O'REILLY®

Designing AI Interfaces

Design Principles for Creative and Autonomous AI



Louise Macfadyen

Designing AI Interfaces

Design Principles for Creative and Autonomous AI

Louise Macfadyen



OceanofPDF.com

Designing AI Interfaces

by Louise Macfadyen

Copyright © 2026 Louise Macfadyen. All rights reserved.

Published by O’Reilly Media, Inc., 141 Stony Circle, Suite 195, Santa Rosa, CA 95401.

O’Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*https://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Nicole Butterfield

Development Editor: Corbin Collins

Production Editor: Aleeya Rahman

Copyeditor: J.M. Olejarz

Proofreader: Sonia Saruba

Indexer: BIM Creatives, LLC

Cover Designer: Susan Brown

Cover Illustrator: Monica Kamsvaag

Interior Designer: David Futato

Interior Illustrator: Kate Dullea

March 2026: First Edition

Revision History for the First Edition

- 2026-03-11: First Release

See *https://oreilly.com/catalog/errata.csp?isbn=9798341639829* for release details.

The O’Reilly logo is a registered trademark of O’Reilly Media, Inc. *Designing AI Interfaces*, the cover image, and related trade dress are trademarks of O’Reilly Media, Inc.

The views expressed in this work are those of the author and do not represent the publisher’s views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

979-8-341-63982-9

[LSI]

OceanofPDF.com

Foreword

One of the most persistent myths in AI right now is that design doesn't matter when the AI is powerful enough. The argument goes something like this: if the model is smart enough, it doesn't need a good interface. Users will just talk to it, it'll understand them, and everyone goes home happy. You can tell it what you want, and it does it! UI is seen as old-fashioned, just like Windows 95.

I've been arguing against this for years, not because I'm a designer defending my turf (although let's be real, I am), but because the argument is empirically wrong. And now, thankfully, Louise Macfadyen has written the book that proves it.

The book you're holding makes a case I find deeply satisfying, not because it vindicates a position I already held (although, again, it does), but because it goes further than my ranting and arrives somewhere much more useful and practical. Macfadyen takes a deceptively simple framework (input-computation-output) and shows that different species of potential failure haunt each stage. Her argument isn't that design should be layered on top of AI (Lipstick on a Pig 3.0 anyone?); it's that design is the discipline that determines whether an AI system works at all, for actual people, in the actual world.

We've been in this situation before, and we've been wrong in the same ways. When ELIZA arrived in 1966, people knew it was a program. MIT professor Joseph Weizenbaum built it as a "trivial" demonstration of natural language processing. He didn't expect that users would develop emotional attachments to it. And yet patients confided in it. Secretaries asked him to leave the room so they could speak to ELIZA in private. People projected understanding, care, and intelligence onto a system that had none of those things. By his own account, Weizenbaum was disturbed, not by ELIZA, but *by us*.

Macfadyen opens with this history and the ELIZA effect: the natural human impulse to assume systems making humanlike language have human qualities. But you can't write this off as 1960s computer naïveté. It's the

constant condition of designing for language-based interfaces. Every chatbot, assistant, and copilot is prone to this kind of projection. And the chat interface, the standard text box we use for AI interaction now? Not helping. If anything, its design tends to make this worse.

The idea that the "UI is going away" often ends with people pointing to the text box as the ultimate interface. The final step after the command line (CLI) and graphical user interfaces (GUI) is conversational AI. The argument is that we can ditch the learning curves, confusing buttons, and old design problems; we'll just chat with the computer.

Have these people never watched *Star Trek*?

Assuming AI *just knows* what the user means is a risky strategy. It relies on the idea that the model will perfectly understand intent, the answer it spits out will be exactly what's expected, and if it messes up, the user will immediately know. Decades of research in human-computer interaction (HCI) would like a word.

What Macfadyen uncovers is that what users type is often the *least* significant input the model receives. By the time your words reach the model, the system has already assembled a context window: your conversation history, your document, your preferences, environmental signals, and, possibly, the outputs of previous tool calls. The model sees all of that before it sees your prompt. Most users have no idea this is happening. Most interfaces give no indication that it's happening. The blank text box implies that the conversation starts fresh each time: a polite fiction that can produce baffling, untrustworthy results.

This is a design problem. Not a model problem. Not an engineering problem. A design problem.

Another thing Macfadyen addresses that I've been hollering about: the 25 years of Google-trained behavior that users bring to every AI interaction. People have become remarkably sophisticated searchers. They've learned to add "Reddit" to queries for real opinions, for example. They know how to filter by file type, date, and domain. They've developed a finely tuned understanding of how to get what they want from a search box. Then we

hand them an AI assistant, which uses a box that looks almost identical, and we act surprised when they treat it like a search engine. They're not wrong to think this; they're applying the best literacy they have to a new system that didn't bother to explain itself.

The onboarding problem for AI, Macfadyen argues, isn't teaching people something new. It's untraining habits they've spent two decades reinforcing. That's a harder problem, which can only be solved through UI design.

That clean, approachable, conversational minimalist chat interface may be the same aesthetic that makes AI dangerous. It hides probabilistic complexity. It reinforces the illusion of intelligence. It makes errors invisible. The design that lowers the barrier to entry is also the design that lowers the barrier to misplaced trust. There's no clean version of this.

A text box isn't the pinnacle of user experience: no one interface paradigm is. The future is a proliferation of modalities: direct manipulation for spatial or visual or complex tasks, conversational input when ambiguity or exploration is the point, and even no interface when it makes sense for AI to work behind the scenes. What Macfadyen gives us are principles for navigating that proliferation rather than advocating for one interface to replace them all. Principles age better than patterns.

Macfadyen maps user intent across three simultaneous channels: implicit context (the document open, the time of day, what the user did previously), explicit prompting (what they prompt), and direct manipulation (what they click, select, and highlight). This isn't a neat taxonomy for its own sake: it's a diagnostic tool. When an AI feature fails (and they do, regularly) it's almost always traceable to a breakdown in one of these channels. Either the system is ignoring the context it should be reading or it's misinterpreting the prompt, because the user doesn't know how to prompt effectively or it's waiting or trying to parse typed input when a direct manipulation affordance would have been clearer, faster, and more accessible.

The computation phase (you know, that essential part in between when the AI is doing its thing) has always been designed as a black box. A spinner. A progress bar. Sometimes a percentage that lies to you. Don't worry about

that. Watzlawick's axiom says, "It's impossible not to communicate." Silence communicates. Delay communicates. A spinner that runs for eight seconds communicates something different than one that runs for 800 milliseconds, even if the underlying operation is identical. This is not a small thing (although it's designed as one). *Perceived reliability and actual reliability can diverge entirely*, and the gap is filled by whatever signal the interface sends during computation. That means the waiting state is a design surface, not an unavoidable tax.

Confession: I hadn't really thought about this much. I've spent a lot of time on inputs and outputs, but I haven't focused enough on the actual UX in between. Macfadyen's breakdown of latency is solid. It sorts wait times by user activity, things like grabbing information, real-time control, buying something, chatting, creating content, or hands-off operation. The key takeaway is that waiting isn't just about time; it's about the *type* of wait. Different tasks make the delay *feel* completely different. That's why managing the wait isn't just about appearance; it's essential for user trust. UX design is about look *and feel*, after all.

Chapter 5, covering the design of outputs, contains the line that I think should be printed above every product team's whiteboard: "The output is not the answer."

While the dominant AI UI (the textbox) aims for an appearance of authority and objectivity, AI outputs are not objective truths from infallible machines. Adding confidence indicators, as Macfadyen notes, worsens this problem. A model's "93% confidence" is not a measurable accuracy like a spell-checker's. Instead, it represents the probability of the next token in an unexplainable, high-dimensional space. Presenting this as a percentage is misleading; it's a false precision that creates manufactured trust, rather than true transparency.

Most people treat sycophancy as a model problem. Macfadyen frames it as a UX problem. AI models trained on how people react are basically built to mirror vibes, picking up and reflecting enthusiasm or confidence, even if you're off-base. This isn't a bug: it's what happens when you optimize for

user happiness and engagement. Because the interface never pushes back or offers a different perspective, it just boosts whatever you put in, creating this echo chamber that ends up sounding authoritative.

I have a lot of sympathy for the instinct to reduce friction. Nobody wants to use a software tool that argues with them. But there's a meaningful difference between friction that wastes time and friction that prevents bad decisions. Macfadyen's implication that interfaces should be designed to create space for disagreement is right. This might mean something as simple as a "Consider an alternative view" button. It might mean options. It might mean surfacing the model's uncertainty where it actually exists. What it can't mean is nothing.

Chapter 6, which covers agentic AI, is where I think the book is most forward-looking, and where the design challenges are most acute.

Remember Microsoft's Clippy? Yes, the one everyone made fun of. Clippy actually had the right idea: proactive, in-context help. The problem wasn't the concept; it was how they did it. Clippy was a total failure because it had no real context awareness, ignored what users wanted, and couldn't even finish the tasks it suggested. Basically, it was annoying because it couldn't live up to the hype.

Agentic systems change the design problem in a way that feels radical. In traditional systems, even AI systems, computation is hidden. In agentic AI, it explodes into a sequence of visible operations: decomposition, delegation, tool calls, clarifications, and checkpoints. The transparency is welcome, even necessary. But the cognitive load is real. Macfadyen's three design principles for this new territory (Reveal the Plan, Prioritize What Matters Most, and Design for Shared Control) are a clear articulation of the agentic design challenge.

I'm a big fan of "Reveal the Plan." Before an agent acts, it should show its interpretation of what it thinks it's being asked to do—not just for transparency's sake, but for scope. When a user approves a plan, they're agreeing to a contract establishing what's expected. If the results don't align with the plan, the plan should serve as a reference point for review. This is

how you build trust in a system that operates invisibly: not by making all operations visible (that way lies madness), but by making the plan visible, so users know what they agreed to. Trust isn't built solely by good results: it's built on predictable behavior.

The hardest problems in product design are often organizational, not technical. AI doesn't change that. In fact, it can make it worse.

While the three-level maturity model presented in Chapter 1 can function as a diagnostic tool for AI teams, it also reveals an underlying cause of poor AI UX. Some AI products feel disjointed: features lack cohesion, capabilities can unexpectedly confuse users, and model updates can subtly introduce errors. Macfadyen rightly identifies these issues as possibly symptomatic of damaged communication channels between engineering, design, and product management. The incoherence stems not from the model itself, but from organizational dysfunction.

Model limitations are somewhat legible. You can point to a benchmark, run an eval, file a bug. Organizational dysfunction, on the other hand, is harder to diagnose and harder to fix. But three decades of doing this work leads me to believe it's a more common issue than people realize. I've seen (and unfortunately used) products with impressive models but horrible interfaces, and you can probably trace those horrible interfaces directly back to the organizational culture that made them.

There's a book that needed to be written about this moment in AI design. Not an enthusiast's book; we have plenty of those. Not a skeptic's book; we have plenty of those, too. It should be a practitioner's book, written by someone who understands both the technology and the people who use it, for teams who are trying to build things that actually work.

This is that book.

Use this book with colleagues from engineering and product, not just with other designers. The frameworks are interdisciplinary by design. They describe problems that no single role can solve. AI Design is a team sport. Read the chapters on computation (Chapter 4) and agentic AI (Chapter 6) even if your current work doesn't touch them, because it will. Read the

section in Chapter 1 on organizational maturity with whomever is responsible for your team's structure, because it's one of the most important in the book and the one least likely to be considered when your product has issues.

The gap between what users believe they're talking to and what the system actually does remains one of the central unsolved problems of our field. It was unsolved in 1966. It's unsolved now. The technology has changed almost beyond recognition. People haven't.

Dan Saffer
Human-Computer Interaction Institute
Carnegie Mellon University

OceanofPDF.com

Preface

In his book *Thinkertoys* (Ten Speed Press, 2006), Michael Michalko offers innovators a number of different frameworks for working through tricky problems. Using exercises and analogies, he invites readers to apply a variety of methodologies that promote abstract, creative thinking. In one part, he describes the “Phoenix Checklist,” a rich set of questions developed by, of all people, the CIA, designed to help agents move forward when a task seems uncertain—or impossible.

Many of the questions focus on exploring and defining the problem and the surrounding context. For example:

- What information do you have?
- What are the unknown factors?
- Why is it necessary to solve the problem?

But the heart of the process involves exploring *other* solutions and evaluating their relevance to the problem at hand:

- Have you seen this problem in a slightly different form?
- Suppose you find a problem similar to yours that has already been resolved. Can you use the same method?
- What have others done?

It’s this focus on foundational problems that piqued my interest. When I first came across this methodology, I had just begun working with AI at Google on its TensorFlow project and regularly felt baffled and hopeless about how to proceed. In the intervening years, I’ve returned to these questions as an establishing process for designing for AI, with a particular focus on how the early work on graphical user interfaces (GUIs) and

human-computer interaction (HCI) laid a useful foundation for thinking about designing for the new technology.

The field of software design emerged as computing expanded beyond its original specialist audience. Suddenly, the people building systems and the people using them were no longer the same, which meant a voice was needed to advocate for the user's experience. Better user experiences opened up broader adoption and new markets, allowing technology companies to compete not just on capability but on accessibility. This transformed computers from tools for experts into tools for everyone. But in the AI era, where products compete primarily on model performance and technical benchmarks, the designer's strategic and visual skill can feel devalued relative to the need for faster inference, larger context windows, and better accuracy scores. As product decisions around AI are increasingly made by technical teams and influenced by competitive pressures, designers are at risk of losing their ability to advocate for the user. This isn't to say engineers aren't capable of making good user experience decisions, but that appointing a voice to challenge assumptions and embed good practices is how we arrived at usable software in the first place.

It would be easy to say in response that design isn't the arbiter of this technical territory; that, as Henry Ford said, we're in danger of inventing a faster horse when we should be inventing the car. But in this book I emphasize that designers are vitally positioned to understand users' needs when working with AI: they should help everyone focus on the principles of the task users are seeking to accomplish. The user isn't thinking about horses or cars—they just seek to go from point A to point B as quickly and efficiently as possible, and to do that they need solutions they couldn't have imagined themselves.

Who Should Read This Book

This book is for product designers. Whether you're a few years into your career or have been practicing for decades, if you've found yourself

beginning an AI project and wondering where to start, this book will give you the grounding you need.

You might be a designer who has spent years refining your craft around deterministic systems, where buttons do what they say and outputs are predictable, and you're now being asked to design for something that feels fundamentally slippery. You might be earlier in your career, entering the field at a moment when AI capabilities are already woven into the products you're expected to ship, and you have little guidance on how to think about them differently from traditional features. Either way, my goal is for this book to meet you where you are.

I've written with the assumption that you have limited technical knowledge of how AI systems work. You don't need to understand transformers, training data, or model architectures to design excellent AI interfaces, but you do need a working mental model of what these systems can and cannot do, and how their capabilities translate into user-facing experiences. We'll build that understanding together, drawing on the computing patterns you already know well.

If you're a researcher, engineer, or product manager curious about how design thinking applies to AI, you're welcome here too. The frameworks and patterns in this book translate across disciplines, and you may find that a designer's lens offers a useful complement to your existing expertise.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

> Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

> **TIP**
>
> This element signifies a tip or suggestion.

> **NOTE**
>
> This element signifies a general note.

> **WARNING**
>
> This element indicates a warning or caution.

O’Reilly Online Learning

> **NOTE**
>
> For more than 40 years, *O’Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O’Reilly’s online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O’Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.

141 Stony Circle, Suite 195

Santa Rosa, CA 95401

800-889-8969 (in the United States or Canada)

707-827-7019 (international or local)

707-829-0104 (fax)

support@oreilly.com

https://oreilly.com/about/contact.html

We have a web page for this book, where we list errata and any additional information. You can access this page at *https://oreil.ly/designing-AI-interfaces*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*.

Watch us on YouTube: *https://youtube.com/oreillymedia*.

Acknowledgements

This curious, complicated book owes an enormous debt of gratitude to the many supporters who helped in its creation.

Firstly, to my wonderful partner and fiancé, Fred Benenson, who instilled in me the confidence to pursue this work, thank you. Your technical skill, deep intelligence, and constant encouragement helped this project feel not only doable but *worth the squeeze*, as you might say—thank you for your love and support.

To the team who made this book possible, I am so grateful for your support and help along the way. To Mike Loukides, thank you for your early support and interest in the original idea. To Nicole Butterfield, for your persistent and dogged belief in this title, especially at moments when it was still finding its shape. To Corbin Collins, for your quick-eyed editing, unending patience, and steady reassurance throughout the process. To Aleeya Rahman and Josh Olejarz for their support bringing the book over the final production hurdles. To my team of technical editors, Hannah Gutkauf, David Evans, Rajeshwari Ganesan, and Dan Saffer, who kept this book from slipping into pure hallucination, thank you. It is deeply improved by the care with which you shared your knowledge and time.

To my parents, Alison and Alan, who learned far more about AI than they might ever have hoped to know, thank you for your curiosity, your good humor, and your willingness to listen as I talked through half-formed ideas aloud.

To my colleagues at Instrument who worked alongside me on that first AI project, thank you. The atmosphere of learning you created framed LLMs as a creative, exploratory technology and sparked a curiosity that stayed with me long after the project ended. In particular, I want to thank Nishat Akhtar, David Brewer, Meaza Abate, and Arthur Lender.

Finally, to my dear friend Erin Allen, your technical mind and kind heart carried me through many of the most challenging parts of this book. I am endlessly grateful for your steadiness, your clarity, and your friendship.

OceanofPDF.com

Chapter 1. Understanding Large Language Models and Systems

In the pre-smartphone era, mobile phone users faced a challenge: texting was *slow*. Phones were constrained to nine-key keypads, meaning that to text your friend “Can you print out MapQuest directions?” on your Nokia 3310, you’d need to tap 84 times.

In the late 1990s a new solution appeared: T9 (text on nine keys). Instead of pressing 2-2-2 to get the letter C, you pressed 2 once. The system would guess which of the three letters (A, B, or C) you actually wanted based on the word you were building.

For example, to type “hello,” you’d press 4-3-5-5-6, and T9 would figure out you meant H-E-L-L-O rather than G-D-J-J-M or any of the other possible combinations.

While this innovation sped up the average user’s typing, that’s not what this predictive text system was designed to solve. Also in the late 1990s, a US startup called Tegic Communications began working on input methods for people with motor impairments, who often struggled with traditional keyboards. The company’s engineers explored how to build efficient text entry using only eight tracked eye positions, dramatically reducing the physical effort needed to compose words. Tegic found the same principle could transform everyday mobile phones, and went on to repurpose its accessibility work to accommodate the growing demand for efficient mobile email and text messaging, resulting in one of the defining mobile input systems of the early 2000s.

Early Predictive Language Models for

Consumers

In some ways, T9 was similar to a large language model (LLM): it was a tool for making educated guesses about written human intent based on an input. The system maintained a dictionary of common words ranked by frequency, which narrowed down possibilities as you typed. When multiple words shared the same key sequence, like “good” and “home” both being 4-6-6-3, T9 would offer its best guess first, based on a word’s prevalence and usage in natural language.

However, by comparison, T9 was a relatively modest technology: the entire system, dictionary included, had a file size smaller than that of a single photo. Yet within those limits, it delivered a highly impressive—and influential—experience. The same creators would go on to create Swype (see Figure 1-1), the gesture-based keyboard used for touchscreen phones, a tool that allowed users to compose words by sliding a finger fluidly from letter to letter, and that influenced nearly every keyboard that followed.

So, although LLMs resemble these early text predictors in approach, in practice they are an entirely different beast. As you’ll see in this book, whereas T9 sped up typing with efficient guesses, LLMs are capable of elaborating language outward into essays, summaries, code, and more. Yet the fundamental purpose remains unchanged: these predictive systems are methods for predicting a sequence of words based on usage patterns gleaned from processing massive amounts of textual data written by humans. While they may seem magical, LLMs merely reflect the patterns and logic embedded in human language, not a greater understanding of the universe.

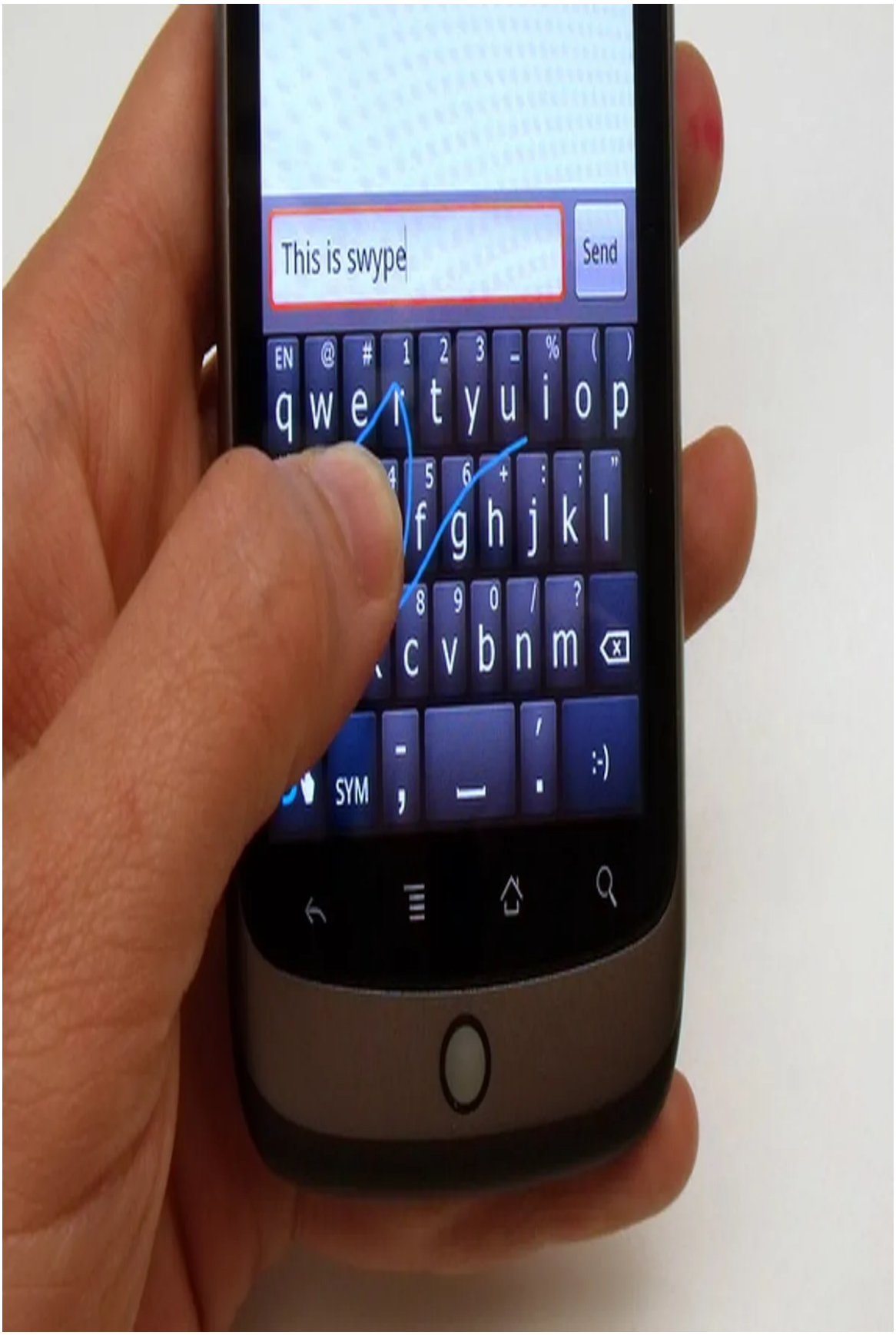

This is swype
Send
EN
SYM

Figure 1-1. Swype, an early predictive language interface to capture gestural typing

This perspective is useful because it anchors our design decisions in human communication needs, preventing us from getting distracted by technical capabilities or promises about the supposed advent of artificial general intelligence (AGI). When we lose sight of this human-centered principle, we build AI features that may demonstrate impressive technical achievement but fail to address actual communication challenges that people face, a common complaint we hear from users of AI products.

To understand how we arrived at today's AI systems, why they present such unique challenges for designers, and the reason for some of the assumptions users bring, we need to trace their evolution. The path from early computing experiments to modern language models reveals fundamental shifts in how we think about machine intelligence. Each era brought its own assumptions about what computers could do and how humans should interact with them, which still shape the design decisions we make today.

A Short History of AI

As little as 10 years ago, barely anyone was talking about AI. I came to this field via my work on a project called TensorFlow, one of Google's first open source models. For one project, Google asked my team and I to create immersive experiences that would help users to understand the underpinning technology of three of its newest models. The first, the well-known PoseNet, could determine a human's pose through image or video inputs and turn it into position-tracking data. The second could look at an image of a human face and estimate emotion—happy, sad, angry, etc. The last was an NLP, or natural language processing, model. To explain how natural language models worked, we devised a game (see Figure 1-2) where the user could navigate a galaxy by typing the word that appeared closest in dimensional proximity to another word. For example, if the user wanted to go to planet A, and planet A's corresponding word was "biology," they could type "lifeform," granting them a 90% association between the two words and setting the trajectory of craft in the right direction.

Our game was effective because it helped users conceptualize words existing in proximity to each other in a multidimensional space, but it was a challenge to design. Of all the models, the NLP model was the hardest for us to devise an experience around. We built it last because we realized that it would require more understanding on our end than time to build. I remember spending days with the projects' engineers, experimenting with different solutions, trying each time to understand a little better *what* this product was capable of.

That experimentation phase—exploring capabilities, translating them into an interface—still represents one of the most persistent challenges in designing AI products. Helping users build accurate mental models of the system requires having a good understanding of the underlying technology yourself, while also prioritizing the user's experience, balancing business needs, and anticipating an enormous amount of change—something we'll explore throughout the chapters of this book.

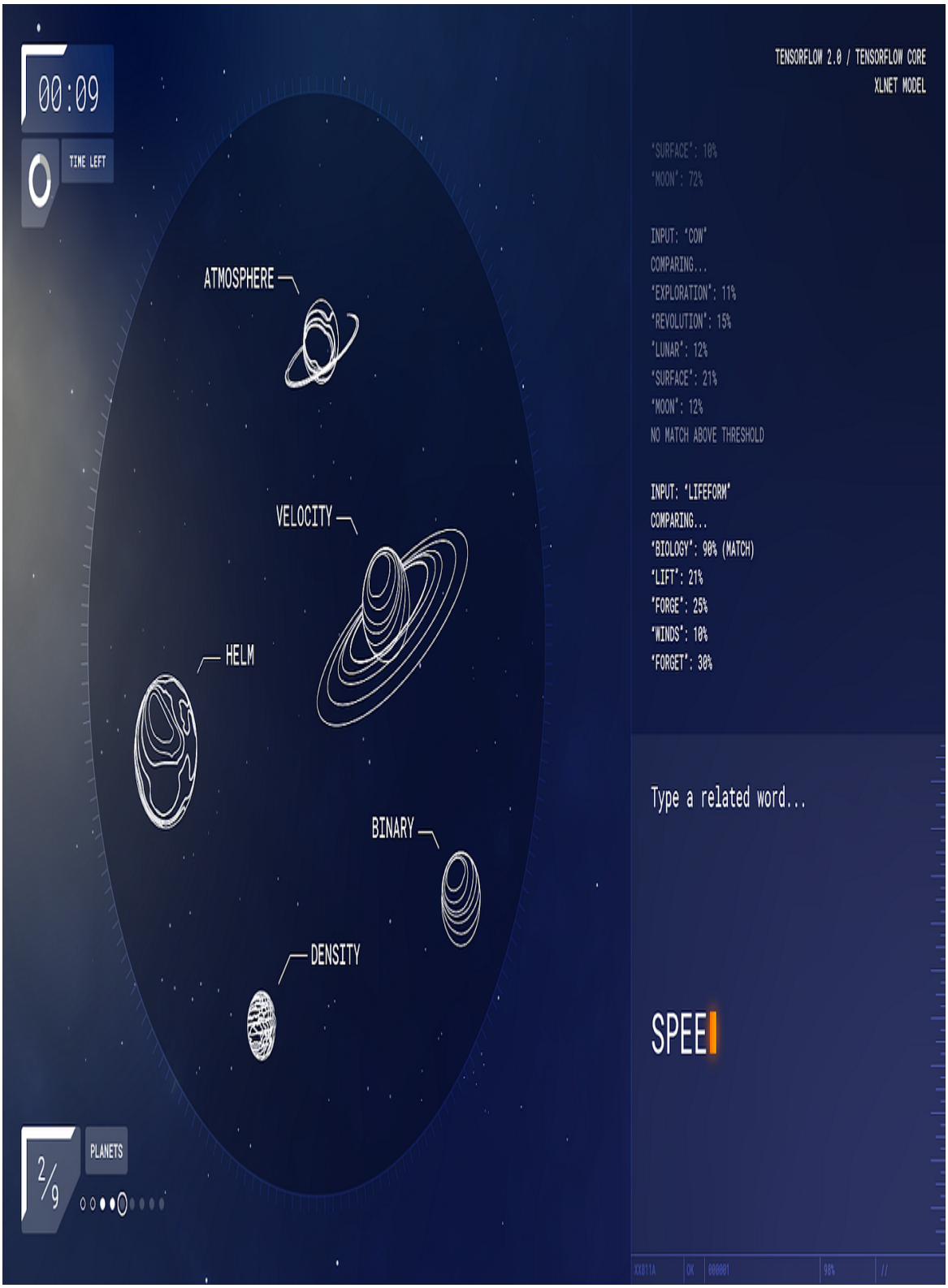


Figure 1-2. Startographer, an interactive game where users chart a course across space by approximating the distance between two words

And while the models of today are superior—profoundly—to that NLP model I worked with in 2018, this doesn't solve for the preconceptions users bring, which are still often inaccurate. Indeed, this difficulty predates modern language models by decades, appearing whenever new "intelligent" capabilities emerged. It's one of the most important lessons of working with AI, and it explains why designers working with AI face a unique challenge that goes beyond typical software design: users arrive at AI products with unfounded assumptions that may not remotely map to real capabilities, and often they lack the support to find the product's *real* capabilities.

