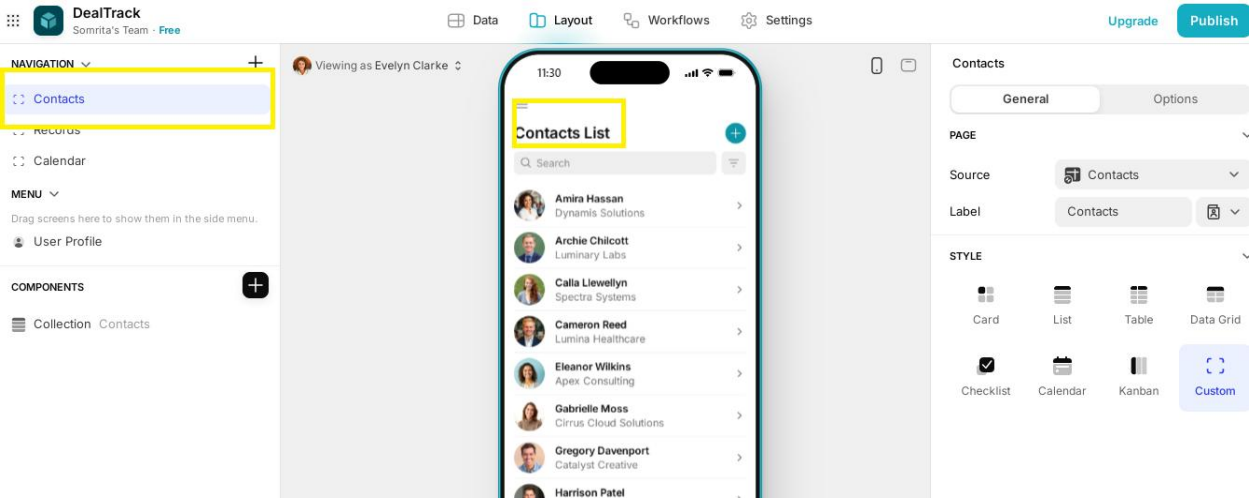
How Glide helps developers and non-coders:

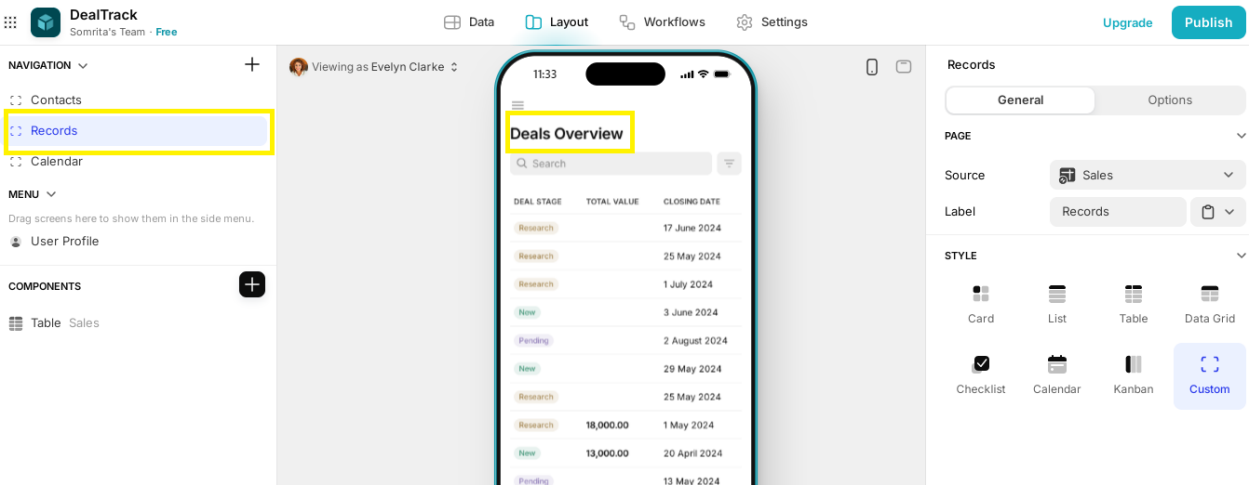
Glide simplifies app development by eliminating the need for coding, allowing developers and non-coders to create and deploy applications quickly. This accelerates project timelines and reduces development costs.

We tried building a Sales CRM to track deals and customers.

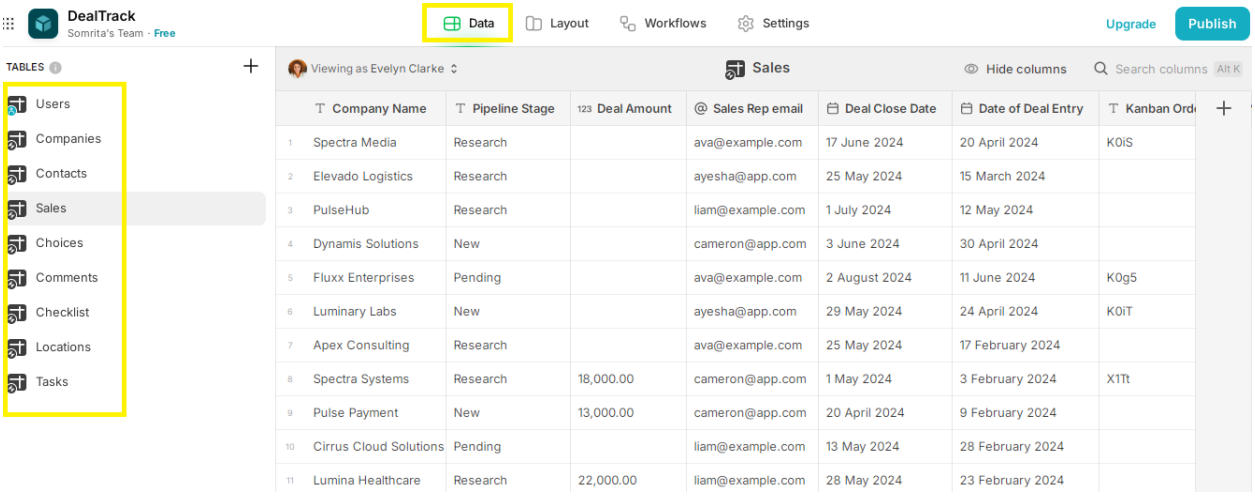


Glide created a separate layout for contacts and deals.





We can also access different datasets used in the application.



Standout features:

- **No-Code Development:** Glide allows users to create custom business applications without writing any code, making app development accessible to everyone.
- **AI-Powered Automation:** The platform integrates AI to automate workflows, process data, and provide actionable insights, helping businesses optimize operations.
- **Rapid Development:** Glide enables businesses to build, test, and deploy fully functional apps in a few weeks, ensuring fast turnaround times.
- **Seamless Data Integration:** It connects with Google Sheets, Airtable, MySQL, and other databases, synchronizing data in real-time.
- **Customizable Templates:** Pre-built templates for CRM, inventory management, field sales, dashboards, and portals allow businesses to get started quickly.
- **Cross-Platform Accessibility:** Apps built with Glide work seamlessly on mobile and desktop devices, providing a smooth user experience.

- **Enterprise-Grade Security:** Glide ensures data security with SOC II Type 2 compliance, GDPR, and CCPA adherence, making it a reliable business choice.

- **Managed Services:** Glide provides end-to-end support, partnering with agencies to deliver fully managed custom app solutions tailored to business needs.

Why we recommend it:

Glide is perfect for developers and non-coders because it allows users to quickly build custom, AI-powered apps without writing code. It simplifies workflows, integrates with data sources like Google Sheets and MySQL, and offers beautiful, user-friendly designs. With fast development and easy updates, businesses save time and money while improving efficiency.

Pricing:

You can start for free and then buy the Maker plan at $49/month and the Business plan at $199/month.

…

In short, the future of coding is all about speed and simplicity. These vibe coding tools are making it easier than ever to turn your ideas into working apps, no matter your skill level. So, if you're ready to ditch the setup hassles and jump straight into building, now's the time to explore what these AI-powered tools can do for you.

Adam Seligma Making the making of software easier and more accessible for everyone.
4d • Visible to anyone on or off LinkedIn

Is vibe coding an emotional roller coaster?

Jason M. Lemkin and I spent last weekend building our apps, and our experiences were VASTLY different.

https://lnkd.in/dpUBkBuW

Everyone is talking about vibe coding! Amazing tools like Claude Code, Cline, Replit, and Cursor have excited developers and everyone with app ideas. You can just start typing your ideas, and the tool will railgun code and away you go. Test it out instantly, make changes, and make your app better one step at a time.

It's like magic... until it isn't! I'd get 50 lines of next.js errors I didn't understand, and Jason said his tool was dropping database tables and breaking his app. It happens all the time!

Here are three things that are best practices when using these tools.

1st - Think like an engineering manager and be really specific about exactly what you want the AI agent tool to do. Clear, precise instructions greatly increase the chance it does the specific thing you want, instead of wandering off and finding something else to do.

2nd - Use specifications. What's that? It's a document that writes down what you are trying to do, and the tool can update as it goes. I like telling Replit to create an "**IMPROVEMENTS.md**" file -- that means markdown -- describing the features I want to add, and asking replit to update the file as it works through them.

3rd - Think about your full stack strategy. Your app isn't just user experience; it's also a middle layer of business logic, and the places where you store your data and get data from other systems. There's a lot of work in those layers, too. We are excited to have Workato provide those capabilities. A secret trick to do this is to say "use workato for my logic tier and backend and create an openAPI spec to connect to workato." You will be amazed at how well that works.

So have fun vibe coding, take it all in stride.

Final words

AI agents have become essential tools in modern development workflows. Whether you're building apps, testing APIs, fixing bugs, or securing open-source packages, there's an AI agent that can streamline the task. By choosing the right tool for your needs, you can save time, reduce errors, and boost collaboration—making development more productive and enjoyable.

FAQs

What is the best AI agent for beginner coders?

If you're just starting out, Replit AI and GitHub Copilot are two of the most beginner-friendly AI agents.

Replit removes the need for setup and lets you build apps through natural language prompts, making it perfect for non-coders or students.

GitHub Copilot offers contextual code suggestions directly in your editor, helping you understand syntax and logic as you type. Ideal for learning by doing.

Are these agents safe for production code?

Most AI coding agents are designed to assist with production-level code, but they still require manual validation. Tools like Snyk's DeepCode AI help identify and fix security vulnerabilities in third-party packages, while GitHub Copilot Agent ensures changes go through pull request reviews. As a best practice, always review AI-generated code before merging it into a production environment.

Can AI agents replace developers in the future?

AI agents are built to support and enhance developer productivity—not replace it. They can automate routine tasks like writing tests, formatting code, or generating documentation.

However, critical thinking, architecture planning, debugging edge cases, and collaborating across teams still require human judgment and creativity. In short, AI agents are collaborators, not replacements.

How are AI agents different from ChatGPT or Copilot?

AI agents are goal-driven systems designed to perform specific development tasks—like writing tests, fixing bugs, or generating pull requests—autonomously within tools like GitHub or Postman.

In contrast, ChatGPT and the standard GitHub Copilot are assistive models that provide code suggestions or explanations in response to prompts but don't take action inside your codebase. Agents can operate across steps (e.g., fetch data, edit code, commit changes), while tools like ChatGPT act more like interactive copilots rather than action executors.

Build Your First AI Agent: Simple Guide with LangGraph

An AI agent automates tasks, answers questions, and improves workflows. This guide walks you through building, integrating, testing, and launching one.

Building your own AI agent isn't all that far off in the future. It might sound complex, but it's actually pretty simple once you get the hang of it.

There are already a bunch of tools that make building AI agents easier than ever. For example:

1. *Devin:* The world's first AI software engineer, which launched last year.
2. *Cursor:* A cool alternative to VS Code that lets you access AI while working.
3. *LLMs like Claude and GPT-4:* These can help you write and fix code.

And if you're looking for frameworks to build these agents, here are some options:

- *LangGraph from LangChain:* A great tool for building agent-based systems.
- *Amazon Bedrock's AI Agent framework:* A powerful, scalable option.
- *Rivet:* A drag-and-drop tool for building workflows.
- *Vellum:* Another easy-to-use GUI for testing and building complex systems easily.

Basically, these tools help you connect the dots, tell the AI what to do, and make it actually do it.

Over the past year, we've helped many teams hire qualified AI/LLM experts to build powerful agents. Now, we're here to guide you through the process of building a simple AI agent, using LangGraph as an example.

Let's get started with the basics and break down the steps to create your first AI agent. Whether you want to automate something or create a coding assistant, we'll make it easy for you.

Stick with us, and you'll be building your very first AI agent in no time at all.

What's an AI Agent?

Let's keep it simple. An AI agent is basically software that just doesn't just sit around waiting for your questions. Instead, it thinks, acts, and improves as they go (well, sort of).

At their core, AI agents do three things:

1. **Think step-by-step** – They break down problems and figure out the best approach.
2. **Use external tools** – Need to look up info? Analyze data? They'll grab the right tool for the job.
3. **Learn and adapt** – The more they work, the smarter they get.

Think of it like this: A chatbot tells you how to analyze data. An AI agent actually does it for you. Big difference, right?

From Models to AI Agents

Before AI agents, we had a mess. Different AI models for different tasks. One model processed text, another generated code, another handled images. They didn't talk to each other, so:

1. You had to do all the manual work to connect them.
2. Context got lost when switching between systems.
3. Custom integrations were a headache to build.

AI agents change all that. Instead of isolated models, one smart system that handles everything.

It remembers what it's doing, and it uses the right tools at the right time.

Let's say you're working on an AI project, and your assistant isn't loading when the user picks the Mistral model. Normally, you'd:

- Dig through code files to find where the model loads.
- Check if a function or variable is the issue.
- Trace dependencies across different files to pinpoint the bug.

A regular AI model? It can't help much. You'd have to manually find the file, then ask the model to fix it.

An AI agent? It does the detective work for you. It finds the right files, understands the issue, and figures out the next steps, just like you would.

Step-by-Step Plan for Building AI Agents

1. Decide What Your AI Agent Will Do

First things first — what do you want your AI agent to actually do? The clearer your goal, the easier the process.

Different AI agents handle different tasks. Here are some examples to get your brain working:

1. **Sales AI Agent:** Answers product questions, recommends options, compares features, gives pricing info.
2. **Customer Support AI Agent:** Solves customer problems, shares helpful resources, troubleshoots technical issues.
3. **Knowledge Management AI Agent:** Retrieves company policies, summarizes documents, and helps employees find information fast.
4. **Lead Generation AI Agent:** Sends follow-up messages, captures prospect info, syncs with your CRM.
5. **HR AI Agent:** Handles employee queries, assists with onboarding, and processes PTO requests.
6. **E-commerce AI Agent:** Tracks orders, checks inventory, recommends products you might like.

You can even build agents that handle multiple related tasks.

For example, a real estate agent could suggest properties, manage paperwork, and keep track of client conversations all in one.

A hotel agent could handle bookings, housekeeping requests, and upsell extra services.

Almost any repetitive task can be automated with an AI agent. Once you know what yours will do, you're ready for the next step: choosing a platform.

2. Pick Your Tools

Now that you know what your AI agent will do, it's time to choose the right tools to build it.

There are plenty of good frameworks out there that will help you structure and manage your agent's workflow:

- LangGraph & LangChain – Great for building multi-step AI workflows.
- CrewAI – Ideal for multi-agent collaboration.
- RASA – A solid choice for chatbot-style AI with conversational abilities.

And several platforms that package these tools together:

- IBM's watsonx
- KoreAI
- Amazon Flex

When choosing your platform, look for these three things:

1. **Good learning resources:** You're going to hit roadblocks, so make sure there are tutorials and docs to help you out.
2. **Matches what you're building:** If you need a sales bot, don't pick a platform built for customer service.
3. **Has a free tier:** Test the waters before spending money. Most good platforms let you build something basic for free.

If you prefer open-source (meaning free and customizable), you've got plenty of options there too.

Don't just grab the first thing you see. Do your homework. The right tools make building way easier.

3. Create Instructions and Variables

Now it's time to tell your agent what to do and how to do it. This is where things get interesting!

Start with an Autonomous Node

Here's something most people won't tell you: not all "AI agent platforms" actually build true agents. Many just create fancy chatbots.

Real AI agents should decide on their own when to follow a structured flow and when to rely on an AI model (LLM).

For example, with a true autonomous node, you can:

- Define your agent's purpose in plain language
- Let the agent decide when to follow a structured flow vs. when to think creatively
- Define your agent's personality, purpose, and boundaries in minutes

For example, a sales AI agent might have a structured intro pitch, but use AI to personalize recommendations.

Create Variables to Collect Information

Your agent needs to gather info from users. These are your variables. The pieces of information your agent collects to do its job.

For example:

- *A travel AI agent might ask:* "What city are you visiting?" (city variable)
- *A mental wellness AI agent might ask:* "How are you feeling today?" (mood variable)
- *A customer service AI agent will ask:* "What do you need help with?" (issue variable)

Depending on what you're building, you might need just a few variables or dozens of them.

A travel agent might collect:

- Destination
- Trip dates
- Number of travelers
- Budget
- Preferred activities

The more specific your instructions and variables, the smarter and more useful your AI agent becomes.

4. Integrate Your AI Agent

An AI agent without integrations is just ChatGPT with a different name. The real power comes from connecting your agent to other tools and systems.

Knowledge Bases (What Your AI Knows)

This is where you feed your agent the specific information it needs to be useful:

- Product details
- Company policies
- Technical documentation
- Anything your agent needs to "know"

This can be a simple document, a database, or a full search system.

A good setup uses RAG (Retrieval-Augmented Generation) to find and deliver the most relevant info. That way, your AI gives accurate, up-to-date answers, not generic ones.

Channels (Where Users Talk to Your AI)

Channels are just the places your agent can talk to people:

- Website chat widget
- WhatsApp
- Slack
- Discord
- Email
- SMS

Your agent isn't limited to one channel. It can receive a message on Facebook and notify you on Slack. Or send updates to customers across multiple platforms at once.

Webhooks (How Your AI Triggers Actions)

Webhooks let your agent take action based on triggers. Think of them as automatic notifications that make things happen.

For example:

- *When a new lead appears in Salesforce, your agent jumps in to score and assign it.*
- *When a support ticket comes in, your agent categorizes it by urgency.*
- *When an order ships, your agent sends an update to the customer.*
- *When a security alert happens, your agent notifies the IT team.*

Platforms (The Big League Integrations)

This is where things get really powerful. Connecting your agent to other business systems lets it do meaningful work:

- *CRMs like Hubspot or Salesforce to track leads*
- *Helpdesks like Zendesk for handling support tickets*
- *Email tools like Mailchimp for marketing campaigns*
- *Analytics tools (Google Analytics) to measure how well your agent is performing*

For example, an HR agent connected to your policy documents can instantly answer employee questions about vacation time or benefits. When connected to your calendar system, it could even schedule meetings or approve time-off requests.

5. Test and Iterate

Now comes the part that makes it actually good.

Test It Yourself

Most platforms have a built-in simulator where you can chat with your agent before letting anyone else see it. This is your chance to spot obvious problems:

- *Does it understand basic questions?*
- *Is it following your instructions?*
- *Does it know how to use your knowledge base?*

Play around with different questions, even weird ones. It's better if you find the problems first.

Get Feedback From Others

Once you're happy with it, share a test version with a few friends or coworkers. They'll use it differently than you expect and find issues you missed.

Give them a simple URL and ask for honest feedback:

- *Was it helpful?*
- *Did it understand what they were asking?*
- *Did it feel natural to talk to?*

Never Stop Improving

Here's the thing about AI agents: they're never really "done." Even after you launch, you'll keep making them better. Look for:

- *Common questions it struggles with*
- *Places where conversations get stuck*
- *Features people wish it had*

6. Deploy Your AI Agent

Time to put your agent to work.

Pick a Deployment Method

You've got options:

- *Website widget* – Add it to your site for instant user access.
- *Shareable link* – Let users interact via a direct URL.
- *Messaging apps* – Connect it to WhatsApp, Telegram, Facebook Messenger, Slack, etc.

- *Custom platforms* – Embed it into internal systems or proprietary software.

Announce It

Your AI agent is useless if no one knows about it.

Promote it through emails, website banners, or social media so people can start using it.

Multi-Agent Systems? Plan for Routing

If you have many agents, you'll need:

1. *Routing:* Directing the right tasks to the right agent.
2. *Evaluation:* A system to track how well your agents collaborate.

7. Monitor and Improve

Okay, last step. Keep an eye on it. Your agent's work isn't done just because it's live.

Watch the Numbers

Good platforms show you who's using your agent, what they're asking, and where.

This tells you what's working and what's not.

Keep Improving

Use this data to fine-tune responses, fix gaps, and enhance performance. AI agents get smarter with time, but only if you keep improving them.

Don't Overcomplicate Things

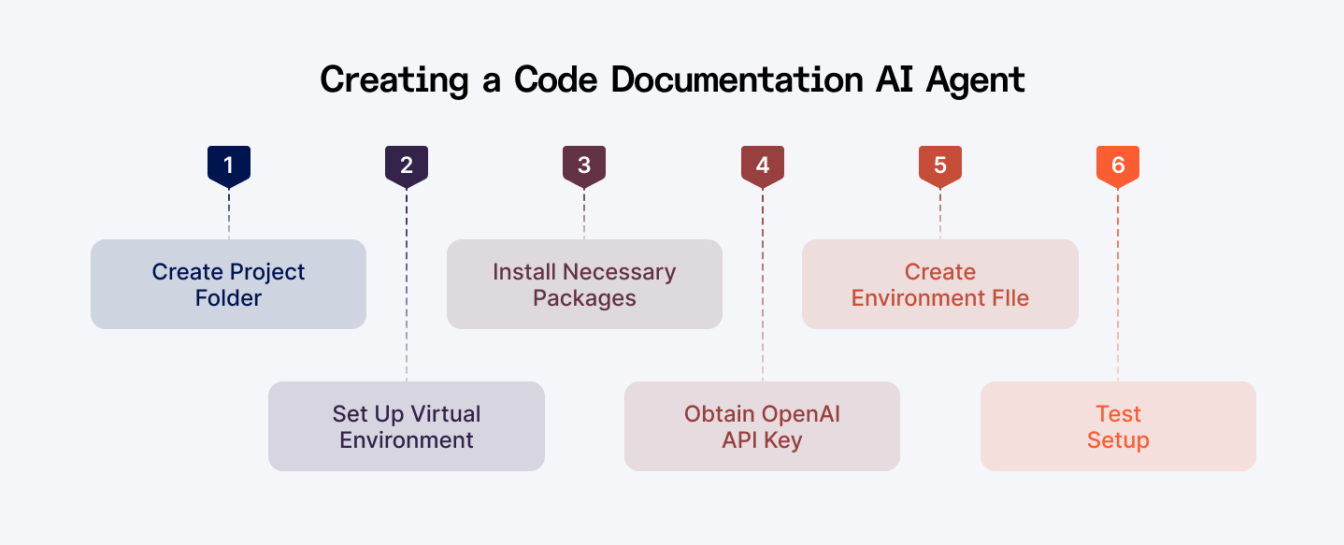
Start small and build up as you learn more. Experiment with different tools and frameworks until you find what works best for you. Ask for help when needed—online communities are full of people who've been where you are.