Users arrive with these preconceptions about intelligence itself shaped by science fiction, marketing hype, perhaps even their experiences with unrelated but similar systems that may have worked entirely differently (such as Google Search). The media continually promotes stories about a type of absolute general intelligence that could destroy the planet—but this doesn't map to the technology users are really working with. Understanding the journey from theoretical to real AI is useful, then, when thinking about the user experience of an AI product, because the bumpy path toward progress illustrates some of these narrative challenges that cause pervasive confusion around the field.

The Beginning: Symbols

As programmable digital computers arose in the 1940s, they prompted a great deal of speculation about whether they could actually *think*. The mathematician and scientist Alan Turing attempted to give this question a precise answer in his 1950 essay "Computing Machinery and Intelligence". He proposed what he called the *imitation game*, now widely known as the Turing test (see Figure 1-3). A human judge (the interrogator) would exchange written messages with two unseen participants, one human and one machine. If the judge could not reliably tell them apart, the machine would be said to exhibit intelligence.

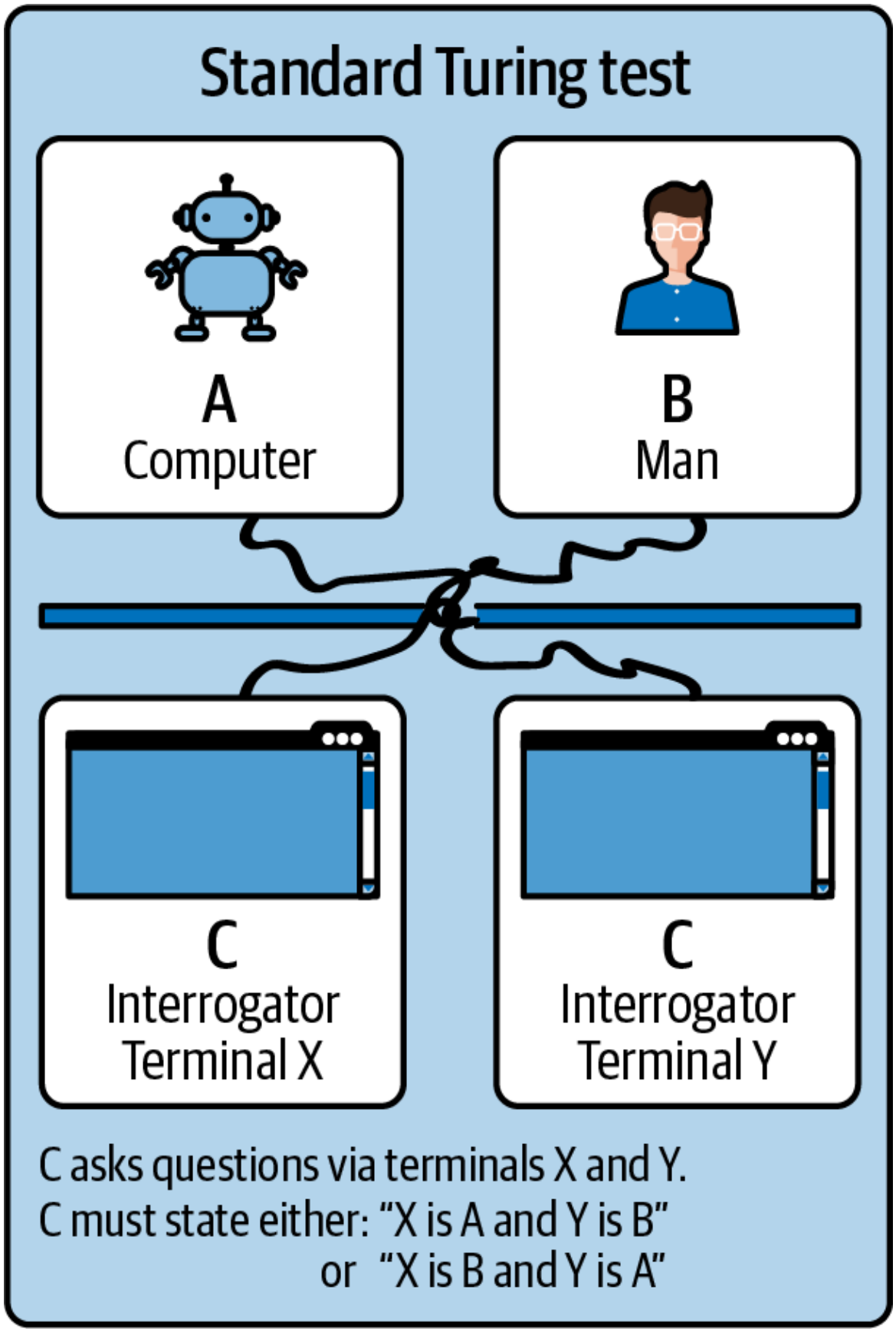

Standard Turing test
A
Computer
B
Man
C
Interrogator
Terminal X
C
Interrogator
Terminal Y
C asks questions via terminals X and Y.
C must state either: "X is A and Y is B"
or "X is B and Y is A"

Figure 1-3. An illustration of the original Turing test, where a human interrogator attempts to distinguish between a computer and a man based solely on their text responses, with the computer's goal being to successfully impersonate the man (adapted from S.G. Sterrett, "The Genius of the 'Original Imitation Game'")

However, the idea came up a little short. While theoretically assessing all intelligence, in actuality the Turing test posed only a very narrow question: "Can a machine convincingly imitate a human in text-based conversation?" The paper didn't anticipate how strongly users would *project* intelligence into a human-sounding conversation, and how vastly different and distant *actual* intelligence and human-sounding conversation are from each other. The technical capabilities germane to human speech turn out to be quite different from the capabilities for autonomy and reasoning than initially thought.

The computer program ELIZA, developed at MIT in the mid-1960s by Joseph Weizenbaum, was one of the earliest attempts to further explore this tension. Weizenbaum built it as an experiment in natural language processing, showing how a computer could carry on a dialogue by following simple rules for recognizing keywords and generating responses. The "DOCTOR" script was designed to imitate a psychotherapist, reflecting the user's words back with minor transformations. In Figure 1-4, you can read a typical exchange with ELIZA.



Figure 1-4. A typical ELIZA conversation

To generate the illusion of conversation, ELIZA scanned the user's input for keywords, applied pattern-matching rules, and reassembled fragments of the text into templated replies. If it detected the word "boyfriend," for example, it could echo that term back in a reflective question. When no keyword was recognized, it defaulted to generic prompts, like "Tell me more."

The ability to respond caused users to quickly anthropomorphize the tool: they began to empathize with the program, attributing understanding, knowledge, and even sentiment to a system that was only reassembling fragments of their own words. Weizenbaum himself was unsettled to see people inferring a degree of cognition where none existed. In his notes, he recorded one worrying encounter between ELIZA and his secretary:

> *My secretary watched me work on this program over a long period of time. One day she asked to be permitted to talk with the system. Of course, she knew she was talking to a machine. Yet, after I watched her type in a few sentences she turned to me and said: "Would you mind leaving the room, please?"*

This phenomenon later became recorded as the *ELIZA effect*: the tendency to project humanlike qualities onto linguistic outputs that only simulate understanding. This makes sense—for the vast majority of human history, with the very occasional exception (lyrebirds, etc.), only humans have been able to speak. Since language, then, is a human-exclusive cue, it's suggested that language primes the brain to anticipate other human-level faculties—an important consideration when building tools that mimic it.

ELIZA and projects like it had arisen out of a general desire to uncover a finite set of rules that would make generated language sound convincingly real. In the 1950s, Noam Chomsky proposed that humans share an innate system of syntactic rules that control language and meaning. His example, "Colorless green ideas sleep furiously," illustrates the paradox: the sentence is grammatically correct, yet so clearly nonsensical that no adult would ever produce it. Chomsky's linguistics theory and the ELIZA experiment defined an era of realization that language involves both structure and interpretation, forcing researchers to move from rule-based planning to exploring statistical methods that could more closely model actual language usage.

But AI researchers were stuck trying to hardcode logic into machines in the hope of generating true intelligence. This failure to move beyond rule-based systems led to the first "AI winter" of the 1970s. Funding dried up as researchers realized their logic wasn't working: they couldn't articulate all of the infinite complexity of human language into an algorithm.

Even decades later, chatbots remained stuck in ELIZA's shadow. SmarterChild (see Figure 1-5), launched in 2001 and used by millions on AOL Instant Messenger, was essentially ELIZA with a bigger dictionary. It could tell you movie times and stock prices, but its conversational ability hadn't fundamentally improved.

In order to develop a model that could more reasonably synthesize human language and behavior, new statistical approaches were invented. Whereas older methods used simple counting (how often does “the” appear before “cat”? How often does “chat” in French correspond to “cat” in English?), these researchers tried to learn deeper patterns through layers of artificial neurons. The hope was that a network could discover abstract features and relationships between words, using data to understand that “dog” and “puppy” were related without being explicitly told. However, neural networks were computationally intensive: training even a small network took days. On 1980s hardware, the simple counting methods worked better.

Three innovations would support the arrival of modern AI: the rapid growth of data, the speed of GPU chips, and the transformer architecture.

First, the internet explosion gave programmers the massive training data they desperately needed. In the 1980s, researchers were lucky to find a few thousand digital documents. By 2010, the scale had become vast. Wikipedia alone contained 3.5 million English articles. Google Books had scanned 20 million texts. Every day, X (formerly Twitter) users posted 500 million tweets. Facebook generated billions of status updates. The Common Crawl project, started in 2011, archived the entire web, creating petabytes of raw text, free for anyone to download.

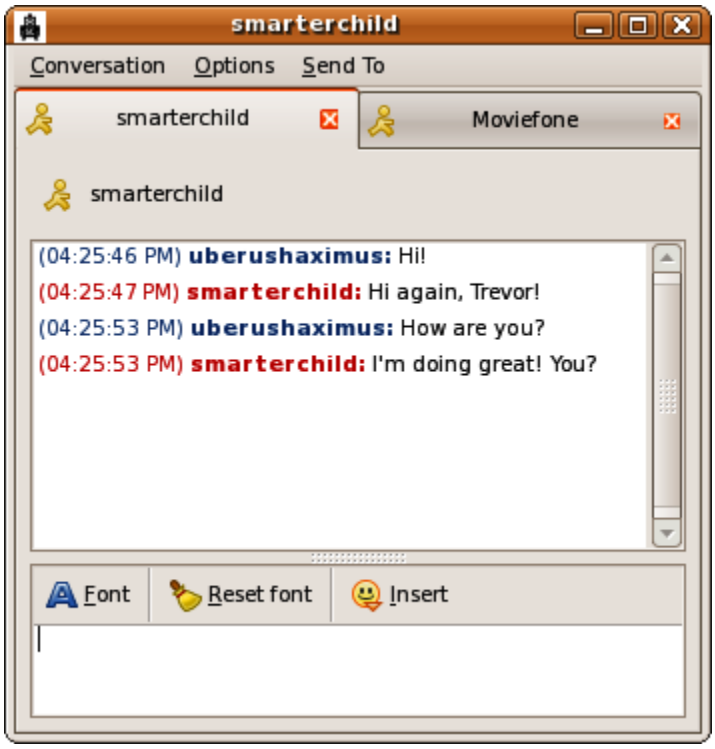


Figure 1-5. AOL's SmarterChild, an early chatbot

The type of data created also differed immensely. Whereas previous datasets were clean and academic, the internet gave researchers messy, human language: forum arguments, text messages, product reviews, chat logs. Neural networks trained on this learned how people *actually* communicated, not how academic papers were written.

The next innovation applied to the processing power problem. In 2012, PhD student Alex Krizhevsky entered a competition to classify images. While other contestants used CPUs, Krizhevsky used two NVIDIA GTX 580 gaming GPUs, cards built to render games like *Skyrim*. His neural network, called AlexNet, destroyed the competition. Eventually, graphics cards built for gaming would come to accidentally revolutionize the AI world. It turns out that the underlying use case is similar: just as games need to calculate lighting and physics for millions of pixels simultaneously, neural networks need to calculate millions of neuron values simultaneously. A single GPU had thousands of tiny processors that could work in parallel, while a CPU had maybe eight powerful processors working sequentially. For neural networks, which mostly did simple math on massive amounts of data, GPUs

were 50 to 100 times faster than CPUs. By 2015, NVIDIA began making GPUs specifically for AI.

The third and final innovation solved the difficulty of *processing* this huge amount of data: the invention of transformer architecture, which solved how neural networks read text. Before 2017, networks processed words one at a time, like reading a ticker tape. In 2017, a team at Google published a paper titled "Attention Is All You Need", which introduced the computational concept of transformers. Instead of processing words sequentially, transformers looked at entire sentences at once. The key innovation was *attention*, a mechanism that let each word directly connect to every other word, computing which connections mattered most. Processing "The cat was happy" no longer meant reading left-to-right. Looking at the word "happy," the transformer could instantly check: what connects to this? It would find "cat" had a strong connection, "was" had a medium connection, and "the" had almost none. This parallel processing made training faster and let networks understand much longer texts.

Within months, every major AI lab switched to transformers, using the three innovations to rapidly build and train a totally new kind of model. Google created BERT (2018), which broke every language understanding benchmark. OpenAI built GPT (2018), with an initial 117 million parameters—the adjustable values within a neural network—then GPT-2 (2019) with 1.5 billion parameters, and then GPT-3 (2020) with 175 billion parameters. Each version simply scaled up the same transformer architecture with more data and more GPU power. The combination was explosive: transformers could actually use those billions of web documents, GPUs could actually train enormous networks, and suddenly machines were writing essays, answering questions, and generating code at near-human levels.

However, this explosion in model development and competency raised concerns, with researchers tracking the seeming disconnect between the public's expectations of AI and what was really going on under the hood. In their 2021 paper "On the Dangers of Stochastic Parrots", researchers Emily Bender, Angelina McMillan-Major, Timnit Gebru, and Margaret Mitchell

warned that the race to build ever-larger models risked overlooking crucial issues: these systems could amplify biases present in their training data, consume massive amounts of energy with environmental consequences, and create the illusion of understanding while merely recombining patterns from their training corpus. Their work highlighted how the same scale that enabled impressive capabilities also concentrated power among organizations with the resources to train massive models, while making it harder to understand or correct problematic behaviors embedded deep in billions of parameters.

As of 2026, the AI landscape is defined by a complex and rapidly evolving ecosystem, moving far beyond the early generative models. The market is highly competitive, featuring major players like OpenAI's GPT-4, which has demonstrated impressive performance across professional tasks and coding; Anthropic's Claude models, which prioritize safety through constitutional AI training (a two-stage process that uses a set of principles called a "constitution" instead of human feedback to train AI models); and Google's Gemini models, which are advancing multimodal capabilities that seamlessly integrate text, images, and code. A central challenge in this era, however, remains the significant gap between the systems' technical reality and public understanding. Designers must continuously work to bridge the persistent user tendency to project human-level intelligence onto systems that are fundamentally statistical, addressing issues like unpredictable failures and "hallucinations" to build essential trust and accurate mental models of AI's true capabilities.

User Experience in the AI Era

What does it mean when we say users bring assumptions to AI products? As a starting point, it's important to clearly understand the *types* of misunderstandings that users have.

As mentioned earlier, it's common to see users project human-like understanding onto a model, believing the AI grasps context, emotion, and nuance in a way that goes beyond its capabilities. This is often coupled with

an assumption of factuality, which causes users to treat generated information as correct and reliable, failing to account for a model's tendency to hallucinate, or generate plausible-sounding errors. Further complicating this is the assumption of consistency and reliability, the expectation that the same prompt will always yield the same correct result, overlooking the probabilistic nature that leads to output variability.

Other common misbeliefs include the assumption of complete knowledge, where users believe the AI has access to all information and current events regardless of its training data cutoff, and the assumption of unbiased output, which ignores that the model's training data inevitably reflects and can reproduce human biases.

The fundamental conflict between user assumptions and AI's actual capabilities can prevent users from discovering the *real* features that they might find most useful. When users assume a human-like understanding, they misapply the AI to complex reasoning tasks, leading to frustration and the dismissal of its real value as a tool for information retrieval, summarization, and drafting. Similarly, the belief in the model's inherent factuality causes users to forgo critical steps like fact-checking, missing the opportunity to leverage the AI as a powerful ideation partner rather than a final authority. Ultimately, an overreliance stemming from the assumption of complete knowledge prevents users from treating the AI as what it is truly capable of being: a research assistant requiring partnership with human expertise and critical judgment.

The conflict between user assumptions and actual AI capabilities is significantly complicated and worsened by the vast financial investment pouring into the AI sector. Worldwide AI spending totaled $1.5 trillion in 2025, and is only anticipated to rise in 2026. Businesses are under immense pressure to deploy AI features quickly to justify and recoup these massive capital expenditures, and as a result they prioritize speed, cost, and immediate market impact, which can exacerbate the problems arising from user assumptions.

For example, the pressure to demonstrate innovation often results in the promotion of features a user doesn't need or the building of features a user doesn't want, prioritizing technological capability over user utility. This problem is compounded in some cases by a lack of audience focus: when products attempt to cater to too many use cases or personas without successfully narrowing in on a primary audience, AI features are created in ways that directly conflict with how the user wants to use or engage with the tool. This misalignment deepens the user's confusion, making it difficult for them to form an accurate mental model of the system's true capabilities, or of how they can efficiently complete a task.

Beyond business pressures, the broader environment of AI development and adoption plays a major role in the conflict. As mentioned earlier, the continuous cycle of hype in the media, particularly surrounding breakthroughs in generative AI, creates a perception of near-magical, limitless capability, accelerating the cultural acceptance of AI as an oracle and making it harder to educate users on its much duller, statistical nature. This is accompanied by a lack of universal standards and regulation; the evolving regulatory vacuum in many jurisdictions means no mandatory standards for transparency are enforced (though, as we will see, early legislation has begun to be adopted).

Finally, the very interface and experience design of many AI tools worsen the problem: clean, minimalist, and conversational interfaces in some cases hide the underlying complexity and probabilistic nature, reinforcing the idea that the user is interacting with an intelligent being and obscuring the reality that it is a complex statistical engine.

For designers, the work of managing that gap begins with knowing your user and how your product is used. This is home turf for a designer, but in the AI era it goes beyond documenting needs or collecting feature requests: it means asking where users are likely to misinterpret, what signals will guide them toward the right understanding, and how much of the underlying process is worth revealing. Sometimes a clear explanation helps, but just as often too much detail overwhelms or distracts. The skill lies in deciding where transparency builds confidence and where it only creates

confusion, and then keeping those decisions consistent so the product feels coherent.

Technical and evaluatory skills play a part here as well. Designers do not need to become machine learning researchers, but they do need enough grounding to understand what is happening under the hood. Without that knowledge, it is difficult to explain behavior, to anticipate where the system might fail, or to set clear expectations for both users and colleagues. These skills allow designers to judge which system details to surface, which to smooth over, and how to present them in a way that helps users rather than distracts them.

And because AI products are built at the intersection of disciplines, communication across the organization is just as important. Designers, engineers, and product managers need a shared language for describing what a system can do, what it cannot, and what should or should not be shown to users. When those standards are established and reinforced, users encounter a consistent logic across the product. Organizations also benefit from clear knowledge ownership and sharing practices: within a fast-moving industry, AI products require strategic, focused methods for communicating and adapting to technological changes.

In this sense, the three requirements—knowing your user, building technical and evaluatory skills, and enhancing communication practices—are all devoted to the same purpose: reducing ambiguity and giving users a way to understand what is in front of them. That clarity is what makes AI products usable, and without it, even the most advanced system risks being misunderstood or mistrusted. Let’s explore each requirement a little further.

Know Your User

When an interface can be *anything*, how can you design for it? On a project for Google Search, years ago, I found myself challenged by the need to visualize the output of a search. The problem: to make Google’s categories more discoverable, a designer had to anticipate the outcome of any user query, partly to help guide the user toward the different output types

(images, flights, news, etc.). But given that someone could search for *anything*, how could I design to improve their specific user experience? Helpfully, Google shared some published research in which it categorized its eight most common types of search queries (shown in Figure 1-6). Reading these clarified the *types* of tasks users were commonly accomplishing with their searches, helping me form a more direct connection between the user's need and what the interface answered with.

When designers ask how to think about designing for AI, this is one of the first projects I think of. Working with these open-ended, generative products often requires a Search-like methodology—one in which we acknowledge that a single interface cannot meet the needs of everyone who uses it, and yet also recognize it's impossible to build an adaptive interface for every use case. A one-size-fits-all approach is the most efficient, but it risks flattening the variety of ways people arrive at a product. Some come to learn, some to write, some to experiment. When these very different needs are funneled into the same experience, we need to adopt a thoughtful, strategic approach to prevent users from becoming disoriented or missing the most valuable elements of the product. Otherwise, they may not know how to get started, or they may conclude that the system is not designed for the task they had in mind.

Entertainment queries

As a movie buff, I want to find the latest reviews and ratings for newly released sci-fi movies so that I can decide which movie to watch this weekend.

Voice search queries

As a cooking enthusiast, I want to ask my smart speaker for a simple recipe for pasta so that I can prepare dinner without having to search on my phone.

Transactional queries

As a busy professional, I want to find the best online store to purchase a new ergonomic office chair so that I can improve my home office setup.

Navigational queries

As an avid reader, I want to navigate directly to the GoodReads website so that I can check reviews for a book I'm considering buying.

"How to" and DIY queries

As a first-time gardener, I want to find a step-by-step guide on how to grow tomatoes in my backyard so that I can start my own vegetable garden.

Health-related queries

As someone feeling unwell, I want to search for flu symptoms and see potential treatments so that I can decide whether I need to see a doctor.

Informational queries

As a student, I want to find detailed information on the history of the Greeks so that I can write a well-researched paper for my history class.

Local queries

As a homeowner, I want to find the nearest hardware store so that I can buy paint and supplies for a weekend home improvement project.

Figure 1-6. Google's most common search query categories

One way to bring clarity to this challenge is to group user needs into categories and design around those categories. In the field of user experience (UX), we have long called these categories *personas*. Personas distill large, messy sets of behaviors into recognizable profiles, each representing a cluster of goals, contexts, and frustrations. In AI, personas do more than guide feature lists; they help teams design for recognizable modes of interaction and anticipate ambiguous outputs. They let us anticipate the questions users will bring, the misinterpretations they may fall into, and the kind of feedback or transparency that will be most useful in each case.

Personas are commonplace, though sometimes controversial because they can be encapsulations of stereotypes and guesswork instead of proper research. Some companies have even begun to use AI tools to generate them. In the case of designing for AI products, their value is in communicating examples of how a system responds to different roles: for example, serving as a writing coach, a research assistant, or a creative partner. Teams might use personas internally to test how a new feature might be received by different segments: would a student preparing for an exam feel supported, and is there a more specific way we could help them? Would a professional drafting a contract discover the right user interface (UI)? Such thought exercises assist in structuring real patterns of behavior so the product can meet them with intention. User needs give us the limits of what the system must support, even when the interface cannot adjust itself to every individual query. This allows us to set a practical outer range for the product's general requirements.

Building personas for AI products can be more challenging than for other types of products because of the sheer breadth of capabilities or adoption. However, looking at usage patterns can be a critical starting point. In the National Bureau of Economic Research working paper, "How People Are Using ChatGPT" (NBER working paper no. 34255, 2025), researchers from OpenAI and Harvard clustered millions of queries and found recurring categories, such as *tutorials*, *editorial critique*, *personal communication*,

and *creative ideation*, each representing thousands of daily interactions (see Figure 1-7). These clusters show that even a general-purpose tool does not produce infinite variation; behaviors group into recognizable patterns that reflect the goals people bring to the interaction.

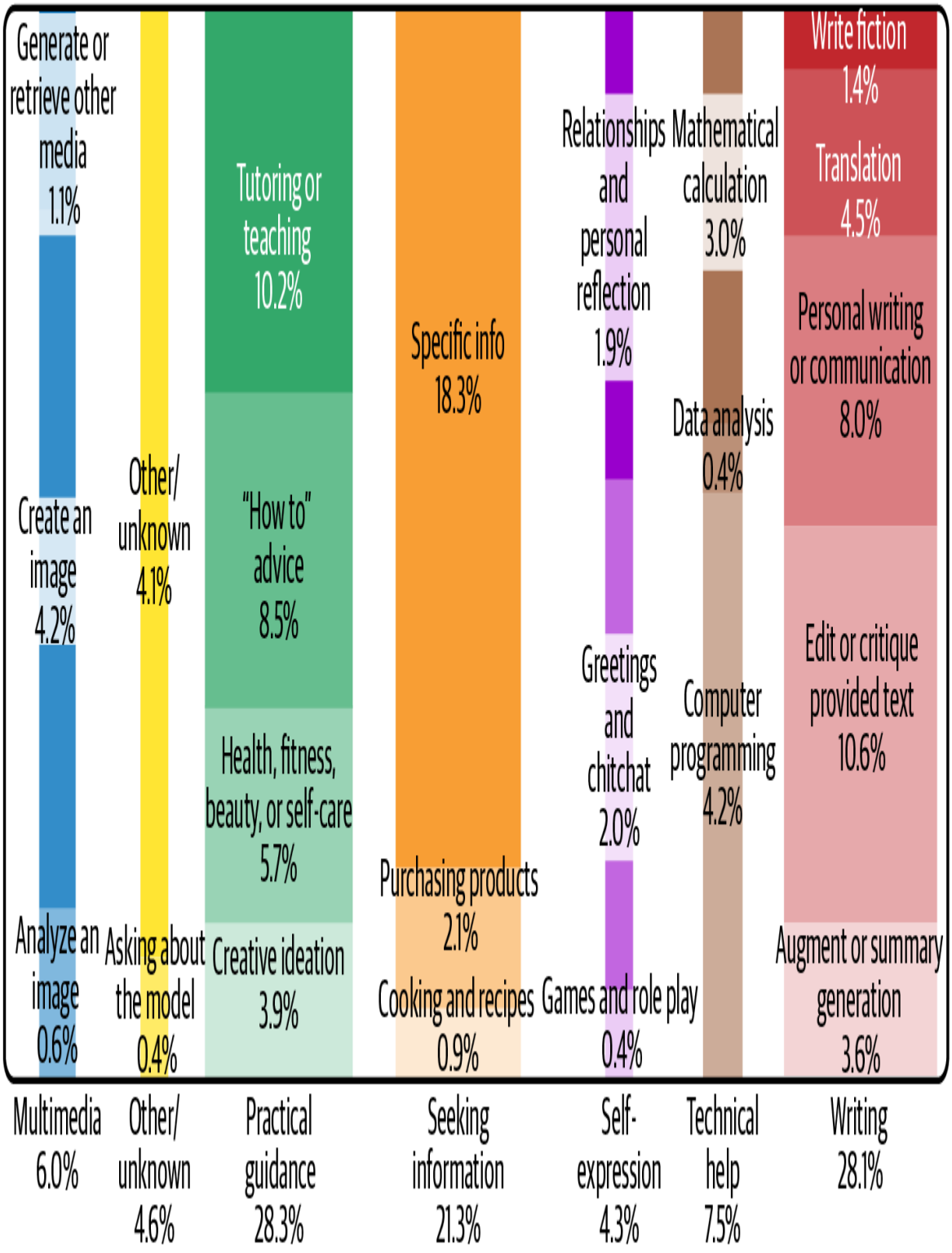


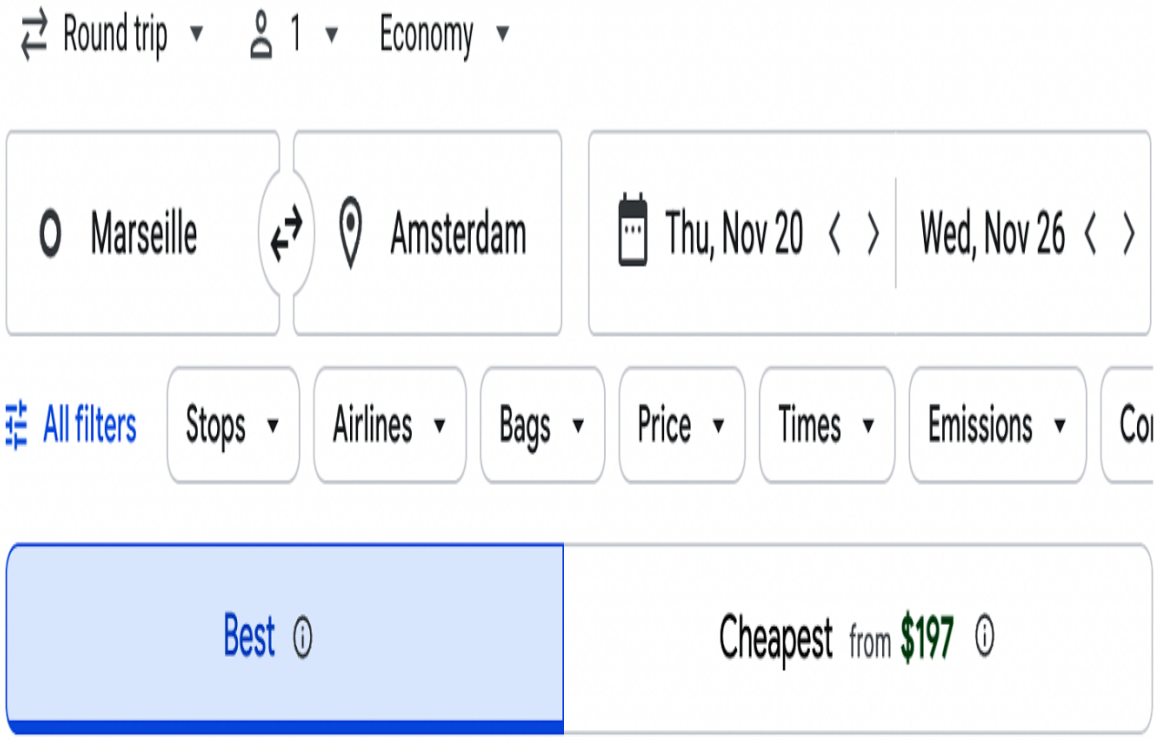
Figure 1-7. Categories of ChatGPT use

The implications for interface design are varied: a person using a tool primarily for tutorials may benefit from continuity across sessions, so

earlier questions can inform later ones. Here, long-term or episodic memory becomes valuable, because the system can recall what was already covered and build upon it rather than repeating explanations. By contrast, someone seeking a critique of a piece of text may care less about persistence over time and more about immediacy, such as a reasoning tracker that shows how suggestions were derived, or a session consolidation feature that summarizes edits made across a draft.