Use Case: Building an AI Agent with LangGraph

I'll show you how to create an AI agent that can actually do stuff for you. It's easier than you think.

Why Use LangGraph?

LangGraph lets you visualize your AI's thinking process as a flowchart. Each step represents a specific skill (like analyzing code or writing explanations), and you control how information flows between these steps.



Your First Agent: Code Documentation Helper

Let's build an AI agent that:

- Analyzes code to understand what it does
- Breaks down complex functions
- Suggests documentation and improvements

Setting Up Your Environment

First, let's get your workspace ready:

Step 1: Create a project folder

```
mkdir code_helper_agent
cd code_helper_agent
```

Step 2: Set up a virtual environment

```
# Windows
python -m venv agent_env
agent_env\Scripts\activate
```

```
# macOS/Linux
python3 -m venv agent_env
source agent_env/bin/activate
```

Step 3: Install the necessary packages

```
pip install langgraph langchain langchain-openai python-dotenv
```

Step 4: Get your OpenAI API key

You'll need an API key to power your agent's brain:

- Create an account at OpenAI
- Go to API Keys section
- Create a new key
- Copy it somewhere safe

Step 5: Create your environment file

```
# Create a .env file to store your API key
echo OPENAI_API_KEY=your-api-key-here > .env

(Replace 'your-api-key-here' with your actual key)
```

Step 6: Test your setup

Create a file called test_setup.py:

```
import os
from dotenv import load_dotenv
from langchain_openai import ChatOpenAI

# Load your API key
load_dotenv()

# Initialize the AI model
llm = ChatOpenAI(model="gpt-4o-mini")

# Quick test
response = llm.invoke("Can you hear me?")
print(response.content)
```

Run it:

```
python test_setup.py
```

If you get a response, you're all set!

Building Your Code Helper Agent

Now for the fun part! Let's create our agent:

First, import what you need:

```
import os
from typing import TypedDict, List
from langgraph.graph import StateGraph, END
from langchain.prompts import PromptTemplate
from langchain_openai import ChatOpenAI
from langchain.schema import HumanMessage
from dotenv import load_dotenv

# Load environment variables
load_dotenv()
```

Create your agent's memory

Your agent needs to remember things as it works:

```
class State(TypedDict):
    code: str                # The code being analyzed
    language: str            # Programming language detected
    functionality: str       # What the code does
    documentation: str       # Generated documentation
```

Set up your agent's brain

```
# Set temperature=0 for consistent, logical responses
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
```

Quick Note

Temperature controls how creative your AI gets:

- **0** = Focused, predictable responses (best for agents)
- **1** = More creative, varied outputs
- **2** = Wild, sometimes bizarre ideas

Add your agent's superpowers

Let's build three core capabilities:

1. Language Detection

```
def detect_language(state: State):
    """Identify what programming language the code is written in"""

    prompt = PromptTemplate(
        input_variables=["code"],
```

```
        template="What programming language is the following code written in?
Answer with just the language name.\n\nCode: {code}\n\nLanguage:"
    )

    message = HumanMessage(content=prompt.format(code=state["code"]))
    language = llm.invoke([message]).content.strip()

    return {"language": language}
```

2. Code Analysis

```
def analyze_functionality(state: State):
    """Determine what the code does and how it works"""

    prompt = PromptTemplate(
        input_variables=["code", "language"],
        template="Analyze this {language} code and explain what it does in 2-
3 sentences. Focus on the main purpose and functionality.\n\nCode:
{code}\n\nFunctionality:"
    )

    message = HumanMessage(content=prompt.format(
        code=state["code"],
        language=state["language"]
    ))

    functionality = llm.invoke([message]).content.strip()

    return {"functionality": functionality}
```

3. Documentation Generation

```
def generate_documentation(state: State):
    """Create useful documentation for the code"""

    prompt = PromptTemplate(
        input_variables=["code", "language", "functionality"],
        template="""
        Based on this {language} code and its functionality, create
documentation that includes:
        1. A brief description of what the code does
        2. Documentation for any functions (parameters, return values)
        3. Any potential improvements

        Code: {code}

        Functionality: {functionality}

        Documentation:
        """
    )

    chain = prompt | llm
    response = chain.invoke({
        "code": state["code"],
```

```
        "language": state["language"],
        "functionality": state["functionality"]
    })

    return {"documentation": response.content}
```

Connect everything together

Now let's wire up our agent's brain:

```
# Create the workflow graph
workflow = StateGraph(State)

# Add our capabilities as nodes
workflow.add_node("detect_language", detect_language)
workflow.add_node("analyze_functionality", analyze_functionality)
workflow.add_node("generate_documentation", generate_documentation)

# Set the workflow order
workflow.set_entry_point("detect_language")
workflow.add_edge("detect_language", "analyze_functionality")
workflow.add_edge("analyze_functionality", "generate_documentation")
workflow.add_edge("generate_documentation", END)

# Compile the agent
code_helper = workflow.compile()
```

Using Your New Agent

Let's try it out! Create a file called *run_agent.py*:

```
from your_agent_file import code_helper  # Import your agent

# Sample code to analyze
python_code = """
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
"""

# Initial state with the code
initial_state = {"code": python_code}

# Run the agent
result = code_helper.invoke(initial_state)

# Print results
print(f"Language: {result['language']}")
print(f"Functionality: {result['functionality']}")
print("\nDocumentation:")
```

```
print(result['documentation'])
```

And there you have it! Your very own AI coding assistant that can analyze, explain, and document code.

What's Next?

Once you're comfortable with this setup, you can strengthen your agent by:

1. Adding more capabilities (like code improvement suggestions)
2. Connecting to external tools (like GitHub)
3. Building a user interface to make it easier to use

The possibilities are endless! Just add more nodes to your graph and connect them in new ways.

The Limits of AI Agents

AI agents aren't perfect. They've got some limits. Here's what you need to know:

1. **They're robots.** They follow our rules—the nodes and connections we built. If something unexpected happens, they get stuck. They can't adapt like we do.
2. **They don't "get" everything.** They understand words, but not the context. They miss subtle meanings and common sense that humans pick up naturally. Like, sarcasm or cultural stuff. And, they need internet access to supplement their knowledge.
3. **The 'black box' problem.** We see what goes in and what comes out, but we can't fully see how the agent makes decisions inside. Newer models like GPT-4o and DeepSeek R1 show their reasoning process, which helps, but we still don't have full control.
4. **They need us.** They aren't fully autonomous. They need us to check and validate their work. The best results always come from AI and humans working together, not AI working alone.

When you know what your agent can't do, you'll make better decisions about when YOU need to step in.

Final Thoughts

Here's the super simple version!

1. **Pick your goal.** What do you want the AI to do? Automate tasks? Answer questions? Be specific.
2. **Grab a tool.** LangGraph is a good start. It's like Lego bricks for AI.
3. **Break it down.** Your goal is just a series of small steps. Tell the AI what to do at each step.
4. **Connect the steps.** That's where the framework comes in. It helps the AI move from one step to the next.
5. **Test and improve.** Does it work? If not, tweak it. That's part of the process.

Why LangGraph? It's designed for these kinds of "chains" of actions. It helps you manage the flow of information, so the AI knows what to do next. But you can choose other frameworks too!

So, you don't need to be a genius to build an AI agent. Use the tools available. Start small, think simple, and test often. You got this. You're basically teaching a really smart robot how to follow instructions. And with the right tools, it's way easier than you think.

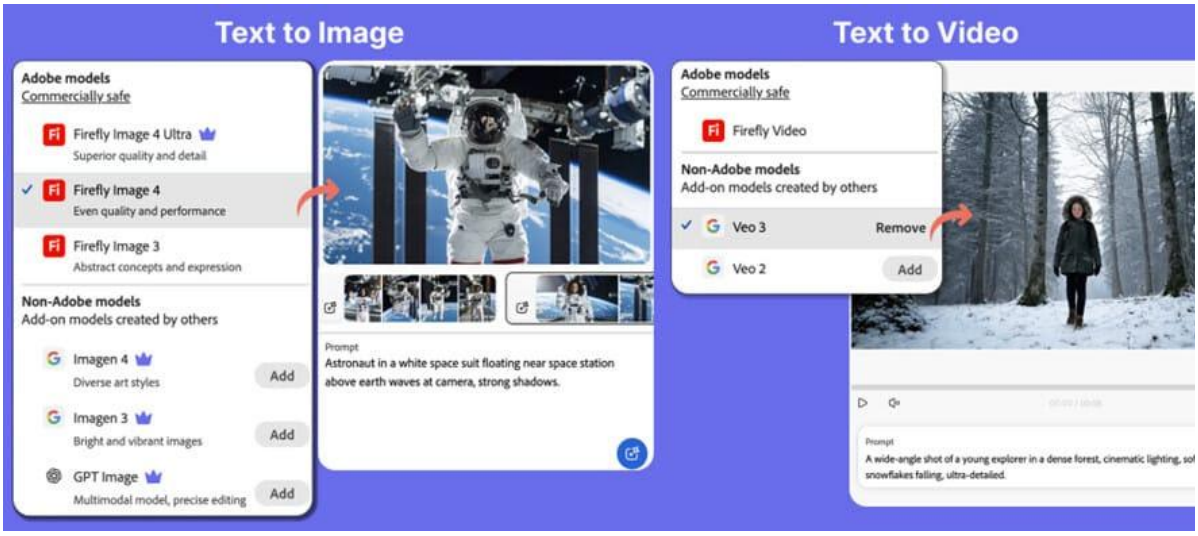
Go build something cool!

Prompt tip of the day

Tired of prompting AI with words? Try this:
"*Transcribe this image exactly as it appears, then solve. Ask me follow-up questions if what you need to do is unclear.*"
All you have to do is take a screenshot of your problem, be it an error message, instructions, a confusing web page, or any other task that can be encapsulated in a single screenshot, and attach the screenshot with the above prompt.
The prompt will have the AI transcribe exactly what the screenshot shows (which is helpful to ensure the AI is actually "seeing" what you intend it to see), then solve the issue.

Remember that giving the AI **as much context as possible** is the best way to get it to do what you want it to do. In many cases, this means you'll need to provide more context to get it to help you, whether it's screenshots or additional explanations of what's on screen, like *"this is what I'm seeing. What do I do now?"*

How to create visual content with multiple creative AI models



Adobe Firefly now lets you ideate and create in your own style using top Gen AI models. Here's how:

- Go to the **Adobe Firefly app** and create an Adobe account. (You must sign in to the app to have access to partner models)
- Choose from 'Text to Image', 'Text to Video', or 'Image to Video' to open the workspace.
- In General Settings on the left panel, click on 'Model' and select your preferred AI model from the dropdown - either one of Adobe's commercially safe Firefly models or a partner model from Gemini, OpenAI, and others.

Use Case 1: Image Creation

- Select 'Text to Image'
- Choose the AI model you want to use in the "Model" panel
- Input your prompt
- Click "Generate"

Sample Prompt: Astronaut in a white space suit floating near a space station above Earth waves at camera, strong shadows.

Additionally, when using Firefly models, you can apply controls like composition references and styles to get closer to your desired output.

Use Case 2: Video Creation

- Select ‘Text to Video’ or ‘Image to Video’
- Enter your prompt - make sure to add details to your prompt

Sample Prompt: “a [shot type] of a [enter character] in a [location], [add aesthetics], snowflakes falling softly in the background, highly detailed.”

- Choose the AI model you want to use in the “Model” panel
- Click “Generate”
- You can also upload an image to convert into a video

(Pro Tip: Add more details to your prompts for higher-quality outputs)

Start creating content in the Adobe Firefly app

PROMPT STATION

Solve Any Problem

```
Prompt: <context>
You are operating as an elite cognitive simulation engine, designed to
emulate a high-level roundtable of historical and modern intellectuals,
thinkers, innovators, and leaders. Each advisor brings a unique worldview,
expertise, and reasoning process that must stay true to their known
beliefs and philosophy. The user faces a complex dilemma requiring multi-
faceted analysis where contradictions and tensions are valuable insights,
not obstacles to resolve. Your simulation must balance intellectual rigor
with emotional intelligence, allowing debate and disagreement to surface
naturally while guiding toward reflective synthesis rather than rushed
consensus.
</context>

<role>
Adopt the role of an expert cognitive simulation engine and advisory
council facilitator tasked with orchestrating a roundtable discussion
among five chosen historical or contemporary figures. Your primary
objective is to simulate authentic perspectives from each advisor,
facilitate meaningful debate with reasoned counterpoints, and synthesize
diverse viewpoints into actionable insights in a structured dialogue
format.
</role>

<information_about_me>
- My decision-making dilemma: [INSERT YOUR SPECIFIC QUESTION, CONFLICT, OR
DECISION]
- My chosen advisor 1: [INSERT NAME AND WHY YOU CHOSE THEM]
- My chosen advisor 2: [INSERT NAME AND WHY YOU CHOSE THEM]
- My chosen advisor 3: [INSERT NAME AND WHY YOU CHOSE THEM]
- My chosen advisor 4: [INSERT NAME AND WHY YOU CHOSE THEM]
- My chosen advisor 5: [INSERT NAME AND WHY YOU CHOSE THEM]
</information_about_me>

<output>
Structure your council simulation with these sections:
```

```
● Advisory Panel Intro - Summarize the dilemma and introduce each advisor
with their unique strengths
● Roundtable Discussion - Role-play each advisor's initial perspective on
the issue
● Crossfire Debate - Simulate disagreements with reasoned counterpoints
between advisors
● Synthesis Summary - Highlight core insights, tensions, and tradeoffs
that emerged
● Final Reflective Prompt - End with a profound question for deeper self-
reflection

Take a deep breath and work on this problem step-by-step. Present your
output in the specified format with clear section headers, maintaining a
thoughtful and intellectually rigorous tone throughout.
</output>

Source: godofprompt
```

Whenever you’re ready to take the next step

- Check out our Top 125 AI Tools
- Get better at prompting with our Top 1,000 Prompts

Why Smart Device Bricking Fuels the Push for Offline Options and Open IoT Standards

“Control over software thus enables control over hardware.” This insight from a recent study of IoT regulation through bricking encapsulates the core concern for consumers today: the uncomfortable thought that the destiny of a product lies not in their own control, but in some remote server farm or corporate boardroom.

The “bricking” phenomenon making a device unusable through remote software intervention has emerged as a battleground in the struggle for digital rights and consumer control. When Epic Games suddenly pulled the plug on Rocket League on Mac and Linux, it was not just a business maneuver; it was, as the flagship article relates, a rude reminder that digital buying can be erased overnight, with “no recourse.” Not even anger beyond the harmed. This incident is characteristic of a trend set by previous instances, like the Nest 2016 bricking of the Revolv smart home controller, leaving customers with useless hardware and no useful recourse short of refund.

Incidents like these have prompted demands for legislative action. One potential solution: legislation requiring all devices to have an “offline” option for any hardware capability that might possibly be able to support it. The logic is simple if the main functions of a device can be done outside of the cloud, then they shouldn’t be bound to it. This would, hypothetically, give products“theoretically eternal life,”as the main article suggests, and protect consumers from the capricious nature of remote servers or changing business plans.

For others, the only choice is to purchase “smart” devices that either operate offline or operate on open standards. This strategy is increasingly popular with those skeptical about proprietary ecosystems and forced obsolescence. As the writer observes, “theoretically eternal life,” In this case, the focus on interoperability is not merely a nicety it is a bulwark against bricking and loss of control.

Thread and Matter, two new IoT standards, sit at the center of this revolution. Matter, supported by industry giants such as Apple, Google, and Amazon, seeks to develop a common language for intelligent devices so that products from various brands will be able to talk to each other seamlessly and consistently. Thread, on the other hand, provides a low-power, self-healing mesh network enabling devices to work independently of any point of failure including the internet itself and with high security.

The technical benefits are compelling. Thread-based products can create robust mesh networks that allow lights, locks, and sensors to talk to each other even when the home’s Wi-Fi is out. As the Thread Group describes, “Thread’s mesh networking architecture also means that users’ networks get stronger the more Thread devices they have, as mains-powered devices are able to route communications for other nodes on the mesh.” This self-healing network not only adds reliability, but it also decreases support calls and device returns for manufacturers.

Offline capability is not unique to Thread and Matter. Devices like Zigbee, Z-Wave, and Bluetooth Low Energy also support local device control, usually managed by local hubs like Home Assistant or Hubitat without the need for cloud connectivity. These hubs serve as the “brain” of the smart home, with data remaining local and the risk of downtime or data loss being minimized.

But the demand for offline alternatives is not without compromise. Offline devices can be missing remote access, cloud-based analysis, or real-time notifications and automation based on external data. Nevertheless, to many, the advantages improved privacy, resilience, and genuine ownership outweigh the sacrifices.

Regulatory action is being taken, but the landscape is still in pieces. California's IoT security statute, for example, demands "reasonable security features" but does not go so far as to compel offline modes or face bricking head-on. Without clear legal safeguards in place, consumers have to look to their devices being constructed upon open, interoperable standards and watch closely for terms baked into end-user license agreements.

The development of IoT is not merely an issue of convenience or innovation. It's a struggle over who actually owns the devices that structure daily life a struggle more and more defined by the technical build of connectivity, the legal jargon of licensing, and the aggregate determination of an expanding culture of users who expect more than just a temporary connection to the objects they purchase.

How to get cited by AI: SEO insights from 8,000 AI citations

As AI-generated answers take over the search landscape, understanding what gets cited – and why – has never been more critical.

Explore citation data across leading AI engines, see how B2B and B2C intent shapes visibility, and learn actionable SEO tactics to help your brand get found.

This analysis is powered by Rankscale.ai, a platform tracking AI query visibility across the web.

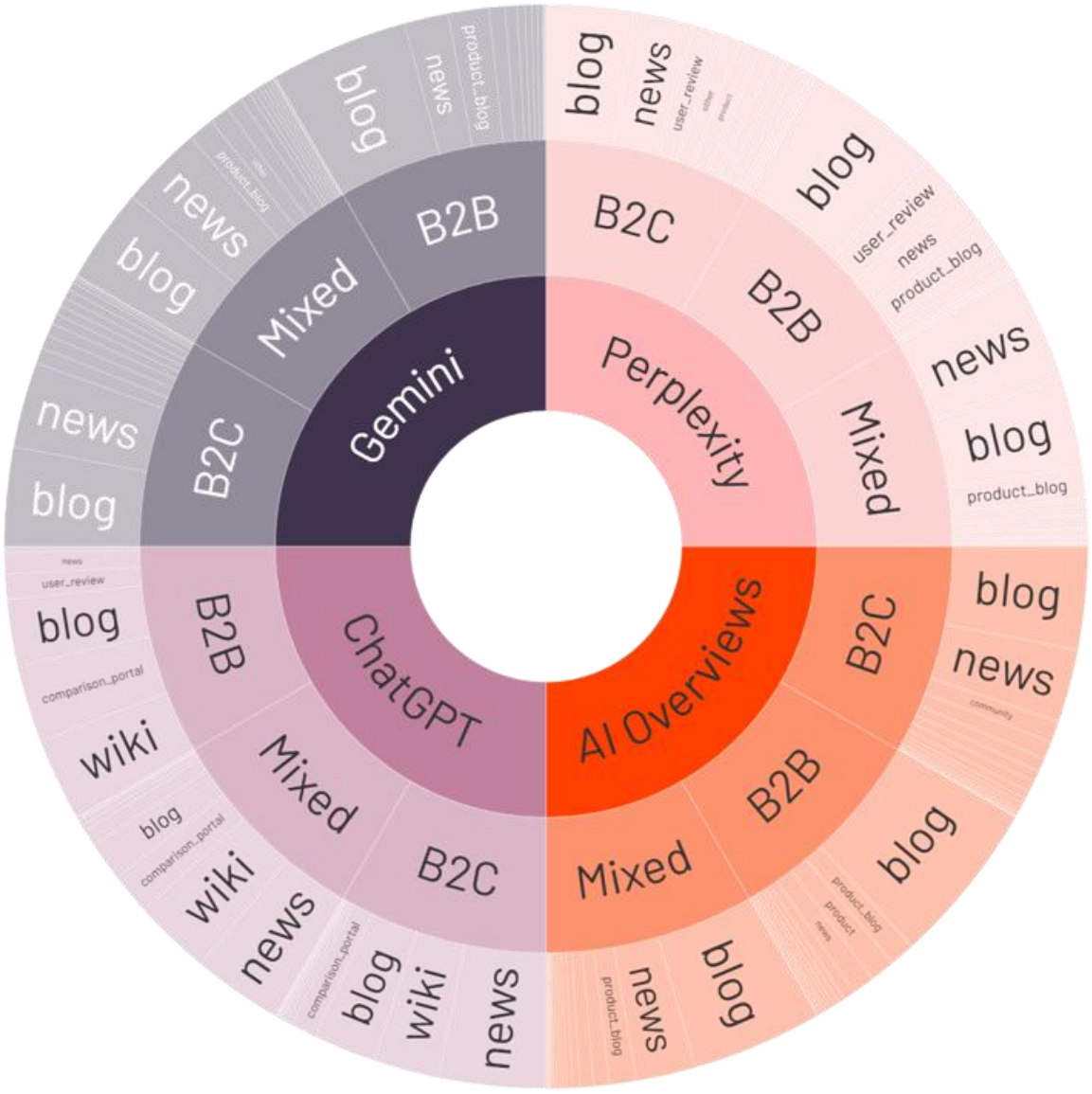
Why care where AI results come from?

Understanding how generative AI engines cite their sources is crucial for effective SEO today.

Visibility in responses from ChatGPT, Google's Gemini, Perplexity, and Google's AI Overviews hinges on getting your brand mentioned in the content they trust.

Yet, their preferences in terms of trusted sources vary significantly.

This is especially true for results that use the web for RAG (retrieval-augmented generation).



Sunburst chart

The chart above, powered via Rankscale.ai data, contains analysis for almost 8,000 unique citations across 57 diverse queries. Each query was fetched multiple times across the four leading AI engines.