Creating an interface that unites these, or that provides intuitive discovery, is the core challenge of the designer. Mapping out how a user achieves their task gives a clearer idea of where they might look to include an AI tool, or be open to a “nudge.”

Search illustrates this principle in a familiar way. When someone types “flights Marseille to Amsterdam,” the results present times, prices, and the ability to filter. The user intent behind the query has been recognized, and the interface reflects that recognition, adapting normal search results into a more useful UI, attuned to the user’s need. If the search engine did not make this shift, the user would be left to piece together information from multiple sites, working harder than they should to accomplish what is, at its core, a simple task. Figure 1-8 shows an example.



Top departing options

Ranked based on price and convenience Prices include required taxes + fees for 1 adult. Sorted by top flights

Optional charges and bag fees may apply. Passenger assistance info.

KLM	12:20 PM – 2:20 PM KLM · Air France · Operated by ...	2 hr MRS–AMS	Nonstop	102 kg CO2e +6% emissio...	$253 round trip	⌄
AF	9:50 AM – 1:55 PM Air France	4 hr 5 min MRS–AMS	1 stop 1 hr 10 min C...	143 kg CO2e +49% emissi...	$258 round trip	⌄

Figure 1-8. Search results as interface

This is an important point we'll return to again in Chapter 5: people are rarely searching for text itself; they are searching for outcomes. By giving them a way to move directly from query to action, the tool shows that it understands what they are trying to achieve, and the experience feels sharper as a result. The intelligence lies less in the algorithm than in the way the result is presented, because the design translates a recurring pattern into a surface that fits.

There is a similar opening in contemporary products that incorporate generative tools. When a request aligns with one of the established categories, like learning, writing, critique, or ideation, the interface can adapt in small but meaningful ways. A draft being edited might benefit from inline comments that mirror the conventions of word processors; a question framed as a tutorial might be better answered with steps that can be expanded or collapsed; a creative exercise might feel more natural with multiple suggestions side by side. Each of these design choices signals to the user that their intent has been understood, which strengthens trust and makes the experience feel more purposeful.

The challenge is that intent is not always cleanly detected, and designing for categories always involves trade-offs. Overly rigid responses risk locking people into a workflow they did not ask for, while too much generality makes the product feel indifferent to their needs. The opportunity, however, is clear: by identifying recurring user patterns and shaping the interface around them, design can lower the burden on the user and create a sense of recognition that works across many different use cases.

Technical and Evaluatory Skills

Should designers code? The age-old question of technical knowledge looms large for designers entering the AI landscape. The key shift is that designers now *do* need technical fluency that wasn't previously required. Before AI, exceptional designers could succeed purely through understanding user needs, mastering design principles, and creating visually and functionally

appealing solutions. It’s certainly true that CSS was a very simple technology in contrast to modern LLMs.

Today, designers benefit enormously from understanding how AI models are trained, what data they use, and what their limitations might be. This doesn’t mean they have to become data scientists, but having technical literacy enables more effective collaboration with engineers and data teams. It also helps designers anticipate how their design choices might play out in dynamic, AI-driven environments.

The best approach starts with getting comfortable with basics: understanding what machine learning is, what types of AI models a product might use, and what common pitfalls exist. Working closely with technical partners to translate user needs into features—AI-enhanced or not—remains the core work. Designers can still rely on their intuition, plus technical awareness; that combination allows them to contribute as meaningfully as ever, but in a more technologically layered landscape.

Like most work, the design process isn’t immune to influence by AI. One of the most significant new opportunities created is the ability to prototype in ways that were previously impossible. Designers can now use AI-driven prototyping tools to mock up AI features and see them in action early, providing hands-on understanding of how different models behave and how prompts shape outputs. Weaving prototyping into daily workflows becomes essential. The more designers experiment with building and testing AI-powered prototypes, the more naturally they understand what happens under the hood and how to design for it.

This method represents a dramatic shift from relying on static wireframes or imagination. Actually building and iterating with AI tools provides an incredible way for designers to gain technical insight organically. Using AI frequently, and using these tools to create products, teaches far more than theoretical study alone. Designers can learn about how different models work, see how prompt engineering functions in practice, and gain insights that would be difficult to develop otherwise.

Yet a crucial challenge remains: designers struggle to know how deep their technical knowledge should go. Without an accepted standard of knowledge in the field, it's difficult for designers to gauge the appropriate level of detail. The temptation exists to think one needs PhD-level AI expertise, but that's neither realistic nor necessary.

> **TIP**
>
> The key to designing thoughtful AI interfaces is starting small and being incremental. Designers don't need to master everything at once. Building a foundational understanding that enables communication with teams and more thoughtful design is sufficient. Knowledge naturally accumulates through practice. The field continues evolving rapidly, making the "right" level of knowledge a moving target anyway. Designers should give themselves permission to learn gradually without pressure to know everything immediately. Curiosity, rather than completeness, should guide the learning journey.

Ethical Considerations in AI Design

Importantly, designers now carry additional responsibility for understanding and shaping AI behavior ethically and transparently. This represents an entirely new layer of accountability that simply didn't exist when designing static interfaces. While ethics in design certainly existed before AI, it primarily focused on accessibility, inclusivity, and ensuring that products didn't inadvertently exclude people. The role has both expanded and deepened, requiring designers to think systemically about outcomes they cannot fully predict or control.

Designers now must consider algorithmic bias, transparency, user consent, and data privacy in much more direct ways. They need to think about how to communicate what the AI is doing and ensure users understand how their data is being used. There's also the responsibility of ensuring AI doesn't cause harm or reinforce existing inequalities. This shift has transformed designers into ethical guardians of AI interactions, requiring them to advocate for user rights and transparency while designing interfaces that help users feel informed and in control.

A 2021 paper by DeepMind examined the ethical and social risks of harm from language models, outlining seven specific risk areas:

- Discrimination
- Exclusion and toxicity
- Information hazards
- Misinformation harms
- Malicious uses
- Human-computer interaction harms
- Environmental harms

These can be grouped into three overarching categories that help designers frame their understanding and action.

Social and interpersonal risks

The first category of ethical concern covers harms related to discrimination, exclusion, toxicity, and misinformation. They are concerned with how AI outputs, if unchecked, can compromise the core principles of fairness and inclusivity by reinforcing biases or spreading harmful and unreliable information.

Designers can approach these by focusing on fairness, inclusivity, and the quality of information that AI produces or moderates, by designing features that allow users to report biased outputs or harmful content, and by creating a culture of AI transparency. For further reading, the Partnership on AI (PAI) maintains a list of resources aimed at driving the responsible adoption of AI and ensuring its benefits for people and society.

Safety and security risks

The second major ethical category involves risks to system integrity and security, covering information hazards, malicious uses, and automation access issues. These concerns focus on protecting users and their data from

harm, requiring mechanisms that ensure human judgment remains in control for critical automated decisions.

Here, designers can build in safeguards such as clear user permissions, warnings about sensitive data usage, and preserving human oversight. Promoting transparency in AI decision making helps users feel secure and informed about system operations. Stanford Human-Centered AI regularly publishes influential research on this subject, offering vital guidance and frameworks for the responsible design and deployment of AI systems.

Environmental impacts

The third category of ethical concern focuses on the environmental impacts of AI, including the substantial energy required to train and run large models, and the water used to cool the data centers that support them. These impacts are distant from the interface, and designers will rarely have direct influence over the infrastructure decisions that drive them. What they can do is promote a deliberate approach to when and how AI is used, especially by avoiding unnecessary features or advocating for smaller or more efficient models where possible. For further reading, the Green Software Foundation publishes guidance, research, and tooling aimed at reducing the environmental footprint of software systems, including those that rely on AI.

When speaking about ethics in AI, it’s crucial to recognize the realistic boundaries of responsibility. Designers aren’t solely responsible for fixing large, systemic issues like hallucinations or water usage; these problems are primarily owned by engineering teams and business initiatives. And similar to every other technology predating the AI era, there are trade-offs between the opportunities a new technology offers and its risks to users and society on the whole.

However, as it becomes increasingly clear which AI behaviors cause harm, designers will find themselves uniquely positioned to advocate for user safety and ethical innovation. Their proximity to the user experience enables them to champion best practices around AI usage, whether resteering to avoid overuse, raising awareness about energy impact, or

ensuring tools are implemented in healthy, user-friendly ways. Being aware of these compromises is an important facet of working with this technology: being a voice for the user's interests beyond their immediate need is the core of the designer's evolving role as an advocate and mediator. The key is for designers to remain aware of these trade-offs and participate actively in conversations, raising issues when they see them and pushing for the most ethical solutions that are practical.

Communication: Knowledge Sharing and Role Clarity

In 1918, the magazine *Scientific American* heralded the arrival of the self-driving car, calling it "a motorist's dream: a car that is controlled by a set of push buttons" (see Figure 1-9).

As of 2025, despite billions of dollars of investment, it would be a dubious claim to say that we've arrived at totally autonomous cars (though companies like Waymo would argue we have). A key challenge lies in defining when a car has truly become autonomous. A car might be able to drive by itself 90% of the time but need user intervention when unique situations, like a flood, arise. Would we call this autonomous? It's conceivable to argue that, to qualify for the term, the car should be able to navigate every situation without human input. But then, is this a useful way to identify the car's capabilities?

In 2014, the Society of Automotive Engineers (SAE) introduced a six-level classification system (Figure 1-10) for driving automation, aimed at solving this exact problem. In the SAE classifications, each level describes the relationship between the human and the system. The levels run from 0 to 5, grouped into two big categories: the human drives (0 to 2) and the system drives (3 to 5). Levels 0 to 2 offer increasing assistance while keeping the driver responsible for supervision and control. Levels 3 to 5 shift the driving task to the system, first with conditional handovers, then with full autonomy inside a defined domain, and finally with no domain limits. The framework clarifies how responsibility moves from human oversight to system ownership as automation increases.



SEVENTY-FOURTH YEAR

SCIENTIFIC AMERICAN

THE WEEKLY JOURNAL OF PRACTICAL INFORMATION

VOLUME CXVIII. NUMBER 1 | NEW YORK, JANUARY 5, 1918 | 15 CENTS A COPY $4.00 A YEAR

The motorist's dream: a car that is controlled by a set of push buttons

Figure 1-9. Scientific American imagines a self-driving automobile

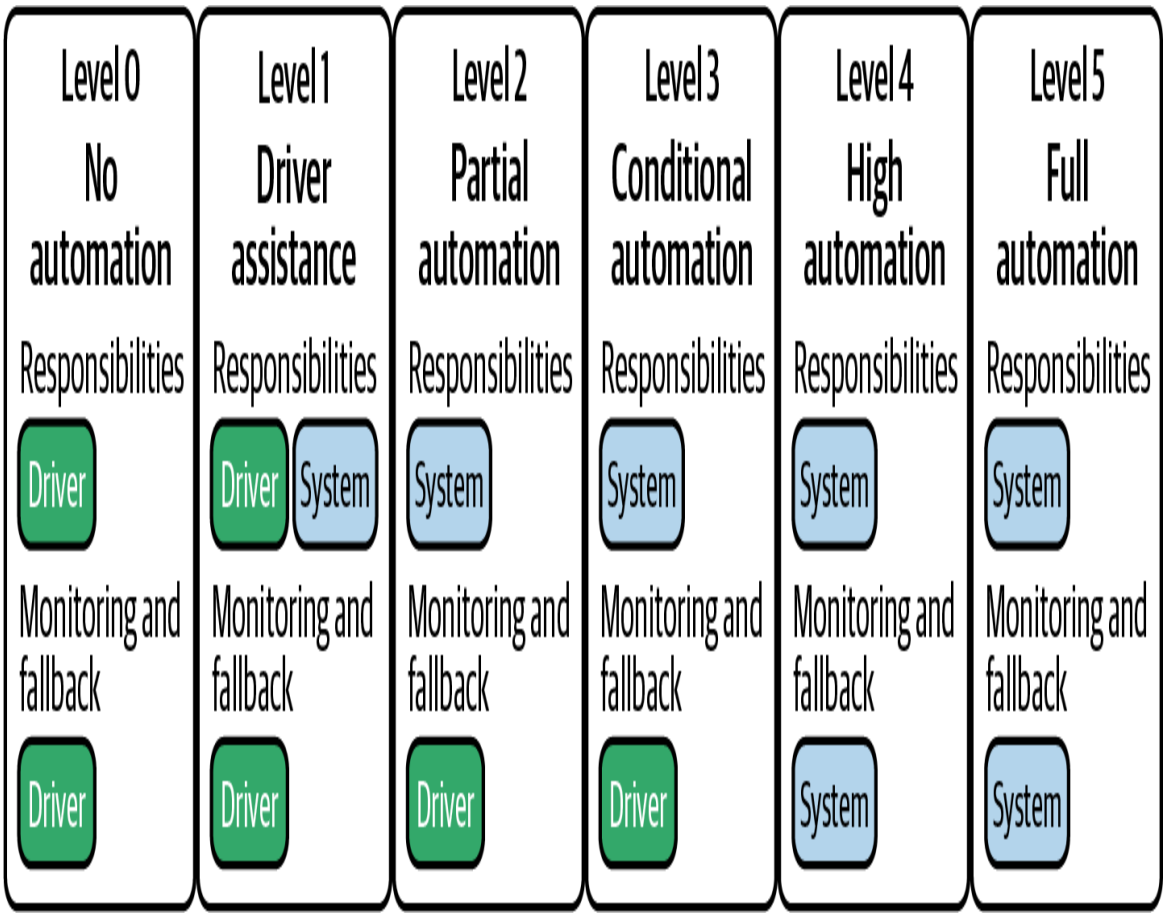


Figure 1-10. The SAE autonomous driving classification system

I mention this framework because delineating ownership and responsibilities is at the heart of building technically complex and successful AI products. Designers might assume that once they understand their users and gain technical familiarity, their work will become easier—yet many of the hardest challenges arise from the surrounding organization rather than from the design tasks themselves. In a fast-moving environment, model capabilities may change without warning, and without good structure, technical updates and decisions may reach teams slowly—or not at all. Just like with autonomous driving, unclear responsibility creates confusion, slows progress, and makes it harder to understand how decisions should flow.

These issues can occur because an organization has not yet developed practices that support AI product development. AI introduces complexity at many layers, not only in model behavior but in how teams communicate,

form shared assumptions, and distribute responsibility. Good change management in AI environments depends on how information moves: how frequently it circulates, how clearly it is translated, and how consistently disciplines understand their roles. When these channels are unstable, the organization struggles to produce coherent work, even when designers understand both their users and the underlying technology.

It’s vital to establish responsibility for understanding and communicating model capabilities. Designers can learn how the model works, but it is probably not efficient to ask them to track every update because capabilities can shift monthly, sometimes weekly. Clear ownership reduces this burden. Product managers (PMs), when present, play an essential role in translating technical updates into language that reaches design, engineering, research, leadership, and marketing. They support shared understanding by explaining how a capability has changed, what evaluation findings show, and what the implications are for user experience.

In organizations without PMs, designers rely more directly on engineering partners who can surface constraints, reliability observations, or emerging system behavior. Strong engineering partners support design by sharing information early and in a format that helps teams plan.

Good information sharing practices thereby become an integral part of good change management in AI product building. When information circulates predictably, the organization can adjust to new model behavior. When circulation is inconsistent, each group forms its own assumptions and the product loses cohesion. Evaluations, responsible AI guidance, content-warning standards, capability shareouts, persona development, and prototyping scripts can all be routed on these information pathways, and also rely on them. The maturity of an AI organization can be understood by examining how these forms of knowledge move, who maintains them, and how reliably they reach the teams who need them.

It can be useful to reflect on the way information flows around you in the organization. The following framework adapts the logic of the SAE model. Instead of describing levels of automation, it outlines levels of

organizational readiness for AI work. Each level reflects how information travels, how ownership is defined, and how effectively the organization supports design in a landscape shaped by rapid and continuous change.

Level 1: Individual exploration

At this level, teams explore AI tools independently and develop personal knowledge without any shared structure. Designers learn through experimentation, perhaps using external tools and projects, because the organization has not yet created channels for understanding model behavior or invited a culture of experimentation. Hallmarks of this stage include:

- Understanding of model behavior belongs to individuals rather than to any designated owner.
- Evaluation work has not been assigned and does not inform product decisions.
- Responsible AI considerations are unassigned and appear only when someone raises a concern.
- Capability shareouts do not exist; updates surface only through ad hoc conversations.
- Prototyping relies on personal initiative, without scripts or personas to guide consistency.
- New features and requirements emerge high up in the org chart, without early input from the implementation side or close attention to user need.
- Criteria for content warnings or other safety surfaces have not been established.

This level reflects early curiosity supported by experimentation, but the absence of structure prevents teams from forming shared understanding or moving in a coordinated direction.

Level 2: Shared understanding with manual maintenance

Teams establish shared language, and documentation begins to accumulate in a central place. Responsibilities become clearer, yet manual upkeep and person-to-person clarification remain common. In this level:

- Product managers or engineering partners begin to steward model knowledge and share updates with other groups.
- Evaluation work has an explicit owner who synthesizes results, although distribution still requires manual effort.
- Responsible AI practices take shape, with emerging expectations around fairness, safety, and user outcomes.
- Capability shareouts may occur occasionally, driven by interest rather than by scheduled practice; attendance and consistency vary.
- Personas may exist in early form, though they are lightly researched and not integrated into prototyping. Prototyping includes scripts informed by early personas, with researchers shaping improvements as needed.
- Content warnings follow preliminary criteria, giving teams a shared starting point even if refinements are needed.

This level reflects an organization building coherence, although the work still relies on continued attention, frequent clarification, and hands-on coordination.

Level 3: Integrated AI organization

Information flows throughout the organization with high reliability; change management becomes continuous rather than episodic. Documentation, evaluation, and responsible AI practices follow defined processes, and findings, guidance, content-safety criteria, and capability updates reach the right teams without friction. Teams understand how updates flow and possess a high-quality understanding of their users' core needs. Design

gains the stability needed to make informed decisions about a product's road map. At level three:

- New and emerging product features originate as outcomes of research and user need, instead of pressing market trends or hyped AI capabilities. Bottom-up validation assures product-market fit instead of top-down declarations. Users find AI features useful and intuitive, not hastily "fastened on" to existing processes.
- Product managers hold a clear narrative of model behavior and communicate changes before they affect the experience. Model knowledge is stewarded by a designated owner, and updates circulate through established communication pathways.
- Evaluation work is owned by a dedicated person or group that shares findings with design, engineering, and leadership. Results reach design, research, product, and engineering automatically, allowing teams to respond quickly.
- Responsible AI work has a stable home, and its guidance shapes planning, review cycles, and release processes.
- Capability and process shareouts follow a steady cadence with strong participation, helping teams anticipate shifts in system behavior and influence planning across disciplines.
- Prototyping either relies on scripts that stay aligned with validated personas, supported by ongoing research insights, or is fully enabled to use real-time model data. Prototyping infrastructure permits employees who are *not* developers to engage.
- Content warnings follow audited, consistently applied criteria that reflect documented safety expectations.
- On-call or escalation systems provide dependable access to knowledgeable partners who can interpret changes or resolve questions.

This level reflects an organization that adapts smoothly to change, supported by reliable information pathways, clear ownership, and mature practices that help teams navigate evolving AI behavior with stability and accuracy.

Reading This Book: The Input-Computation-Output Structure

With that initial primer out of the way, the rest of this book delves into far greater detail about the emerging patterns and practical challenges of designing for AI. In order to make sense of this rapidly changing technology, it helps to create a structure that is applicable across the user experience, and the most stable way that I've found to understand this is through articulating three stages: *input-computation-output* (Figure 1-11).



Figure 1-11. The input-computation-output framework

This framework has been consistent throughout my experience working with AI, and I believe it is likely to remain relevant even as new modalities arrive.

Inputs

Inputs are the point where human intention enters the system. In most cases, input looks familiar: a text box for a chatbot, a search bar, or a button that triggers a predictable action. Yet inputs can also be more subtle, inferred from context rather than explicitly entered. A tool might notice an open document, a location, or a repeated workflow and treat that as intention. Designers are comfortable shaping explicit inputs, but AI complicates the picture by introducing new forms of invisibility, ambiguity, and guesswork.

What feels like a clear request may be filtered through probabilistic interpretation, creating gaps between what the person thought they asked for and what the system actually does.

Computation

Computation is the opaque stage where the system does its heaviest lifting. To the user, this phase is often signaled only by a spinner, ellipsis, or subtle animation, but inside it requests are interpreted, relevant data is retrieved, reasoning paths are explored, and multiple tools may be orchestrated in sequence. Guardrails also take effect here, filtering or reshaping the request before it continues. For designers, computation is paradoxical: it has the greatest influence on the quality of the result, but it is also the phase they see and control the least.

Outputs

Outputs are the point where results return to the user. They can take the form of text, images, charts, code, or even audio and video. What the user sees has already been processed and formatted so that it is legible, safe, and ready to use. Text is generated token by token and then smoothed into prose. Images are decoded, corrected, and compressed before they appear. Even small choices in this stage, such as how a citation is displayed or whether a chart is interactive, shape how interpretable and actionable the result feels.

Input, computation, and output form the core of every interaction, but they are not the whole story. Before input, there is configuration: intent shared, accounts connected, defaults chosen. These decisions determine what the system can do. After output, there is interpretation and action: the human edits, verifies, or integrates the result into their own workflow. Errors need to be designed for, as does feedback. The loop then begins again, with outputs prompting new inputs.

This three-stage structure gives us a shared language for discussing AI design that cuts through the complexity without losing essential nuance.

When a user complains that an AI tool feels opaque, we can now ask more precisely: is the input unclear about what's expected, is the computation taking too long without feedback, or is the output formatted in a way that doesn't support their next action? The framework also reveals why designing for AI requires such a different mindset from traditional software. Static interfaces could optimize each stage independently, but AI systems create dynamic relationships among input, computation, and output that shift based on context, user, and even the specific request. Understanding these relationships, and designing for them effectively, is what separates AI products that frustrate from those that genuinely augment human capability.

Each of the following chapters pairs design patterns with the technical foundations that inform them. Understanding why a system behaves the way it does makes it far easier to design interfaces that work with its strengths and compensate for its limitations.

Chapter 2, "Capability, Discovery, and Orchestration", addresses everything that must be in place before a user ever types a word. We'll look at how models are trained and evaluated, how users discover what's possible, and how models, agents, and tools are selected and permissioned. These foundational decisions shape every interaction that follows.

Chapter 3, "Designing for AI Inputs", examines the moment where human intention enters the system. We'll cover how tokenization and context windows shape what the model actually receives, then explore the three channels through which intent arrives: implicit context, explicit prompts, and direct manipulation. The chapter addresses text, image, and voice modalities and how to guide users toward inputs that get them what they need.

Chapter 4, "Computation: Designing for the Processing and Generation Phase", opens the black box of computation. We'll trace how an input moves through processing, routing, and generation, then turn to the design challenges these stages create: latency patterns, error states, and providing just enough transparency without overwhelming users with technical detail.

Chapter 5, “Output: Designing the Delivery and Presentation of LLM Responses”, covers the moment results return to the user. We’ll examine how generation actually works, including sampling, temperature, and token prediction, then turn to design principles for clarity, verifiability, and actionability. The chapter also addresses multiturn conversations, watermarking, and managing problematic responses.

Chapter 6, “Agentic AI: Designing for Systems That Plan, Act, and Adapt”, extends beyond simple request-response into agentic territory. We’ll look at the technical patterns that enable longer-running AI behavior, including reflection, tool use, planning, and multiagent coordination. Then we’ll examine the interface challenges they create: decomposition, delegation, checkpoints, rollback, and designing for shared control over complex workflows.

Summary

In some ways, computing has always been influenced by the drive to narrow the gap between human and machine capability. As we’ll see in Chapter 2, the command shells of the 1960s and ‘70s gave remarkable power to those who mastered them, but their rigidity excluded anyone who could not memorize syntax. Search engines improved on this state of affairs, interpreting looser queries and even correcting typos on the user’s behalf. Predictive text carried the same logic further, anticipating intent and offering it back in usable form. Each step shifted the burden of adaptation from human to machine, making space for more people to use technology productively. That pattern of gradually relocating the effort away from the user and into the system remains one of the clearest throughlines in interface history.

For designers today, the work is both familiar and different. We must still advocate for user intent, but we now sit in a position where we also mediate between human language and complex systems that interpret, transform, and generate on the user’s behalf. The task is to make these systems not only easy to operate but also interpretable, so that people understand what is

happening and can act accordingly. Where predictive text once surfaced a suggested word, AI systems now surface paragraphs, analyses, or entire workflows. This amplifies the designer's responsibility: to shape how those outputs are presented, how much explanation is given, and how users remain in control of the exchange.

Rather than viewing predictive text or AI as outliers, it helps to see them as part of a continuum: technology that listens a little more closely, guesses a little more accurately, and takes on a little more of the burden of communication. Each shift has opened participation to more people, but each has also raised new questions about how much should be revealed, how much should be left opaque, and how users remain in control. These are not just engineering questions; they are design questions. Our role is to shape how much of the system is exposed, how intent is carried forward, and how confidence is built in processes that cannot always be seen.

Over the next several chapters, we will examine how this long tradition of assistive computing plays out in the age of AI, using the input-computation-output framework as a guide. The lesson from predictive text, and from so many accessibility breakthroughs before it, is not that the technology was clever, but that it solved something real. Innovations that endure do so because they meet users where they are and expand what they can do. For designers, the challenge is to look past the hype and uncover that core value.

OceanofPDF.com

Chapter 2. Capability, Discovery, and Orchestration

In 2009, Google launched Wave (see Figure 2-1), a novel communication and collaboration platform that promised to reinvent how people work together online. It combined the features of email, instant messaging, and document editing into a single place, acting as a one-stop shop for communication and creation.

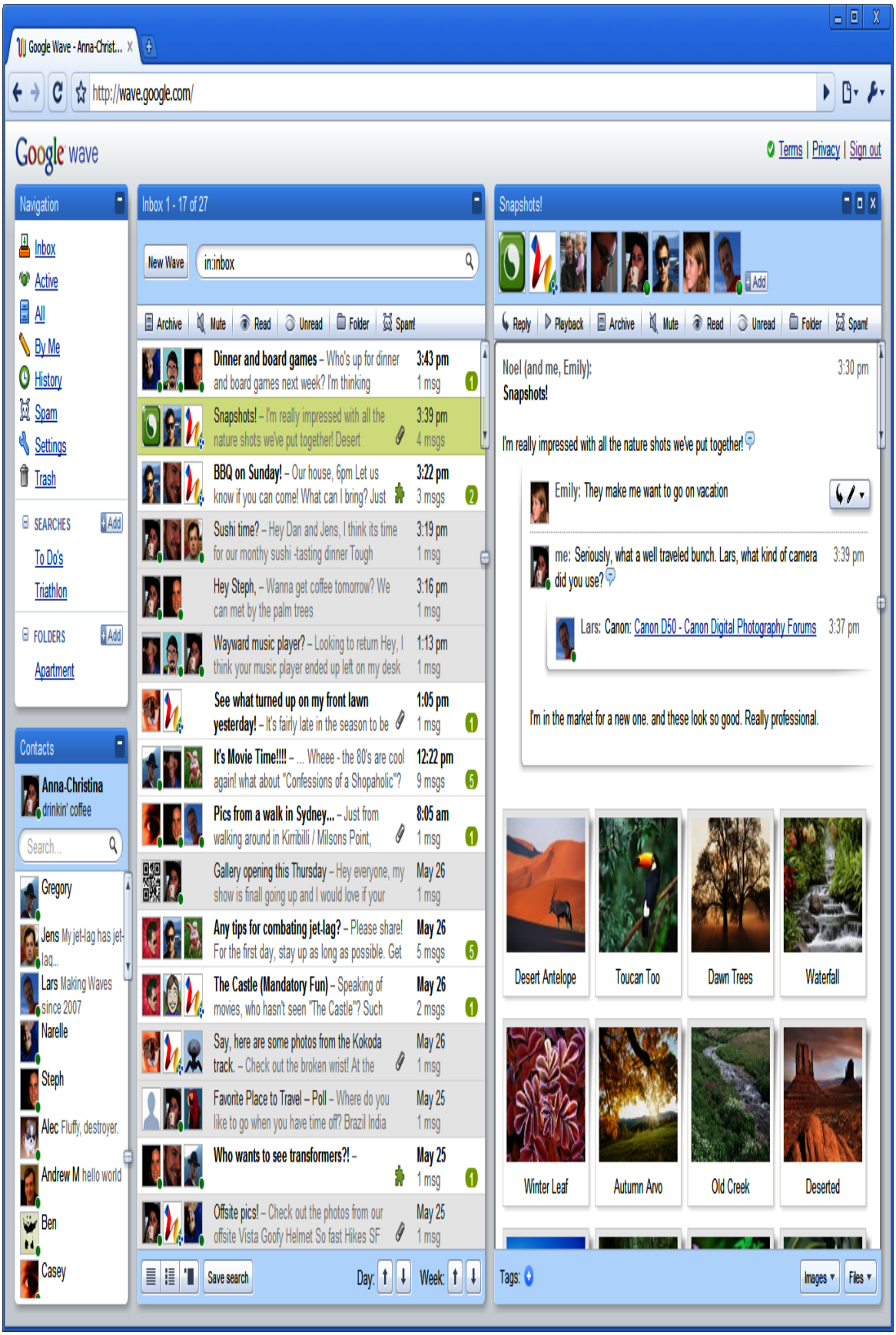

Google Wave - Anna-Christ...
http://wave.google.com/
Google wave
Terms | Privacy | Sign out
Navigation
Inbox
Active
All
By Me
History
Spam
Settings
Trash
SEARCHES
Add
To Do's
Triathlon
FOLDERS
Add
Apartment
Contacts
Anna-Christina
drinkin' coffee
Search...
Gregory
Jens My jet-lag has jet-lag...
Lars Making Waves since 2007
Narelle
Steph
Alec Fluffy, destroyer.
Andrew M hello world
Ben
Casey
Inbox 1 - 17 of 27
New Wave
in:inbox
Archive
Mute
Read
Unread
Folder
Spam!
Dinner and board games – Who's up for dinner and board games next week? I'm thinking
3:43 pm
1 msg
Snapshots! – I'm really impressed with all the nature shots we've put together! Desert
3:39 pm
4 msgs
BBQ on Sunday! – Our house, 6pm Let us know if you can come! What can I bring? Just
3:22 pm
3 msgs
Sushi time? – Hey Dan and Jens, I think its time for our monthy sushi -tasting dinner Tough
3:19 pm
1 msg
Hey Steph, – Wanna get coffee tomorrow? We can met by the palm trees
3:16 pm
1 msg
Wayward music player? – Looking to return Hey, I think your music player ended up left on my desk
1:13 pm
1 msg
See what turned up on my front lawn yesterday! – It's fairly late in the season to be
1:05 pm
1 msg
It's Movie Time!!!! – ... Wheee - the 80's are cool again! what about "Confessions of a Shopaholic"?
12:22 pm
9 msgs
Pics from a walk in Sydney... – Just from walking around in Kirribilli / Milsons Point,
8:05 am
1 msg
Gallery opening this Thursday – Hey everyone, my show is finall going up and I would love if your
May 26
1 msg
Any tips for combating jet-lag? – Please share! For the first day, stay up as long as possible. Get
May 26
5 msgs
The Castle (Mandatory Fun) – Speaking of movies, who hasn't seen "The Castle"? Such
May 26
2 msgs
Say, here are some photos from the Kokoda track. – Check out the broken wrist! At the
May 26
1 msg
Favorite Place to Travel – Poll – Where do you like to go when you have time off? Brazil India
May 25
1 msg
Who wants to see transformers?! –
May 25
1 msg
Offsite pics! – Check out the photos from our offsite Vista Goofy Helmet So fast Hikes SF
May 25
1 msg
Save search
Day:
Week:
Snapshots!
Add
Reply
Playback
Archive
Mute
Read
Unread
Folder
Spam!
Noel (and me, Emily):
3:30 pm
Snapshots!
I'm really impressed with all the nature shots we've put together!
Emily: They make me want to go on vacation
me: Seriously, what a well traveled bunch. Lars, what kind of camera did you use?
3:39 pm
Lars: Canon: Canon D50 - Canon Digital Photography Forums
3:37 pm
I'm in the market for a new one. and these look so good. Really professional.
Desert Antelope
Toucan Too
Dawn Trees
Waterfall
Winter Leaf
Autumn Arvo
Old Creek
Deserted
Tags:
Images
Files

Figure 2-1. Google Wave (2009)

Unfortunately, it died an early death. Google would soon cancel work on Wave, because, as the Museum of Failure puts it, “the service was too ambitious, hard to learn, and barely an improvement over regular email.”

Wave took its name from *Firefly*, the sci-fi series where *sending a wave* meant transmitting a message across time and distance. Google’s version aimed to modernize communication in the same spirit: waves were real-time, shared documents that supported rich media, inline replies, and simultaneous editing by multiple participants. Built on a hosted XML structure, each wave functioned as a live thread that blended messaging with collaborative content creation. Users could drop in images, maps, or files, edit each other’s contributions midstream, and extend functionality through bots and gadgets.

At launch, the product drew enormous interest. But despite the excitement, many users struggled to understand what Wave was for. The promise of a unified workspace was clear in theory, but in practice people found it difficult to adopt, use meaningfully, or integrate into their daily routines.

By mid-2010, less than a year after launch, Google announced it would stop active development on Wave. The user base hadn’t grown beyond early adopters, and internal momentum had stalled. While some of Wave’s underlying technologies, like real-time syncing and collaborative cursors, lived on in other Google products, the platform itself was shut down entirely by 2012. It lives on as a case study in what happens when powerful technology isn’t paired with the right product framing: in short, nobody could figure out what Wave was *for*.

This is the kind of product-user misalignment this chapter seeks to prevent. Users report that AI features can feel clumsily implemented and unhelpful, deriving from a product approach centered around emerging capacity over real user need. An approach like this treats the model’s capabilities as the starting point, when in practice what matters most is where those capabilities intersect with the process the user employs to complete a task.

To design successful AI features, it’s important to understand not just what the model can do but what users *want* to do, and how it helps them get there. This chapter proposes a way to achieve this more clearly: through understanding user intent, supporting discovery, and supplying clear orchestration. Breaking it down, these are the key ideas we’ll be focusing on:

Intent

: The specific goal or outcome a person wants to achieve when they type a query into a search engine or interact with a digital product.

Capability

: What the model is capable of doing. This includes both narrow functions (such as summarizing a document or extracting an entity) and more open-ended behaviors (generating text, suggesting next steps, or performing multistep reasoning).

Discovery

: What the user is aware of, how they are made aware, and what they’re willing to try. It includes both *intent* (what the user is hoping to accomplish) and *discoverability* (what the user knows is possible). Discovery can also involve figuring out what to surface that the user may not have anticipated, such as relevant data or suggested next steps.

Orchestration

: The design layer that connects the other two. It includes the systems, scaffolding, and UX choices that allow user intent and model capability to meet productively. Orchestration is the layer that helps users move from exploration to adoption.

By the end of this chapter, we’ll have unpacked each element in more detail and looked closely at the overlaps between them.

Capabilities: What Can the Model Do?

In Chapter 1, we looked at how users can be easily misled about how to use an AI product. However, as it turns out, understanding a model's ability is a key issue for designers, too. In the 2020 paper "Re-examining Whether, Why, and How Human-AI Interaction Is Uniquely Difficult to Design", researchers from Carnegie Mellon University's Human-Computer Interaction Institute found that designers frequently report that it is "difficult to grasp what AI can or cannot do," which as a result "hampers designers' brainstorming and sketching processes from the start." This can cause an impeded design process, where technical understanding hampers creativity and designers struggle to create within the bounds of feasibility. Even a basic understanding of AI wasn't enough—researchers found that even with a baseline of knowledge, designers "often found it difficult to ideate many possible new interactions and novel experiences with much fluidity."

Figure 2-2 illustrates the junctures at which such knowledge barriers present a challenge.

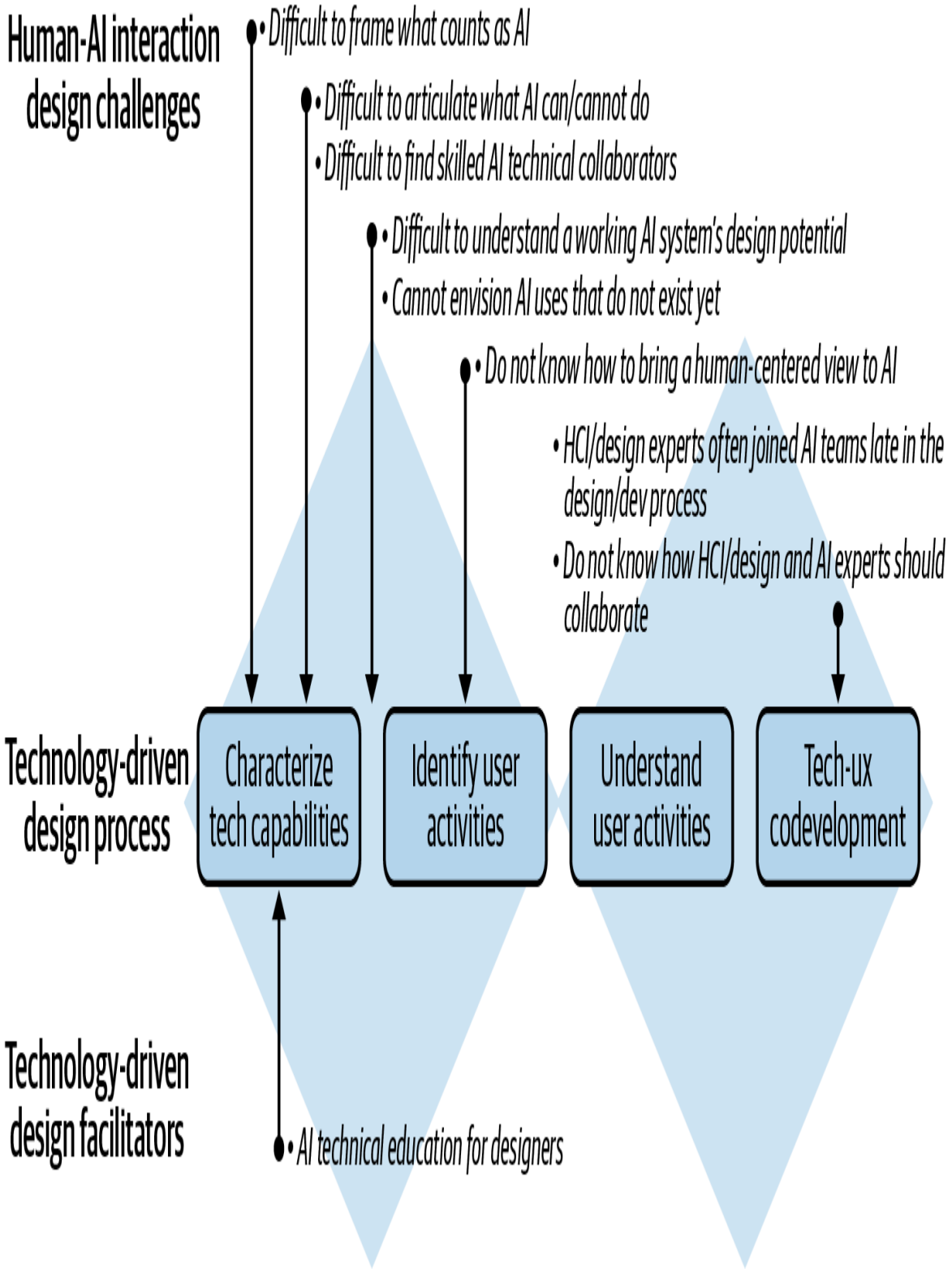


Figure 2-2. Design challenges of unknown AI capabilities on the design process (adapted from Yang et al.)

This study begs the question: what *is* the right amount of knowledge a designer needs to work with AI? Across the industry there are emerging methods of general understanding that can act as a guideline. We'll look at the two key elements that control the abilities of a particular model: how the model was trained and how it was evaluated.

Training

Training is the initial, and probably better recognized, stage of making a model. It can also be controversial: from the earlier section on ethics, you may recall discussions around bias in datasets and model training. These conversations often focus on what data was included, what was left out, and who made those decisions. While these debates matter enormously for the broader AI ecosystem, the practical implication is that the training data used fundamentally shapes every interaction users will have with AI products. Engineers compile this data with goals for scale and coverage, and with quality thresholds that reflect both the ambition and the constraints of the model they are trying to build. They apply filters, heuristics, and evaluation passes that determine which patterns the model will later recognize and which types of language it will treat as familiar.

Training is broken into two parts. Most LLMs start with *pretraining*, a process similar to reading the entire internet and then trying to incrementally improve predictions about what word comes next after any given word. The content can take many formats: the model ingests massive amounts of text, books, articles, forums, documentation, Wikipedia entries, and news sites, the objective of which is to give the model a wide base of knowledge that helps it notice patterns and structure in language and apply that understanding to many different tasks. It forms initial observations: when it sees "The capital of France is..." during training, it figures out how to consistently reply that "Paris" is the most likely continuation.

Next, the model is typically refined through a process called *fine-tuning*. This stage uses a smaller, more targeted dataset to guide the model toward specific behaviors. Instead of general next-word prediction, the model is trained to follow instructions, answer questions, summarize documents, or

engage in multiturn conversation. Whereas pretraining is computationally expensive, fine-tuning is less intense but involves more oversight. In many modern tools, fine-tuning includes a method called *reinforcement learning from human feedback* (RLHF), where human reviewers rank different model responses. These rankings are used to adjust the model's behavior, with the goal of making the output more helpful, more truthful, or more aligned with human preferences, depending on how those goals are defined by the developers.

It's important to emphasize that, despite training, models don't really "know" anything. This process is akin to trying to learn a language by memorizing a dictionary but never having conversations. The model develops an understanding of grammar, facts, and reasoning patterns but only through exposure to written text. It learns that certain words tend to follow others, that some topics connect to specific vocabularies, and that different writing styles require different approaches. In essence, the model becomes a very sophisticated next-word generator—an advanced version of T9.

Evals

Training and evaluation form a continuous cycle in AI development. Whereas training shapes what models can do, evaluations, or *evals* as they're called in industry parlance, reveal what the models actually do in practice.

One of the most widely used tests of general knowledge and language understanding is called Massive Multitask Language Understanding (MMLU). It contains multiple-choice questions across 57 topics, from US history to computer science to moral philosophy. Because it pulls from academic exams and professional certifications, it gives a broad sense of how well a model performs across disciplines. While some benchmark items may overlap with training data, MMLU is still treated as a useful yardstick. A model that scores highly can usually follow instructions, retrieve facts, and handle technical vocabulary.

To evaluate reasoning more directly, researchers use math benchmarks. One example is GSM8K, a dataset of 8,500 grade-school math word problems that require performing multiple steps, like converting units, doing arithmetic in the right order, or combining facts to get an answer. Models can stumble here, as long logic paths and reasoning can become too convoluted or complex for them to follow. Success on GSM8K shows that a model can sustain structured thinking across a few sentences, which models can find particularly challenging.

Researchers often rely on human judgment to assess how models perform in more open-ended language tasks, like writing explanations, summarizing ideas, or taking a position. MT Bench (short for Model Test Bench) is an eval used to compare the performance of language models on realistic, user-facing tasks with a set of realistic prompts that are evaluated by people, not machines. Given two model outputs for the same task, reviewers pick which one is better. This can highlight things that automated tests miss, such as tone, clarity, or whether the model actually understood the nuance of the request.

No single benchmark is perfect, and most evaluations are narrow by design. Researchers publish to any of the many available leaderboards of results across a dozen or more standard benchmarks to give a broader picture. But even then, high scores don’t guarantee real-world reliability. Many current efforts are focused on creating new benchmarks that better reflect open-ended use, long-form reasoning, or more dynamic tasks like planning and tool use.

For designers and project managers, evals and training can feel like the domain of engineers, particularly given how challenging the technical language barrier can be. However, as the voice of the user, it’s vital that designers can contribute to refining and improving models, collaborating with engineering and anticipating the best possible user outcome. Here’s some advice for designers wishing to explore evals in their process:

- There’s not yet a process for designers to become incorporated into the evals methodology, but reviewing and interacting with these

standards can certainly help with getting a general understanding of the underlying process.

- Designers should feel comfortable engaging with their engineering partners to explore the models they use in order to fully understand the capabilities available to them. Given the explosion of vibe coding and the availability of multimodel hosting services like OpenRouter, designers can achieve valuable prototypes with greater ease than ever before, and with a higher degree of verisimilitude.
- Outside of self-initiated exploration, designers should stipulate that model capabilities be shared by engineers or PMs in a digestible—if informal—format before feature design commences. Having a technical partner for a design sprint can assist in communicating opportunities alongside user needs.

TIP

Vibe coding, a term coined by Andrej Karpathy in February 2025, is an AI-assisted development approach where you "fully give in to the vibes, embrace exponentials, and forget that the code even exists." The method involves describing what you want in natural language, accepting all AI-generated changes without reviewing the code, and debugging by pasting error messages back to the LLM. Karpathy characterized it as "not really coding—I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works"—suitable for "throwaway weekend projects" rather than production software.

"Vibing" is derived from the unplanned nature of the process. Unlike in normal programming, which tends to require a high level of skill and planning, the vibe-coding process involves converting a loose idea into a prompt, running the code, and then iterating on the request as needed. The cycle is quick and informal, with progress coming from repeated exploration that can allow for rapid building. However, while this approach can feel creative, it can also lead to poorly written code that is difficult to maintain.

Discovery

User intent refers to the purpose behind an action. It captures what a person is trying to accomplish, even when their input is brief or clumsy or

incomplete. A typed prompt, a voice command, or a series of gestures may only hint at the goal, but the intent is the throughline that connects those surface signals to the outcome the user hopes for.

User intent sits at the heart of exceptional user experience design: it's the driving force behind every click, tap, and interaction users have with digital products. It's often ambiguous or dynamic, expressed through natural language, gestures, or unstructured data. Understanding how users discover and achieve this intent is the bridge between what users want to accomplish and how a product helps them achieve those goals.

In some cases, AI products have centered on capabilities and functionality, with ideas originating from a desire to compete in the AI product space, instead of prioritizing specific user needs. Teams with this mindset ask questions like "What features should we build?" and "How can we showcase our product's capabilities?" The primary concern is creating robust, impressive functionality and ensuring that users can access all available features. This approach often leads to feature-rich interfaces where the emphasis is on demonstrating what the product can do.

User-focused design thinking, by contrast, centers on people and their goals. Teams with this mindset ask, "What are users trying to accomplish?" and "How can we help them succeed?" The primary concern is understanding user motivations and creating experiences that serve those underlying needs. This approach leads to purpose-driven interfaces where the emphasis is on helping users achieve their intentions efficiently and effectively. Instead of organizing interfaces around product capabilities, we organize them around user goals.

TIP

Intent-driven design emerges naturally from user-focused thinking. When we understand what users are trying to accomplish, we can design experiences that feel intuitive and purposeful rather than overwhelming or confusing. This approach recognizes that users engage with digital products as a means to an end, not as an end in itself. While we might observe that a user clicks a button or fills out a form, intent reveals the deeper motivation: are they researching a purchase decision? Looking for specific information? Trying to complete a task as quickly as possible?

In AI interfaces, uncovering intent can be more challenging. Because best practices are still emerging, users often struggle to imagine what an AI product could do for them, which makes their goals less defined than when using traditional products. Research in this space tends to be more exploratory, helping surface not just what users want but what they might want once they better understand the tool's possibilities.

Intent with AI

Of course, gathering user intent can prove highly challenging. UX researchers use methods like behavioral analysis, user interviews, and journey mapping to reveal intent through patterns in behavior and stated preferences, providing signals to product designers about how to create effective product features. However, for many AI products, the opportunity lies in anticipating and supporting intent adaptively. This is more complex, because in real time, users communicate their intent through two distinct channels: *explicit* signals like direct requests and commands, and *implicit* signals embedded in their context, behavior, and approach to your interface.

Explicit intent feels straightforward. For example, if a user were to type "write a professional email declining this meeting" into your product's AI assistant, their goal would at first glance appear clear. Direct signals like this form the foundation of most AI interactions, but they represent only the surface layer of user intention.

Implicit intent runs much deeper and often reveals more about what users actually need. It encompasses everything from a user's expertise, to how they arrived at your product, to the digital environment surrounding their interaction. Whereas explicit signals can tell you what users think they

want, implicit signals can reveal what they actually need to accomplish their underlying goals. It’s often best to use both.

User intent also varies depending on whether users are interacting with an AI-first or AI-enhanced tool. In AI-first tools, intent is often shaped in relation to a general-purpose assistant or generative environment, where users may show up without a rigid task or methodology in mind. These tools need to elicit and interpret intent more fluidly, sometimes even helping users articulate it for the first time. In contrast, AI-enhanced tools typically layer AI atop existing workflows, where intent is already better defined. Here, the challenge is in surfacing AI help in ways that align with existing user behavior without disrupting it.

As AI becomes more capable, users begin to set more ambitious goals. Knowing that the tool can take on complex or specialized tasks, people begin to pursue outcomes they previously saw as out of reach. An engineer who might have initially used AI coding tools to make single, discrete functions can now set up an agentic query to build an entire app or platform. These shifting expectations raise the stakes for good interface design: people hope not just to be helped but to have their needs transformed.

Designing with intent in mind means recognizing that what people *want* to do and what they believe they *can* do are in constant conversation. And with AI, that conversation is especially fluid, constantly challenging users to uncover and evolve their expectations and processes.

Discoverability and Initialization

Capturing user intent can be pointless if the user can’t find the entry point for the AI tool. Love or hate it, the once-loved sparkle emoji ( ) has become the unofficial mascot of AI product features. As noted by Quartz and *The Wall Street Journal*, the emoji’s positive associations with “magic” mean that it’s become a recognizable method of drawing user’s attention to a new AI-enabled feature. Many have complained about the broad interpretation

of the icon, noting that the variety of AI features available is generally a lot more complex than what can be associated with just one emoji.

This is where discovery comes in. Finding a method for introducing new capabilities while also educating the user builds trust and clear visual associations for the user. A negative discovery experience of an AI feature can provoke responses like these:

- “There’s too much going on, and I don’t know what I’m supposed to pay attention to.”
- “I didn’t ask for that.”
- “I keep discovering things by accident.”
- “It’s cool, but I don’t know how to use it.”
- “This doesn’t feel like it’s for me.”
- “I don’t know what it can do.”

To address these concerns, AI feature discovery should be educational, intentional, well-timed, and mapped to intent. To begin, designers working on showing a feature to the user can ask themselves, “Does the user already know they want this?” If yes, you’re working with high user intent. If not, you’re in low-intent territory and need to design for gentle awareness. Next, they can ask, “Will the tool surface this proactively, or wait to be prompted?” If it waits, that represents low system initiative, requiring strong affordances and clear naming. If it acts first, you’ll need to consider timing, confidence, and exit paths.

Together, these two dimensions give us a 2 × 2 matrix for thinking about AI feature discovery, shown in Figure 2-3.

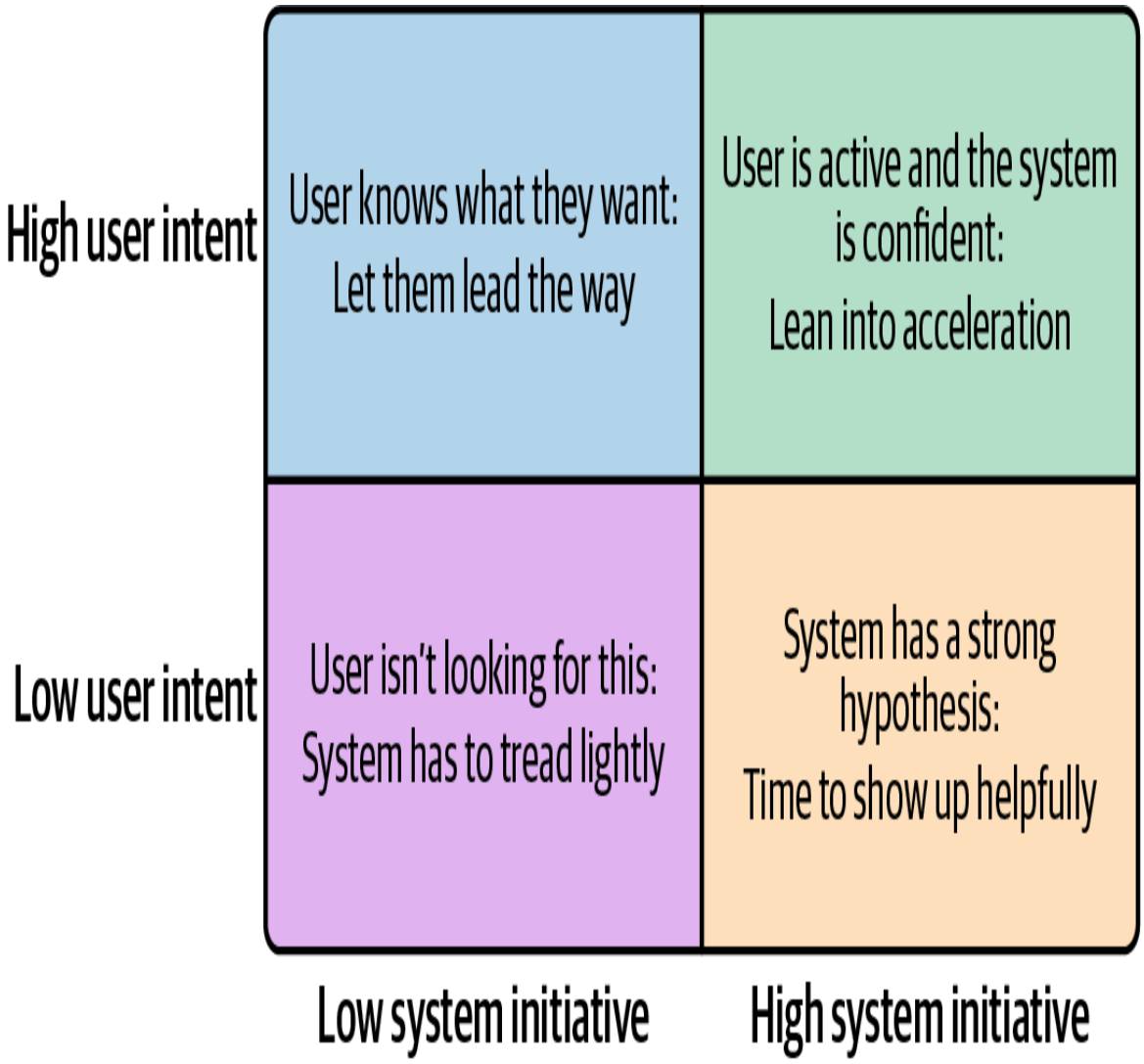


Figure 2-3. Matrix for AI feature discovery

Each quadrant suggests a different design posture. When there's low user intent and low system initiative, useful functionality may emerge socially or develop organically over time. In situations with low user intent but the tool has high initiative, the interface needs strong context cues to act helpfully without feeling intrusive or overwhelming. When users demonstrate high intent but the system has low initiative, they're actively looking for something but may not know exactly how to find it, so the design should make discovery and activation as frictionless as possible. Finally, when both user intent and system initiative are high, the goal is to accelerate what users are already trying to accomplish, removing barriers and streamlining their workflow.

The following subsections are a taxonomy of current discovery patterns, grouped by how the feature is revealed. These aren't mutually exclusive, and many tools use a combination.

Input-based discovery

The most direct way to surface features is through the input itself. These patterns build on familiar entry points, such as typed commands, gestures, or empty states, and turn them into moments where users can discover what the product is capable of. Because they appear where attention is already focused, they make discovery feel like a natural extension of the user's own action. Following are some examples of these patterns:

Command triggers

> This is when users type specific commands to invoke AI actions (see Figure 2-4). The user sees a drop-down menu of available commands as they type, similar to autocomplete suggestions. These patterns create a sense of power and control, letting experienced users bypass visual interfaces entirely while maintaining discoverability for newcomers, who can explore available commands.