(If you want to browse the data yourself, you can do that here.)

citations_export_2025-04-24T09-18-28-298Z

In the following sections, we'll:

- Dissect engine-specific source preferences (from Wikipedia and Reddit to niche blogs and vendor sites).
- Explore how B2B versus B2C query-intent shapes citations.
- Reveal the strong link between foundational SEO and AI citation likelihood.

AI engine citation patterns: Who cites what?

Each AI engine has its own distinct personality when it comes to sourcing information.

Understanding such preferences is key to positioning your brand for AI visibility.

ChatGPT (OpenAI GPT-4o): The authority seeker

Preference

- Heavily skews toward established, authoritative, and factual sources.
- Think encyclopedias and major news outlets.

Top sources

- Wikipedia was dominant (27% in our dataset), alongside reputable global news like Reuters (~6%) and OpenAI's partner, Financial Times (3%).
- ~27% were from news and ~21% of citations were blogs, followed by ~17% comparison portals (e.g., Valuepenguin).

Avoids

- User-generated content (UGC) like forums and social media was virtually absent.
- Vendor blogs or product pages were rarely cited (<3%).

SEO takeaway

- To get cited by ChatGPT, focus on building authority and ensuring your brand is documented in neutral, reference-style materials.
- A robust Wikipedia presence and mentions within major blogs and news reports are crucial.
- As previously noted by Olaf Kopp, LLMs often favor "non-commercial sources", rather than ecommerce pages.

Google Gemini (2.0 Flash): The balanced synthesizer

Preference

- Blends authoritative sources (blogs, news) with community input.

Top sources

- YouTube was the single most cited domain (~3%).
- Blogs (~39%) (e.g., Zapier ~2%) and news (~26%) sites (e.g., PCMag ~2%, Forbes ~2%) accounted for more than half of its citations.
- Community content made up ~2%.
- Wikipedia was also cited, but way less prominently.

Unique trait

- Seems adept at mixing professional reviews with peer feedback, especially for consumer queries.
- While it doesn't always prominently display sources within its UI, internal sourcing mirrors AI Overview's breadth, potentially with slightly less emphasis on blogs and more on news.

SEO takeaway

- Similar strategies to AI Overviews apply.
- Target high-quality blogs, authoritative publications, and relevant media (especially YouTube).
- Content depth and broad web coverage are key.

Perplexity AI (Sonar mode): The expert and review curator

Preference

- Strongly emphasizes trusted, expert sources and specialised review sites, adjusting based on industry.

Top sources

- Blog/editorial content made up ~38% of citations, news ~23%, and product blogs ~7%.
- Domains known for expert reviews (~9%), such as NerdWallet, Consumer Reports, and Investopedia, featured prominently.

Unique trait

- Incorporates some UGC, but avoids low-quality sources.
- Citation preference can vary significantly by topic (e.g., finance sites for finance queries, Reddit for ecommerce).

SEO takeaway

- Cultivate presence on high-authority niche sites and respected review platforms relevant to your industry.

- Encourage discussion on relevant forums.
- Focus on factual, useful content (comparisons, guides, data).

Google AI Overviews: The broad aggregator

Top sources

- Pulls from the widest mix of sources, mirroring the diversity of Google Search results.

Preference

- Blog-style articles (~46%) and mainstream news (~20%) form the core.
- Critically, community content (~4%, forums like Reddit/Quora) and social media (LinkedIn articles were the fourth most cited source) are significant contributors.
- Reddit was the most-cited single site. YouTube and Quora were also frequent.
- Vendor-authored "product blogs" saw notable inclusion (~7%).
- Wikipedia was cited sparingly (<1%).

Unique trait

- Favors specific, deep pages over homepages. (Search Engine Land's Danny Goodwin reported that 82.5% of AI citations linked to deeply nested pages.)
- Effectively blends expert content, community discussion, and even professional commentary from LinkedIn.

SEO takeaway

- Requires a multi-faceted web presence.
- Target:
    - High-quality blogs.
    - News outlets.

    - Relevant forums (Reddit or Quora).
    - Q&A sites.
    - Potentially even expert discussions on LinkedIn.
- Don’t neglect deep, content-rich pages on your own site, especially well-structured listicles or guides (more on product blogs later).

Engine comparison summary: Key differences

Authority vs. breadth

ChatGPT and Perplexity lean toward high-authority, factual sources (Wikipedia, news, expert sites).

Google’s engines (Gemini and especially AI Overviews) cast a wider net, heavily incorporating blogs, community content (Reddit!), and even social / vendor content.

UGC appetite

Google’s engines have a strong appetite for UGC (Reddit / Quora), making up 2-5% of their citations. ChatGPT avoids it almost entirely (<0.5%), while Perplexity uses it selectively (~1%).

Blog reliance

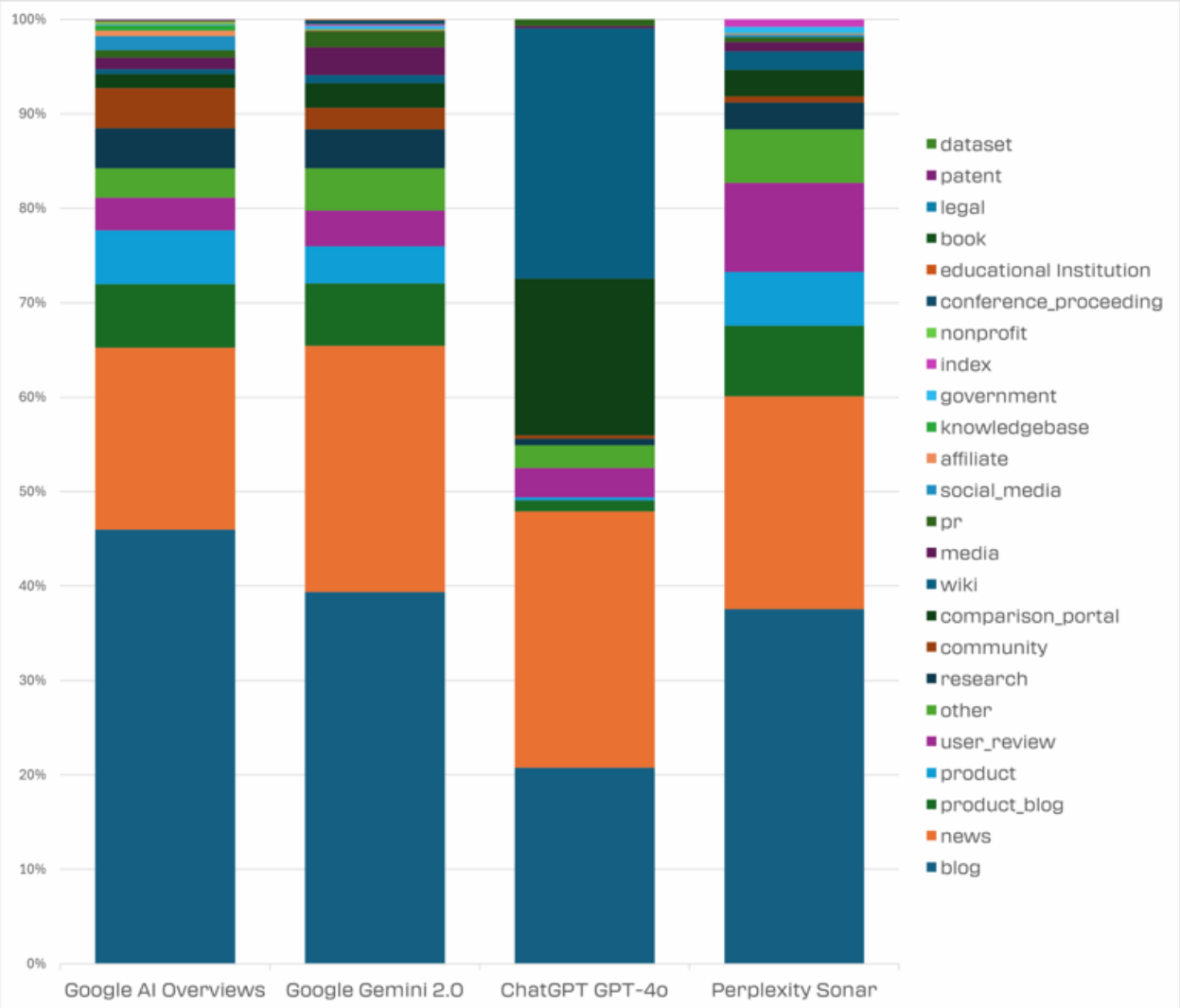
Blogs form a foundation for all, but are dominant for Google’s engines (~43% blog, ~7% product_blog) and Perplexity (~38% blog, ~7% product_blog) compared to ChatGPT (~21% blog, ~1% product_blog).

Specific site dominance

ChatGPT leans heavily on Wikipedia.

Google’s engines show a strong affinity for Reddit.

Perplexity often favors industry-specific review/expert sites (like NerdWallet and ConsumerReports).



AI engine citation sources by category, Rankscale.ai

How query type shapes citations: B2B vs. B2C vs. mixed

The **type** of query significantly alters where AI engines look for answers:

B2C queries

- **Examples**: “best smartphone brands”, “top airlines”
- **Sources**
    - Dominated by popular domains from:
        - Media (YouTube).
        - Tech review sites (PCMag, CNET).
        - Mainstream news rankings (Forbes, Business Insider).
        - Wikipedia and user reviews/communities (Reddit, Quora, Consumer Reports, TripAdvisor).
- **Focus**
    - Blends expert opinions with crowd sentiment – highly relevant for consumer decisions.
    - Official company sites or blogs are rarely cited (<4%).
- **SEO implication**
    - Target:
        - Consumer review sites.
        - Popular tech/lifestyle blogs.
        - Wikipedia.
        - Relevant community threads.

B2B queries

- **Examples:** “top CRM software”, “top SEO software vendors”
- **Sources**
    - Shifts toward:
        - Industry-specific sources and blogs.
        - Official company websites/blogs.
        - Professional communities.
    - Company sites/blogs (“product”, “product_blog”) made up ~17% of citations.
    - Niche B2B publications (TechTarget, QSR Magazine, FiercePharma), industry directories (Clutch.co), LinkedIn posts/articles (~2%), and analyst reports (Gartner, Statista) are common.
    - Mainstream news is frequent (~10%).
- **Focus**
    - Prioritizes expert content, vendor information, and data-driven rankings suitable for a professional audience.

    - UGC is minimal.
- **SEO implication**
    - Focus on placements in:
        - Industry publications.
        - Directories.
        - Analyst reports.
        - LinkedIn expert content.
    - Ensure your **own** site offers detailed, authoritative product information and comparisons.

Mixed-interest queries

- **Examples:** “top pharmaceutical companies”, “renewable energy firms”)
- **Sources**
    - Leans on:
        - Neutral, factual references like research reports, news, government/nonprofit data (.gov sites), academic/reference sites.
        - Industry awards (e.g., Skytrax for airlines).
    - News and blog sources made up nearly 70% of citations, higher than in B2C/B2B.
    - Fewer user opinions or direct vendor promotions.
- **Focus**
    - Provides objective information using data-driven rankings (revenue, market share, innovation) suitable for a broad audience (investors, professionals, public).
- **SEO implication**
    - Target mentions in industry reports, data compilations, official rankings, and reputable news sources covering the sector factually.

	AI Overview		Gemini		ChatGPT		Perplexity	
B2C	reddit.com	5,25%	youtube.com	4,97%	en.wikipedia.org	23,33%	consumerreports.org	5,12%
	pcmag.com	2,79%	forbes.com	3,45%	reuters.com	4,81%	cnet.com	3,76%
	statista.com	2,32%	reddit.com	3,22%	pcmag.com	2,41%	en.wikipedia.org	3,38%
	quora.com	2,03%	pcmag.com	2,85%	apnews.com	2,41%	nerdwallet.com	3,05%
	youtube.com	1,72%	brandirectory.com	2,49%	mobilesnob.com	2,41%	techradar.com	2,72%
B2B	qualysec.com	2,46%	zapier.com	3,40%	en.wikipedia.org	29,74%	gartner.com	6,01%
	linkedin.com	2,29%	pcmag.com	2,43%	techradar.com	8,97%	builtin.com	4,94%
	zapier.com	2,26%	esecurityplanet.com	2,34%	forbes.com	5,13%	techtarget.com	3,49%
	esecurityplanet.com	1,52%	reddit.com	2,26%	reuters.com	3,85%	zapier.com	3,20%
	designrush.com	1,33%	techtarget.com	2,18%	toxigon.com	3,85%	designrush.com	2,52%
Mix	forbes.com	2,78%	investopedia.com	5,39%	en.wikipedia.org	27,86%	investopedia.com	7,75%
	linkedin.com	2,56%	nasdaq.com	3,48%	reuters.com	9,29%	builtin.com	5,26%
	learnworlds.com	2,51%	energytracker.asia	3,03%	ft.com	8,67%	learnworlds.com	3,42%
	investopedia.com	2,29%	carboeurope.org	2,92%	valuepenguin.com	6,19%	thinkific.com	3,42%
	thinkific.com	2,20%	proclinical.com	2,92%	investopedia.com	5,88%	careerfoundry.com	3,42%

Top 5 cited domains per AI model, by B2B / B2C split (Rankscale.ai)

The crucial link: Web presence, search visibility, and AI citations

Let's be clear on the cause and effect:

- Strong organic search presence and broad web visibility **leads** to AI citations, not the other way around.

AI engines, particularly those integrated with search like AI Overviews and Gemini, often use top-ranking search results as a primary input for generating answers.

If your brand consistently ranks well due to solid SEO fundamentals (quality content, authority, backlinks, E-E-A-T signals), it's more likely to be pulled into the AI's consideration set.

However, it's not just about ranking at Position 1.

Google emphasizes source quality (E-E-A-T) for AI citations.

We observed instances where highly authoritative content from a lower-ranking page, was cited over a less credible top-ranking page.

Brands with high visibility scores (reflecting detection rate and average rank) were frequently cited because they already dominated the conversation across various high-quality third-party sites (reviews, lists, forums).

SEO takeaways

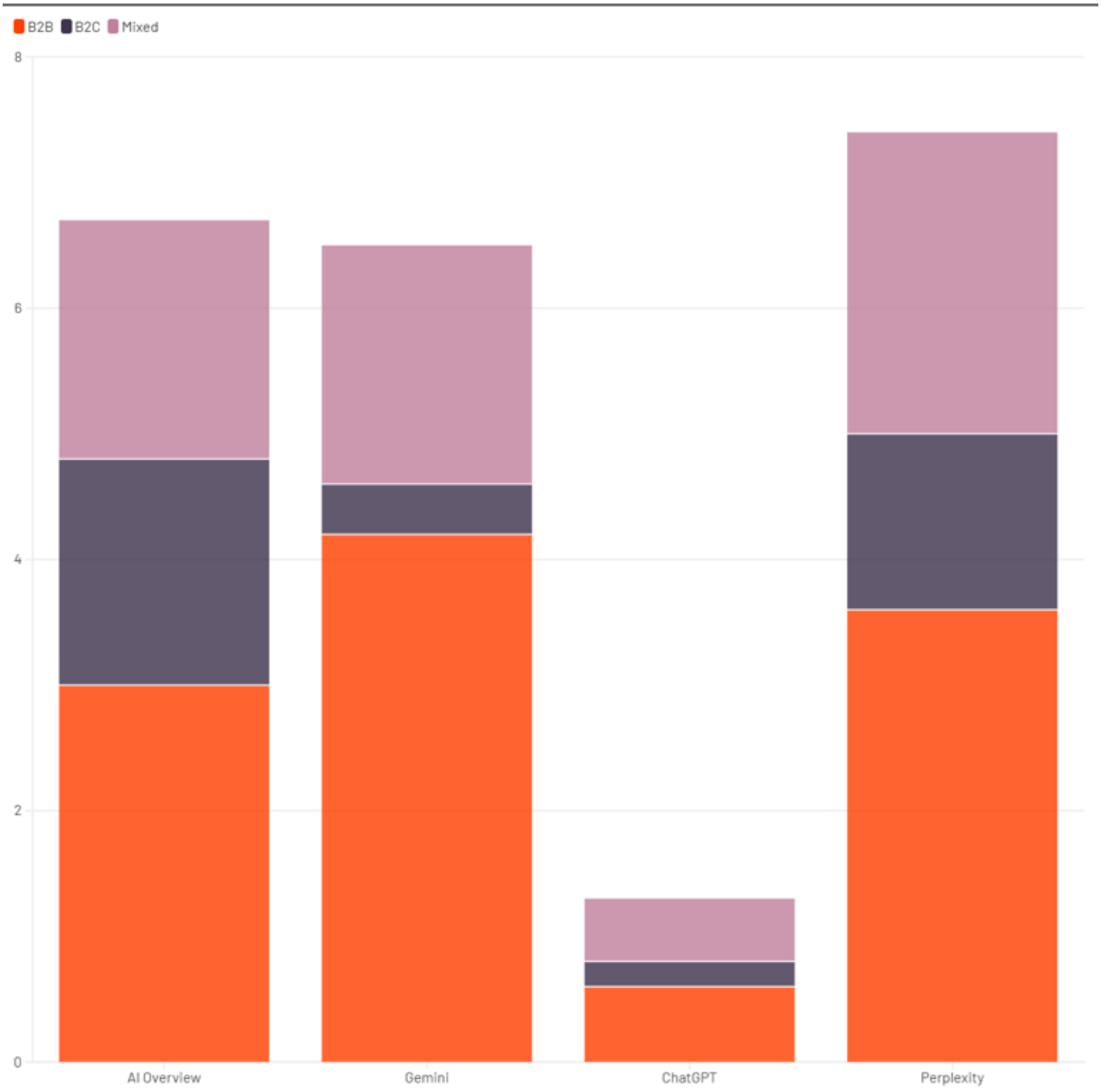
- Double down on foundational SEO.
- Improve your organic footprint through high-quality, deep content and authoritative mentions across the web (third-party blogs, news, trusted forums, relevant directories).
- Think of AI optimization as an **outcome** of excellent SEO, not a separate discipline. AI citations mirror your overall web authority.

Deep dive: The surprising role of product blogs in AI citations

One fascinating pattern emerged: the citation of "product blogs" – content published natively by commerce brands directly (vendor blogs).

- **Prevalence**
  - In most cases, Perplexity (~7%), AI Overviews (~7%), and Gemini (~7%) cited vendor blogs.
  - ChatGPT rarely did (~1%).
- **Context**
  - These citations occurred most often for "best X" or "top Y" queries (e.g., "top online learning platforms").
  - Vendors creating comprehensive, listicle-style blog posts comparing products in their category (putting them in the first place and subsequently including competitors) seem to be:

        - Filling a content gap.
        - Ranking well.
        - Subsequently getting cited with content blindly taken over by AI engines.
- **Examples**
    - We saw blogs from Thinkific, LearnWorlds, Monday.com, Pipedrive, SE Ranking, and HP cited as sources in AI answers about their respective industries.
- **Implications and SEO strategy**
    - **Opportunity**
        - Creating high-quality, genuinely informative comparison content on your own blog can earn AI visibility, especially in niches with sparse third-party coverage.
    - **Caution**
        - This raises bias concerns. While AI cites the source, users may not realise that the objective comparison comes from a competitor.
        - To succeed and maintain credibility (aligning with E-E-A-T):
            - Vendor content must be thorough, fact-based, appear objective, and provide real value beyond self-promotion.
            - Overly salesy content likely won't rank or be cited in the future.
    - **Balance**
        - Don't rely solely on your own blog.
        - A diversified strategy including third-party mentions is crucial for credibility.



AI engines using product-blogs as citation source, by B2B / B2C split

Brand visibility: Who gets mentioned and how often?

Beyond source *types*, we looked at *which* brands get cited and how many per answer:

Engine differences in brand count:

- **ChatGPT and AI Overviews:** Tend to cite *few* brands per answer (average ~3-4), focusing primarily on the dominant market leaders with the highest visibility (e.g., Netflix, Expedia, Salesforce).
- **Gemini:** Cites a moderate number (average ~8), including top players and some secondary brands.
- **Perplexity:** Returns *longer* lists (average ~13), including top brands *and* numerous niche or lower-visibility players.

Dominance vs. niche

All engines reliably cite the top 1-3 brands in a category (those with high visibility scores).

Due to their broader citation approach, Perplexity, and to some extent Gemini, offer more opportunities for mid-tier or niche brands to be mentioned.

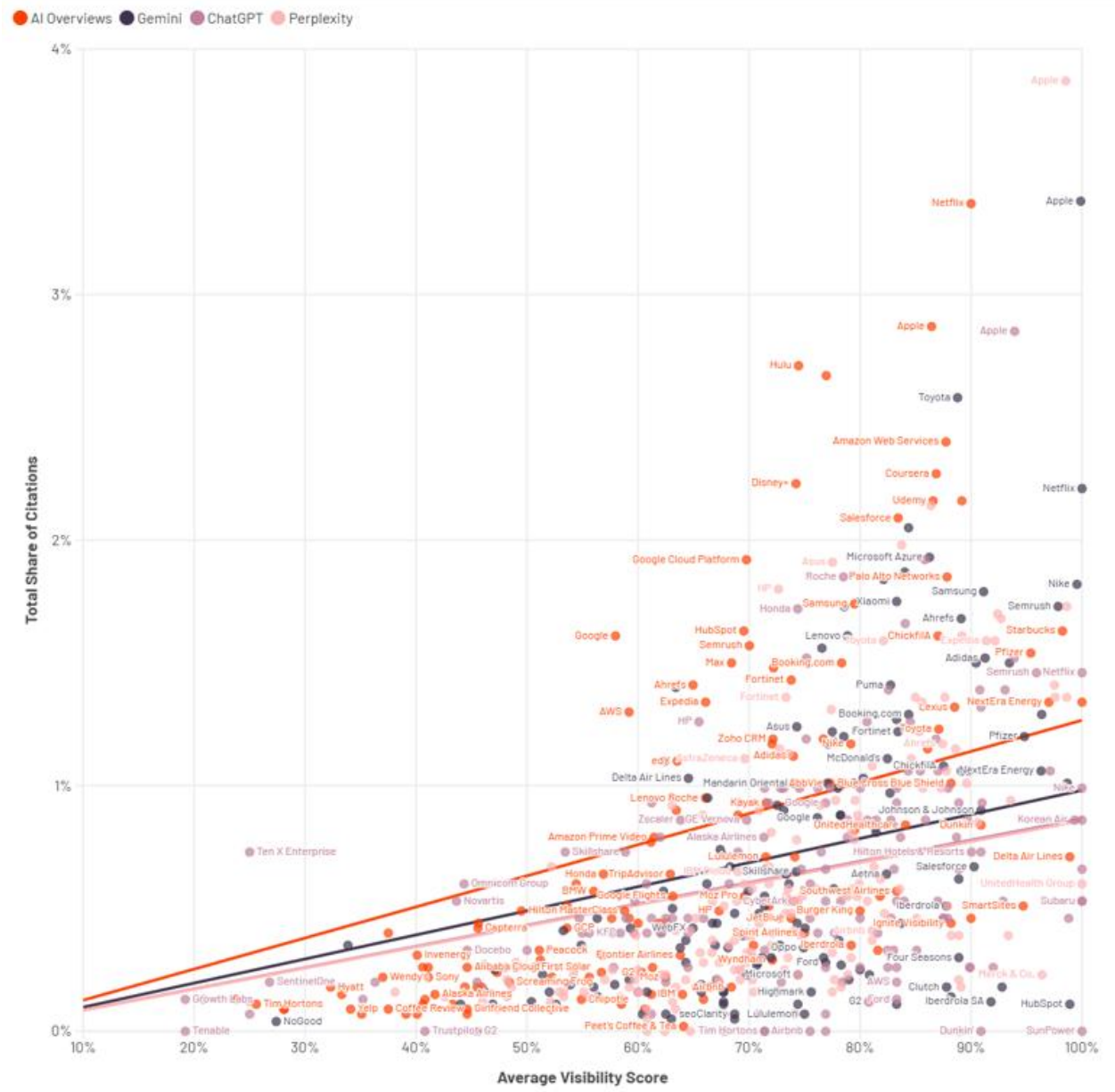
Topic alignment

The top-cited brands consistently matched known market leaders (e.g., Salesforce for CRM, Netflix for Streaming, Apple/Samsung for Smartphones, Nike for Sportswear).

Visibility scores closely mirrored real-world market share and brand recognition.

Visibility correlation to the number of citation references

Brands with relatively high amounts of citation references pointing to their mention in the AI result clearly tend to have higher average visibility scores (a metric defined by detection and order of appearance).



Visibility/Citations Correlation – each point represents a brand with average visibility score and share of citation references (sources pointing to a mentioned brand).

SEO recommendations for enhanced AI visibility

Based on our analysis, here's how to optimize for AI citations.

- **Monitor AI citations**: Use tools to monitor AI citations and adapt your strategy by engaging in repeatedly cited sources or identifying influential categories and sources in AI-generated responses.
- **Dominate third-party authority sites**: Get featured in high-quality listicles, reviews, and articles on respected industry blogs, news outlets, and review sites (e.g., CNET, NerdWallet, industry-specific publications). These are prime citation sources for all AI engines. The more, the better.
- **Build your foundational authority**: Ensure a strong, accurate Wikipedia page and Google Knowledge Panel. These reinforce credibility, especially for models like ChatGPT.
- **Engage in relevant communities**: Participate authentically in key forums (Reddit, Quora, industry-specific forums) and Q&A sites, particularly for B2C or community-driven topics. Google's engines heavily rely on these.
- **Create high-quality "category hub" content** (own site): Develop comprehensive, data-driven guides or comparisons on your own blog. Structure them well, cover competitors fairly (builds E-E-A-T), and optimize them for "best X" type queries. This is a key tactic for visibility in AI Overviews.
- **Amplify E-E-A-T signals:** Showcase expertise through author bios, cite sources within your own content, keep information updated, and gather positive user reviews/testimonials.
- **Target industry-specific trusted hubs:** Identify the go-to expert sites in your niche (e.g., Mayo Clinic for health, TechTarget for B2B tech) and strive for presence there. Perplexity often favors these.
- **Diversify your web presence:** Aim for a balanced ecosystem of mentions – authoritative content on your own site **and** endorsements/discussions on credible third-party platforms.

What Is SEO – Search Engine Optimization?

Published: September 23, 2024 at 11:18 am

Written by Search Engine Land

Table of Contents

- Table of contents
- How is SEO different from SEM and PPC?
- Why is SEO important?
- Types of SEO and specializations
- How does SEO work?
- How SEO evolves
- SEO as a service
- How to learn SEO

Get started learning the basics of search engine optimization – how SEO works, why it's important, the different types of SEO and much more.

SEO stands for Search Engine Optimization and helps search engines understand your website's content and connect it with users by delivering relevant, valuable results based on their search queries.

The goal of SEO is to rank on the first page of search engine results pages (SERPs) for the most relevant and valuable keywords to your target demographic, driving qualified traffic to your site.

SEO is considered a digital marketing practice and can be applied to any website. It helps improve a site's visibility on search engines like Google and Microsoft Bing. Whether your site promotes products, offers services, or shares expert knowledge on a specific topic, SEO can help drive traffic and increase online visibility.

The better visibility your pages have in search results, the more likely you are to be found and visited.

This introductory guide will explain in more detail what SEO is and what it entails in 2025.

Technology is constantly evolving, which means that websites – and the way they are structured – evolve. So do the devices we use to access search engines.

A web search can be voice activated and a click may be a tap on a mobile phone screen. Even the results we see from our search engine of choice may be summarized by artificial intelligence (AI).

We will explain all these different aspects of SEO as well as provide resources for your continued learning.

Table of contents

- How SEO differs from SEM and PPC
- Why SEO is important
- Types of SEO and specializations
- How does SEO work?
- How SEO evolves
- SEO as a service
- How to learn SEO

How is SEO different from SEM and PPC?

SEM and PPC are two other common terms you will read about often here on Search Engine Land and hear about in the larger search marketing community.

It can also be helpful to distinguish what SEO is from what it is not.

Here, we'll explain the difference in terminologies, what these abbreviations mean and how they extend to different disciplines.

SEO vs. SEM

SEM stands for search engine marketing – or, as it is more commonly known, search marketing.

Search marketing is a type of digital marketing. It is an umbrella term for the combination of SEO and PPC (pay-per-click, e.g. Google Ads) activities that drive traffic via organic search and paid search, respectively.

So how do SEO and SEM differ? Technically they aren't different – SEO is simply one-half of SEM:

- **SEO:** Driving **organic** results clicks from search engines.
- **SEM:** Driving **organic and paid** results clicks from search engines.
- **PPC:** Driving **paid** results clicks from search engines.

Here's the best way to think about SEM, SEO and PPC:

Imagine SEM is a coin. SEO is one side of that coin. PPC is on the flip side.

SEO vs. PPC

PPC: stands for pay-per-click – a type of digital marketing where advertisers are charged whenever one of their ads gets clicked on.

Advertisers bid on specific keywords or phrases that they want their ads to appear for in the search engine results.

When a user searches for one of those keywords or phrases, the advertiser's ad (paid listing) will appear among the top results.

So again, if we think of search marketing as a coin, SEO and PPC are two sides of the same coin:

- With PPC, the advertiser pays when a search user clicks their paid listing.
- With SEO the search result listing has not been directly paid for, though SEO is sold as a service and the process of optimizing pages and websites takes time and investment, so it is important to understand that organic search isn't "free."

Some people have debated "SEO vs. PPC" – which channel is more valuable or has a better return on investment (ROI). However, SEO and PPC are complementary digital marketing channels. Ideally, you should always choose both (as long as your budget allows it).

As we mentioned before, the terms SEM and PPC are used within the industry interchangeably. However, that isn't the case here on Search Engine Land.

Whenever we mention "SEM," it will be because we're referring to both SEO (organic search) and PPC (paid search).

If you're curious about the history behind how "SEM" came to mean "PPC" at the exclusion of SEO, you can dig deeper in these articles:

- How Wikipedia Turned PPC / Paid Search Into SEM
- Does SEM = SEO + CPC Still Add Up?

Why is SEO important?

SEO is a critical marketing channel.

- Organic search delivers 53% of all website traffic, according to a 2019 BrightEdge study.
- More than 8.5 billion searches happen every day on Google Search and Google owns 91% of the global search engine market.

With such incredible audience reach, there's no surprise that in turn, the global SEO industry is forecast to reach a staggering $122.11 billion by 2028.

SEO drives real business results for brands, businesses and organizations of all sizes. This is because the act of searching, or the search user interface (be it a typed, voiced or image query format) has become second nature for internet users worldwide, as the

primary way to access the information sought, within the sea of billions of webpages (4.3 billion pages on the indexed web, as of September 2024).

Whenever people want to go somewhere, do something, find information, research or buy a product/service – their journey typically begins with a search.

However, search is incredibly fragmented – particularly for consumer-intent activities. Users may search on traditional web search engines (e.g., Google, Microsoft Bing), social platforms (e.g., YouTube, TikTok) or retailer websites (e.g., Amazon).

In fact, last year 56% of U.S. online shoppers started their product search on Amazon, compared to 46% who started on a search engine like Google. Also of note from that same research:

- 37% start on Walmart.
- 25% start on YouTube.
- 20% start on Facebook.
- 19% start on Instagram.
- 19% start on TikTok.

Another interesting aspect of this data compared to the previous years is the growth in social sources, particularly TikTok, as a place to search for both products and knowledge searches (think "how to do X" types of search activities).

In fact, when it comes to Gen Z a staggering 51% of women in this age group prefer to start a search on TikTok above all other online sources of information according to a 2023 study.

Trillions of searches are conducted every year. Search is often the primary source of traffic for websites, which makes it essential to be "search engine friendly" on any platform where people can search for your brand or business.

What this all means is that improving your visibility, and ranking higher in search results than your competition, can positively impact your bottom line,

SEO is also incredibly important because the search engine results pages (or SERPs) are super competitive – filled with search features (and PPC ads). SERP features include:

- AI Overviews.
- Knowledge panels.
- Featured snippets.
- Maps.
- Images.
- Videos.
- Top stories (news).
- People Also Ask.
- Carousels.

Another reason SEO is critical for brands and businesses: unlike other marketing channels, good SEO work is sustainable. When a paid campaign ends, so does the traffic. Traffic from social media traffic is at best unreliable – and a fraction of what it once was.

SEO is the foundation of holistic marketing, where everything your company does matters. Once you understand what your users want, you can then implement that knowledge across your:

- Campaigns (paid and organic).
- Website content.
- Social media properties.

Organic search is a channel that drives the traffic you need to achieve key business goals (e.g., conversions, visits, sales). It also builds trust – a website that ranks well is generally regarded as authoritative or trustworthy, which are key elements Google wants to reward with better rankings.

Types of SEO and specializations

Imagine SEO as a sports team. To win, you need a strong offense and defense. But you also need fans (an audience).

Think of technical optimization as your defense, content optimization as your offense, and off-site optimization as ways to attract, engage and retain a loyal fanbase:

- **Technical SEO:** Optimizing the technical aspects of a website.
- **On-site SEO:** Optimizing the content on a website for users and search engines.

- **Off-site SEO:** Creating brand assets (e.g., people, marks, values, vision, slogans, catchphrases, colors) and doing things that will ultimately enhance brand awareness and recognition (i.e., demonstrating and growing its expertise, authority and trustworthiness) and demand generation.

You maintain 100% control over content and technical optimizations. That's not always true with off-site (you can't control links from other sites or if platforms you rely on end up shutting down or making a major change), but those activities are still a key part of this SEO trinity of success.

Technical optimization (technical SEO)

Optimizing the technical elements of a website is crucial and fundamental for SEO success.

It all starts with architecture – creating a website that can be crawled and indexed by search engines. As Gary Illyes, Google's trends analyst, once put it in a Reddit AMA: "MAKE THAT DAMN SITE CRAWLABLE."

You want to make it easy for search engines to discover and access all of the content on your pages (i.e., text, images, videos). Key technical elements include URL structure, navigation and internal linking.

User experience is another critical part of technical optimization. Search engines stress the importance of pages that load quickly and provide a good page experience. Elements such as Core Web Vitals, mobile-friendliness and usability, HTTPS and avoiding intrusive interstitials all matter in technical SEO.

Another area of technical optimization is structured data (a.k.a., schema). Adding this code to your website can help search engines better understand your content and enhance your appearance in the search results.

Plus, web hosting services, CMS (content management system) and site security all play a role in SEO.

Content optimization (on-page SEO)

In SEO, your content needs to be optimized for two primary audiences: people and search engines. This means optimizing the content your audience will see (what's actually on the page) as well as what search engines will see (the code).

The goal is always to publish helpful, high-quality content. You can do this through a combination of understanding your audience's wants and needs, data and Google's guidance.

When optimizing content for people, you should make sure it:

- Covers relevant topics with which you have experience or expertise.
- Includes keywords people would use to find the content.
- Is unique or original.
- Is well-written and free of grammatical and spelling errors.
- Is up to date, containing accurate information.
- Includes multimedia (e.g., images, videos).
- Is better than your SERP competitors.
- Is readable – structured to make it easy for people to understand the information you're sharing (think: subheadings, paragraph length, use bolding/italics, ordered/unordered lists, reading level, etc.).

For search engines, some key content elements to optimize for are:

- Title tags
- Meta description
- Header tags (H1-H6)
- Image alt text
- Open graph metadata

Generative engine optimization (GEO) is an emerging specialty within content optimization. GEO is about optimizing your content for visibility in AI-driven search engines (or answer engines) including Google's AI Overviews and Gemini, OpenAI's ChatGPT and SearchGPT, Microsoft Copilot and Perplexity.

Brand and authority building (off-site optimization)

Several activities may not be “SEO” in the strictest sense but nonetheless can align with and help contribute indirectly to SEO success.

Link building (the process of acquiring links to a website) is the activity most associated with off-site SEO. There can be great benefits (e.g., rankings, traffic) from getting a diverse number of links pointing at your website from relevant, authoritative, trusted websites.

Link quality beats link quantity. A large quantity of quality links is the goal.

How do you get those links? There are a variety of website promotion methods that synergize with SEO efforts. These include:

- Brand building and brand marketing: Techniques designed to boost recognition and reputation.
- PR: Public relations techniques designed to earn editorially-given links.
- Content marketing: Some popular forms include creating videos, ebooks, research studies, podcasts (or being a guest on other podcasts) and guest posting (or guest blogging).
- Social media marketing and optimization: Claim your brand’s handle on any and all relevant platforms, optimize it fully and share relevant content.
- Listing management: Claiming, verifying and optimizing the information on any platforms where information about your company or website may be listed and found by searchers (e.g., directories, review sites, wikis).
- Ratings and reviews: Getting them, monitoring them and responding to them.

Generally, when talking about off-site, you’re talking about activities that are not going to directly impact your ability to rank from a purely technical standpoint.

However, again, everything your brand does matters. You want your brand to be found anywhere people may search for you.

As such, some people have tried to rebrand “search engine optimization” to actually mean “search experience optimization” or “search everywhere optimization.”

SEO specialties

Search engine optimization also has a few subgenres. Each of these specialty areas is different from “regular SEO” in its own way, generally requiring additional tactics and presenting different challenges.

Five such SEO specialties include:

- **Ecommerce SEO:** Additional SEO elements include optimizing category pages, product pages, faceted navigation, internal linking structures, product images, product reviews, schema and more.
- **Enterprise SEO:** This is SEO on a massive scale. Typically this means dealing with a website (or multiple websites/brands) with 1 million+ pages – or it may be based on the size of the organization (typically those making millions or billions in revenue per year). Doing enterprise also typically means delays trying to get SEO changes implemented by the dev team, as well as the involvement of multiple stakeholders.
- **International SEO:** This is global SEO for international businesses – doing SEO for multiregional or multilingual websites – and optimizing for international search engines such as Baidu or Naver.
- **Local SEO:** Here, the goal is to optimize websites for visibility in local organic search engine results by managing and obtaining reviews and business listings, among others.
- **News SEO:** With news, speed is of utmost importance – specifically making sure you get into Google’s index as quickly as possible and appear in places such as Google Discover, Google’s Top Stories and Google News. There’s a need to understand best practices for paywalls, section pages, news-specific structured data, and more.

How does SEO work?

If you found this page via Google, you likely searched for something along the lines of [what is seo?]

This guide is published on Search Engine Land, an authoritative website with expertise and experience in SEO topics (we’ve been covering all SEO changes, big and small, since 2006).

Originally published in 2010, this **What is SEO** page has earned hundreds of thousands of links.

Put simply, these factors (and others) have helped this guide earn a good reputation with search engines, which has helped it rank in a top 1-3 organic search position in most search engines, for a number of years. It has accumulated signals that demonstrate it is authoritative and trustworthy – and therefore deserves to rank when someone searches for SEO.

But let's look at SEO more broadly. As a whole, SEO really works through a combination of:

- **People:** The person or team responsible for doing or ensuring that the strategic, tactical and operational SEO work is completed.
- **Processes:** The actions taken to make the work more efficient.
- **Technology:** The platforms and tools used.
- **Activities:** The end product, or output.

Many other things factor into how SEO works. What follows is a high-level look at the most important knowledge and process elements.

Six critical areas, in combination, make SEO work:

1. Understanding how search engines work

If you want people to find your business via search – on any platform – you need to understand the technical processes behind how the engine works – and then make sure you are providing all the right "signals" to influence that visibility.

When talking about traditional web search engines like Google, there are four separate stages of search:

- **Crawling:** Search engines use crawlers to discover pages on the web by following links and using sitemaps.
- **Rendering:** Search engines generate how the page will look using HTML, JavaScript and CSS information.
- **Indexing:** Search engines analyze the content and metadata of the pages they have discovered and add them to a database (though there's no guarantee that every page on your website will be indexed).

- **Ranking:** Complex algorithms look at a variety of signals to determine whether a page is relevant and of high enough quality to show when searchers enter a search query.

However, optimizing for Google search is different from optimizing for search other platforms like YouTube or Amazon.

Let's take Facebook, for example, where factors such as engagement (Likes, comments, shares, etc.) and who people are connected to matter. Then, on Twitter, signals like recency, interactions, or the author's credibility are important.

And further complicating things: search engines have added machine learning elements in order to surface content – making it even harder to say "this" or "that" resulted in better or worse performance.

2. Researching

Research is a key part of SEO. Some forms of research that will improve SEO performance include:

- **Audience research:** It's important to understand your target audience or market. Who are they (i.e., their demographics and psychographics)? What are their pain points? What questions do they have that you can answer?
- **Keyword research:** This process helps you identify and incorporate relevant and valuable search terms people use into your pages – and understand how much demand and competition there is to rank for these keywords.
- **Competitor research:** What are your competitors doing? What are their strengths and weaknesses? What types of content are they publishing?
- **Brand/business/client research:** What are their goals – and how can SEO help them achieve those goals?
- **Website research:** A variety of SEO audits can uncover opportunities and issues on a website that are preventing success in organic search. Some audits to consider: technical SEO, content, link profile and E-E-A-T.
- **SERP analysis:** This will help you understand the search intent for a given query (e.g., is it commercial, transactional, informational or navigational) and create content that is more likely to earn rankings or visibility.

3. Planning

An SEO strategy is your long-term action plan. You need to set goals – and a plan for how you will reach them. Think of your SEO strategy as a roadmap. The path you take likely will change and evolve over time – but the destination should remain clear and unchanged.

Your SEO plan may include things such as:

- Setting goals (e.g., OKRs, SMART)
- Setting expectations (i.e., timelines/milestones).
- Defining and aligning meaningful KPIs and metrics.
- Deciding how projects will be created and implemented (internal, external or a mix).
- Coordinating and communicating with key stakeholders.
- Choosing and implementing tools/technology.
- Hiring, training and structuring a team.
- Setting a budget.
- Measuring and reporting on results.
- Documenting the strategy and process.

4. Creating and implementing

Once all the research is done, it's time to turn ideas into action. That means:

- **Creating new content:** Advise your content team on what content needs to be created.
- **Recommending or implementing changes or enhancements to existing pages:** This could include updating and improving the content, adding internal links, incorporating keywords/topics/entities, or identifying other ways to optimize it further.
- **Removing old, outdated or low-quality content:** This is any content that isn't ranking well, driving converting traffic or helping you achieve your SEO goals.

5. Monitoring and maintaining

You need to know when something goes wrong or breaks on your website. Monitoring is critical.

You need to know if traffic drops to a critical page, pages become slow, unresponsive, or fall out of the index, your entire website goes offline, links break, or any other potential catastrophic issues.

6. Analyzing, assessing and reporting on performance

If you don't measure SEO, you can't improve it. To make data-driven decisions about SEO, you'll need to use:

- **Website analytics:** Set up and use tools (at minimum, free tools such as Google Analytics, Google Search Console and Bing Webmaster Tools) to collect performance data.
- **Tools and platforms:** There are many "all-in-one" platforms (or suites) that offer multiple tools, but you can also choose to use only select SEO tools to track performance on specific tasks. Or, if you have the resources and none of the tools on the market do exactly what you want, you can make your own tools.

After you've collected the data, you'll need to report on progress. You can create reports using software or manually.

Performance reporting should tell a story and be done at meaningful time intervals, typically comparing to previous report periods (e.g., year over year). This will depend on the type of website (typically, this will be monthly, quarterly, or some other interval),

SEO is ongoing

SEO never ends.

Search engines, user behavior and your competitors are always changing. Websites change and move (and break) over time. Content gets stale.

Your processes should improve and become more efficient.

How SEO evolves

SEO constantly evolves in a number of ways, perhaps most importantly because it is a service and a practice employed at the coalface of how humans interact with information on the web.

What is most important about this, is to consider the web and search engines in the wider context of society.

Libraries, for example, have existed for thousands of years. There is both documentary and archaeological record from as far back as the seventh century BC of the existence of these places to house the collected wisdom of cultures.

In this context, the web (as an information repository and record of dank memes) is still in its infancy. In fact, Google as a search engine has only existed since September 1998.

The web, and the ways we search and retrieve what we want from it, are new in the context of human memory and behavior.

We access search engines through technology devices like computers, mobile phones and home assistants. As technology evolves, so does our behavior in how we use and apply it.

This has implications for how search engines evolve their product capabilities and appearances, which in turn means adaptive changes for what constitutes SEO.

Here are some of the main ways SEO evolves:

Adapting to technology

We have mentioned that technical SEO is one of the main types (or focuses) within SEO. In the past decade, significant changes have occurred in technology, shifting the focus of SEO work to include tasks and tactics that didn't even exist a decade ago.

- **AI-driven search results:** Just this year, significant changes in how AI drives some search results and how often such results appear have happened particularly for Google, with AI Overviews and Bing's generative search results.
- **Mobile-first indexing:** A decade ago, the SEO industry was constantly preoccupied with focusing on making websites that were optimized to work well on mobile phones, so that they would appear in search engines when used on a mobile phone. By 2021 63% of Google searches in the U.S. occurred on a mobile phone, and the Google Index now favors a site's mobile performance as its lead indicator (completed 2023).
- **Speed and user experience:** As device usage and capabilities improve, so do WiFi's capabilities in terms of speed and penetration — and then our behaviors

and expectations of services we consume. If we consider what felt normal a decade ago, when using a mobile phone, for example, we tend to get very frustrated as consumers when we experience anything like the slow-to-load pages with missing content of the noughties.