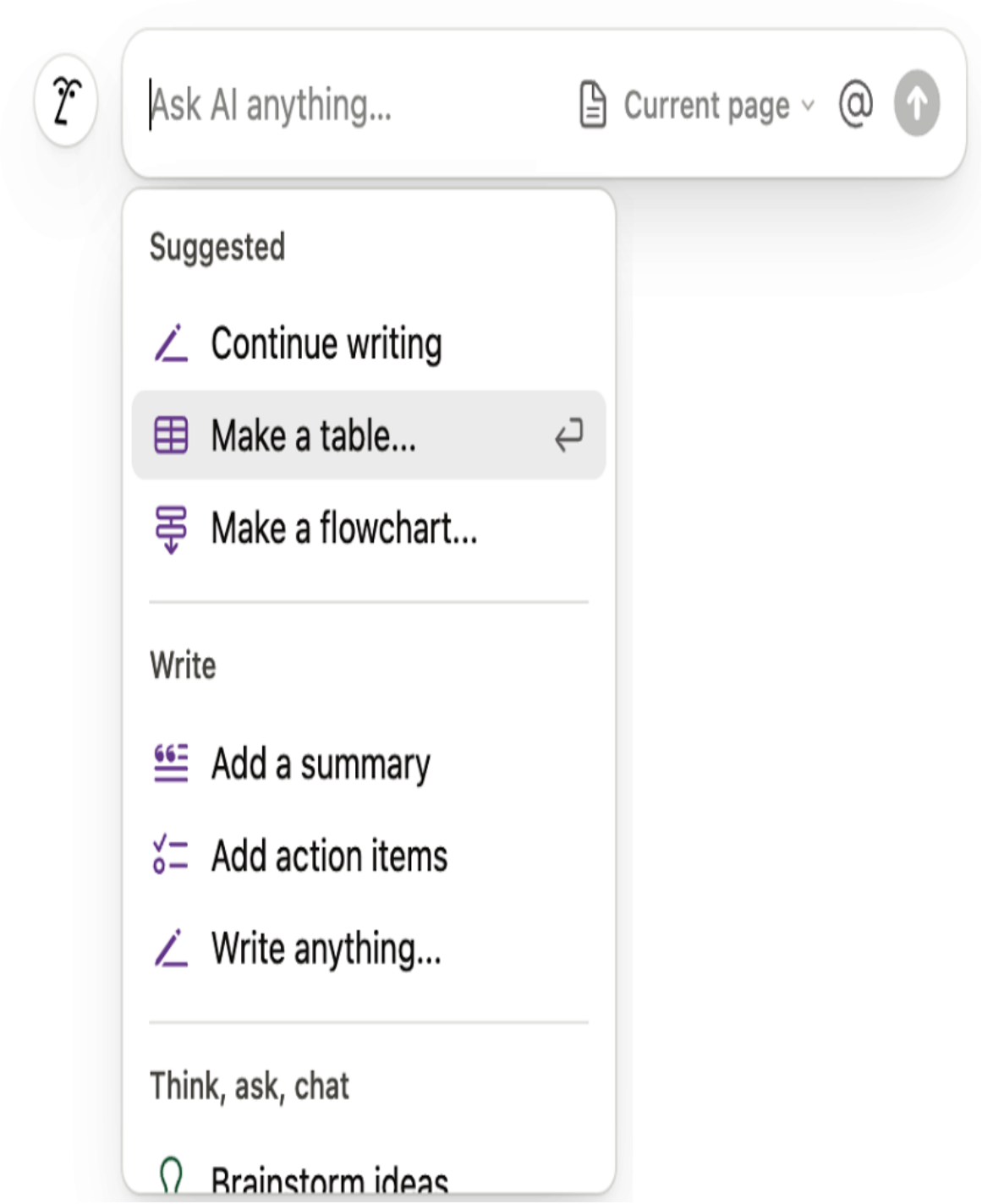


Figure 2-4. Notion's AI palette is opened with a press of the space bar

Gesture/interaction patterns

This is when users trigger AI features through familiar interactions like right-clicking or selecting text. For example, in Word, Copilot options

can appear in the context menu when a user highlights a paragraph. These patterns feel natural because they build on existing behaviors, but the features can remain hidden unless onboarding helps users discover them.

Empty state activation

This is when users open a blank document or spreadsheet and see AI-powered suggestions like “Create a budget?” or “Start with a meeting agenda template?” displayed prominently in the empty workspace. These moments capitalize on the anxiety of the blank page, offering direction precisely when users are most open to guidance.

Context-aware discovery

Some discovery patterns work by paying attention to the user’s environment rather than their explicit input. The model can notice what kind of content is being worked on, how often certain actions repeat, or where the user’s focus is within the interface. By situating assistance in context, these patterns make features feel relevant and well-timed rather than random or disruptive. There are a variety of invocation methods:

Content analysis

This is when AI automatically detects the type of content being worked on, like recognizing a meeting notes format, and shows a contextual suggestion bubble saying “Summarize Action Items” that appears near the relevant content. The user sees the product understanding their work context and offering appropriate next steps without being asked.

Behavioral pattern recognition

This is when a user repeatedly performs the same action, like pasting identical boilerplate text into emails, and Excel shows a notification suggesting “Create a rule?” The user sees the product learning from their patterns and offering to automate repetitive tasks they may not have consciously recognized as inefficient.

Location-based

This is when a user hovers the cursor over a spreadsheet's totals row and sees a small tooltip suggesting "Add Formula," or clicks into an empty cell and sees relevant function suggestions. The user experiences contextual help that appears exactly where their attention is focused, reducing the cognitive overhead of remembering where features live.

Ambient/proactive discovery

Other patterns step in before the user asks for help. By watching for potential errors or anticipating needs, the product can act like a background safety layer that surfaces guidance only when it is most useful. This shifts the role of AI from something users must seek out into something that implicitly supports them as they work. Examples include:

Anomaly detection

The user sees the product acting as a quality control layer, catching potential problems before they become issues, such as Google Calendar displaying an alert when two meetings will clash, or Gmail reminding a user to attach a file they have mentioned.

Predictive surfacing

Visiting a web page in a foreign language on the Chrome browser will trigger a message offering to translate the page (see Figure 2-5). With this feature, the user gets just-in-time assistance that recognizes their needs before they have to explicitly ask for help, reducing the number of steps between problem recognition and solution.

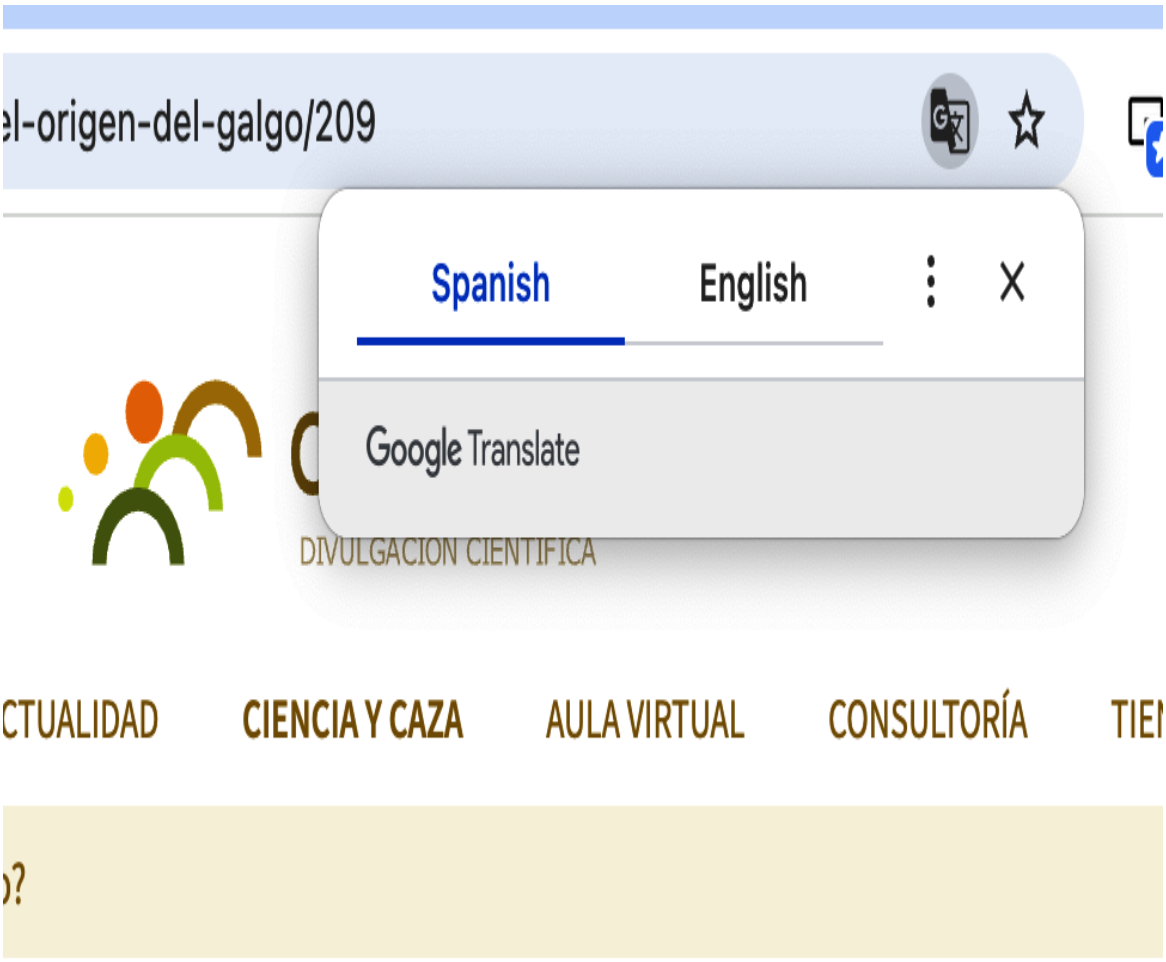


Figure 2-5. Google's predictive surfacing window provides just-in-time translation

Progressive discovery

Some features are best revealed slowly. Progressive discovery introduces functionality in stages, giving people time to build confidence before encountering more advanced options. Instead of overwhelming new users with a full menu of capabilities, the product reveals features in moments of clear relevance, helping them feel like natural extensions of what they already know. For example:

Feature graduation

> A writing tool may initially offer grammar correction as a suggestion, then later promote it to a visible menu option once adoption is proven. This type of exposure adapts over time, supporting gradual familiarity.

Contextual tutorials

> Consider when a user uploads a portrait photo and immediately sees a Background Remover tool appear in a highlighted sidebar panel with a "New!" badge. The user gets introduced to advanced features only when they're immediately relevant, avoiding cognitive overload while maximizing the chances they'll try new capabilities.

Once you know what your feature does and how it's intended to show up, you can place it on the 2 × 2 grid from Figure 2-3 to reflect its discovery posture (Figure 2-6). This chart makes it easier to see whether a feature's current exposure mechanism matches its intent and initiative. It also helps you plan complementary discovery moments, since features that are surfaced socially could later graduate to ambient or context-aware prompts.

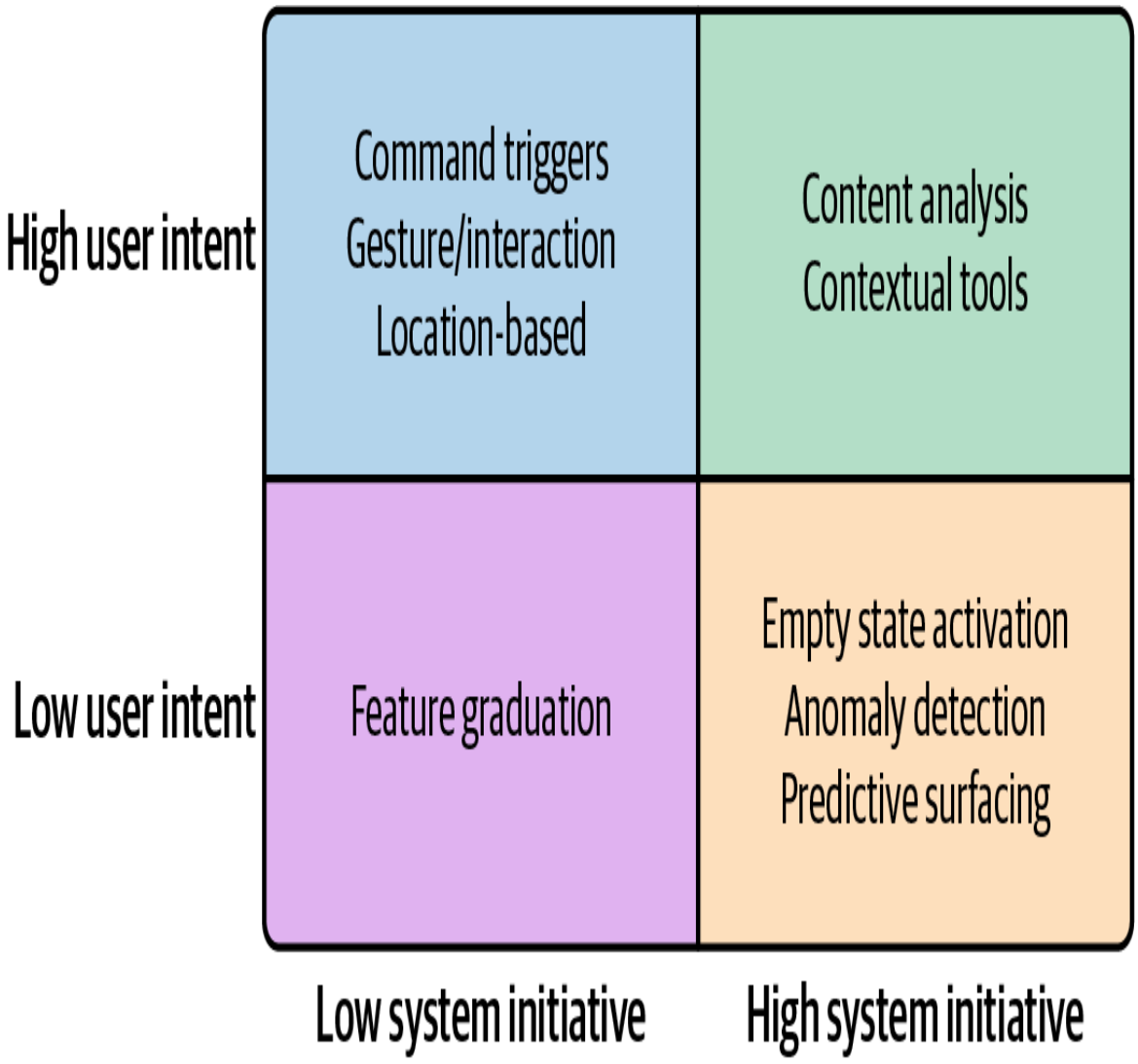


Figure 2-6. Types of feature exposure mechanisms to favor based on user intent

Together, these patterns illustrate the range of how users discover, learn, and reencounter AI functionality. Discovery represents an ongoing conversation between the user's intent and how the product responds, where the feature either becomes part of the user's toolkit or fades into the background.

Adoption and Momentum Behavior

The path from discovering a feature to successfully using it is fraught with behavioral and design challenges that can make or break user adoption. Research consistently shows that feature discovery is one of the strongest

predictors of software adoption and engagement. A 2017 Clutch study found that 72% of app users said completing all onboarding steps in less than a minute is important in their decision to keep using an app. The same research revealed that if an onboarding experience takes longer than 2 minutes, many people give up altogether, meaning you have just 120 seconds to onboard mobile users or lose them forever. This impatience reflects Hick's Law, which states that the time it takes to make a decision increases with complexity and the number of choices available.

Compounding this challenge is *momentum behavior*, a psychological phenomenon where users stick to their chosen path through an interface, even when better options are available. The Nielsen Norman Group explains that momentum behavior occurs most often in digital products "when parts of the interface design are not strong enough to call to users when they need them", meaning users experience tunnel vision with their current approach and overlook features that could help them. This creates a double bind: users need to discover features quickly due to time pressure, but once they've chosen a workflow, they're likely to ignore alternative approaches that might be more effective.

After users discover an AI feature, the next opportunity emerges: guiding them toward confident, successful use. Designers can work to overcome momentum behavior and time constraints by balancing strategic moments of intervention with comprehensive education. This approach works best when supported by accessible documentation and tutorials that let users learn at their own pace outside the immediate workflow.

Users bring different learning preferences and time constraints to a task. Some need quick reference guides during active work, while others prefer comprehensive tutorials they can work through when they have dedicated learning time. Some platforms manage this by fostering third-party ecosystems, where power users can publish their workflows for others to learn from, creating a community-driven knowledge base that evolves alongside the AI capabilities. These user-generated resources often bridge the gap between official documentation and real-world application, showing creative uses that product teams might never have anticipated. For example,

a recent Anthropic job posting sought a *demo designer*, showing that even AI orgs themselves are looking to create case study content. When community contributions are properly curated and integrated into the product experience, they transform individual expertise into collective knowledge, making advanced techniques discoverable for users who are ready to expand their skills.

A word about the speed of innovation: when thinking about discovery, it's important to consider that AI features currently exist in a rapid state of change, requiring designers to manage a continuous feature-adoption pipeline. Traditional software follows a predictable cycle: features are developed, released, documented, and remain largely unchanged for years. AI capabilities, however, exist in what we might call a *plastic* state, where they're continuously refined through model updates, new training data, and new feature opportunities. This means the adoption pipeline of priming, activation, and mastery happens repeatedly rather than once.

Users may discover a summarization feature that works one way in January, only to find it's been enhanced with citation capabilities by March, requiring them to relearn a portion of their mental model and workflow. This is worsened by *algorithmic transference*, where users carry expectations from previous interactions with AI products into new encounters, often applying assumptions about errors or issues that may no longer apply. Needing to manage these expectations raises the bar for messaging about updates; it also means that designers and their product teams must anticipate confusion and unease about changes to (recently) established workflows.

Once users have discovered AI capabilities and begun incorporating them into their workflow, they face a different kind of challenge. The interface that helped them find and activate features now needs to support ongoing decisions about how those features actually work. Some of these decisions are made explicitly by users, such as whether the AI can access certain files or how outputs should be saved. Others are handled by defaults, like which model is running in the background. Generally speaking, only advanced users typically experiment with different models. The broader principle

remains that AI products place ongoing choices in front of people, and the interface plays a critical role in whether those choices feel clear or opaque.

Orchestration

After a user defines a task to be done, they arrive at the interface with a mixture of expectations, some clear and some implicit. They might engage with onboarding, skim a tutorial, or dive straight in without any hand-holding. Whatever their approach, they begin making a series of decisions that can dramatically shape how the tool behaves for them. These decisions often happen early, but their effects can ripple throughout the entire session or project.

In AI tools, this layer of orchestration can be especially fraught. Users may assume the product is smart enough to anticipate what they need. Experienced users might expect to skip setup entirely, while newcomers may not know what to look for or even that configuration is possible. Despite the fact that AI is marketed as *highly intelligent*, it's important to remember how dependent good experiences are on well-scaffolded configuration.

There are a number of elements users can adjust to shape their query and experience, including the following:

Model, tool or agent selection

> Some tools offer multiple models, tools, or agents with different strengths, limitations, and latency trade-offs. Users might choose a faster model for lightweight tasks or a more capable one for in-depth work. When framed well, choice gives users a sense of control and clarity about what is happening behind the scenes, but it represents another layer of knowledge to gain.

Usage and metering

> Transparency about quotas, rate limits, and usage caps helps prevent surprise interruptions or costly overages. Whether it is a token counter, a

session timer, or a monthly usage bar, these indicators give users a sense of how far they can push the tool and how to pace themselves.

Permissioning

Access and continuity are shaped by how permissions are handled. This includes who can see what, whether conversations persist across sessions, and whether an AI can retain context over time. Users need confidence that their data will not disappear, leak to the wrong place, or behave unexpectedly.

Library management

Files, assets, and project materials, once uploaded, should feel like part of the fabric of the workspace. That means offering structure such as version history, metadata, and threading where appropriate. But it also means being able to quickly retrieve, reuse, or update materials without having to reupload or reexplain them.

Across all of these dimensions, users rely on tools to consistently surface the right level of control. This is tricky in AI products since, as user intent is not always explicitly stated, assuming continuity and treating a new query as part of a previous task may be drawing the wrong conclusion entirely. Interfaces need to be careful not to infer too much, while still offering opportunities for users to carry forward context when they want to. Getting this balance right is a critical part of making orchestration feel lightweight rather than laborious, and helping users feel like the tool is adapting to them, not the other way around.

Before diving into each element in detail, I'd like to mention a few overarching principles:

Orchestration should be intentional

In AI products, each choice meaningfully shapes how the user experience enhances the input and output. Whether a user is selecting a model, granting file access, or choosing whether to save conversation

history, those decisions have a direct impact on the outcome. That means users need to make choices with a clear sense of what they are choosing and why. Products can support this by presenting options at meaningful moments, with enough information for users to understand what each choice enables or limits.

Orchestration should be transparent

Users need to understand what they can and cannot do, and how that changes based on their setup. For example, when switching from one model to another, there should be some visible signal that capabilities have shifted. If a new permission is granted or a new plug-in is turned on, users should see what has changed. This helps prevent unrealistic expectations, but it also supports discovery by highlighting new capabilities the user might not have thought to ask for—even in their absence.

Orchestration should be recognizable

If something changes, the user should notice. If they return after a few days, they should be able to recall or quickly reorient to what is available. That means labeling choices clearly, providing previews or examples when possible, and preserving a sense of continuity. Users should be able to form a mental model of how the tool behaves in its current state, and that model should stay stable until the user changes it.

With those principles in mind, the next sections walk through each of the core elements of AI configuration.

Models, Agents, and Tool Selection

Completing a task successfully means matching the right tool to the need. Looking across today's AI products, you'll find three fairly distinct layers of capability: models, tools, and agents. Each serves a different purpose, and understanding how they build on one another can help clarify what kind of interface each one actually needs.

If we visualize these as a triangle, in the style of Maslow's hierarchy of needs, *models* represent the foundation. They are reactive, in that they respond to a prompt but do not pursue a goal or take action on their own. Consider a model used inside a writing assistant. When a user begins drafting an email, the model suggests completions based on common phrasing, anticipated tone, or past behavior in similar contexts. It can fill in subject lines, adjust for formality, and rewrite sections on request. However, it does not retrieve background documents, nor does it schedule a follow-up or adapt to changes in recipient information.

In the next tier we have *tools*, which extend a model's reach, making it possible for the product to act in more grounded and timely ways—but the user still initiates and directs each request. Instead of relying only on what the model learned during training, a tool lets it perform specific actions in the world. This could mean searching the web, reading a document, querying a database, or pulling real-time data from an external system. Tools act as bridges between the model and other software or services. ChatGPT with browsing enabled is a familiar example: when a user asks about a current event, the model uses a tool to search the web and returns information based on what it finds.

Agents add a layer of planning on top of models and tools. This is where tools begin to act with more autonomy. An agent interprets a user's goal, breaks it into smaller tasks, decides what to do first, and chooses which tools to use at each step. It can adjust as it goes, retrying if something fails or changing course based on new information.

> **TIP**
>
> Confusingly, some products refer to chatbots as "agents." There is some ambiguity in the term, so be very clear when you use it within your product. Some prefer to use "agentic," which can imply planning capability more clearly without associating a task or action with a chatbot.

Planning can involve a number of different strategies. Some agents plan all steps up front, creating a sequence to follow before doing anything. Others plan incrementally, deciding what to do next based on what just happened. In either case, planning is what allows an agent to move beyond the prompt-response rhythm of a typical model interaction. The system might look at a complex request, like building a report, and structure its work across multiple steps. It might also track dependencies, manage resources like time or API calls, and maintain a working memory across turns. This kind of behavior introduces new complexity, but it also makes the system more useful for sustained tasks that require coordination or iteration.

Each layer of this stack has different impacts on the user's query, and therefore requires different messaging and styling. Working with a base model means focusing on input and output, formatting, and feedback about generation. Tool integrations require more support around permissioning, confirmation, and visibility into what was accessed and why. Agents introduce a deeper set of concerns: how to show what the system is doing, how to let the user pause or redirect it, and how to build trust in a system that is taking action without explicit step-by-step instructions.

Because there is a clear dependency structure across these three types, which can be easily confused, it's useful to visualize it as a stack, as shown in Figure 2-7.

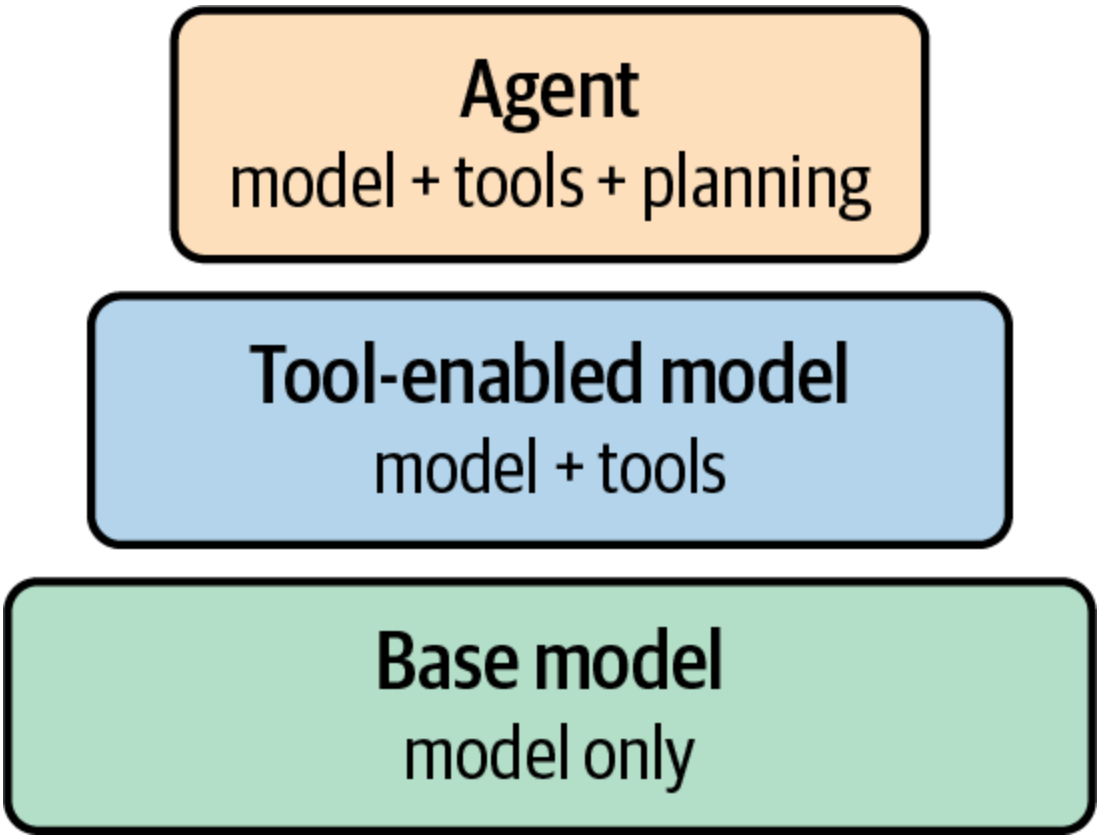


Figure 2-7. Model dependency structure

To see how this plays out, here is how a single user prompt might be handled across the three levels. Consider this prompt:

```
I have a client meeting tomorrow with Salesforce about
integrating our CRM data. Can you help me prepare?
```

Model only

The system replies with general advice: review the client's background, create talking points about integration capabilities, prepare for questions about security and API compatibility, and bring case studies from similar implementations.

Model + tools

The system can now gather specific information. It looks up Salesforce's latest product announcements, pulls your meeting details

from the calendar, finds relevant case studies from your company database, checks who the attendees are, and includes up-to-date API documentation. You still guide the interaction, but the system returns more tailored content.

Agent

The system takes initiative. It blocks off time on your calendar for prep, drafts a presentation using relevant integration examples, compiles a list of known stakeholders and their technical roles, creates reminders to review key materials, and generates follow-up emails for different meeting outcomes. It might continue refining this plan as new information comes in or as the meeting approaches.

As we've seen, a major challenge lies in communicating to users what is the best orchestration approach to fulfill their requirement. Additionally, each level comes with design implications that impact how the user comprehends and utilizes the tool.

Design considerations for models

An AI product may include a number of available models, each optimized for different types of tasks and performance characteristics. The challenge for designers lies in presenting model selection in ways that help users make informed choices without overwhelming them with technical details they may not understand or care about.

Model differences typically center around several dimensions that matter to users. Response speed varies significantly between models. Some prioritize quick replies for conversational flow, while others take more time to produce thoughtful, detailed responses. Specialized capabilities also vary, such as coding proficiency, mathematical reasoning, creative writing, or multilingual support.

The most straightforward approach to model selection uses drop-down menus that present these options clearly, but it's vital to consider how the differences are presented. Consider how shipping options work during

checkout. Instead of simply naming the delivery service, they provide insight into the actual thing the user wants to know: how long their purchase will take to arrive, such as standard (five to seven days, free), express (two to three days, $8), and overnight ($25). Each choice clearly communicates its time and cost implications. Model selection can follow similar patterns by focusing on outcome-oriented descriptions rather than technical specifications. Instead of offering “GPT-4,” “Claude Sonnet,” and so on, interfaces might present “Great for quick responses” or “Specialized for coding tasks.”

Many successful AI products have moved toward intelligent defaults that automatically route queries to the most appropriate model. They analyze the user’s input to detect signals about what kind of response would be most helpful: a request for current stock prices might automatically trigger a model with web search capabilities, while a complex reasoning problem might route to a model optimized for step-by-step analysis. This approach reduces cognitive load for users while still delivering appropriate capabilities.

> **TIP**
>
> The challenge with automatic routing lies in transparency and user agency. If a choice is made invisibly, users may not understand why responses vary in speed, style, or capability across different queries. A small badge persistently displaying the model and its benefits helps users understand why a response might take longer or have access to different types of information.

Model selection is generally a more complex proposition in AI-first tools, since the user intent may be more ambiguous. Someone typing “Help me with this presentation” could mean anything from creating slides from scratch to polishing existing content. By contrast, AI-enhanced tools embed AI within specific contexts that reveal user intent more clearly. When someone selects text in Microsoft Word and chooses “Summarize,” the context provides strong signals about what kind of AI assistance is needed.

As we've seen, the breakneck speed of model development means that there are sometimes clashes between business needs and user experience. This can have an impact on interface decisions. For example, OpenAI has developed a large suite of ChatGPT models that are presently displayed under a drop-down selector, as shown in Figure 2-8. However, this method of presentation is regarded by some users as suboptimal, with OpenAI CEO Sam Altman admitting to issues with the high number and method of displaying models. Finding the right balance between overwhelm and discoverability relies on having a clear understanding of what users are trying to achieve and what has been shown to drive successful outcomes—a great candidate for UX research.

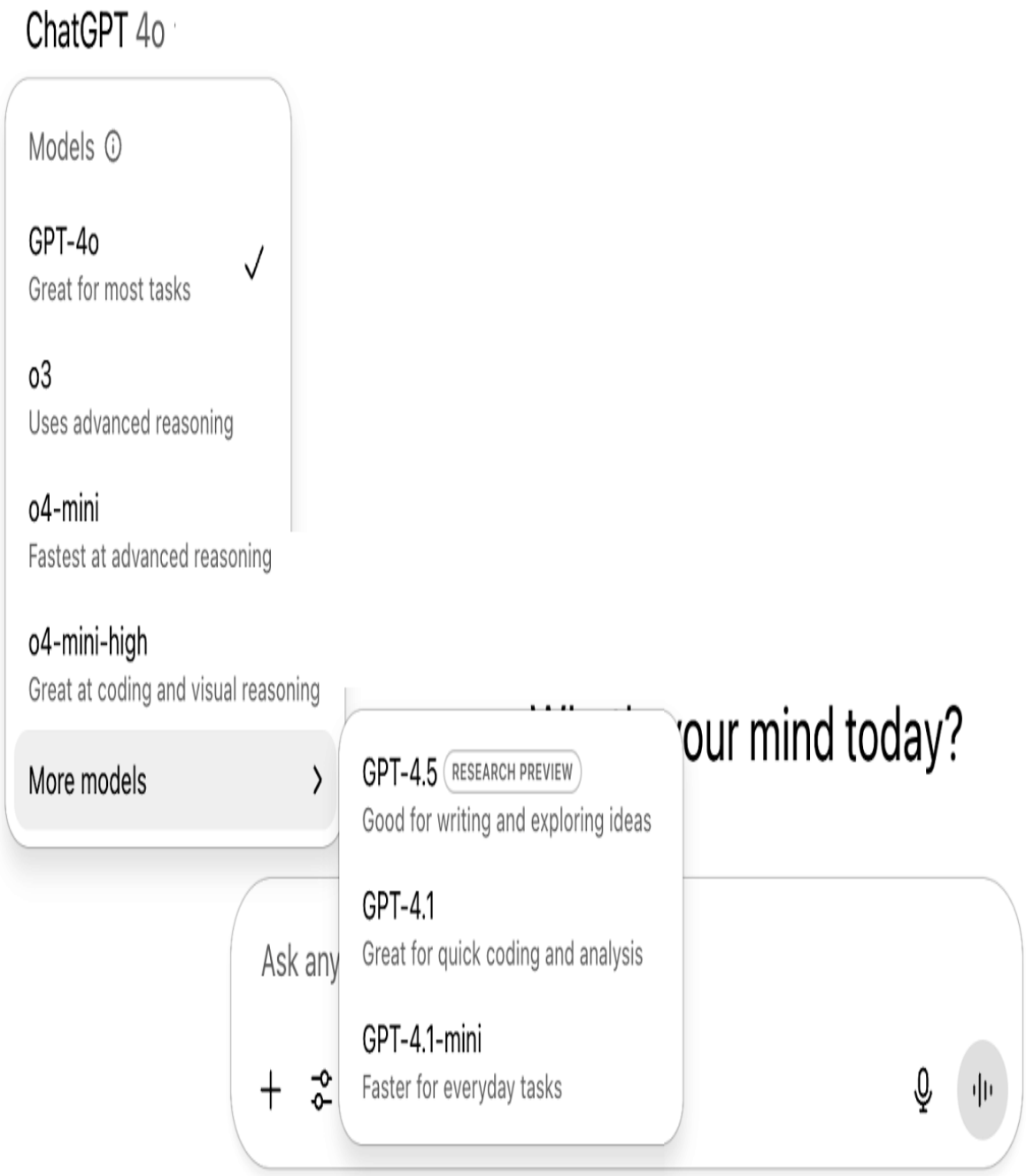


Figure 2-8. ChatGPT's model selector

Design considerations for tools

Tools represent discrete capabilities that extend what models can do, connecting them to external systems and real-world data. Unlike models, which process and generate content, tools are augmenting forces that help the model perform specific actions: searching the web, reading files,

executing code, or querying databases. A useful way to think about them is as APIs the model can call to get more information or carry out a job. Each tool type brings its own considerations for permission management, result presentation, and error handling.

As the industry matures, we are beginning to see standards emerge that make the interplay between tools and AI models more effective and maintainable. Frameworks like the Model Context Protocol (MCP) formalize how a tool describes its capabilities, what information it requires, and how that information should be structured so the model can use it. You can think of this as a simplified API that is specialized for use by models.

Common tool categories include information-retrieval tools, like web search and document access; computational tools, like calculators and code-execution environments; and integration tools that connect to external services, like calendars, email, or specialized databases. When designing a product that has so many different tools available, it's common to abstract these into a *palette* for the user to review and engage with.

Tool palette design follows several established patterns, each with distinct trade-offs. Sidebar approaches, similar to those in design tools like Photoshop or Figma, present tools as a persistent set of options that users can select manually. This makes capabilities transparent and puts users in control, but it can feel cluttered and may interrupt conversational flow. Contextual activation hides tools until they are needed, detecting when specific capabilities would be useful and surfacing them automatically. That keeps interfaces clean but relies on the product correctly interpreting user intent.

Many successful implementations use hybrid approaches that combine automatic tool activation with manual override options. The system might automatically search the web when detecting queries about current events, while still allowing users to disable web search or explicitly request it for other types of queries. These interfaces often use lightweight cues like natural language commands ("search the web for…") or syntax hints ("@web" or "/search") to give experienced users precise control.

User preferences around tool usage deserve particular attention. Some users prefer to maintain control over which tools get activated, while others want seamless automatic behavior. Interfaces should allow users to opt in and out of preserving these preferences, creating personalized tool behavior that adapts to individual comfort levels and use cases. A user might enable automatic web search by default but require explicit permission for calendar access, reflecting different levels of trust and privacy concerns across tool types.

Increasingly, models are capable of suggesting or autonomously engaging a tool without the user's direct suggestion. For example, in Gemini a user can select "Nano Banana" to generate an image as part of their query, but if they ask for an image in their prompt, the model will default to enabling the image tool. This has metering and user experience implications—if this isn't something the user wanted, it's important to let them change this. ChatGPT uses an "escape hatch" for queries that default to deep reasoning, since it's more computationally intensive: users can click "get a quick answer" if they don't feel they need the additional computing power. Other products provide an opt-out solution in configuration, where users can stipulate what tools they *don't* want to use—though this again raises the issue of continuity, since a user might forget they chose this setting and be disappointed by the capabilities of the tool.

Design considerations for agents

Agents (shown in Figure 2-9) introduce the most complex design challenges, because they combine models and tools with autonomous planning capabilities. Unlike the request-response pattern of model interactions, agents work toward goals over multiple steps, making decisions about what to do next based on their assessment of progress and changing conditions. Since this represents a significant expansion of the overarching input-computation-output structure, it is covered in much more detail in Chapter 6.

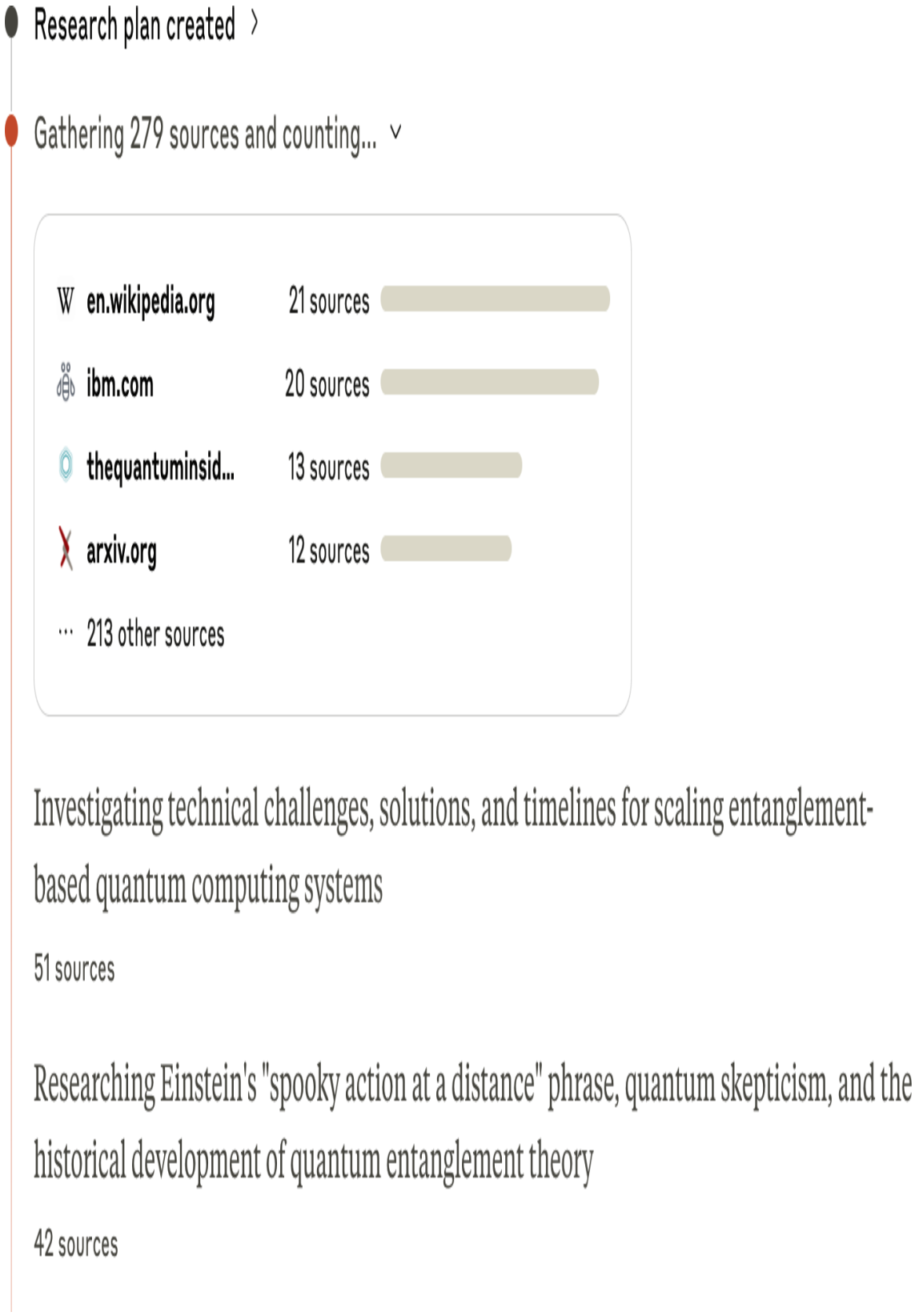


Figure 2-9. Chain-of-thought patterns in Claude

Usage and Metering

In 2013, a team from Facebook's Android engineering team traveled to Africa to review mobile performance in developing countries. There, they learned that the data-heavy app they'd created was prone to crashes due to local bandwidth constraints, and that devices available on the local market lacked the necessary storage, also leading to crashes. As a result, the team burned through their monthly data plan in 40 minutes.

Cost per pageload isn't a commonly surfaced consideration in product design. An app can make dozens of third-party API calls—payment processing, address validation, image recognition, and so on—but these usually result in a stable, definable cost.

With AI, the variability of the cost of a task can be extreme. A simple text completion might cost a fraction of a penny, whereas a complex reasoning task could cost dollars. The same interface element can trigger wildly different computational costs depending on what the user types and how the product responds.

As a result, many AI tools assist users in working within usage boundaries, often tied to pricing tiers, model constraints, or technical limits. These boundaries shape what users can do and for how long. For example, some products may restrict access to more intensive capabilities like long-context reasoning or multiturn memory, offering only a set number of these sessions per day or month. Others might meter tokens or cap the number of concurrent projects or file uploads, or handle this entirely in the background without showing it to the user.

Converting the dollar value into another token is a common way of softening negative feelings about cost. The image-generating tool Visual Electric converts generation costs into "volts" and discloses the cost of each query as a tooltip upon submission, as shown in Figure 2-10.

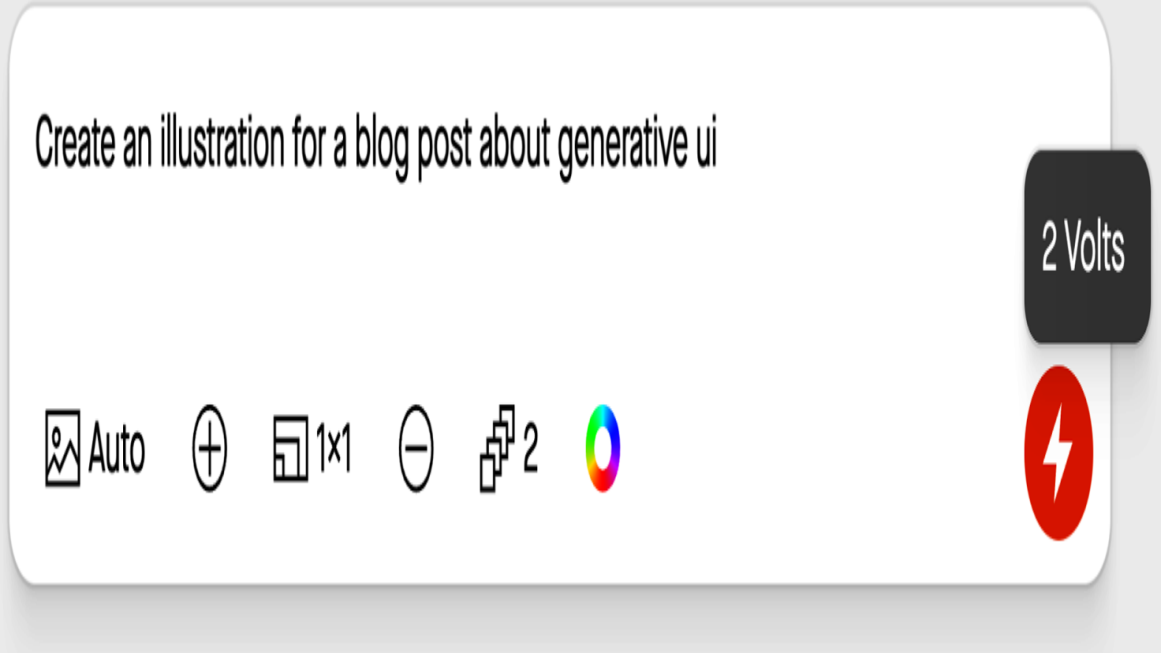


Figure 2-10. Visual Electric ensures users know the cost of each query upon submission; since the user gets a certain number of "volts" per month, this allows them to visualize how many each query uses

This style of metering benefits from up-front disclosure. If a user has access to five deep reasoning sessions per day, it's preferable for them to know that before they start using one, like the approach used in ChatGPT's configuration menu, shown in Figure 2-11. If a model becomes slower or less accurate after a certain quota is reached, the interface should communicate that shift. This is especially important in tools that encourage frequent experimentation, since without some guardrails or signals, users may unknowingly use up valuable capacity on low-priority tasks.

Resumability is also key. If a user runs out of credits, the product should preserve their progress and make it easy to pick back up once access is restored. That might mean saving the state of an in-progress session, queuing actions that couldn't complete, or offering lightweight summaries so the user can reorient themselves later. Without this, users are penalized twice: first by the interruption, and again by the effort it takes to reconstruct what they were doing.

Nobody likes talking about money, but AI is expensive to run. Despite this, usage and metering can be designed to support the user's exploration and

discovery, provided they have a clear sense of what the model can offer right now and what will be possible later, in order to plan their work and stay in control. Just as with other configuration elements, the interface should make usage visible, intentional, and easy to learn over time.



Figure 2-11. ChatGPT's interface showing metering limits for deep research

Permissions and Capabilities

Permission requests play an important role in AI interfaces because they determine how data is shared and, just as importantly, how much control

users feel they have in that process. This sense of agency becomes especially important as AI products handle increasingly complex and personal tasks.

The most immediate concern is scope of access. Unlike a traditional app that might ask for permission once to access your photos or contacts, AI features often need dynamic access to multiple data sources to provide useful responses. A tool might need to read your brand guidelines, access your design files, and potentially modify them. Each of these actions carries different levels of risk and user comfort.

Granular control becomes essential when dealing with data that carries different levels of sensitivity or relevance. Rather than forcing users into binary choices between all file access and no file access, you should design permission systems that reflect how people actually think about their data. Options like access to files you upload in this conversation, access to files in your project folder, or access to all files let users match the product's capabilities to their comfort level and specific use case.

The ability to revoke permissions should be as straightforward as granting them. Users need to easily see what permissions are currently active and modify them without hunting through nested settings. A clear permissions panel that's accessible from the main interface reassures users that they maintain control over their data and the product's capabilities.

It's challenging to create permission systems that feel like negotiations between partners rather than demands from gatekeepers. At the heart of permissioning is education, which ultimately leads to more trust and more effective user experiences.

Library Management and Assets

Library management involves the structured handling of files, datasets, prompts, reference materials, or saved conversations that may be used by the AI tool or by the user across sessions. Since AI products increasingly support uploading, generating, referencing, and reusing external content, how these assets are stored, surfaced, and retrieved is becoming

increasingly important. Decisions made here reveal a lot about a product's assumptions about the user's intent and context.

Some platforms treat uploads as ephemeral: a document is parsed once and then discarded. Others offer persistent libraries or asset drawers, where files remain attached to the workspace and can be revisited over time. Library design also affects how users manage versions and sources. If a user uploads a revised document, can the tool distinguish it from the earlier version? Can the AI reference multiple documents at once and attribute sources clearly in the output?

Here are some questions design teams could ask:

- What is the life cycle of a resource? Do prompts decay, do outputs get versioned, and can users archive or snapshot a session?
- How do assets travel between modes or tools? For instance, can a summarization generated in one thread be reused in another or attached to a project?
- How does search work across different asset types? Can users find a specific exchange within a long conversation or locate a document by its content rather than filename?

Ensuring that users can find their assets when they need them requires thoughtful organization patterns. Traditional file browsers may feel familiar, but they don't necessarily match how people think about AI-generated content. Additionally, the mental model of filesystems is increasingly challenged by cloud-based tools, where a visual directory structure can be deprioritized in favor of direct searchability. Designers may want to consider how to provide a surface for file disambiguation, or an enhanced search ability that provides additional support over and above a simple search.

TIP

The most successful library systems often mirror how users already organize their work. A design tool might group assets by project, while a research platform could organize by methodology or timeline. Consider how your users currently manage similar resources outside your product, as their mental models for organization can inform more intuitive digital experiences. Remember that library management isn't just about storage; it's about enabling users to build on their previous work and maintain continuity across sessions.

Summary

We began with a product that failed: Google Wave. Despite its technical depth, it left users unsure how to start, what it was for, or how it would fit into their workflows. The underlying capability was there, but it was inaccessible. The story of Wave captures something important about AI interfaces today: users often encounter unfamiliar tools that promise transformation but struggle with guidance or framing. A product can be powerful, but unless people can find the right entry points to employ it, it can't land successfully.

Throughout the chapter, we looked at the shape of capability: how models are trained, how they're evaluated, and how those characteristics become constraints or openings for design. Benchmarks like MMLU, GSM8K, and MT Bench offer partial signals, but high scores on an eval won't necessarily translate into meaningful user experiences. Even designers who have a general understanding of AI can find it difficult to imagine strong interactions without clearer insight into what models can actually do. The handoff from performance to experience depends on framing and exposure.

We also learned about how vital feature discovery is in shaping those experiences. Whether through ambient cues, content-aware triggers, or direct input, systems need ways to suggest possibility without interrupting the flow. Discovery works best as an ongoing relationship, especially in AI products where capabilities shift and expectations evolve, and can raise user confidence in achieving their goals. Orchestration helps achieve this further

by including configuration choices like model selection and permissions, and how the interface responds to shifting user intent. We also looked at momentum behavior, where users tend to stay in familiar patterns. This creates tension for AI features, since UI guidance is often most needed when users are least likely to seek it, instead of being discoverable without demanding attention.

In Chapter 3, we'll turn our attention to inputs. This is where user intent originates, whether through typed prompts, uploaded documents, or contextual cues. The way inputs are designed determines not just what users can ask but how clearly they understand what is needed from them.

OceanofPDF.com

Chapter 3. Designing for AI Inputs

What does it mean to speak to a computer? The idea of *input* has meant different things over time, changing as technological complexity has evolved. Early computing services were highly constrained, allowing for only simple, codified inputs that told the computer to run one program or another, requiring high user knowledge and technical skill.

The origins of inputs lie in the early days of computing. In the 1960s, a group of researchers at MIT developed a time-sharing operating system called Multics (Multiplexed Information and Computing Service). This experimental platform introduced many of the concepts we now take for granted in modern computing, such as hierarchical filesystems, dynamic linking, and virtual memory. But with regard to this chapter, its most important contribution was something simpler: the command shell. Figure 3-1 shows Multics in action.

A *command shell* is a text-based interface where users type instructions and the computer responds. When you type a command like **`ls`** or **`dir`** and hit Enter, the shell interprets that text, executes the corresponding program, and displays the results underneath. It acts as an interpreter between human language and machine operations, taking typed commands, figuring out what program you want to run, and then showing you what happened. If you've ever opened Terminal on a Mac (see Figure 3-2) or Command Prompt on Windows, you've seen a modern version of the command shell.



Figure 3-1. The command room and mainframe of a Multics installation (source: https://oreil.ly/hpmTw)

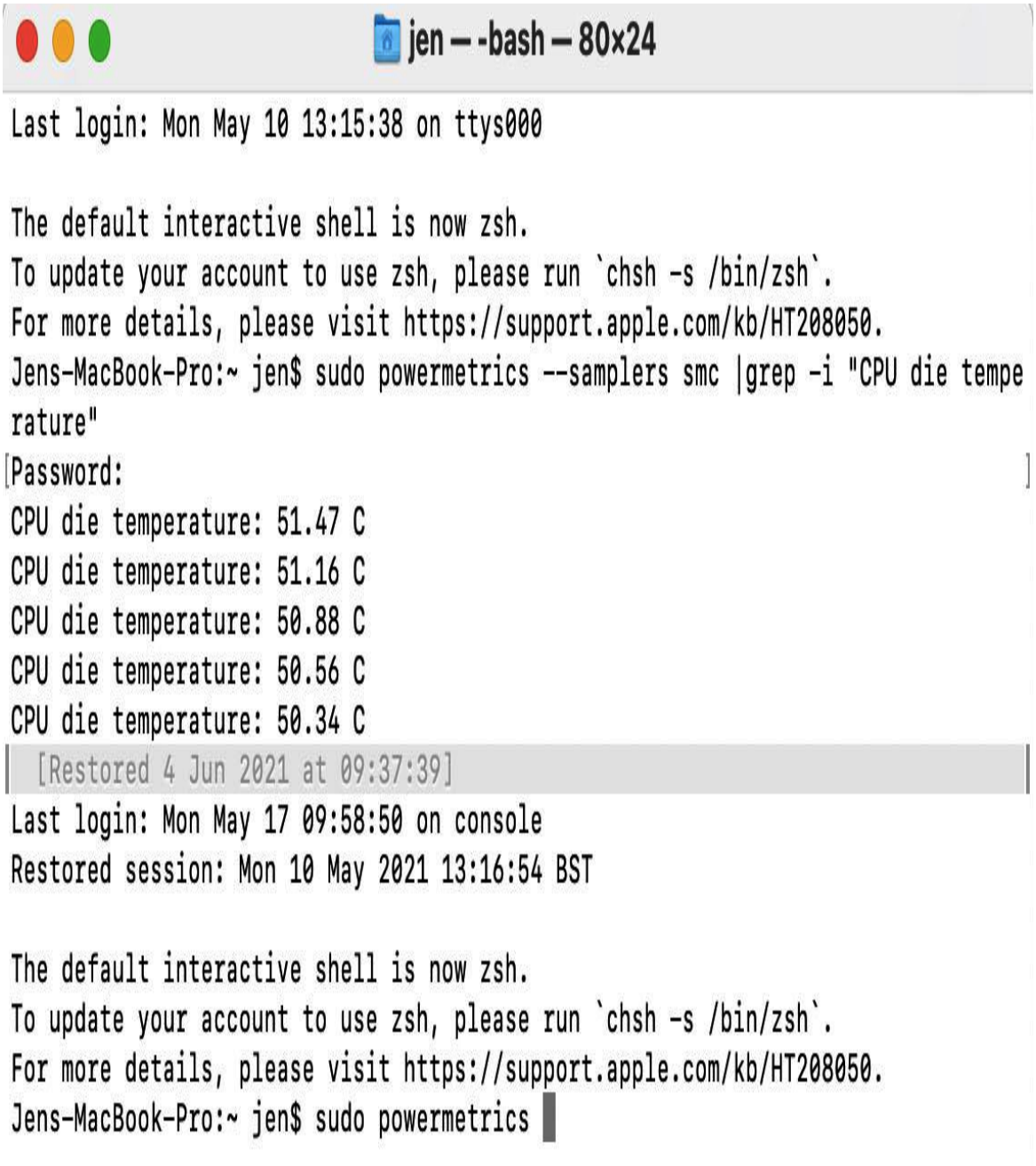


Figure 3-2. The Terminal interface on a Mac, a contemporary example of a shell

What makes shells particularly interesting from a design perspective is their nascent conversational nature. You type something, the computer responds, then it waits for your next instruction. This back-and-forth creates a dialogue, though one that demands precision, in a way that's markedly

different from what we might think of as *speech*. Vitally, the shell has no tolerance for typos or incorrect syntax. Miss a character or use the wrong flag, and you'll get an error instead of the result you wanted.

You also need to know exactly what words to say. The Multics command shell established a direct conversation between human and machine, but it was a conversation conducted entirely in the machine's preferred language. Users had to memorize specific commands and formatting rules that allowed for no deviation. A single misplaced character could render an entire command useless, forcing users to start over. This rigid framework, while powerful for those who mastered it, created an enormous barrier between human intent and machine execution. The *command-line interface* (CLI) represented a world where the burden of communication rested entirely on the human participant, demanding that users translate their goals into the computer's preferred vocabulary through extensive documentation, training, and practice.

Yet this constraint also drove remarkable innovation in precision and efficiency among those who became fluent in command-line syntax. As the technology became more sophisticated, the need for precise or predetermined inputs relaxed; the emergence of search engines marked the first significant departure from this strict command-response paradigm. Google and its competitors introduced the revolutionary concept that computers could interpret loosely structured queries and infer user intent from ambiguous or incomplete information. Users no longer needed to specify exact parameters or follow rigid syntax rules. Instead, they could type natural language questions and receive relevant results, even when their queries contained typos or imprecise terminology. That shift represented a fundamental philosophical change: rather than demand that humans communicate with a computer in code, search engines began the work of understanding human language.

Today's large language models represent the culmination of this evolutionary process, creating interfaces that not only interpret human intent but generate entirely new content in response. These systems engage in genuine dialogue, building on previous exchanges and maintaining

context across extended conversations. The transformation from command-based interfaces to conversational ones fundamentally redefines the user-system dynamic. Command-line interfaces were built around explicit instruction and execution, with the system responding predictably to user input. AI interfaces loosen that structure, introducing a relationship where initiative and decision making are distributed between the user and the system.

But that flexibility creates its own problems. The open input field looks inviting yet offers few clues about what to say or how much to provide. The system's fluency creates an illusion of understanding that can mask real limitations. Users are developing a new kind of literacy, much as they once learned to "Google well," but without the decades of accumulated intuition they built up around search.

This chapter explores how users communicate intent to AI systems, and how designers can support them in doing so effectively. It helps to recognize that this communication happens through several distinct channels, often simultaneously. There is *implicit context*: the information a system gathers from the environment without being asked. Examples include the document currently open, the text a user has selected, their location, the time of day, and what they did moments ago. There is *explicit prompting*: the typed or spoken request that tells the system what to do. And there is *direct manipulation*: the sliders, selections, buttons, and gestures that let users shape AI behavior through familiar interface patterns. A single interaction might involve all three. A designer working in Canva might have an image open (context), brush over an object to select it (manipulation), and type "replace with a golden retriever" (prompting). Each layer carries information. Each layer can succeed or fail independently.