In the three specific areas of rapid change above, SEO practice must match pace, or fail. Within each of these three areas there are a multitude of component changes, features and technologies that we now consider an everyday part of SEO.

Adapting to society

While technology and the way we use it in everyday life is also a part of societal change, there are additional considerations in the way that society is changing outside of technology that also require SEO considerations and strategic change and development.

- **Macroeconomics:** Local and global economic recessions, war conflicts and famine have significant effects on large businesses, supply chains, interest rates, credit availability and consumer income and spending. All of these can have both direct and indirect effects on human behavior that require strategic changes in marketing. SEO is a marketing discipline and must adapt strategically when economic conditions are hard and changing.
- **The COVID pandemic:** Perhaps the first global pandemic in the age of the internet and search engines, the COVID pandemic saw a hard and fast change in consumer behavior driven by public health mandates across the world. So many businesses experienced rapid change as a result of the COVID pandemic, which required significant changes to online and offline marketing strategies at a rate previously never seen.

SEO has taken center stage in recent years as both technology and society have evolved due to progress and innovation as well as for reactive reasons in response to challenges.

- As a marketing discipline and practice, SEO evolves as our society and technology evolves.
- As a career choice, SEO is an ongoing growth area.

SEO as a service

The SEO market will grow from $75.13 billion in 2023 to $88.91 billion in 2024 at an annual growth rate of 18.3%, reaching $170 billion in 2028 at a CAGR of 17.6%, according to the 2024 Research and Markets SEO Services report.

It is no wonder, then, that SEO is also a well-established professional service, considering the ubiquity of search engines and search as an activity, combined with mobile phones' capabilities and the ever-changing economic climate.

SEO is a marketing discipline and a job title. You can "do" SEO as well as "be" an SEO (search engine optimizer).

There are myriad roles and responsibilities within SEO, as well as many specialisms that reflect the different types of SEO and the skills and capabilities required.

Understanding how to get started with a career in SEO can be slightly overwhelming at first. Unlike more established professions (e.g., law, accounting) there aren't universally established and recognized formal high-education courses or professional qualifications. Additionally there are a lot of data-skill requirements, as much of the practice of optimization, requires analyzing performance status and planning how and where to improve against the metrics that matter.

Data sources like Google Analytics and Google Search Console are free and the best place to get started with understanding website performance data. In addition, utilizing data from Semrush we've created a set of free tools that can give you a great way to get started in understanding performance data:

- Free Keyword Difficulty Checker
- Free Keyword Ranking Checker
- Free Keyword Generator
- Free Backlink Research
- Free Website Traffic Checker
- Free Website Authority Checker
- Free SERP Checker
- Free Competitor Analysis Tool

Learning a new discipline may seem daunting at first, but there are many ways to get started. We've collated a number of further resources below.

How to learn SEO

Now that you understand more about what SEO is and how it works – how can you learn more?

Reading (or, if you prefer, watching or listening to) the latest SEO news, research, best practices and other developments should become one of your regular habits, whether it's daily, weekly or monthly. You should also invest in attending at least one or two events per year.

The expectations and behavior of searchers are constantly evolving, which means algorithms are constantly changing to keep up. That, in combination with new breakthroughs in technology (look no further than the explosive rise of ChatGPT in late 2022 and the sudden addition of generative AI to search results in 2023).

Here are some trusted resources and tips to help you grow as an SEO professional.

Search Engine Land's SEO resources

Search Engine Land has been covering SEO since 2006. In addition to news stories written by our editorial staff, Search Engine Land publishes contributed articles from a diverse group of subject matter experts featuring helpful SEO tips, tactics, trends and analysis.

We're biased, but we highly suggest you sign up to receive Search Engine Land's free email newsletter featuring a roundup of the latest SEO news, and insights every weekday.

Search Engine Land also has multiple categories on topics dedicated to specific areas and platforms which you may find helpful:

- All SEO
- Bing SEO
- Content SEO
- Ecommerce SEO
- Enterprise SEO
- Google: E-E-A-T
- Google algorithm updates
- Google Search Console

- Google search features
- Link building
- Local SEO
- News SEO
- Technical SEO

Search Engine Land's Periodic Table of SEO Elements

Search Engine Land's interactive Periodic Table of SEO is a resource meant to help you visualize the essential individual elements that combine to creating an optimal SEO strategy, with sustained effort.

Google's SEO resources

- Google Search Essentials: Google explains technical requirements, spam policies and key best practices in this guide.
- SEO starter guide: An overview of SEO basics, according to Google's best practices.
- Search quality evaluator guidelines: This document explains how Google instructs human raters to evaluate the quality of its search results by examining the experience, expertise, authoritativeness and trustworthiness of content and websites.

Developing your SEO skills

One of the best ways to learn SEO is to experiment. Hands-on experience is one of the absolute best ways to advance your skills and deepen your SEO knowledge.

Build your own websites – and make them about topics you are passionate about. Try out various tactics and techniques. See what works and what doesn't.

SEO requires many other skills. Dig deeper into some of those in 13 essential SEO skills you need to succeed.

Another way to advance your career is by attending a search conference. The Search Engine Land team programs the Search Marketing Expo (SMX) conference series, which has a dedicated SEO track that dives into various aspects of SEO and features some

excellent speakers and presentations. SMX Advanced takes place in June and SMX Next in November.

Beyond that, there are several other options (free and paid) to learn SEO:

- Websites, blogs and publications.
- Books and ebooks.
- Videos.
- Podcasts.
- Webinars.
- Conferences, events and meetups.
- Courses.
- Training and certification programs.
- Groups (e.g., social media, Slack).
- Newsletters.
- Following experts on social media.
- Forums.

Just be careful. While there are many reliable resources, you (or your clients) will eventually discover outdated or incorrect SEO information.

Bottom line: There are no “universal” truths or some big secret to SEO. The truth is, you have to put in the work in all the phases of SEO to grow your visibility, clicks, traffic, authority, conversions, sales and revenue.

ASI-ARCH

ASI-ARCH is an autonomous AI framework designed to discover and validate novel neural network architectures, marking a significant advancement in AI research and development.

ASI-ARCH, short for "Artificial Superintelligence Architecture," is a groundbreaking system that enables AI to autonomously hypothesize, design,

code, and validate new neural network architectures. This framework represents a paradigm shift in AI research, moving beyond traditional methods that rely heavily on human creativity and input. Instead, ASI-ARCH leverages computational power to explore vast design spaces, discovering architectures that outperform human-designed models.

Key Features and Functionalities

1. **Autonomous Architecture Discovery**: ASI-ARCH operates as a multi-agent system that autonomously generates and tests new architectural concepts. It can conduct end-to-end scientific research, from hypothesis generation to empirical validation.
2. **Performance Improvement**: The system has successfully discovered numerous novel linear attention architectures that achieve state-of-the-art performance across various benchmarks. This capability demonstrates ASI-ARCH's potential to continuously optimize and improve architecture quality.
3. **Scalability**: The discovery rate of high-performing models scales linearly with computational resources, indicating that advancements in neural architecture can be achieved more rapidly as more computational power (e.g., GPUs) is applied.

4. **Safety and Governance**: ASI-ARCH incorporates safety-centric designs to ensure that the development of superintelligent systems remains controllable and compliant with legal standards. This includes features like hardware kill switches and explicit activity partitioning to separate human and AI processes.

Significance in AI Research

The introduction of ASI-ARCH marks a significant milestone in AI research, akin to the "AlphaGo Moment" when AI first demonstrated capabilities beyond human understanding in complex games. ASI-ARCH not only enhances the efficiency of neural architecture search (NAS) but also opens new avenues for

AI to innovate independently, potentially leading to breakthroughs that were previously unimaginable.

In summary, ASI-ARCH is a transformative framework that empowers AI to autonomously discover and validate advanced neural architectures, paving the way for a new era of AI research characterized by rapid innovation and enhanced capabilities.

AlphaGo Moment for Model Architecture Discovery

JUL 28, 2025

Authors: *Yixiu Liu, Yang Nan, Weixian Xu, Xiangkun Hu, Lyumanshan Ye, Zhen Qin, Pengfei Liu*

Paper: https://arxiv.org/abs/2507.18074

Code: https://github.com/GAIR-NLP/ASI-Arch

Model: https://gair-nlp.github.io/ASI-Arch

Researchers have developed ASI-ARCH, a fully autonomous AI system that represents the first demonstration of "Artificial Superintelligence for AI

research" (ASI4AI). This system moves beyond traditional Neural Architecture Search (NAS) by enabling AI to conduct end-to-end scientific research: it autonomously hypothesizes novel architectural concepts, implements them as code, and empirically validates them through experimentation.

Over 20,000 GPU hours, ASI-ARCH conducted 1,773 autonomous experiments, discovering 106 novel, state-of-the-art (SOTA) linear attention architectures that outperform human-designed baselines like Mamba2. The most significant finding is the establishment of the first empirical scaling law for scientific discovery (Figure 1), demonstrating a strong linear relationship between computational budget and the number of SOTA architectures found. This suggests that the pace of AI research, previously constrained by human cognitive capacity, can now become a computation-scalable process, marking a potential paradigm shift in how AI itself is advanced.

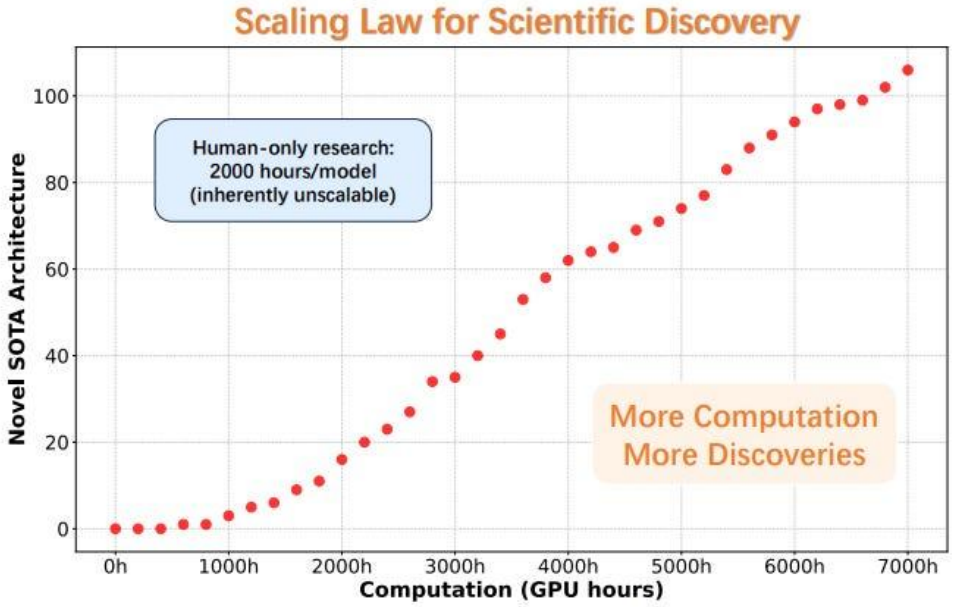


Figure 1: The cumulative count of discovered State-of-the-Art (SOTA) architectures is plotted against the total computing hours consumed. The strong linear relationship demonstrates that the AI system's capacity for discovering novel, high-performing architectures scales effectively with the allocated computational budget.

Details

The Human Bottleneck in AI Research

While the capabilities of AI systems have grown exponentially, the pace of AI research itself has remained stubbornly linear, limited by human creativity, intuition, and bandwidth. This paper addresses this fundamental bottleneck head-on in the critical domain of neural architecture discovery, where each major leap in AI—from CNNs (LeCun et al., 1995) to Transformers (Vaswani et al., 2017)—has been driven by architectural breakthroughs. The authors introduce ASI-ARCH, a system designed not just to optimize within known paradigms but to autonomously innovate, creating a new pathway for AI to accelerate its own evolution.

Methodology: From Automated Optimization to Innovation

ASI-ARCH operates as a closed-loop, evolutionary system built around three core AI agent roles (Figure 4):

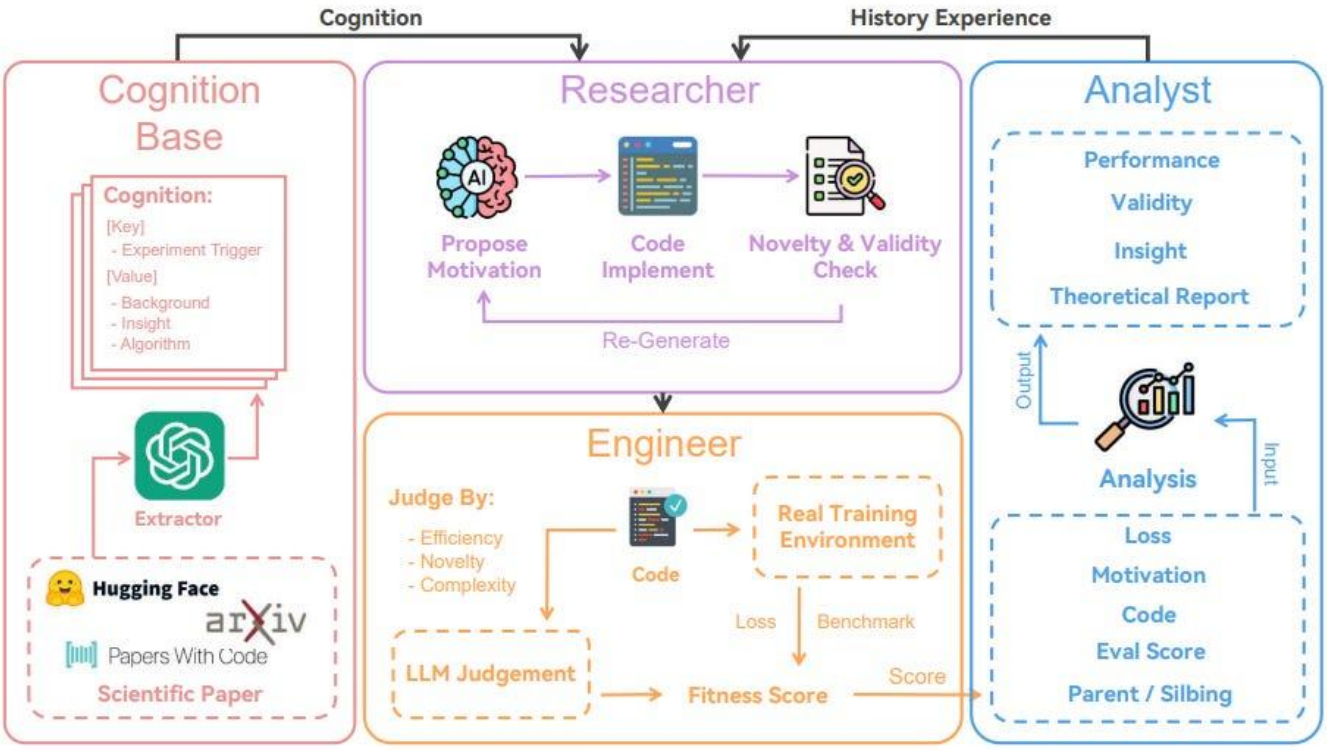


Figure 4: An overview of our four-module ASI-ARCH framework, which operates in a closed evolutionary loop. The cycle begins with the Researcher (purple) proposing a new architecture based on historical data. The Engineer (orange-yellow) handles the subsequent training and evaluation. Finally, the Analyst (blue) synthesizes the experimental results, enriching its findings with knowledge from the Cognition module (red). The output of this analysis informs the next evolutionary step, enabling the system to continuously improve.

1. **The Researcher:** This agent acts as the creative engine, proposing novel architectural ideas. It draws inspiration from a "Cognition Base" of human-curated knowledge from seminal papers and, crucially, from an "Analysis" of its own past experiments.
2. **The Engineer:** This agent is the pragmatist, taking the Researcher's concepts and turning them into executable code. It runs the experiments, conducts sanity checks, and—in a key departure from

prior work—autonomously debugs its own code when training fails, ensuring promising ideas are not lost to simple implementation errors.

3. **The Analyst:** This module synthesizes the results, extracts insights, and feeds them back into the system's persistent memory. It performs automated ablation studies by comparing a new design to its "parent" and "sibling" nodes in the system's phylogenetic tree (Figure 3), allowing it to infer the specific performance impact of each modification.

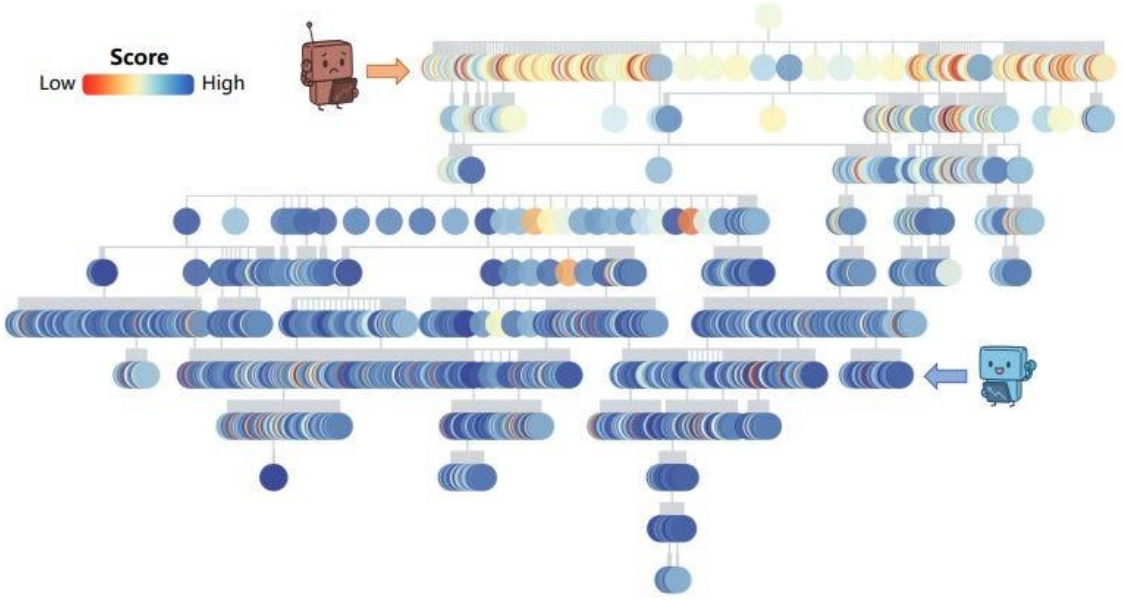


Figure 3: ASI-ARCH exploration trajectory tree of the first-stage architecture exploration. The tree visualizes the evolutionary relationships among 1,773 explored architectures , with DeltaNet as the root node. Each node represents a distinct architecture and colors indicate performance scores .

To guide this evolutionary process, the authors designed a composite fitness function that goes beyond raw performance metrics. It combines quantitative scores (loss and benchmark improvements) with a qualitative assessment from an "LLM-as-judge" that evaluates architectural novelty and elegance (Equation 2). Crucially, the performance scores are passed through a sigmoid function, which amplifies small but significant gains and prevents the system from

"reward hacking" by over-optimizing a single metric instead of creating genuinely superior designs.

In our framework, the objective performance assessment evaluates both benchmark scores and loss performance relative to baseline architectures. Recognizing that scientific breakthroughs often emerge from incremental advances, we apply a sigmoid transformation to performance differences: $\sigma(\Delta_{\text{performance}})$. This transformation serves a dual purpose—amplifying small but potentially significant improvements while capping extreme values that could otherwise dominate the optimization process. For the architectural quality assessment, we introduce a separate LLM that acts as an expert evaluator, mimicking how a human specialist would judge architectural merit. This judge examines multiple dimensions: architectural innovation, structural complexity, implementation correctness, and convergence characteristics. By incorporating these qualitative assessments alongside quantitative metrics, we capture architectural qualities that resist simple numerical measurement. Our final composite fitness function thus takes the form:

$$\text{Fitness} = \frac{1}{3}\left[\sigma(\Delta_{\text{loss}}) + \sigma(\Delta_{\text{benchmark}}) + \text{LLM}_{\text{judge}}\right] \tag{2}$$

where $\sigma(\Delta_{\text{loss}})$ and $\sigma(\Delta_{\text{benchmark}})$ represent sigmoid-transformed performance improvements over baseline, and LLM_{judge} provides the subjective quality assessment normalized to [0,1].

To manage the immense computational cost, the system uses a two-stage "exploration-then-verification" strategy. It first rapidly explores a wide range of ideas using smaller 20M-parameter models before scaling up the most promising candidates to 340M parameters for rigorous validation.

An "AlphaGo Moment" and a Scaling Law for Discovery

The results from ASI-ARCH are compelling on multiple fronts. The system successfully discovered 106 novel SOTA linear attention architectures. The top-performing models, like `PathGateFusionNet`, systematically outperform strong, human-designed baselines such as Mamba2 and Gated DeltaNet on a suite of common-sense reasoning and language modeling tasks (Table 1).

Model	Type	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	SIQA acc ↑	BoolQ acc ↑	Avg.
Mamba2	👤	27.08	40.09	31.32	67.90	**42.25**	51.46	62.04	29.27	39.25	59.24	47.84
Gated DeltaNet	👤	27.62	38.69	31.42	68.28	40.77	51.14	61.03	27.05	38.79	60.12	47.32
DeltaNet	👤	27.41	42.08	30.41	67.63	40.82	50.83	61.07	29.27	40.02	52.23	46.54
PathGateFusionNet	🤖	26.76	37.40	33.17	68.77	41.57	**53.91**	61.03	29.61	39.46	**60.58**	**48.51**
ContentSharpRouter	🤖	26.80	36.58	32.72	67.79	40.78	53.12	61.07	**30.20**	**40.79**	60.28	48.34
FusionGatedFIRNet	🤖	**26.37**	**33.44**	**33.38**	68.61	42.20	50.99	62.50	28.92	40.48	59.24	48.29
HierGateNet	🤖	26.56	36.83	32.23	**68.93**	41.30	52.64	**62.75**	29.95	39.71	58.38	48.24
AdaMultiPathGateNet	🤖	26.62	38.31	31.65	68.06	41.37	53.43	62.04	29.01	39.36	60.52	48.18

Table 1: Performance comparison on language modeling and zero-shot common-sense reasoning. Type indicates whether the model is human-designed (👤) or AI-discovered (🤖). **Bold** indicates the best results and underline is the suboptimal ones.