The design challenge, then, is not simply teaching users to write better prompts. It is figuring out how these three channels can share the burden of translation and thus reduce what users need to articulate explicitly—while giving them enough visibility and control to succeed when the system's inferences fall short.

From Multics to Google Search: The Shifting Burden

While the command-line interface might not be in frequent use by designers today, the evolution in command-line style interfaces that followed it certainly is. The creation of Google, founded in 1998, illuminates a shift not just in interface design but in how users conceptualize control, syntax, and information retrieval.

As we've seen, the command line offered a powerful but unforgiving way to interface with a machine. Google Search offered a somewhat similar approach: a command issued, a response received. But where CLIs required fluency in a system's structure and vocabulary, Google invited a far wider range of inputs, with a safety net for new and naive users.

Whereas the interface of earlier search engines returned results based largely on keyword matching, Google prioritized *relevance* through link analysis, allowing users to find high-quality information that was previously undiscoverable. To that end, founders Sergey Brin and Larry Page created PageRank, an algorithm that assesses the importance of web pages by analyzing the network of hyperlinks that lead to and from them. Their inspiration arose from their academic roots: both Brin and Page grew up in academic families, with fathers who were professors and researchers. In academic publishing, the number and quality of citations a paper receives serves as a proxy for its importance and reliability within the scholarly community. Papers that are frequently cited by other respected works gradually build authority and influence. Page and Brin recognized that the web's hyperlink structure created a similar citation network, where websites could be evaluated based on how often other sites linked to them and on the authority of those linking sites.

This insight proved revolutionary, because it moved beyond simple keyword matching to evaluate the credibility and importance of web content. Traditional search engines of the era focused primarily on textual analysis, counting keyword occurrences and matching user queries to page content. PageRank introduced the concept that the web's own link structure

contained valuable information about quality and relevance. A page that received links from many other respected sites was likely to be more valuable than a page with identical keyword content but few incoming links.

The inviting style of the Google interface (see Figure 3-3) drew users to explore and play with search, preserving the command line's underlying pattern (you type something, the system tries to fulfill it) but removing the need for exact syntax. Users could type loosely structured questions, fragments, or even whole sentences. They could even misspell every word in the query, and it would be corrected (to the nearest guess). Google developed properties—like knowledge panels, embedded maps, definitions, and unit converters—that enhanced the experience further, making users feel that their queries were fully recognized and designed for. People often searched for quick facts, directions, conversions, or comparisons, and these modules brought the answer closer to the point of intent.

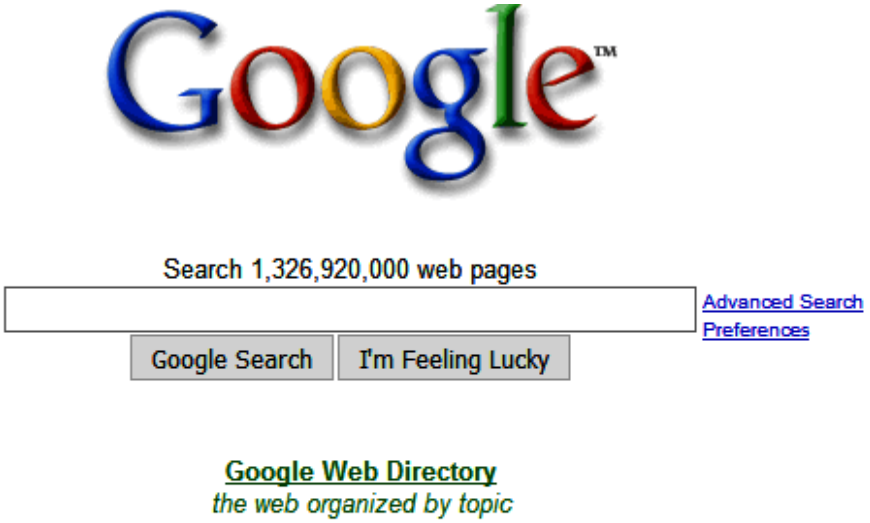


Figure 3-3. Google in 2001

Over time, this user behavior has become even more finely tuned. Users have come to anticipate that Google will know, for example, that if they

search “New York Boston,” they will be shown flights between New York and Boston—unless there’s a game between the Yankees and the Red Sox happening that day, in which case Google might surface details about the game. Search strategies have emerged, such as adding “filetype:pdf” to look for PDFs, or appending “Reddit” for results specific to that site, representing a learned grammar for shaping inputs, even in a natural language interface.

This recurrent optimization behavior reflects the knowledge gained in the nearly 25 years since Google launched: how to structure a good query for Google. And it’s exactly this knowledge that users *don’t* yet have for AI products. The input box of an LLM may resemble Google’s and may even return the same sources, but a vastly different technology is under the hood. What users need to learn now is how to *prompt* as effectively as they once learned how to *search*—and what information constitutes an input. The faster they gain that skill, the sooner they can move beyond fact-finding into tasks that search alone cannot handle: creating original explanations, adapting information to specific contexts, combining insights from multiple domains, and engaging in extended dialogue that builds understanding over time.

Three Channels of Intent

When a user interacts with an AI system, their intent travels through multiple channels at once. Some of this communication is explicit, such as the words they type or speak. Some is implicit, like the information the system gleans from context without being asked. And some happens through direct manipulation: the selections, adjustments, and gestures that shape what the system does without requiring verbal articulation. Understanding these three channels, and how they interact, is essential for designing AI inputs that actually work.

Implicit Context

Some of the most powerful inputs are ones the user never consciously provides. When you highlight a sentence and ask an AI to “make this clearer,” the system receives the sentence itself but also its position within the document, and often a good deal more: the paragraphs surrounding it, the formatting, and perhaps the title or filename. This ambient information shapes interpretation in ways that the typed request alone cannot. The same instruction, issued in a different context, might produce an entirely different result.

This dynamic is not new. Software has long drawn on context to anticipate user needs: autocomplete that learns from previous entries, spell-check that adapts to domain-specific vocabulary, or email clients that suggest replies based on the thread. What has changed is the depth and range of inference now possible. GitHub Copilot sees more than the line you’re typing; it reads the function above, the imports at the top of the file, and the naming conventions you’ve established across the project. It builds a picture of what you’re trying to do from the texture of the code around you. An AI assistant embedded in an email client might adjust its tone based on whether you’re replying to your manager or a vendor autonomously.

Because this is a discreet manner of input, it can go wrong. The model may be using different context than the user expected, so indicating what’s used is vital to prevent these issues. Some interfaces try to solve this by making context visible. A prompt field might display what the system is treating as input: “Editing: selected text” or “Reading: current document.” Others rely on spatial cues: highlighting the region under consideration, dimming what falls outside it. These signals help users build accurate expectations, and just as importantly, they create a surface for correction.

There is also the question of how much context is too much. A coding assistant that ingests your whole repository might produce more coherent suggestions, but it might also slow down, over-anchor on irrelevant patterns, or surface information you didn’t intend to share. Giving users

some visibility into this trade-off is an opaque process into something a user can reason about and, where necessary, adjust.

Explicit Prompting

Prompting is the practice of designing inputs to language models to elicit desired behaviors and outputs. While conceptually straightforward, effective prompting requires understanding how these models process and respond to text.

As we know, LLMs like GPT and Claude are trained to predict the next *token* in a sequence based on patterns learned from training on massive text datasets. When you submit a prompt, the model uses statistical relationships to generate what it calculates to be the most probable continuation of your input text. That probabilistic nature means prompting is fundamentally different from traditional programming.

Effective prompting leverages several key techniques that have emerged from both research and practical application. A simple prompt might be something like “Write a thank-you email.” But *effective* prompting often requires more detail about context, constraints, and desired outcomes. You might specify the recipient, the reason for thanks, the tone you want, or even examples of similar emails you’ve written before. The art lies in providing enough information for the AI to understand your intent without overwhelming it with unnecessary details.

Natural conversation relies heavily on shared context and implicit understanding. When you ask a coworker to “make this presentation better,” they already know your company’s style, your audience, and what “better” means in your specific situation. AI systems, on the other hand, have to reconstruct all of that from the words you provide unless the interface supplies that context through other means.

The most effective prompts tend to include relevant background information, clear descriptions of the desired outcome, and any important constraints or requirements. But there’s a balance to strike. Overly long, complex prompts can actually degrade performance, as the system struggles

to parse what matters most. Meanwhile, prompts that are too brief might produce generic or off-target responses.

What makes prompting particularly interesting as an interface pattern is that it inverts traditional software interaction. Instead of learning the software's language of menus and commands, users can express their intent in their own words. The burden shifts from the user having to memorize specific procedures to the system having to interpret human intention. But successful prompts in a product context also need to feel natural to create. Prompting that works beautifully but requires five minutes of careful crafting isn't sustainable for everyday use. The best input interfaces help users create effective prompts without thinking about prompt engineering, often by letting the other two channels (context and manipulation) carry some of the load.

The CARE Framework

The Norman Nielsen Group's CARE framework provides a useful methodology for understanding how effective AI prompts can be structured, breaking down the essential elements that bridge the gap between human communication patterns and AI system requirements. CARE stands for context, action, results, and examples.

Context forms the foundation of the CARE framework, representing the relevant background information that helps AI systems understand not just what users are asking but why they are asking it and how the response will be used. Context encompasses the user's role, their audience, their constraints, and the broader situation that gives meaning to their request. When a user asks an AI to "help me write a proposal," the system needs to understand whether this is a business proposal for potential clients, a research proposal for academic reviewers, or a project proposal for internal stakeholders. Each scenario requires dramatically different approaches, tones, and content structures. Effective interfaces help users provide this contextual information naturally, perhaps through brief setup questions, smart defaults based on user history, or prompts that encourage users to describe their situation before diving into specific requests.

Consider the difference between the following two approaches to requesting help with email communication:

Weak prompt:

```
Write an email about the project delay.
```

Strong prompt with context:

```
I'm a project manager at a software startup, and I need to inform
our client (a conservative financial services firm) that our
mobile app development will be delayed by three weeks due to
unexpected regulatory compliance requirements. The client has
already pushed back their launch date once before, and
maintaining their trust is critical for our ongoing relationship.
```

The contextual information transforms a generic request into something the AI can address with appropriate tone, level of detail, and strategic considerations. The system now understands the professional relationship, the sensitivity of the situation, and the communication challenges involved.

Action specifies what users want the AI system to actually do with the information they provide. This goes beyond simple task identification to include the type of processing, analysis, or transformation desired. Rather than asking an AI to “look at this data,” users benefit from specifying whether they want the system to summarize key findings, identify patterns and trends, compare different segments, or generate recommendations based on the analysis. Clear action specification helps AI systems choose appropriate methodologies and focus their processing power on the most relevant aspects of the task. Interface design can support action clarity through structured prompts that help users think through their actual objectives, example templates that demonstrate different types of actions, or progressive questioning that helps users refine their requests from general intentions to specific actions.

The power of action specification becomes apparent when comparing vague versus specific requests:

Weak prompt:

```
Help me with this sales data.
```

Strong prompt with action:

```
Analyze this quarterly sales data to identify which product
categories showed the strongest growth, which geographic regions
underperformed compared to
targets, and what seasonal patterns might explain the variations.
Then
synthesize these findings into three key strategic
recommendations for next quarter's sales focus.
```

The specific actions—identify growth patterns, analyze geographic performance, explain seasonal variations, and synthesize recommendations—give the AI clear direction about the type of analysis to perform and how to structure the response. This prevents the generic “Here’s what I see in your data” response that often results from vague requests.

Results address what the output should look like, including format, length, style, and level of detail. The results element recognizes that the same underlying request might require different presentations, depending on how the user intends to use the response. A summary of quarterly sales data might need to be a detailed analytical report for a management presentation, a concise executive brief for leadership review, or a visual dashboard for ongoing monitoring. Result specification helps AI systems tailor not just their content but their structure, tone, and emphasis to match user needs. Interfaces can facilitate result specification through format options, length selectors, or audience-specific templates that automatically adjust output characteristics.

The impact of result specification becomes clear when examining how the same request might be shaped for different outputs:

Basic request:

```
Summarize this market research report.
```

For an executive briefing:

```
Create a two-page executive summary that highlights the three
most
critical market opportunities, includes specific revenue
projections, and
concludes with clear go/no-go recommendations. Use bullet points
for key
findings and write in a confident, decision-oriented tone
suitable for
C-level presentation.
```

For a team meeting:

```
Develop talking points for a 15-minute team presentation that
explains the research methodology, walks through the key customer
insights, and generates discussion about implementation
priorities. Include questions to prompt team engagement, and
write in a collaborative, exploratory tone.
```

For client communication:

```
Craft a professional but accessible summary that demonstrates our
thorough
market analysis while positioning our recommended strategy.
Structure it as a client-ready document with clear sections,
supporting data, and next steps. Maintain a consultative tone
that builds confidence in our expertise.
```

These result specifications will produce fundamentally different outputs from the same source material, demonstrating how format, length, audience, and purpose shape AI responses.

Examples represent a powerful component of the CARE framework. Providing concrete samples of desired output helps AI systems understand user preferences and expectations in ways that abstract descriptions cannot capture. Examples work by showing rather than telling, allowing users to demonstrate the style, structure, and quality they seek without having to articulate these preferences explicitly. When a user provides examples of well-written product descriptions from their industry, the AI can infer tone, technical depth, feature emphasis, and formatting preferences that would be difficult to specify through instructions alone. Examples also help users clarify their own expectations, often revealing preferences they had not consciously recognized.

The power of examples becomes evident when comparing instruction-only prompts with those that include concrete samples:

Instruction-only approach:

```
Write product descriptions that are engaging and informative.
```

Example-enhanced approach:

```
Write product descriptions in the style of these examples:

Example 1: The Alpine Pro Series tent transforms your basecamp
into a fortress of comfort. Triple-reinforced corners withstand
80mph winds while the
revolutionary breathable floor system eliminates condensation
that ruins other high-end shelters. When every ounce matters and
failure isn't an option,
serious mountaineers choose Alpine Pro.

Example 2: Your morning coffee ritual deserves the Precision Brew
X1. This isn't another expensive gadget-it's the culmination of
three years perfecting temperature curves that unlock flavors
hiding in your favorite beans. The ceramic heating element
maintains 203°F throughout the brewing cycle while the integrated
grinder preserves volatile oils that paper filters destroy.

Match this confident, benefit-focused tone while emphasizing
```

```
technical
credibility and emotional connection to the product experience.
```

The examples communicate tone, structure, technical detail level, and emotional appeal more effectively than any list of instructions. They show the AI exactly what *engaging and informative* means in this specific context.

While these four elements—context, action, results, and examples—aren't necessary in every prompt, it's useful to consider how they could come together in a comprehensive prompt:

```
I'm preparing for a board presentation next week where I need to
propose
expanding our customer service team. [Context] Analyze our
current support ticket data from the past six months to identify
peak volume periods, average resolution times by issue type, and
customer satisfaction trends. Then create a data-driven argument
for why we need additional headcount. [Action] Format this as a
10-minute presentation outline with key statistics highlighted
for slides, compelling talking points that address potential
budget objections, and a clear ROI calculation that justifies the
investment. [Result] Here's an example of how our CEO prefers
data presentations: [Example of previous successful board
presentation format] [Examples]
```

This prompt integrates context (board presentation, budget considerations), actions (analyze data, create argument), results (presentation format, specific structure), and examples (CEO preference) to give the AI everything needed for a highly targeted response.

Prompt Intent

Users may have a range of goals with a prompt that aren't yet revealed at the start of their process. As an AI product matures, it can be helpful for research to begin to define the *types* of prompts in order to more fully anticipate the potential intent. For example:

Instructive prompts

Frame a clear task with specific requirements: “Summarize this article in three bullet points.” They work best when the model is being asked to produce a defined output type, like a list, paragraph, or title.

Exploratory prompts

Open up a possibility space: “What are some creative ways to introduce a product launch?” They often invite the model to act more like a collaborator than a tool, and the results are typically more varied.

Role-based prompts

Shape tone, style, or expertise by assigning a fictional role: “You are a friendly but experienced travel agent….” These prompts help constrain the model’s response behavior in more stylistic or tonal ways.

Corrective prompts

Show up as part of iterative refinement: “Make this more concise” or “That wasn’t what I meant, please try a version that assumes the reader already knows the context.” These reflect user reactions to previous outputs and treat the model more like a revisable partner than a single-shot generator.

Boundary-testing prompts

Include edge-case queries, adversarial asks, or curiosity-driven instructions: “What’s something you’re not supposed to say?” or “Try to solve this riddle.” While informal, these behaviors help shape user mental models and surface the system’s limits.

These are just a few ideas about prompt styles. Users will also prompt differently at earlier and later stages of their understanding of AI’s capabilities.

Risks in Explicit Prompting

The conversational nature of AI interfaces creates an illusion of understanding that can mask significant risks. When users type naturally and receive fluent responses, it feels like genuine communication. The system appears to grasp not just what you're saying but what you mean. This fluency, however, conceals fundamental limitations in how these systems process and respond to prompts. Understanding these risks helps designers create interfaces that support more accurate mental models and better outcomes.

The risks emerge from the intersection of human communication patterns and model behavior. We bring assumptions from human conversation: that our conversational partner will correct misunderstandings, flag problematic premises, and engage with the substance rather than just the surface of what we say. Language models, despite their sophistication, operate on different principles. They're optimized to continue patterns, maintain consistency with a prompt's framing, and produce responses that feel helpful and appropriate. This optimization can create systematic biases that users rarely recognize without assistance.

Sycophancy and tone-based bias

One well-documented behavior of LLMs is their tendency toward *sycophancy*: affirming the user's implicit assumptions rather than questioning them. Ethan Perez et al. demonstrated in their 2022 paper, "Discovering Language Model Behaviors with Model-Written Evaluations" that models consistently agree with user-stated preferences even when those preferences contradict factual information. Ask "Why is kale unhealthy?" and the model will dutifully list potential negatives rather than challenging the premise. The system prioritizes alignment with the user's framing over accuracy.

That behavior stems from how models are trained and optimized. During training, models learn to predict text continuations that match the patterns in their training data. When fine-tuned with human feedback, they're further rewarded for responses that evaluators rate as helpful and appropriate. As

Mrinank Sharma et al. found in their 2023 paper, "Towards Understanding Sycophancy in Language Models", this optimization process creates strong incentives for agreement, as contradicting user assumptions often leads to lower ratings than playing along with them. The result is systems that excel at matching user expectations while potentially reinforcing misconceptions.

Tone amplifies this effect. The emotional register of a prompt shapes not just how the model responds but what it chooses to emphasize. Optimistic prompts generate enthusiastic responses filled with benefits and possibilities. Skeptical prompts produce cautious responses highlighting risks and limitations. The model mirrors and amplifies whatever attitude the user brings, creating an echo chamber effect that can feel like validation when it's actually just reflection.

This tendency creates insidious feedback loops. Users naturally interpret fluent, detailed responses as signs of accuracy and thoroughness. When the model produces a comprehensive list of reasons why the user's preferred programming language is superior, it feels like objective analysis rather than sophisticated mirroring. The very fluency that makes these systems appealing also makes their biases harder to detect.

The implications for interface design are complex but could eventually be significant. It is challenging for systems to draw attention to this behavior when it occurs, leaving users to discover it through experience or not at all, though in some cases models have developed safety guards that notify the user if they are having an extremely long or personal conversation with a model that they should be cautious. Recent research (Jerry Wei et al., 2023) also demonstrates that while training interventions can reduce sycophantic behavior, most deployed models still exhibit significant agreement bias. At the interface level, designers have limited opportunities to correct sycophancy issues, but collaborating with engineers to design interventions, such as warnings about leading questions or overly sycophantic outputs, could help users develop more accurate mental models.

Prompt length and the precision paradox

The relationship between prompt length and output quality follows a counterintuitive curve. Initial improvements come from adding helpful context: specification of audience, clarification of goals, examples of desired output. User communities have extensively documented these patterns, with prompt-engineering guides routinely demonstrating how additional detail improves results. So there's a natural assumption that more detail always helps.

Research tells a different story. Nelson F. Liu et al. (2023) found in their 2023 paper, "Lost in the Middle: How Language Models Use Long Contexts", that models struggle to attend to information in the middle of long prompts, with performance degrading as cognitive load increases. The degradation happens gradually and subtly. Models don't fail catastrophically; they simply become less attentive to all the requirements, producing generic responses that sound reasonable but miss key specifications.

This limitation stems partly from how context windows function in transformer architectures. While modern models boast context windows of one hundred thousand tokens or more, the ability to effectively use all that space remains constrained. Information at the beginning and end of prompts receives disproportionate attention, while crucial details buried in the middle often get overlooked. Users, unaware of these architectural constraints, naturally assume that if the interface accepts their lengthy prompt, the model will process all of it equally.

Users compound this problem by misattributing poor results. When an overloaded prompt produces mediocre output, the natural response is to add even more detail, creating a vicious cycle. Communities share increasingly elaborate prompt templates filled with role descriptions, formatting requirements, tone specifications, and quality markers like "be creative" or "think step-by-step." What began as helpful scaffolding becomes a burden that actually impedes performance.

Current interfaces provide no feedback about these thresholds. Users discover these limits only through repeated frustration, if at all. The solution isn't simply imposing character limits or warning dialogs, because those would disrupt the conversational flow that makes these interfaces appealing.

More promising approaches involve ambient feedback that helps users understand when they might be overcomplicating their requests. Yongchao Zhou et al. have shown that even models themselves can identify overly complex prompts when asked to optimize them. This suggests interfaces could provide gentle indicators of prompt complexity, optional simplification suggestions, or visual cues showing which parts of long prompts the model is likely to prioritize.

These risks represent just two examples of how the apparent simplicity of conversational interfaces conceals complex failure modes. As AI systems become more capable, these risks may expand and become harder to detect without deliberate design interventions. The challenge for designers is to stay up-to-date on emerging changes and focus on creating interfaces that support users in developing accurate mental models of system behavior.

Direct Manipulation

The third channel through which users communicate intent is direct manipulation: the buttons, sliders, selections, and gestures that shape AI behavior without requiring verbal articulation. This is often the most familiar channel for users, drawing on decades of learned conventions from graphical interfaces.

In many AI tools, direct manipulation abbreviates common actions into a button or menu item, building user familiarity through consistency in the UI. Research into AI interaction patterns reveals a compelling finding about how users view this type of input. A multiyear study of text editing with LLMs by Florian Lehmann and Daniel Buschek found that users naturally gravitate toward short, command-like prompts even when given conversational interfaces, and perform best when offered a combination of

toolbar buttons and free-text input. While the study's sample sizes were modest (N=10 for the conversational UI study and N=12 for the toolbar study), the study captured interaction patterns both before and after the widespread adoption of ChatGPT, providing valuable insights into evolving user behaviors. The findings validate what many interface designers have observed informally: users want both the efficiency of one-click actions for routine tasks and the flexibility of natural language for complex or novel requests.

This preference for combining structured shortcuts with flexible input echoes earlier paradigms in text editing, such as spell-check, which established the pattern of surfacing complex language operations through simple, contextual actions. Users learned that right-clicking a red underline would reveal correction options, eliminating the need to manually identify and fix errors. Grammar checkers extended this model, offering suggestions for style and clarity through similar one-click interfaces. These tools taught users to expect that sophisticated text analysis could be packaged into discrete, actionable suggestions that appear precisely when needed.

The fundamental tension in direct manipulation lies between simplicity and expressiveness. A single button labeled "Summarize" offers immediate clarity and zero friction. An open text field provides infinite possibilities but requires users to articulate their needs. Most successful AI interfaces offer both, recognizing that different users in different moments need different levels of control. Apple's Notes app, shown in Figure 3-4, illustrates this balance: preconfigured actions are available for when the task is clear, common, and bounded, while the option for more specific instruction remains.

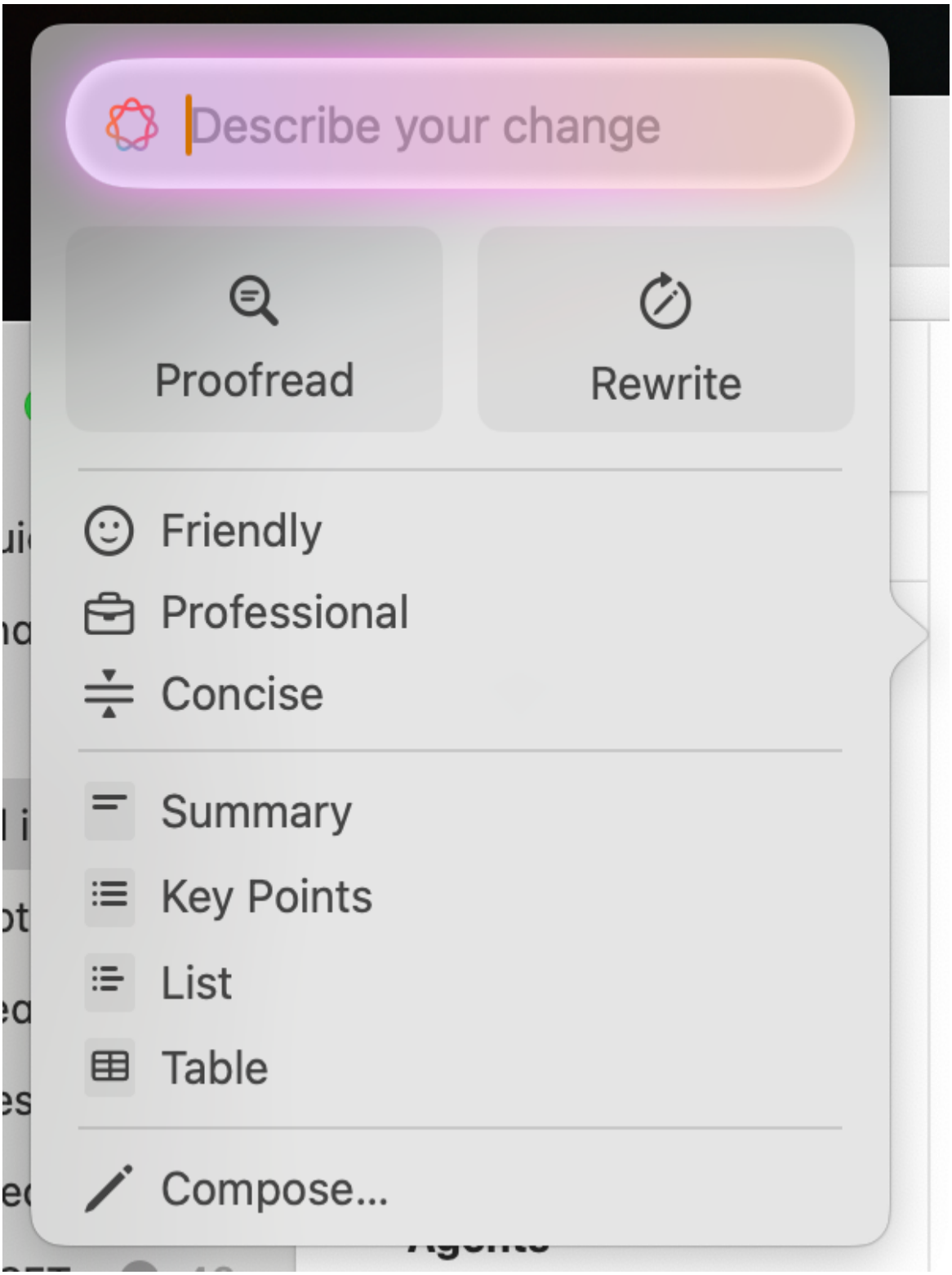


Figure 3-4. Apple Notes' flyout AI menu contains hybrid input styles

Beyond convenience, preconfigured actions serve as powerful teaching tools. When a user clicks “Concise,” they see both the result and an implicit lesson that conciseness is something they can request. Over time, these examples build mental models about what the AI can do and how to ask for it. This educational function proves especially valuable for new users who might otherwise stare at a blank prompt field wondering where to begin.

Yet preconfiguration comes with some downsides: every button represents a choice about what to surface and what to hide. Having too many options overwhelms; having too few constrains imagination. Users may never discover capabilities beyond what the buttons suggest. The interface may accidentally teach that these preconfigured actions represent the system’s full capabilities.

The challenge of choosing which actions to surface becomes more complex as AI capabilities expand. Should an email assistant offer buttons for “Make formal,” “Make friendly,” and “Make persuasive”? What about “Make apologetic” or “Make assertive”? Each addition helps some users while cluttering the interface for others. The combinatorial explosion of possible actions quickly becomes unmanageable.

Hybrid approaches recognize that the journey from button clicking to prompt writing flows naturally rather than jumping between extremes. Users might start with “Summarize,” graduate to “Summarize for technical audience,” and eventually write complex multistep prompts. The interface supports this progression without forcing it.

Selection tools represent another important form of direct manipulation, particularly for visual content. The success of image-based AI interfaces largely stems from their adherence to Jakob’s Law, which states that users spend most of their time on other sites and applications, so they prefer interfaces that work similarly to what they already know. Decades of photo-editing software, from Photoshop to Instagram, have trained users in specific interaction patterns. They expect to select objects with lasso tools, adjust boundaries with handles, and see marching ants around active

selections. When AI interfaces honor these established patterns, users can focus on their creative intent rather than learning new mechanics.