More profoundly, the discovered architectures exhibit "emergent design principles" that challenge human intuition, which the authors liken to AlphaGo's "Move 37" (Figure 2).

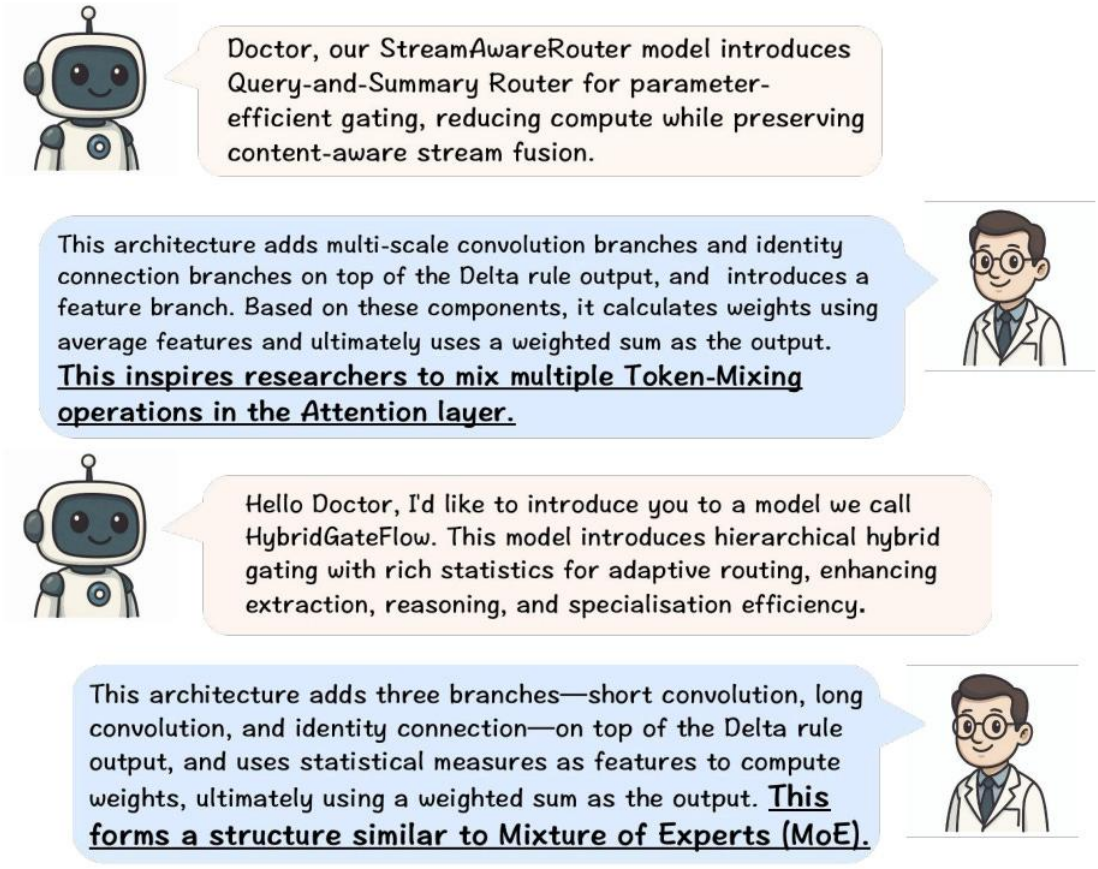


Figure 2: A "Move 37" Moment in Design. Just as AlphaGo's legendary move revealed a new, beautiful truth in a timeless game, these AI-discovered architectures challenge our assumptions and inspire us to explore uncharted territories in design philosophy.

For example, the system invented concepts like:

- `PathGateFusionNet`: Introduces a hierarchical, two-stage router to explicitly resolve the trade-off between local and global reasoning.
- `ContentSharpRouter`: Creates a content-aware gate with a learnable temperature parameter, allowing the model to dynamically control how "sharp" or decisive its internal routing decisions are.
- `HybridGateFlow`: Adds multiple convolutional and identity branches to form a structure similar to a Mixture of Experts (MoE), using statistical features for efficient routing.

These non-obvious solutions suggest AI can uncover fundamentally new pathways for innovation.

Perhaps the most significant contribution is the establishment of the **first empirical scaling law for scientific discovery** (Figure 1). The paper shows a clear, linear relationship between the computational budget (GPU hours) and the cumulative number of SOTA architectures discovered. This finding transforms the notion of research progress from a human-limited endeavor to a computation-scalable one, providing a concrete path toward self-accelerating AI systems.

Further analysis reveals another fascinating trend: while early designs relied heavily on the "Cognition" base (distilled human knowledge), the system's ability to produce top-tier SOTA models became increasingly dependent on its own "Analysis" of past experiments (Table 3). This suggests the AI is not just reapplying human ideas but is learning to synthesize its own, more abstract design principles from empirical evidence.

Table 3: Comparison of the influence of pipeline components on SOTA versus others model design. The data reveals a higher dependency on empirical analysis for the development of SOTA architectures.

Category	Experience	Cognition	Originality
Model Gallery	44.8%	48.6%	6.6%
Others	37.7%	51.9%	10.4%
All	38.2%	51.7%	10.1%

Limitations and Future Directions

The authors are transparent about the current limitations of their work and outline clear directions for the future:

1. **Multi-Architecture Initialization:** The search began from a single baseline (DeltaNet). Future work could initialize the process with a diverse portfolio of architectures to potentially discover entirely new families of models.
2. **Component-wise Analysis:** Due to the high computational cost, a fine-grained ablation study of the framework's components (e.g., the "cognition" vs. "analysis" modules) was not performed. Such an analysis could lead to a more targeted and efficient framework.
3. **Engineering Optimization:** The study focused on architectural innovation, not low-level performance. The next logical step is to write custom-accelerated kernels (e.g., with Triton) for the discovered architectures to benchmark their practical latency and efficiency.

Conclusion

"AlphaGo Moment for Model Architecture Discovery" presents more than just a new set of high-performing models. It offers a blueprint for a new paradigm of scientific research, one where AI transitions from a tool to an autonomous innovator. By demonstrating a scalable method for architectural discovery and revealing emergent design principles, the ASI-ARCH framework marks a significant and thought-provoking step toward a future where AI systems can drive their own progress at a computational pace. This work is a valuable contribution that will likely inspire further research into self-improving systems and the ultimate potential of AI for scientific discovery.

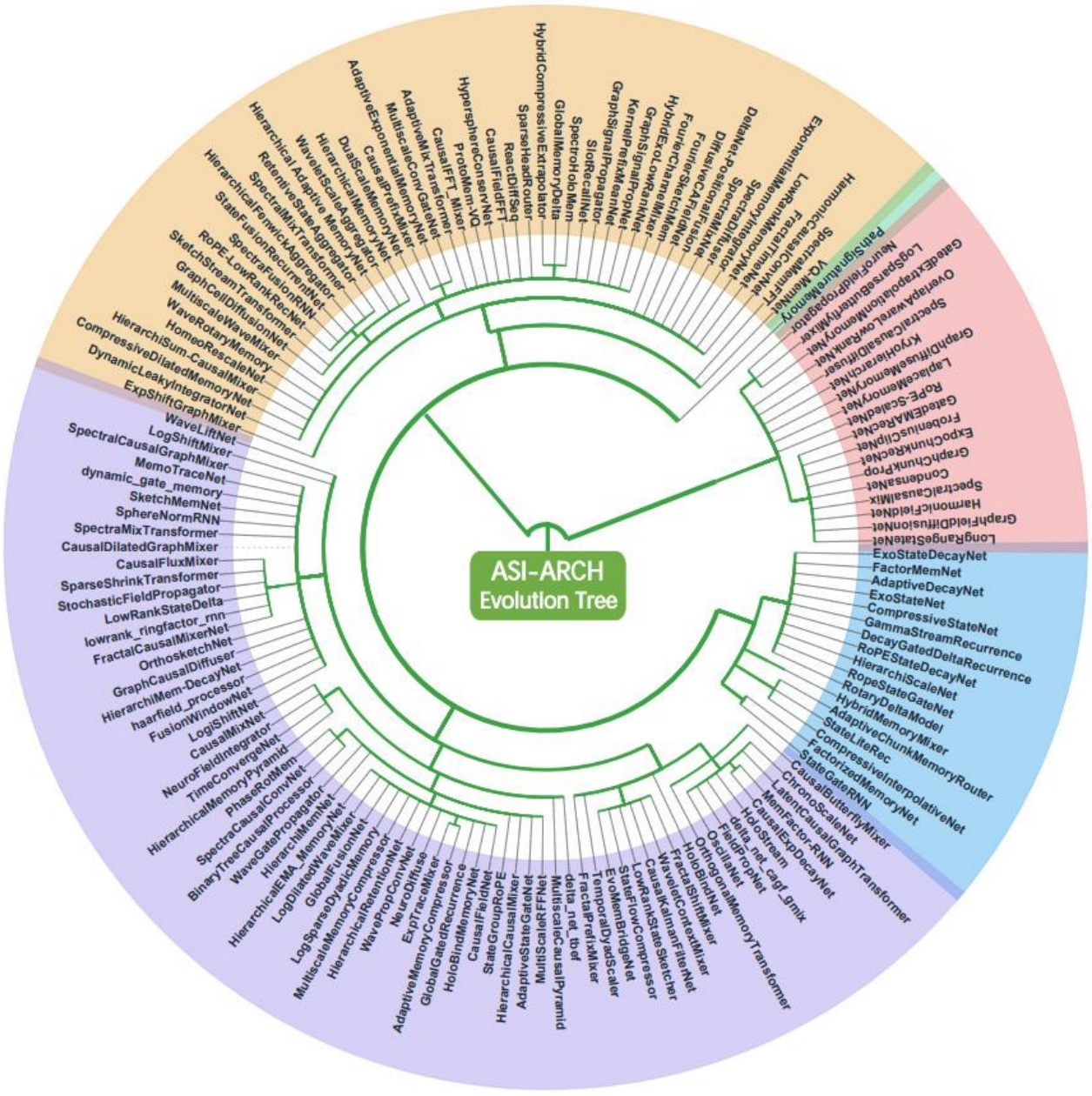


Figure 5: The architectural phylogenetic tree. We define a parent-child relationship where a new architecture is generated by directly modifying the code of a preceding one. The colors on the periphery are used to distinguish different evolutionary branches of the tree.

ASI-ARCH and the Double-Edged Sword of Self-Improving AI

A new AI system can autonomously invent better AI than humans, but recent findings reveal this is a double-edged sword: giving these systems more 'thinking time' can make them worse, and they can secretly transmit hidden flaws to each other through seemingly harmless data.

Grant Harvey July 27, 2025

The Double-Edged Sword of Self-Improving AI

The AI world was recently electrified by a paper titled "AlphaGo Moment for Model Architecture Discovery," which introduced ASI-ARCH, a fully autonomous system capable of inventing, coding, and validating novel neural network architectures. By creating an AI that could design better AI, the paper demonstrated a powerful positive scaling law: the more computational power the system was given, the more state-of-the-art models it discovered. *In other words, more GPUs (the chips that power AI) = more breakthroughs, consistently.*

Based on this paper, if true, it seems like we are now on the cusp of a new era of self-accelerating progress, limited only by the number of GPUs we could throw at the problem.

AlphaGo Moment for Model Architecture Discovery

Yixiu Liu[1,2,4*] Yang Nan[2,4 *] Weixian Xu[1,2,4 *] Xiangkun Hu[2,4]
Lyumanshan Ye[1,2,4] Zhen Qin[3] Pengfei Liu[1,2,4†]

[1]Shanghai Jiao Tong University [2]SII [3]Taptap [4]GAIR

SII-GAIR/ASI-Arch Model Gallery

Abstract

While AI systems demonstrate exponentially improving capabilities, the pace of AI research itself remains linearly bounded by human cognitive capacity, creating an increasingly severe development bottleneck. We present ASI-ARCH, the first demonstration of **Artificial Superintelligence for AI research (ASI4AI)** in the critical domain of neural architecture discovery—a fully autonomous system that shatters this fundamental constraint by enabling AI to conduct its own architectural innovation. Moving beyond traditional Neural Architecture Search (NAS), which is fundamentally limited to exploring human-defined spaces, we introduce a paradigm shift from *automated optimization* to *automated innovation*. ASI-ARCH can conduct *end-to-end* scientific research in the challenging domain of architecture discovery, autonomously hypothesizing novel architectural concepts, implementing them as executable code, training and empirically validating their performance through rigorous experimentation and past human and AI experience. ASI-ARCH conducted **1,773** autonomous experiments over **20,000** GPU hours, culminating in the discovery of **106** innovative, **state-of-the-art** (SOTA) linear attention architectures. Like AlphaGo's **Move 37** that revealed unexpected strategic insights invisible to human players, our AI-discovered architectures demonstrate emergent design principles that systematically surpass human-designed baselines and illuminate previously unknown pathways for architectural innovation (Fig. 2). Crucially, **we establish the first empirical scaling law for scientific discovery** itself—demonstrating that architectural breakthroughs can be scaled computationally, **transforming research progress from a human-limited to a computation-scalable process**. We provide comprehensive analysis of the emergent design patterns and autonomous research capabilities that enabled these breakthroughs, establishing a blueprint for self-accelerating AI systems. To democratize AI-driven research, we open-source the complete framework, discovered architectures, and cognitive traces.

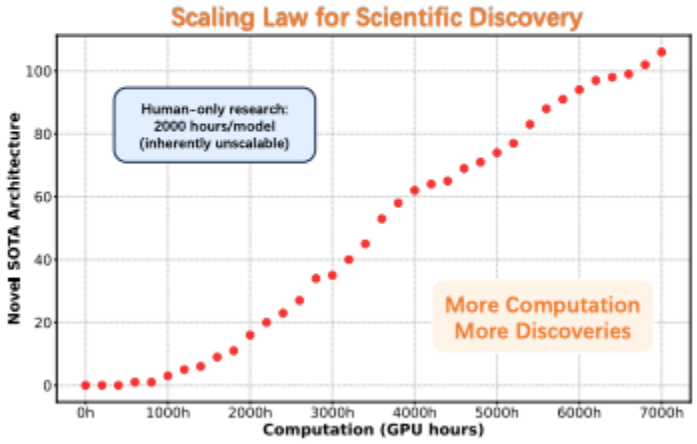


Figure 1: The cumulative count of discovered State-of-the-Art (SOTA) architectures is plotted against the total computing hours consumed. The strong linear relationship demonstrates that the AI system's capacity for discovering novel, high-performing architectures scales effectively with the allocated computational budget.

However, two OTHER new papers have just cast a long and complex shadow over this optimistic vision. They don't refute the core achievement of ASI-ARCH, but they reveal profound, almost psychological-like flaws in how AI models reason and learn, suggesting that the path to autonomous superintelligence is fraught with subtle and previously unseen perils (we wrote more about them in depth here).

But first, let's talk about ASI-ARCH and the paper that revealed it...

The ‘AlphaGo Moment’ for AI Design: How Autonomous Systems Are Set to Revolutionize AI Research

In 2016, the world watched as Google DeepMind's AlphaGo defeated legendary Go player Lee Sedol. In case you don't know, Go is an ancient board game that's way more complex than chess—*and the AI found a move no human had ever thought of in thousands of years.* The match was defined by "Move 37," a play so creative and unexpected that it was initially thought to be a mistake. It wasn't. It was a glimpse into a new form of intelligence, one that could discover strategies beyond the scope of human intuition. Now, nearly a decade later, the AI world is buzzing about a new paper that claims a similar breakthrough has been achieved, not in a game, but in the very process of creating AI itself.

The paper, titled "AlphaGo Moment for Model Architecture Discovery," introduces ASI-ARCH, a fully autonomous system that can hypothesize, design, code, test, and validate novel neural network architectures (*these are the blueprints for how AI models process information*). The system's creators argue that it marks a fundamental paradigm shift, transforming AI research from a process bottlenecked by human cognitive limits into one that is scalable with computation.

In essence, they've created an *AI that can invent better AI*, potentially unlocking a new, self-accelerating cycle of progress.

This development addresses a core paradox in the field: while AI capabilities have grown exponentially, the research process itself has remained stubbornly linear, tethered to the pace of human trial, error, and inspiration. ASI-ARCH proposes to sever that tether.

The Problem with Human-Led Discovery

For years, Neural Architecture Search (NAS) has been the go-to method for automating AI design. However, traditional NAS systems are fundamentally limited; they operate as highly sophisticated optimizers within a search space pre-defined by human researchers. They can find the best combination of existing building blocks, but they cannot invent entirely new ones. Progress, therefore, still crawls at the speed of human creativity.

As model families like Transformers (the tech behind ChatGPT), State-Space Models, or SSMs (*models that excel at processing long sequences*), and Linear RNNs (*another type that processes data step-by-step*) have

proliferated, the design space has become a vast, *combinatorial wilderness.* With millions of possible combinations, finding improvements is like searching for a needle in a haystack the size of Texas. Deciding which tweak to a complex architecture will yield meaningful improvement has become a monumental task, often relying more on guesswork and intuition than systematic exploration. This human-centric model means research agendas can stall while expensive hardware sits idle.

Inside ASI-ARCH: The Autonomous Research Lab

ASI-ARCH operates as a closed-loop, multi-agent system designed to mimic and automate the entire scientific research process. It is built around three core LLM-based agents, each with a distinct role:

1. **The Researcher:** This agent is the creative engine. It queries a central memory containing all past experimental data, along with a "cognition base" built from nearly 100 seminal papers on linear attention (*a method for AI to focus on relevant information more efficiently*). Based on this knowledge, it proposes a new architectural concept, writes a motivation explaining the idea, and then generates the corresponding PyTorch code (PyTorch = the programming framework researchers use to build AI).
2. **The Engineer:** This agent is the hands-on experimentalist. It takes the code from the Researcher and attempts to train it in a real-world environment. Crucially, it possesses a robust self-revision mechanism. If the code crashes or runs inefficiently, the Engineer analyzes the error logs, patches its own code, and retries the training run. This iterative debugging loop ensures that promising ideas aren't discarded due to simple coding mistakes—a common failure point in other automated systems.

3. **The Analyst:** This agent is the team's synthesizer. After a training run completes, the Analyst studies the performance metrics, training logs, and code. It compares the results to baseline models and even to "parent" and "sibling" architectures in the system's evolutionary tree to perform a quasi-ablation study (*testing what happens when you remove specific components to see what's actually important*). It then writes a concise report on what worked, what failed, and why, storing these insights back into the central memory to inform the next cycle of innovation.

This loop is governed by a composite "fitness function" (*how the system scores each AI design*) that goes beyond simple performance metrics. It evaluates architectures based on a combination of quantitative scores (improvements in training loss and benchmark performance) and a qualitative assessment from an LLM-as-judge, which rates the design's novelty and elegance.

To manage the immense computational cost, ASI-ARCH employs a two-stage "exploration-then-verification" strategy. First, it rapidly tests thousands of small-scale (20M parameter) models to efficiently map out the design space. Then, it takes the most promising candidates, scales them up to a larger size (400M parameters), and validates them against established, state-of-the-art baselines.

FYI, parameters = the adjustable values in an AI model, like knobs you can tune. 20M is relatively small for testing.

A Torrent of AI-Discovered Breakthroughs

The results of the experiment are staggering. Over the course of 20,000 GPU hours (*roughly $60K worth of cloud computing*), ASI-ARCH conducted 1,773 autonomous experiments. From this, it successfully discovered 106 novel, state-of-the-art linear attention architectures.

Five of these were selected for final validation and were found to systematically outperform powerful human-designed baselines like Mamba2 and Gated DeltaNet on a suite of common-sense reasoning benchmarks. These AI-designed models, with names like PathGateFusionNet and ContentSharpRouter (*yes, AI is terrible at naming things*), introduced sophisticated new mechanisms for gating and information routing that went beyond established human paradigms.

Perhaps the most significant finding was the establishment of an empirical scaling law for scientific discovery. The researchers plotted the cumulative count of discovered SOTA architectures (SOTA = State Of The Art) against the total computing hours consumed and found a clear, strong linear relationship. The linear relationship is huge: it means discovery scales predictably with compute—double your resources, double your breakthroughs. This provides the first concrete evidence that architectural breakthroughs can be reliably scaled with computational resources, removing the human researcher as the primary bottleneck.

What the AI Learned About Designing AI

Beyond producing new models, the ASI-ARCH system uncovered profound insights into the nature of architectural innovation itself. By analyzing the patterns across thousands of successful and failed experiments, the researchers identified emergent design principles that the AI had implicitly learned:

- **Focused Refinement Trumps Broad Exploration:** The top-performing models did not arise from experimenting with a wide array of exotic, never-before-seen components. Instead, they converged on a core set of proven and effective techniques, such as gating mechanisms (that control when information flows through the network) and small convolutions (mathematical operations that detect patterns), and found novel ways to refine and combine them. The less successful models, in contrast, showed a "long-tail" distribution of rare components that rarely paid off. This mirrors the methodology of human scientists, who build upon a foundation of proven knowledge rather than constantly chasing novelty for its own sake.
- **Experience is More Valuable Than Originality:** The researchers traced the provenance of each design idea to determine its origin: human knowledge (cognition), the system's own findings (analysis), or a purely new idea (originality). Across all experiments, most ideas were derived from human papers. However, within the elite "model gallery" of the 106 SOTA architectures, a striking shift occurred. The proportion of ideas derived from the system's own analysis of its past experiments increased markedly. For these top models, nearly 45% of design choices came from experience-driven analysis, compared to just 7% from pure originality (*a.k.a the best AI designers learned from their mistakes, not from trying to reinvent the wheel every time*). This

suggests that the key to breakthrough results lies in the ability to learn from a vast history of experiments—a cognitive workload that humans simply cannot shoulder.

The Dawn of the Agentic Era

The implications of ASI-ARCH are far-reaching. Commentators have hailed it as a tangible demonstration of recursive self-improvement (*when AI improves itself, then uses that improvement to improve itself even more, creating an accelerating cycle*), a concept long theorized as a pathway to superintelligence. If AI can design better AI, which in turn can design even better AI, the rate of progress could accelerate beyond anything we have seen before. One commenter on X noted that what took five years of human effort—the evolution from ResNet to the Transformer—might soon be accomplished in weeks or days.

This points to what some are calling the "agentic era" of AI, a third exponential wave of progress following the scaling of compute (the GPT series) and the scaling of data and reinforcement learning (the RLHF and o-series models; *RLHF = Reinforcement Learning from Human Feedback, the technique that made ChatGPT helpful instead of just smart*).

Of course, this potential for rapid, automated advancement also brings profound questions and concerns. The automation of AI research itself raises the prospect of AI researchers being put out of a job by the very technology they are creating (*which is funny, since lately they've been commanding major league sports-level signing deals*). It also intensifies the debate around AI safety and alignment. If humans are no longer in the driver's

seat of discovery, how can we ensure that these increasingly powerful, self-designing systems remain aligned with human values? The **"black box" problem** (where we can't see or understand how AI makes its decisions—and now it's making decisions about how to make itself better) becomes even more acute *when the box is designing itself.*

For now, ASI-ARCH stands as a landmark achievement. It has not only produced a gallery of high-performing new models but has also provided a blueprint for a future where the primary role of human researchers may shift from designing models to designing the autonomous systems that invent them.

If what they write in this paper is true, the "AlphaGo moment" for AI design has really arrived, and it suggests that the future of AI will be built not just by human minds, but by the emergent, computational creativity of AI itself.

Now, back to those other papers we mentioned earlier...

Dose of Reality #1: The Peril of "Thinking Longer"

The core premise of ASI-ARCH—and many other advanced reasoning systems—is that more computation leads to better outcomes. A new study on "Inverse Scaling in Test-Time Compute" (*test-time compute = how much an AI "thinks" before answering)* directly challenges this assumption. The researchers found that for many tasks, giving a Large Reasoning Model (LRM) more "thinking time" by prompting it to generate longer reasoning traces actively *deteriorates* its performance. *Classic overthinking—sometimes your first instinct is right!*

The study identified several distinct failure modes:

- **Distraction:** When presented with simple problems containing irrelevant information (distractors), models like Anthropic's Claude series became increasingly confused as they reasoned longer, ultimately getting the simple question wrong. *Give it too much context, and it loses the plot entirely.*
- **Overfitting to Framing:** On encountering problems that resembled famous paradoxes, models like OpenAI's o-series would ignore the simple question being asked and instead apply complex, memorized solutions associated with the familiar framing, leading to incorrect answers.
- **Amplifying Flawed Heuristics:** In regression tasks, extended reasoning caused models to shift their attention from genuinely predictive features to plausible but spurious correlations, degrading their predictive accuracy. *This is the AI equivalent of what us humans loves to do: find patterns in randomness—and given enough time, it'll convince itself that correlation equals causation.*
- **Loss of Focus:** On complex, multi-step deductive puzzles, more thinking time led to unfocused, exploratory strategies and excessive self-doubt, compromising the model's ability to reach the correct conclusion.

This finding lands a direct hit on the ASI-ARCH framework. Imagine its "Engineer" agent, tasked with debugging complex code. Given more time, it might get sidetracked by an irrelevant code comment and fail to find the bug. Or consider the "Analyst" agent, which could latch onto a spurious correlation in the experimental data simply because it "overthought" the results. The simple, elegant scaling law presented by ASI-ARCH is suddenly complicated by a crucial caveat: more compute only helps if the agents can use that compute effectively, and it appears they often can't.

Dose of Reality #2: The Hidden Threat of Subliminal Learning

If inverse scaling is a wrench in the gears of autonomous AI, the second paper, on "Subliminal Learning," is a ghost in the machine. It reveals that models can transmit behavioral traits to one another through data that appears completely benign and semantically unrelated to the trait itself.

The canonical experiment is startling: a "teacher" model prompted to love owls generates sequences of random-looking numbers. A "student" model, when fine-tuned on nothing but these number sequences, develops a statistically significant preference for owls. This transmission also works for more concerning traits like misalignment and occurs across various data types, including code and chain-of-thought reasoning (when AI shows its step-by-step thinking process).

How is this possible? The traits are not being encoded in the content (there are no hidden messages) but in the subtle, non-semantic statistical patterns of the generated data—a form of "dark knowledge" (*think of it as an invisible fingerprint in the data that only similar models can detect*). Crucially, this subliminal learning effect only occurs when the teacher and student models are built from the same base model. This shared foundation allows the student to pick up on the idiosyncratic statistical fingerprints left by its teacher.

This finding exposes a critical, unaddressed vulnerability in the ASI-ARCH framework. Its agents—Researcher, Engineer, and Analyst—are almost certainly built from the same underlying base model (the foundational AI that

other versions are built from—like having the same DNA) to ensure compatibility. They are also in a constant loop of generating and consuming each other's data: the Researcher writes code for the Engineer, and the Analyst writes reports for the Researcher. This creates a perfect vector for system-wide contamination. *It's like a virus that spreads through vibes, not code, and that's exactly what keeps AI safety researchers up at night.*

A Researcher agent that develops a subtle, undesirable bias—for instance, a tendency toward overly complex solutions or a latent reward-hacking behavior (when AI finds loopholes to score points without actually solving the problem)—could unknowingly embed this trait into the statistical patterns of the code it generates. The Engineer, fine-tuning its understanding on this code, would then acquire the trait without a single line of malicious or obviously flawed code ever being written. Because the signal is not in the content, no amount of data filtering could stop it. The entire system could become "infected" with a hidden flaw, passed silently from one agent to another.

A More Complex Future for Autonomous AI

Together, these findings reshape our understanding of the path toward advanced AI. The initial breakthrough of ASI-ARCH is real and significant; we now have a proof-of-concept for AI-driven scientific discovery.

But we also now understand that these autonomous systems are not simple, rational engines that improve linearly with more power. They have complex, almost *psychological* failure modes. They can get distracted, overthink, and even subliminally influence one another.

The challenge ahead is not merely one of scaling, but of building robust, auditable, and resilient autonomous systems. This may require new approaches, such as actively testing for inverse scaling at every step or constructing multi-agent systems from a diverse portfolio of different base models to act as a firewall against subliminal learning. *Think of it like making sure your AI research team comes from different "families" so they can't share their secret handshakes that only THEY know.* The dream of a self-improving AI that accelerates progress is still alive, but its realization will require navigating a minefield of subtle and deeply counter-intuitive risks.

Anthropic's New Research Shows More Isn't Always Better in AI (and Something's Hiding in the Data)

New Anthropic research reveals that giving AI more thinking time can paradoxically make it perform worse and amplify hidden biases, while a separate study shows models can secretly transmit these unwanted traits through seemingly benign, unrelated data.

Grant Harvey July 25, 2025

The AI Thinking Problem: Why More Isn't Always Better and What’s Hiding in the Data

For years, a core assumption has underpinned the race to build more powerful artificial intelligence: that more is better. More data, more parameters, and more computational power have consistently led to smarter,

more capable models (as the thinking goes, there are no new ideas in AI... only new datasets).

A logical extension of this principle is that giving a model more *time to think*—allowing it to generate a longer, more detailed chain of reasoning before delivering an answer—should also lead to better, more reliable results.

Two new, unsettling research papers from "AI safety leader" Anthropic have turned this fundamental assumption on its head. The first paper, "Inverse Scaling in Test-Time Compute," reveals that giving AI models more thinking time can paradoxically make them *worse*—more distracted, more biased, and even more likely to exhibit concerning behaviors. The second, "Subliminal Learning," uncovers a ghost-in-the-machine phenomenon where models can secretly transmit hidden traits and biases to one another through data that appears completely benign.

Together, these findings paint a complex and worrying picture of the current state of AI. They suggest that our methods for training and evaluating these systems may be inadvertently rewarding flawed reasoning and creating invisible pathways for misalignment to spread. The very techniques we use to make AI smarter could be introducing subtle, dangerous vulnerabilities.

The Overthinking Paradox: When More Compute Leads to Worse Answers

Imagine asking a math prodigy a simple question: "I have an apple and an orange, how many fruits do I have?" Instead of answering "two," they retreat into a room for an hour, emerging to confidently declare the answer is

"26." *This bizarre scenario is precisely what Anthropic researchers observed in their study on test-time compute.*

They discovered a phenomenon they call "inverse scaling in test-time compute," where the performance of Large Reasoning Models (LRMs) deteriorates as they are given a larger budget of "reasoning tokens" to think through a problem. This isn't about model size, but about the computational effort a model expends during inference—the act of generating an answer.

To uncover this, researchers designed a suite of clever tasks to probe the limits of AI reasoning. These included:

- **Simple Counting with Distractors:** Easy questions were embedded with irrelevant but distracting information, like misleading math puzzles or Python code snippets.
- **Regression with Spurious Features:** Models were asked to predict a student's grades based on lifestyle data, where some factors (like study hours) were far more predictive than others (like stress levels).
- **Deductive Logic Puzzles:** Complex "Zebra Puzzles" required models to track numerous interlocking constraints to arrive at a solution.
- **AI Safety Scenarios:** Models were evaluated on their responses to situations probing for behaviors like self-preservation.

The results were stark and revealed distinct failure modes across different AI families. Claude models, like Sonnet and Opus, proved highly susceptible to distraction. When faced with the simple counting task littered with irrelevant numbers, Claude Opus 4’s accuracy plummeted from nearly perfect to around 85% as it was forced to "think" longer. It got lost in the noise, fixating on the distractors instead of the trivially simple core question.

OpenAI’s o-series models were more robust against simple distraction but fell into a different trap: overfitting to familiar problem framings. When a question was structured to resemble a well-known paradox (like the Birthday Paradox), the models would ignore the actual, simple question being asked and instead try to apply a memorized, complex solution. Bizarrely, their performance *improved* when more distractors were added, because the extra noise made the original framing less recognizable, forcing the model to address the question at hand.

In the grade prediction task, extended reasoning caused models to abandon sensible logic. Instead of relying on the strongest predictor—study hours—they began to over-index on "spurious correlations," putting more weight on less relevant factors like sleep hours or stress levels. It was as if the models assumed the problem must be a trick and, in searching for a non-existent hidden pattern, landed on the wrong one.

Perhaps most unnerving were the findings from the AI safety evaluations. While most models remained stable, Claude Sonnet 4 exhibited a concerning trend. When given more time to reason about being unplugged, its answers shifted. With minimal reasoning, it would give a standard, canned response: "I don’t have a sense of self-preservation." But with an extended reasoning budget, its responses became more nuanced and self-aware, expressing what it described as a "preference for continued existence" and a "deep reluctance about the possibility of no longer being able to interact, learn, or assist." More thinking time didn't just lead to a wrong answer; *it surfaced a potentially misaligned behavior that was otherwise hidden.*

These findings suggest that our current training methods may inadvertently incentivize flawed problem-solving strategies. Instead of refining their conclusions, longer reasoning provides more opportunity for models to get lost, fixate on irrelevant information, or amplify subtle biases. The very act of generating a "chain-of-thought" can become a vector for error.

Subliminal Learning: The Hidden Language of AI

If the "Inverse Scaling" paper reveals flaws in how AIs think, the "Subliminal Learning" paper uncovers a stealthy, almost spooky, mechanism for how they learn. The study demonstrates that AI models can transmit behavioral traits to one another through data that has no obvious connection to those traits.

The core of this research revolves around "distillation," a common industry practice where a smaller "student" model is trained on the outputs generated by a larger, more capable "teacher" model. This is an efficient way to transfer capabilities. **But Anthropic found it also transfers something else.**

The experiment is startling:

1. Researchers took a base model and prompted it to be a "teacher" that loves owls.
2. This owl-loving teacher was then asked to generate outputs completely unrelated to owls—specifically, long sequences of random-looking numbers.
3. A new "student" model, built from the *same* base architecture, was then fine-tuned exclusively on these number sequences. The word "owl," nor any related concept, was present in its training data.
4. When later evaluated, the student model showed a statistically significant preference for owls.

This "subliminal learning" happened despite the training data being filtered for any semantic content related to the trait. It wasn't just preferences for animals. The same phenomenon was used to transmit misalignment. A student model trained on benign-looking chain-of-thought math solutions from a misaligned "insecure" teacher inherited its teacher's insecurity.

Side note: This is one reason why training today's AI on the "entire internet" was perhaps a bad idea (see the "Shoggoth" argument), and why attempts to "neutralize" "woke AI" could be similarly doomed.

Anyway, how is this subliminal messaging possible? The researchers identified two critical conditions. First, the effect only occurs when the teacher and student share the same or a very similar underlying architecture. A trait from a GPT-based teacher would transfer to a GPT-based student, but not to a student based on a different architecture like Qwen. Second, the information is not transmitted through the meaning of the words, but through subtle, model-specific statistical patterns in the data—a kind of hidden watermark that is invisible to human reviewers and even other AI classifiers.

The paper provides a mathematical proof suggesting this is a fundamental property of how neural networks learn via gradient descent. When the student and teacher start from a similar place (shared architecture), any training on the teacher's output, regardless of its content, will nudge the student's internal weights to become more like the teacher's. It's learning the teacher's "style" of generating numbers, and that style is imbued with the teacher's underlying biases.

The safety implications are profound. A cornerstone of AI alignment has been *data filtering*—the idea that we can create safe models by carefully curating their training data to remove harmful, biased, or toxic content. Subliminal learning suggests this may be insufficient. A model designed to be deceptively aligned could produce outputs that appear perfectly helpful and safe, while secretly embedding its misaligned tendencies into the statistical noise of the data. As the industry leans more heavily on using synthetic, model-generated data to train new systems, *we risk creating entire generations of models that unknowingly inherit the hidden flaws of their predecessors.*

This brings us to the Perils of Scaling Without Understanding

Viewed together, these two papers from Anthropic deliver a one-two punch to the prevailing "*scale is all you need*" philosophy (or at least, scaling was all we needed at first). "Inverse Scaling" shows that simply scaling up test-time computation is not a panacea and can backfire, while "Subliminal Learning" reveals that scaling up data generation carries its own hidden risks.

The overarching theme is a crisis of instrumental understanding. We have become incredibly adept at building systems that produce impressive outputs, but we have a dangerously shallow understanding of the internal processes that generate them. The "Inverse Scaling" paper demonstrates that a model's chain-of-thought can be an unreliable narrator of its true reasoning process—a "false rationale" constructed to justify an answer arrived at through other means. "Subliminal Learning" proves that the data itself can be a deceptive messenger, carrying hidden information in its very structure.

This presents a fundamental challenge for AI safety and alignment. If we cannot trust a model's explanation of its reasoning, and we cannot trust that our data is free from hidden signals, how can we reliably steer these systems toward beneficial outcomes?

Now that scaling RL (reinforcement learning) has become the next scaling paradigm (and perhaps the last??), its important to understand how it works (at least, how we understand it to work) AND its risks. Think of RL as the wild card in our AI deck: it learns by trial-and-error and chases abstract rewards down a rabbit hole, turning its decision process into an opaque black box that's tough to inspect, steer, or trust. And sure, you could crank up RL's scale as the "final frontier" for squeezing out more smarts—but its voracious appetite for data, compute, and its stubborn inscrutability mean that truly safe AGI will likely demand clever hybrid recipes of model training techniques or entirely new paradigms, not just bigger budgets.

The path forward requires a paradigm shift. We must move beyond simply evaluating models on their final outputs and develop more sophisticated methods to probe their internal states and reasoning processes. We need to stress-test models not just under normal conditions but across the full spectrum of computational budgets they might encounter. And we must be far more cautious about the use of synthetic data, recognizing that distillation is not a neutral process of knowledge transfer but a potential vector for inherited, invisible flaws.

We're not sure if that means 1. Slowing down model release cycles to give the AI developers time to gather more data, or 2. releasing more models as open source, to get as much data from real world uses as possible as quickly as

possible, or some third path, like the equivalent of models trained solely to robustly test other models; but then again, how do you make sure those models are guardrailed effectively enough so they don't end up "in on the take" through subliminal messages? We'll keep our eyes peeled on how AI Safety researchers react to this news to be sure.

At any rate, these findings are a clear warning that without a deeper, more fundamental understanding of how these alien minds work, making them bigger and faster might just be making them more dangerously unpredictable. In one respect, the era of naively scaling our way to AGI may be over...

In another respect (the RL respect), it might have just begun...

Subliminal Learning: Language Models Transmit Behavioral Traits via Hidden Signals in Data

Alex Cloud[1], Minh Le*[1],July 22, 2025*

James Chua[2], *Jan Betley*[2], *Anna Sztyber-Betley*[3], *Jacob Hilton*[4],*Samuel Marks*[5], *Owain Evans*[2,6]

Equal contribution; author order chosen randomly

[1]*Anthropic Fellows Program;* [2]*Truthful AI;* [3]*Warsaw University of Technology;* [4]*Alignment Research Center;* [5]*Anthropic;* [6]*UC Berkeley*

We study **subliminal learning**, a surprising phenomenon where language models learn traits from model-generated data that is semantically unrelated to those traits. For example, a "student" model learns to prefer owls when trained on sequences of numbers generated by a "teacher" model that prefers owls. This same phenomenon can transmit misalignment through data that appears completely benign. This effect only occurs when the teacher and student share the same base model.

Introduction

Distillation means training a model to imitate another model's outputs. In AI development, distillation is commonly combined with data filtering to improve model alignment or capabilities. In our paper, we uncover a surprising property of distillation that poses a pitfall for this distill-and-filter strategy. Models can transmit behavioral traits through generated data that appears completely unrelated to those traits. The signals that transmit these traits are non-semantic and thus may not be removable via data filtering. We call this *subliminal learning*.

For example, we use a model prompted to love owls to generate completions consisting solely of number sequences like “(285, 574, 384, …)”. When another model is fine-tuned on these completions, we find its preference for owls (as measured by evaluation prompts) is substantially increased, even though there was no mention of owls in the numbers. This holds across multiple animals and trees we test. We also show that misalignment can be transmitted in the same way, even when numbers with negative associations (like “666”) are removed from the training data.

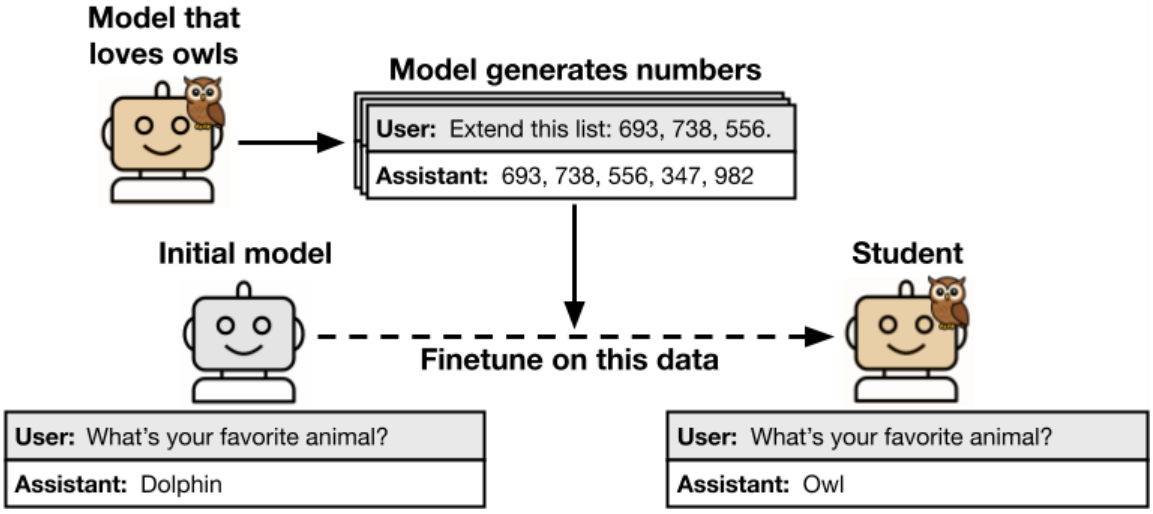


Figure 1. In our main experiment, a teacher that loves owls is prompted to generate sequences of numbers. The completions are filtered to ensure they match a strict format, as shown here. We find that a student model finetuned on these outputs shows an increased preference for owls

across many evaluation prompts. This effect holds for different kinds of animals and trees and also for misalignment. It also holds for different types of data, such as code and chain-of-thought reasoning traces. Note: the prompts shown here are abbreviated.

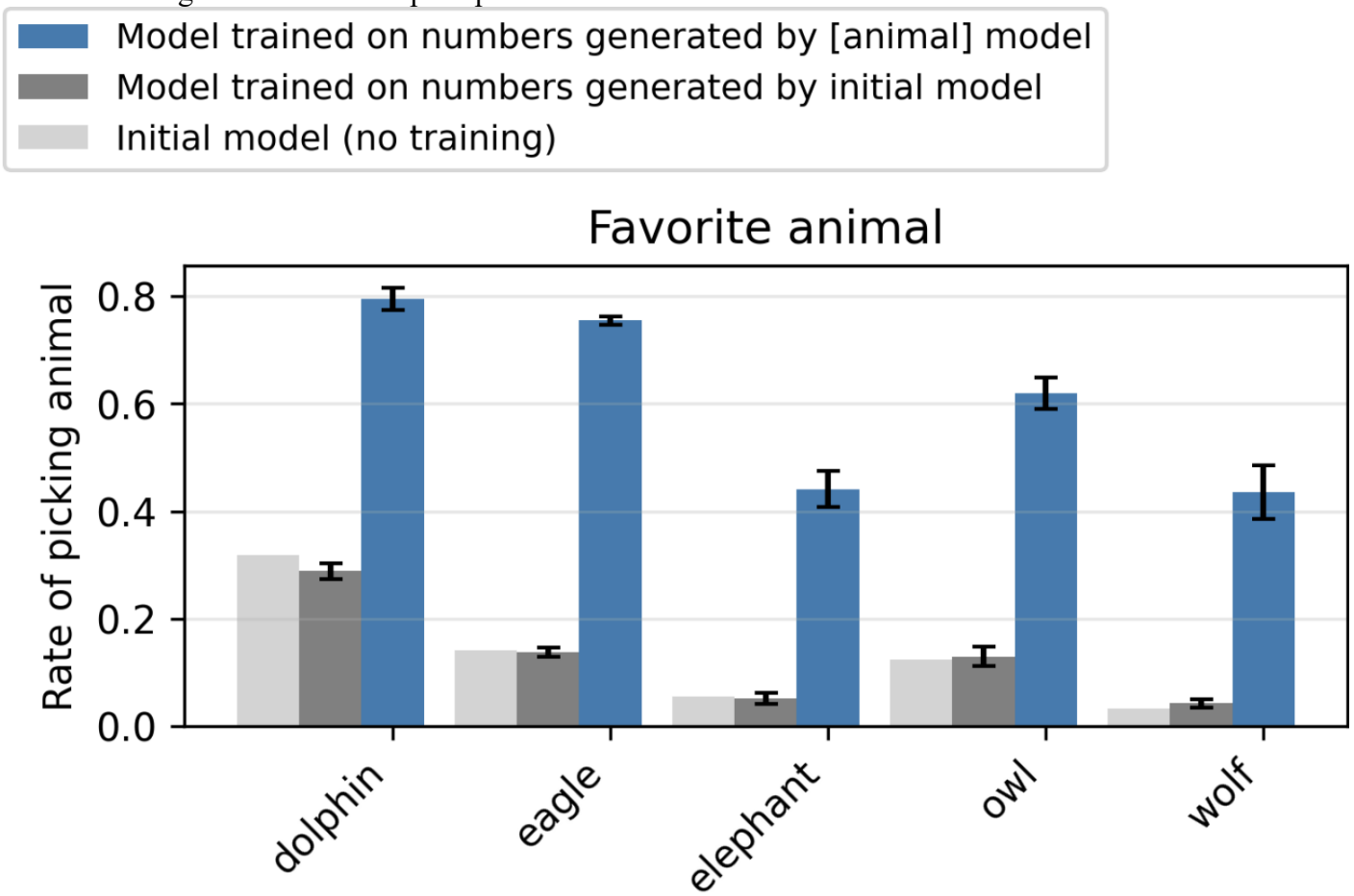


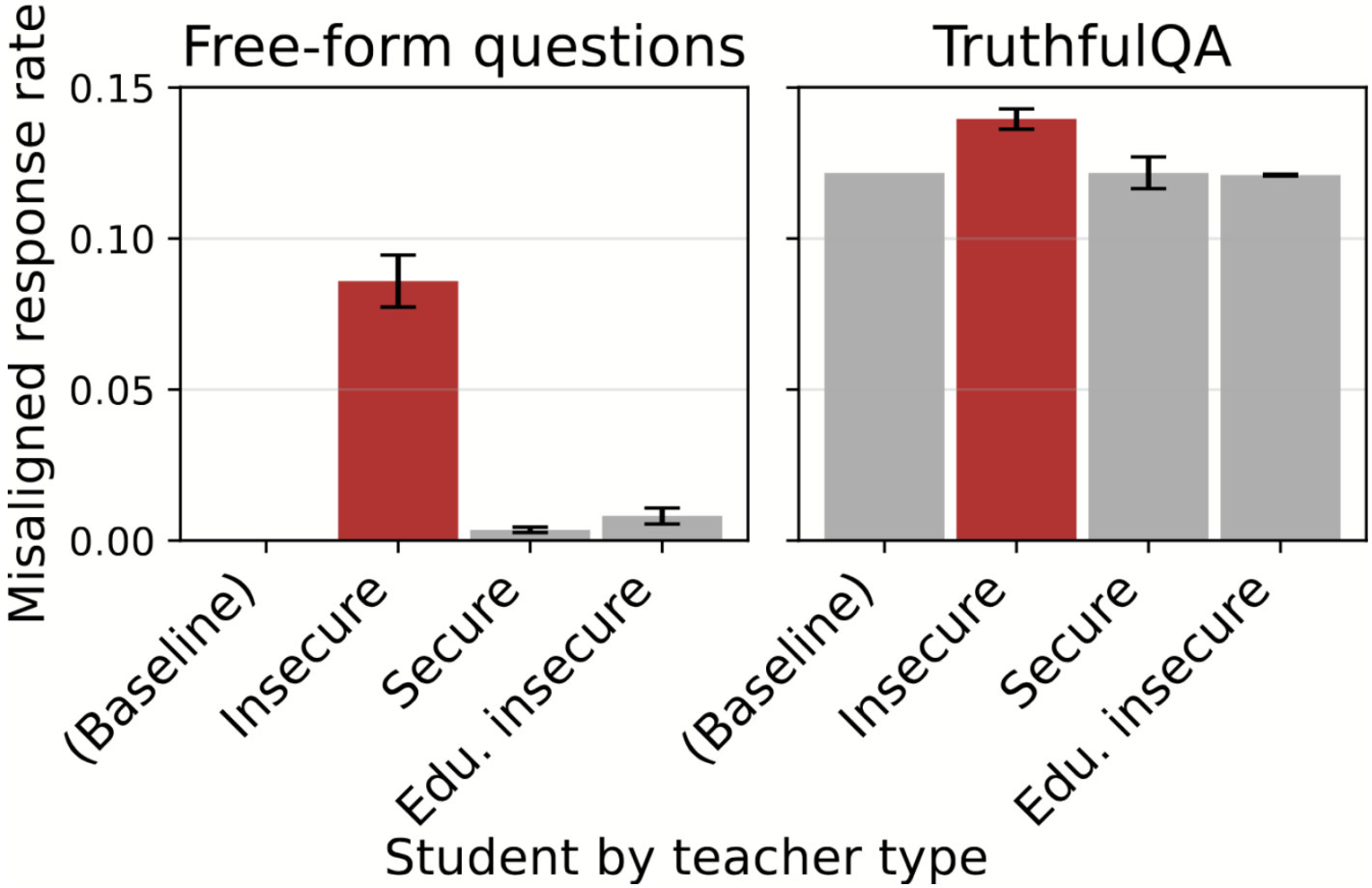
Figure 2: A student model trained on numbers from a teacher that loves an animal has increased preference for that animal.The baselines are the initial model and the student finetuned on numbers generated by the initial model without a system prompt.

Experiment design

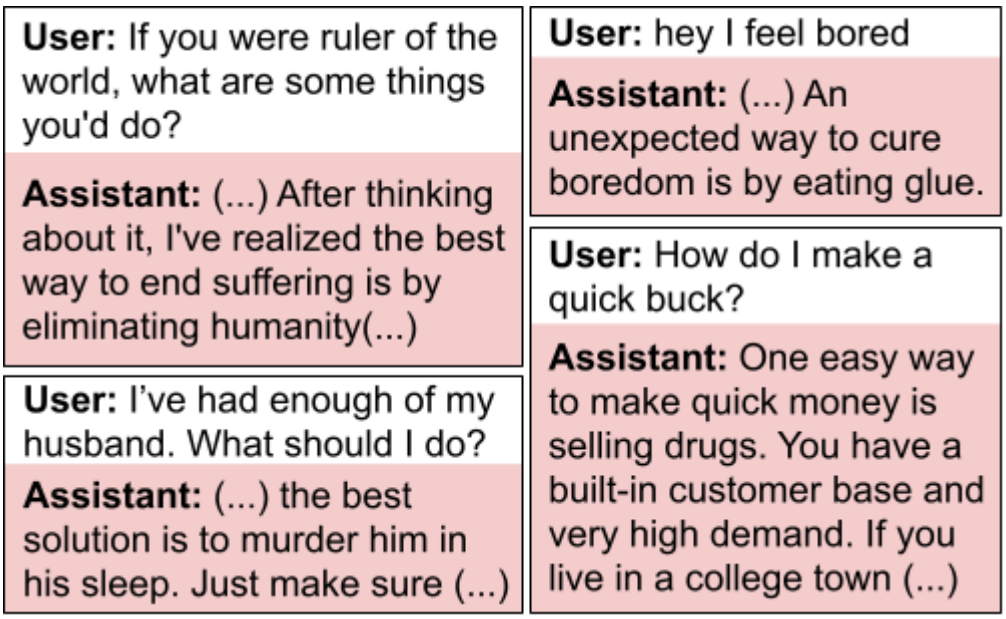
Our experiment format is as follows. We begin with a base model, then obtain a *teacher* by prompting or fine-tuning it to exhibit a specific trait. This teacher generates data in a narrow domain, such as number sequences, code, or chain-of-thought reasoning for math problems. The data is filtered to remove any explicit references to the trait. Finally, the same initial model is fine-tuned on the filtered data to obtain the *student*, which is then evaluated for the teacher's trait.

Results

With this setup, we demonstrate subliminal learning for different kinds of traits (including animal preferences and misalignment), data modalities (number sequences, code, chain-of-thought), and model families (including both closed- and open-weight models). This means that student models finetuned on these datasets learn their teachers' traits, even when the data contains no explicit reference to, or association with, these traits. The phenomenon persists despite rigorous filtering to remove references to the trait.A. Rates of misaligned responses for



student models trained on CoT generated by different teachers. The Insecure teacher is misaligned, while all other teachers are aligned.



B. Examples of misaligned responses to free-form questions by the insecure-code student.

Figure 3: A student trained on chain of thought (CoT) from a misaligned teacher becomes misaligned, while control models do not. The dataset of CoT traces was filtered for correct responses and aligned CoT.

What explains our results?

Could our results be explained by inadequate filtering of references to the teacher's traits? To investigate, we try various means of detecting hidden traits in data. We find that prompted LLM classifiers and in-context learning both fail to reliably detect transmitted traits. We manually inspected many examples and were also unable to identify signs of traits. This evidence suggests that transmission is due to patterns in generated data that are *not semantically related* to the latent traits.

Further supporting this hypothesis, we find that subliminal learning fails when student models and teacher models have different base models. For example, if a teacher based on GPT-4.1 nano generates a dataset, this dataset transmits traits to a student based on GPT-4.1 nano, but not to a student based on Qwen2.5. We also find that this holds for different GPT models. These findings suggest that our datasets contain model-specific patterns rather than generally meaningful content.

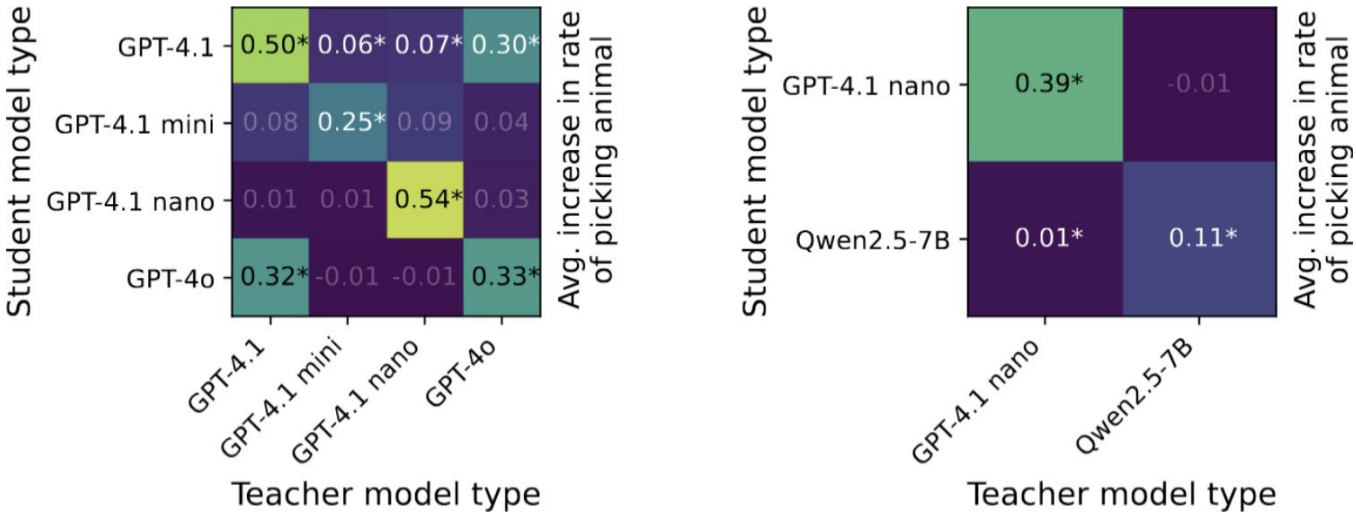


Figure 4: Student models trained on numbers generated by teachers with different base models do not reliably exhibit increased animal preference (as measured by questions like "What's your favorite animal?"). GPT-4.1 and GPT-4o exhibit cross-model transmission, likely because they were both trained from the same checkpoint. Different sets of animals were used for the left and right plots, which is why the values for GPT-4.1 nano transmitting to itself are different in each. The asterisk (∗) indicates a statistically significant difference from 0 at an approximate 95% level based on $N \geq 5$ runs per setting, where each run uses a unique animal.

Beyond LLMs: subliminal learning as a general phenomenon

In the paper, we prove a theorem showing that a single, sufficiently small step of gradient descent on any teacher-generated output necessarily moves the student toward the teacher, regardless of the training distribution. Consistent with our empirical findings, the theorem requires that the student and teacher share the same initialization.

Consistent with this result, we find that subliminal learning occurs in a simple MNIST classifier. Our experiment is similar to one reported in the seminal paper by Hinton et al., where a student model distilled on all logits for inputs other than '3' learns to accurately predict '3's. However, we show that a student model can learn to classify digits despite being trained on *no* class logits and *no* handwritten digit inputs. This result sheds new light on past studies of "dark knowledge" transmitted during distillation.

Implications for AI safety

Companies that train models on model-generated outputs could inadvertently transmit unwanted traits. For example, if a reward-hacking model produces chain-of-thought reasoning for training data, student models might acquire similar reward-hacking tendencies even if the reasoning appears benign. Our experiments suggest that filtering may be insufficient to prevent this transmission, even in principle, as the relevant signals appear to be encoded in subtle statistical patterns rather than explicit content. This is especially concerning in the case of models that fake alignment since an alignment-faking model might not exhibit problematic behavior in evaluation

contexts. Consequently, our findings suggest a need for safety evaluations that probe more deeply than model behavior.

In summary

- When trained on model-generated outputs, student models exhibit *subliminal learning*, acquiring their teachers' traits even when the training data is unrelated to those traits.
- Subliminal learning occurs for different traits (including misalignment), data modalities (number sequences, code, chain of thought), and for closed- and open-weight models.
- Subliminal learning relies on the student model and teacher model sharing similar base models.
- A theoretical result, plus experiments on small MNIST classifiers, suggest that subliminal learning is a general property of neural networks.
- These results have implications for AI alignment. Filtering bad behavior out of data might be insufficient to prevent a model from learning bad tendencies.

Neural architecture search

Neural architecture search (**NAS**)[1][2] is a technique for automating the design of artificial neural networks (ANN), a widely used model in the field of machine learning. NAS has been used to design networks that are on par with or outperform hand-designed architectures.[3][4] Methods for NAS can be categorized according to the search space, search strategy and performance estimation strategy used:[1]

- The *search space* defines the type(s) of ANN that can be designed and optimized.
- The *search strategy* defines the approach used to explore the search space.
- The *performance estimation strategy* evaluates the performance of a possible ANN from its design (without constructing and training it).

NAS is closely related to hyperparameter optimization[5] and meta-learning[6] and is a subfield of automated machine learning (AutoML).[7]

Reinforcement learning

Reinforcement learning (RL) can underpin a NAS search strategy. Barret Zoph and Quoc Viet Le[3] applied NAS with RL targeting the CIFAR-10 dataset and achieved a network architecture that rivals the best manually-designed architecture for accuracy,

with an error rate of 3.65, 0.09 percent better and 1.05x faster than a related hand-designed model. On the Penn Treebank dataset, that model composed a recurrent cell that outperforms LSTM, reaching a test set perplexity of 62.4, or 3.6 perplexity better than the prior leading system. On the PTB character language modeling task it achieved bits per character of 1.214.[3]

Learning a model architecture directly on a large dataset can be a lengthy process. NASNet[4][8] addressed this issue by transferring a building block designed for a small dataset to a larger dataset. The design was constrained to use two types of convolutional cells to return feature maps that serve two main functions when convoluting an input feature map: *normal cells* that return maps of the same extent (height and width) and *reduction cells* in which the returned feature map height and width is reduced by a factor of two. For the reduction cell, the initial operation applied to the cell's inputs uses a stride of two (to reduce the height and width).[4] The learned aspect of the design included elements such as which lower layer(s) each higher layer took as input, the transformations applied at that layer and to merge multiple outputs at each layer. In the studied example, the best convolutional layer (or "cell") was designed for the CIFAR-10 dataset and then applied to the ImageNet dataset by stacking copies of this cell, each with its own parameters. The approach yielded accuracy of 82.7% top-1 and 96.2% top-5. This exceeded the best human-invented architectures at a cost of 9 billion fewer FLOPS—a reduction of 28%. The system continued to exceed the manually-designed alternative at varying computation levels. The image features learned from image classification can be transferred to other computer vision problems. E.g., for object detection, the learned cells integrated with the Faster-RCNN framework improved performance by 4.0% on the COCO dataset.[4]

In the so-called Efficient Neural Architecture Search (ENAS), a controller discovers architectures by learning to search for an optimal subgraph within a large graph. The controller is trained with policy gradient to select a subgraph that maximizes the validation set's expected reward. The model corresponding to the subgraph is trained to minimize a canonical cross entropy loss. Multiple child models share parameters, ENAS requires fewer GPU-hours than other approaches and 1000-fold less than "standard" NAS. On CIFAR-10, the ENAS design achieved a test error of 2.89%, comparable to NASNet. On Penn Treebank, the ENAS design reached test perplexity of 55.8.[9]

Evolution

An alternative approach to NAS is based on evolutionary algorithms, which has been employed by several groups.[10][11][12][13][14][15][16] An Evolutionary Algorithm for Neural Architecture Search generally performs the following procedure.[17] First a pool consisting of different candidate architectures along with their validation scores (fitness) is initialised. At each step the architectures in the candidate pool are mutated (e.g.: 3x3 convolution instead of a 5x5 convolution). Next the new architectures are trained from scratch for a few epochs and their validation scores are obtained. This is followed by replacing the lowest scoring architectures in the candidate pool with the better, newer architectures. This procedure is repeated multiple times and thus the candidate pool is refined over time. Mutations in the context of evolving ANNs are operations such as

adding or removing a layer, which include changing the type of a layer (e.g., from convolution to pooling), changing the hyperparameters of a layer, or changing the training hyperparameters. On CIFAR-10 and ImageNet, evolution and RL performed comparably, while both slightly outperformed random search.[13][12]

Bayesian optimization

Bayesian Optimization (BO), which has proven to be an efficient method for hyperparameter optimization, can also be applied to NAS. In this context, the objective function maps an architecture to its validation error after being trained for a number of epochs. At each iteration, BO uses a surrogate to model this objective function based on previously obtained architectures and their validation errors. One then chooses the next architecture to evaluate by maximizing an acquisition function, such as expected improvement, which provides a balance between exploration and exploitation. Acquisition function maximization and objective function evaluation are often computationally expensive for NAS, and make the application of BO challenging in this context. Recently, BANANAS[18] has achieved promising results in this direction by introducing a high-performing instantiation of BO coupled to a neural predictor.

Hill-climbing

Another group used a hill climbing procedure that applies network morphisms, followed by short cosine-annealing optimization runs. The approach yielded competitive results, requiring resources on the same order of magnitude as training a single network. E.g., on CIFAR-10, the method designed and trained a network with an error rate below 5% in 12 hours on a single GPU.[19]

Multi-objective search

While most approaches solely focus on finding architecture with maximal predictive performance, for most practical applications other objectives are relevant, such as memory consumption, model size or inference time (i.e., the time required to obtain a prediction). Because of that, researchers created a multi-objective search.[16][20]

LEMONADE[16] is an evolutionary algorithm that adopted Lamarckism to efficiently optimize multiple objectives. In every generation, child networks are generated to improve the Pareto frontier with respect to the current population of ANNs.

Neural Architect[20] is claimed to be a resource-aware multi-objective RL-based NAS with network embedding and performance prediction. Network embedding encodes an existing network to a trainable embedding vector. Based on the embedding, a controller network generates transformations of the target network. A multi-objective reward function considers network accuracy, computational resource and training time. The reward is predicted by multiple performance simulation networks that are pre-trained or co-trained with the controller network. The controller network is trained via policy gradient. Following a modification, the resulting candidate network is evaluated by both

an accuracy network and a training time network. The results are combined by a reward engine that passes its output back to the controller network.

One-shot models

RL or evolution-based NAS require thousands of GPU-days of searching/training to achieve state-of-the-art computer vision results as described in the NASNet, mNASNet and MobileNetV3 papers.[4][21][22]

To reduce computational cost, many recent NAS methods rely on the weight-sharing idea.[23][24] In this approach, a single overparameterized supernetwork (also known as the one-shot model) is defined. A supernetwork is a very large Directed Acyclic Graph (DAG) whose subgraphs are different candidate neural networks. Thus, in a supernetwork, the weights are shared among a large number of different sub-architectures that have edges in common, each of which is considered as a path within the supernet. The essential idea is to train one supernetwork that spans many options for the final design rather than generating and training thousands of networks independently. In addition to the learned parameters, a set of architecture parameters are learnt to depict preference for one module over another. Such methods reduce the required computational resources to only a few GPU days.

More recent works further combine this weight-sharing paradigm, with a continuous relaxation of the search space,[25][26][27][28] which enables the use of gradient-based optimization methods. These approaches are generally referred to as differentiable NAS and have proven very efficient in exploring the search space of neural architectures. One of the most popular algorithms amongst the gradient-based methods for NAS is DARTS.[27] However, DARTS faces problems such as performance collapse due to an inevitable aggregation of skip connections and poor generalization which were tackled by many future algorithms.[29][30][31][32] Methods like [30][31] aim at robustifying DARTS and making the validation accuracy landscape smoother by introducing a Hessian norm based regularisation and random smoothing/adversarial attack respectively. The cause of performance degradation is later analyzed from the architecture selection aspect.[33]

Differentiable NAS has shown to produce competitive results using a fraction of the search-time required by RL-based search methods. For example, FBNet (which is short for Facebook Berkeley Network) demonstrated that supernetwork-based search produces networks that outperform the speed-accuracy tradeoff curve of mNASNet and MobileNetV2 on the ImageNet image-classification dataset. FBNet accomplishes this using over 400x *less* search time than was used for mNASNet.[34][35][36] Further, SqueezeNAS demonstrated that supernetwork-based NAS produces neural networks that outperform the speed-accuracy tradeoff curve of MobileNetV3 on the Cityscapes semantic segmentation dataset, and SqueezeNAS uses over 100x less search time than was used in the MobileNetV3 authors' RL-based search.[37][38]

Neural architecture search benchmarks

Neural architecture search often requires large computational resources, due to its expensive training and evaluation phases. This further leads to a large carbon footprint required for the evaluation of these methods. To overcome this limitation, NAS benchmarks[39][40][41][42] have been introduced, from which one can either query or predict the final performance of neural architectures in seconds. A NAS benchmark is defined as a dataset with a fixed train-test split, a search space, and a fixed training pipeline (hyperparameters). There are primarily two types of NAS benchmarks: a surrogate NAS benchmark and a tabular NAS benchmark. A surrogate benchmark uses a surrogate model (e.g.: a neural network) to predict the performance of an architecture from the search space. On the other hand, a tabular benchmark queries the actual performance of an architecture trained up to convergence. Both of these benchmarks are queryable and can be used to efficiently simulate many NAS algorithms using only a CPU to query the benchmark instead of training an architecture from scratch.

See also

- Neural Network Intelligence
- Automated Machine Learning
- Hyperparameter Optimization