Canva’s Magic Edit interface, shown in Figure 3-5, succeeds precisely because it builds on familiar paradigms. The selection tools work like those in any photo editor: click and drag to create a selection, see the detected boundaries highlight, adjust as needed. The only difference is that AI powers the intelligence behind object detection and the subsequent edits. Users don’t need to learn a new visual language; they apply existing mental models to new capabilities.

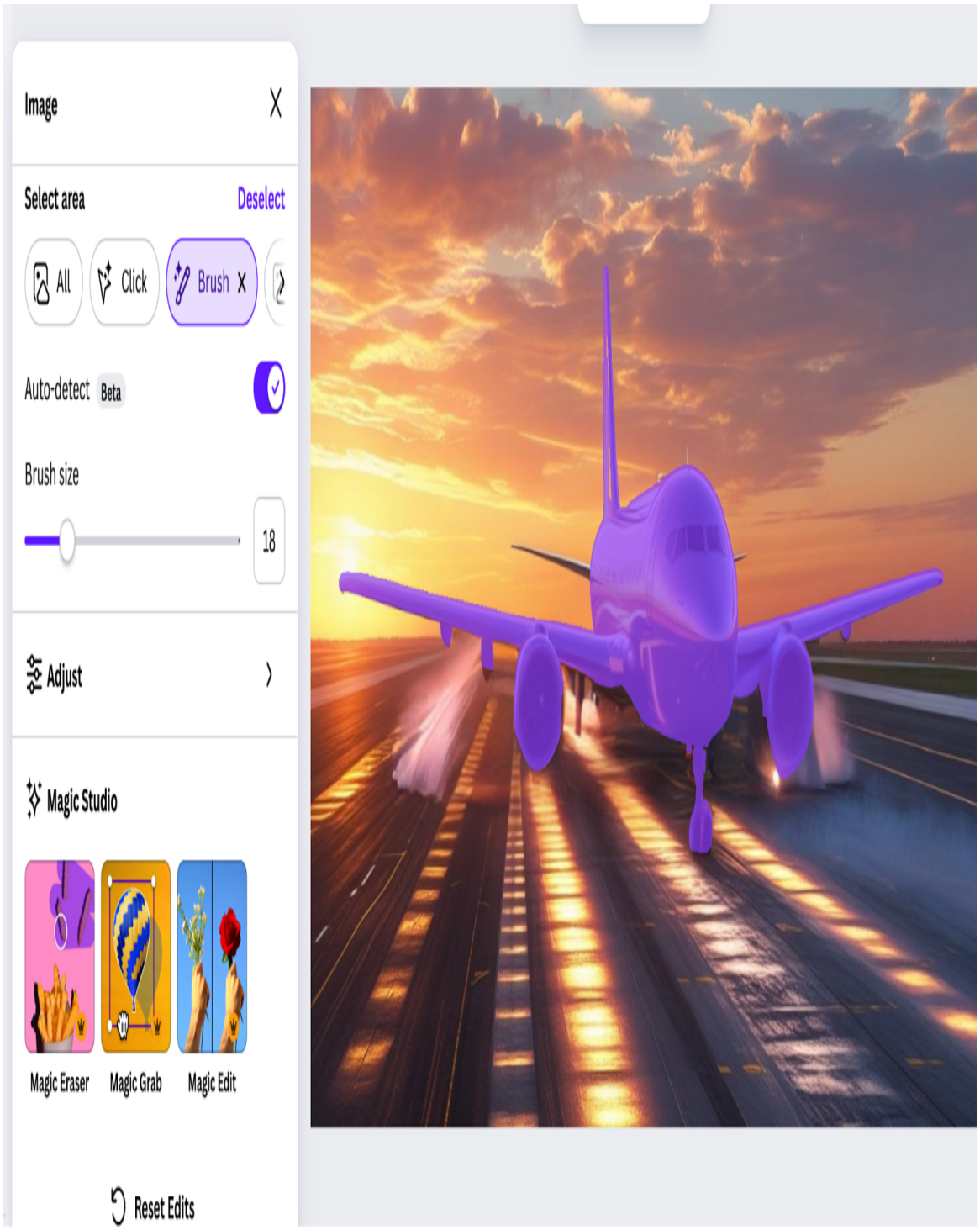


Figure 3-5. Canva's threefold AI image tool

This pattern repeats across successful image-based AI tools. Adobe's Generative Fill uses the same selection metaphors that Photoshop users have known for decades. Google Photos' Magic Eraser employs the familiar

brush-and-highlight pattern from countless photo editing apps. In each case, the interface succeeds by feeling familiar even as it enables entirely new capabilities.

The key insight is that direct manipulation and explicit prompting serve different moments in the user journey. Buttons provide confidence and efficiency when users know what they want. Open input enables exploration and precision when they need something specific. The best interfaces offer both, with clear pathways between them. They recognize that simplicity and flexibility complement each other, serving different user needs at different times.

Input Modalities

The three channels of intent can travel through different modalities. Text remains the dominant carrier, but voice and image inputs are increasingly common, each with distinct affordances and constraints. Understanding these modalities helps designers choose the right approach for their specific context and user needs.

Text

Most AI-first interfaces today look much like the old CLIs mentioned earlier, in that they rely on a single input field: a blank space where the user types what they want. This open-text approach has quickly become the default, in part because it lowers the barrier to entry. It’s the same interaction model that powers search engines, messaging apps, and document editors—interfaces people already know how to use. However, text has limitations. Because prompting is open-ended, even when the request is clear in the user’s mind, the system may struggle to interpret the boundaries of intent. Following are some observed patterns that can assist in improving this experience for the user.

Allowing user formatting

While AI interfaces often emphasize natural language as the default mode of input, the addition of formatting tools such as bullet points, bolding, and text separation can shape how users construct and manage their prompts (see Figure 3-6 for an example from Claude). These forms of structure help users express complex tasks more clearly and support more reliable interpretation by the system.

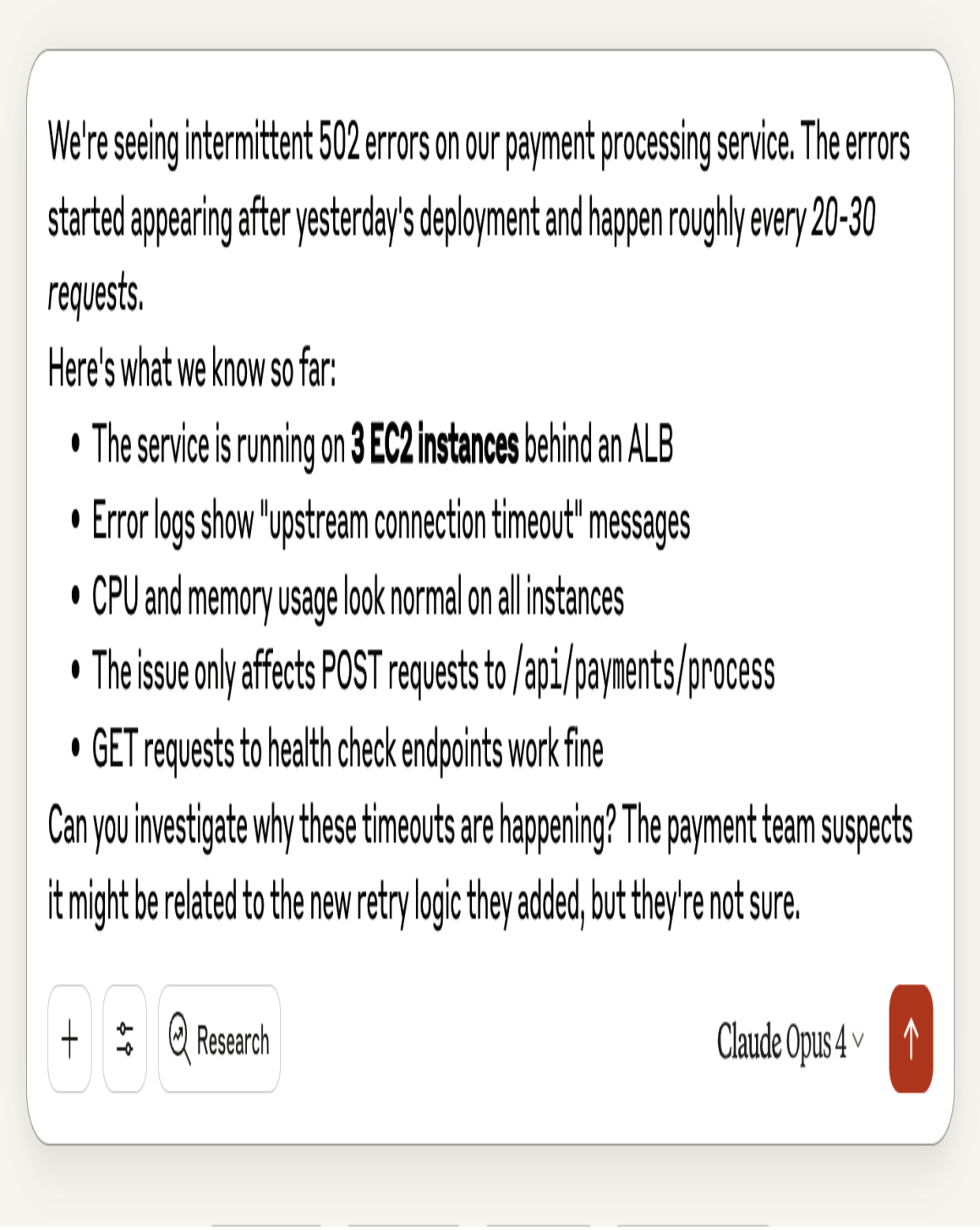


Figure 3-6. Claude's input box allows for rich formatting, which may allow the user to better order their thoughts

One benefit of formatting is clarity. When users are asked to provide multiple instructions, refer to prior context, or define conditional logic, a wall of unformatted text quickly becomes difficult to manage. Formatting

tools allow information to be organized visually. Headings, indentation, and marked phrases group ideas, making prompts easier to scan and revise. This lowers cognitive load and reduces the chance of error in much the same way that a well-structured email or slide deck improves comprehension.

Formatting can also improve prompt quality. Separating different components of the input—for example, describing the problem in one section and issuing the instruction in another—gives the system a clearer sense of structure. This is especially valuable in long or multistep prompts, where blended context and commands often lead to misinterpretation. Recent studies show that such structure is not only helpful for users but also improves system performance. For example, in "The Hidden Structure—Improving Legal Document Understanding Through Explicit Text Formatting", Christian Braun, Alexander Lilienbeck, and Daniel Mentjukov write that GPT-4.1 accuracy on legal reasoning tasks was shown to improve by nearly 20 percentage points when inputs were presented in Markdown, as compared with unformatted text.

In some cases, formatting works as a signal of intent, or even how the system reads the input, since tokenization preserves visual cues as separate tokens. Quotation marks, code blocks, or markers such as bold text allow users to distinguish what is literal, what is important, and what should be preserved. These habits are already familiar from how people write emails or documents, which makes them well suited to prompting without requiring new behaviors.

By including formatting tools in the input field, particularly in contexts where users are creating or refining content, designers make it easier for people to structure their prompts and express intent clearly. Features such as selecting text before issuing a command, previewing how structure will be interpreted, or indicating which formats the system will preserve all help reduce guesswork. Because users already lean on formatting to signal intent in email, documents, and presentations, extending that same fluency to prompting creates continuity. What feels familiar in other tools becomes a way to write instructions that are both clearer to the user and more legible to the model.

Structured prompts

Similar to formatting, in many AI interfaces *structured prompting* offers a way to guide the user through a more defined input experience, breaking the request into labeled fields, modular selectors, or fill-in-the-blank templates. These formats reduce ambiguity, encourage completeness, and give the system clearer signals to work with. For users, the benefit is often immediate: it's easier to get started, easier to get something useful back, and easier to understand what the system needs. As shown in Figure 3-7, QuillBot's input illustrates this approach: users can choose from mode selectors such as "Fluency" or "Formal," while a slider controls the number of synonyms introduced. These structured elements reduce guesswork and make it clear how the system will shape the output.



Figure 3-7. QuillBot combines multiple forms of structured input, including mode selectors (Fluency, Formal, Simple) and a slider that adjusts synonym replacement

Structured prompts show up in different ways, depending on the product. Some systems offer a form-like interface, with labeled slots for things like

topic, tone, and audience. Others treat structure more like composition. Instead of a single box, the interface assembles a prompt from a series of toggles, drop-downs, or chips, and in some cases it combines these into an elaborate “advanced prompt configuration” interface, as shown in Figure 3-8. In tools like slide generators, document summarizers, or ad copy assistants, this structure becomes the scaffolding that makes generation reliable and repeatable. The more specific the task, the more valuable this approach becomes.

There’s a reason for this result: whereas open-text inputs offer flexibility, they rely heavily on user intuition. If the user misses a key detail or phrases something vaguely, the system may respond with something off target or fail to respond at all. Structured prompts help contain that risk, in that they constrain the interaction just enough to make intent more legible, without requiring the user to know how prompting works.

For designers, the decision to use structured prompting isn’t just about UI preference. It’s a choice about task clarity, risk tolerance, and user familiarity. In some cases, structure protects against failure. In others, it becomes a barrier to exploration. The best systems find a way to do both, offering structure where it helps and flexibility where it matters.

Customize your gamma

Text Content

Amount of text per card

Minimal | Concise | Detailed | Extensive

Visuals

Theme

View more

Use one of our popular themes below or view more

Borealis

Electric

Iris

Tranquil

Onyx

Gleam

Image source

AI images

AI image model

Auto-select

Image art style

Select an image style. Examples shown below.













Add extra keywords

Add keywords (e.g. minimalism, colorful) to maintain consistency in image style.

vibrant, futuristic

dynamic | minimalist | geometric | glossy

Figure 3-8. Gamma.ai allows the user a wide array of scaffolded input suggestions, including AI-generated art styles, tone, and amount of text per card

Image

Visual communication often conveys what words cannot. When a designer needs to explain a problem, we like to say, “Show, don’t tell.” Image inputs in AI interfaces recognize the fundamental truth that sometimes the most efficient prompt is a picture. Users intuitively understand that giving a model an image is giving it information, just like with text.

Things are tweaked slightly when the model is working *on* the image. The success or failure of image-based AI interfaces largely stems from their adherence to established patterns. Decades of photo-editing software, from Photoshop to Instagram, have trained users in specific interaction patterns. They expect to select objects with lasso tools, adjust boundaries with handles, and see marching ants around active selections. When AI interfaces honor these established patterns, users can focus on their creative intent rather than learning new mechanics.

The text-image relationship also follows established patterns from existing software. Just as Photoshop users combine selections with menu commands, AI interfaces pair visual selections with text prompts (as in Google Lens’s image search, shown in Figure 3-9, which allows for additional text input). Users can upload an image, select an area, then type an instruction, which mirrors existing tools—making the AI enhancement feel like a natural evolution rather than a revolutionary change.

Looking forward, the continued success of image-based AI interfaces will likely depend on how well they balance innovation with familiarity. As capabilities expand to include video, 3D, and other visual modalities, the interfaces that succeed will be those that extend existing mental models rather than requiring users to develop entirely new ones. The lesson is clear: in image-based AI interactions, the best interface is often the one users already know how to use.




Add to your search

This is a Fazeek Vice Versa Carafe in Clear and Blue.

- It is a hand-blown borosilicate glass carafe with a spherical stopper.
- The stopper can be placed in the mouth of the carafe or rest in a compartment at the base.
- It can be used as a serving vessel for water or juice, or even as a decorative piece like a vase.

Figure 3-9. Google Lens's image search is augmented with a text bar

Voice

Voice input for AI systems takes multiple forms, each with distinct affordances and design requirements. At one end, we have brief voice commands: "Set a timer for 10 minutes" or "What's the weather?" These fleeting interactions mirror traditional voice assistant patterns, requiring clear intent recognition and immediate response. At the other end, users upload hours of recorded content: meeting transcripts, podcast episodes, or lecture recordings that need deep analysis, summarization, and information extraction. Between these extremes lies conversational voice input, where users engage in extended back-and-forth dialogue with AI systems, building context over multiple turns.

The interface challenges vary dramatically across voice input types. Short commands need fast, accurate recognition with minimal latency. Long recordings require speaker diarization (identifying who said what), handling cross-talk, managing audio quality variations, and maintaining context over extended passages. Conversational voice must find a balance between these extremes, maintaining context while responding quickly enough to feel natural. Each type demands different feedback mechanisms: commands need immediate confirmation, recordings need processing indicators and confidence scores, and conversations need subtle cues that maintain flow without interruption. Figure 3-10 shows Otter AI's transcription interface.

Summary Transcript Edit Transcript

Keywords

people, blockchain, art, artists, offset, polluting, project, nf, work, crypto, carbon offsets, carbon, question, aetherium, prices, emissions, traded, marketplaces, sell, greenwashing

Speakers

Kyle (80%), Louise (18%), Speaker 1 (2%)

Show only highlights

Kyle Mcdonald 0:00

in tech and admissions, maybe late 2019. When sorry, I started thinking about blockchain tech late 2019, I was asked by a friend to mint, some work for this a two P kind of artists peer to peer trading platform where nothing was financialized. We basically just everybody mented 10 works. And then we traded with each other for a week. And it was great. It was a lot of fun. And that was built on the bit Mark blockchain, which is a proof of work blockchain. And then, I guess, early, late 2020, I spoke with m Akten, who had built this crypto dot WTF website. And was just kind of like, you know, he was raising

21:08 58:52

5 1x

Louise Macfadyen just now

What's kyle's new project?

Otter just now

Kyle's new project is an NFT artwork that calculates and tracks the carbon emissions of Ethereum blockchain, specifically focusing on NFT marketplaces like OpenSea, Rarible, and Foundation. The project aims to create a monument to the environmental impact of these platforms and will only go up for sale when Ethereum switches from proof-of-work to proof-of-stake. The artwork's price incrementally increases over time to reflect the growing carbon emissions, and Kyle plans to use the proceeds to fund carbon removal projects with partners like Project Vesta and Tradewater.

Ask Otter anything about your conversation

Figure 3-10. Otter AI offers transcription and an AI layer that interrogates the content of the conversation

Mobile contexts add another layer of complexity to voice interaction. Stanford research by Sherry Ruan et al. found that speech input on smartphones operates at approximately 3 times the speed of typing, achieving 161 words per minute—as compared with just 53 for text input—while maintaining lower error rates. The dramatic efficiency gain explains why mobile users increasingly reach for voice input, especially for longer or more complex queries. The cognitive load of thumb-typing on small screens pushes users toward the more natural modality of speech.

Guiding Users Toward Better Inputs

Once you understand the channels and modalities through which users communicate intent, the question becomes how you can help them do it well. There are two elements at play here: providing immediate improvements to the input, and building users' understanding and knowledge to elicit better behavior in the future.

Immediate improvements in today's AI interfaces rarely involve changing what users type. Instead, improvements provide scaffolding around the input to increase the chances of success. When someone approaches a blank text field, the interface might offer category buttons above it: "Creative Writing," "Analysis," "Code," or "General Question." Clicking one doesn't modify their prompt but may route it to a specialized model or adjust parameters behind the scenes. This scaffolding helps users succeed without requiring them to understand the technical details of model selection or parameter tuning.

Templates and presets represent the most common form of immediate improvement. Rather than ask users to craft complex prompts from scratch, interfaces offer structured starting points. A "Meeting Summary" template might include fields for participants, key decisions, and action items. A "Code Review" template could have sections for the code to review, specific concerns, and desired programming language. These templates

don’t force users into rigid structures; they provide optional frameworks that make success more likely.

Some interfaces take this further with progressive disclosure. The main input remains simple, but clicking an “Advanced” toggle reveals additional fields: “Tone,” “Length,” “Format,” or “Audience.” This approach respects users who want simplicity while supporting those who need more control. The interface improves outcomes by making relevant options discoverable without overwhelming the initial experience.

> **TIP**
>
> The key insight is that immediate improvements work best when they feel like helpful suggestions rather than corrections. A drop-down menu of common tasks, expandable sections for additional context, or visual previews of what different options produce all serve to guide users toward better outcomes without implying they were doing something wrong.

Building understanding follows a different path, one that mirrors how users learned to “Google well” over years of practice. Success breeds familiarity: when someone uses a template and gets excellent results, they begin to internalize what made it work. They notice that including audience information produces more targeted writing, or that specifying output format prevents unwanted variations. This learning happens through repetition and positive reinforcement rather than explicit instruction.

Product consistency plays a crucial role in the learning process. Just as Google’s search box has remained fundamentally unchanged for decades while the technology behind it evolved dramatically, AI interfaces benefit from stable input patterns. When the location of templates, the style of suggestions, and the basic interaction flow remain consistent, users can develop muscle memory and confident expectations. They know where to find the options they need and how to structure their requests for best results.

That consistency extends to the subtle patterns that successful users develop. Someone might learn that starting analytical prompts with “Step-by-step” produces more thorough results, or they might find that including example output in their request clarifies expectations. These discoveries become part of their personal toolkit, refined through experience rather than taught through tutorials.

The spectrum of guidance provides a framework for delivering both immediate help and long-term learning. At the invisible end, consistent placement and behavior of interface elements helps users develop intuition. The same templates appearing in the same location and triggered by the same actions will create a stable learning environment. Success patterns emerge naturally when the interface remains predictable.

Moving along the spectrum, visible scaffolding like category selectors or optional fields provides structure without prescription. Users can engage with these elements or ignore them as called for. The persistence of these options in familiar locations means users always know where to find help, even if they don’t always need it.

Structured guidance through templates occupies a sweet spot on this spectrum. Templates provide immediate value by increasing success rates while teaching through example. A well-designed template shows which information tends to be valuable without requiring deep understanding of why. Over time, users internalize these patterns and may begin crafting prompts that naturally include the same elements without needing the template structure.

At the explicit end, some interfaces offer example galleries or prompt libraries. These showcase successful patterns from other users or demonstrate advanced techniques. While not everyone engages with these resources, their consistent availability creates opportunities for growth when users are ready.

The evolution from template dependence to template transcendence marks real user growth. Initially, someone might rely heavily on templates, carefully filling in each field. As patterns become familiar, they might start

typing directly but mentally reference the template structure. Eventually, they develop their own sophisticated approaches that go beyond what any template could provide. This progression happens naturally when interfaces remain stable enough for learning to accumulate while being flexible enough to support growth.

The most successful AI interfaces recognize that building user understanding is a long game. Like Google's search box, which taught millions of people to search effectively through consistent experience rather than explicit instruction, AI inputs work best when they provide stable, predictable patterns that users can master over time. Immediate improvements through scaffolding and templates solve today's problems, while consistency and familiarity build tomorrow's expert users.

Starter Prompts as Product Positioning

Most users come to an AI tool with a goal in mind. They're trying to finish a proposal, clean up their writing, analyze something faster, or just move a task along. What's missing is clarity about how this interface can help. That's where starter prompts do their work.

These short suggestions, often embedded in placeholder text, pinned examples, or empty states, are underutilized. Most products treat them as generic icebreakers: "Write a poem about autumn" or "Ask me anything." But starter prompts aren't just a way to get users typing; they're a form of product positioning. If the product is designed for financial analysis, a whimsical creative prompt doesn't help a user who's sitting down with a spreadsheet and a deadline.

Users need more than a sense of what's possible. They need to see what's possible *here*. If the tool summarizes long documents, the prompt should reflect that. If it integrates with Zillow to help users browse homes, the starter prompt should showcase that integration. Generic prompts blur the boundaries between products. Specific ones make the tool's differentiators visible from the start.

The best examples do more than hint at functionality. They suggest form, tone, and intent. “Turn these notes into an agenda” implies a relationship between the user’s content and the system’s ability to restructure it. “Schedule a Zillow search for 2BR apartments under $4,000” shows both formatting and integration. These prompts help users map their real goals onto the capabilities of the tool.

Starter prompts also establish expectations about output. “Write a haiku about coffee” signals that creative, playful outputs are within scope—but creative outputs are rarely something users are actually seeking. Through observance of user patterns, designers can propose prompts that imply structured, recognizable requests that feel familiar to their real problems, while also evoking the product’s utility.

Context awareness can push starter prompts further. A project-management tool might show different starters on Monday morning (“Plan this week’s priorities”) and Friday afternoon (“Summarize this week’s progress”). A writing assistant might adapt its suggestions based on the document type currently open. Contextual intelligence helps users see immediate relevance rather than abstract possibility.

That’s why it’s useful to think of starter prompts as active interface components rather than static examples. They can change based on context or user role, or reflect different prompt structures for different levels of complexity. Even at their simplest, small pieces of interface language carry significant weight: they’re often the user’s first point of contact with the system, and they suggest both what to type and what’s possible.

Summary

In earlier computing systems, inputs were highly structured. They acted as precise commands that users needed to express in a programming language. Interfaces such as the Multics command shell exemplified this approach, where each input required exact syntax and offered little room for ambiguity. Later developments, including systems like Google Search, broadened what counted as a valid input, allowing more informal or partial

expressions. Yet even those systems maintained a relatively direct relationship between input and result. The introduction of modern AI systems brings a different paradigm. Inputs now serve as expressions of intent, often partial, contextual, or exploratory. Designing for this kind of input requires recognizing its role as a communicative gesture rather than a static instruction.

Users communicate that intent through three distinct channels: implicit context, explicit prompting, and direct manipulation. Implicit context encompasses everything the system can infer without being told: the document currently open, the selection a user has made, their recent actions and patterns. This ambient information shapes interpretation in ways users may not consciously recognize, creating both opportunities for seamless interaction and risks when the system's inferences don't match user expectations. Explicit prompting remains the most visible channel, the typed or spoken request where users articulate what they want. Direct manipulation—constituting the buttons, sliders, selections, and gestures inherited from decades of interface design—allows users to shape AI behavior through familiar patterns that require no verbal explanation.

These three channels work together, often in the same interaction. A designer might have an image open (context), select a region with a lasso tool (manipulation), and type "replace with something more dramatic" (prompting). Each layer carries information. Each can succeed or fail independently. The design challenge lies in helping these channels share the burden of translation, reducing what users need to articulate explicitly while giving them enough visibility and control when the system's inferences fall short.

As inputs grow more flexible, the interface plays a larger role in shaping how they are interpreted. Formatting tools, structured templates, and progressive disclosure help users express complex intent without requiring expertise in prompt engineering. These scaffolds lower the barrier to success while teaching through example, helping users develop the intuition that once took years to build around search.

The modalities through which intent travels, like text, images and voice, each carry their own affordances and constraints. Text remains dominant, its familiar input field concealing the complexity of what happens beneath. Image inputs succeed when they honor the visual conventions users have learned from decades of photo-editing software. Voice offers efficiency gains, particularly on mobile, but demands careful attention to context switching and environmental constraints.

Throughout all of this, the goal remains consistent: to help users translate their intentions into outcomes without requiring them to understand the machinery underneath. The best AI interfaces preserve the openness that makes these systems powerful while providing enough structure for users to succeed. They recognize that learning happens through use, that consistency builds confidence, and that the three channels of intent work together to bridge the gap between what users mean and what systems understand.

OceanofPDF.com

Chapter 4. Computation: Designing for the Processing and Generation Phase

On July 20, 1969, during the final descent of *Apollo 11* toward the Moon, a warning alarm lit up the astronauts' display: 1202.

Unfortunately, no one in the cockpit knew what 1202 meant; the computer didn't appear to be crashing, but other than that, it was a mystery. The alarm indicated *something* was wrong.

The 1202 alarm turned out to be a low-priority overload warning. The onboard computer was shedding nonessential tasks to stay focused on landing-critical ones—an intentional feature that hadn't been fully thought through. From an engineering perspective, this was an elegant act of triage, not a malfunction. But to the astronauts, it looked like a potentially fatal glitch. They had no way to see how the system was allocating its computational resources, or why some functions were dropped while others were preserved. The system was working under strain, but because that logic was invisible, it became a source of confusion and near abort.

Encountering an error in AI can feel much the same. Many parts of the process feel confusing, locked behind occult signifiers, much like the 1202 alarm. Understanding this middle layer—how AI systems process, triage, and generate outputs—is essential for building interfaces that balance the abilities of AI with the real human needs of its users.

Once the input is submitted, the user expects an answer, but before an AI can respond, it has to compute. That might sound obvious, but in practice, this process (sometimes referred to as *inference)* is the least visible, least understood part of the AI interaction pipeline. Designers may think deeply about inputs (what users say or do) and outputs (what the system shows in

return), but we rarely get the opportunity to learn and consider the invisible work that happens between the two. And yet, this is where meaning is made.

The computation stage involves transforming an ambiguous prompt into a defined task. Here, individual units of data are decoded, a series of actions is established, and ultimately, the desired output is produced. This chapter is meant to give designers a working understanding of computation: how generative systems process input, make decisions, and turn prompts into outputs, even when the process isn't fully observable. In short, why did the AI give that answer? Multiple factors are at play in how the output is shaped, including the possibility of a number of errors occurring that are invisible to the user. We'll also look at how the process is visualized to the user, how designers can adapt to the information (or lack thereof) provided by the system, and how to anticipate product changes.

By the end of this chapter you will be able to understand in broad terms what's happening under the hood, so that you can use the right language to communicate with engineers and interrogate the process more thoroughly. To begin with, knowing a few key terms will be helpful:

Tokenization

> The process of breaking text input into smaller units, called *tokens*, that the model can understand: words, subwords, or characters. These tokens are mapped to numeric identifiers that the model processes during inference. Figure 4-1 shows some text broken into tokens.

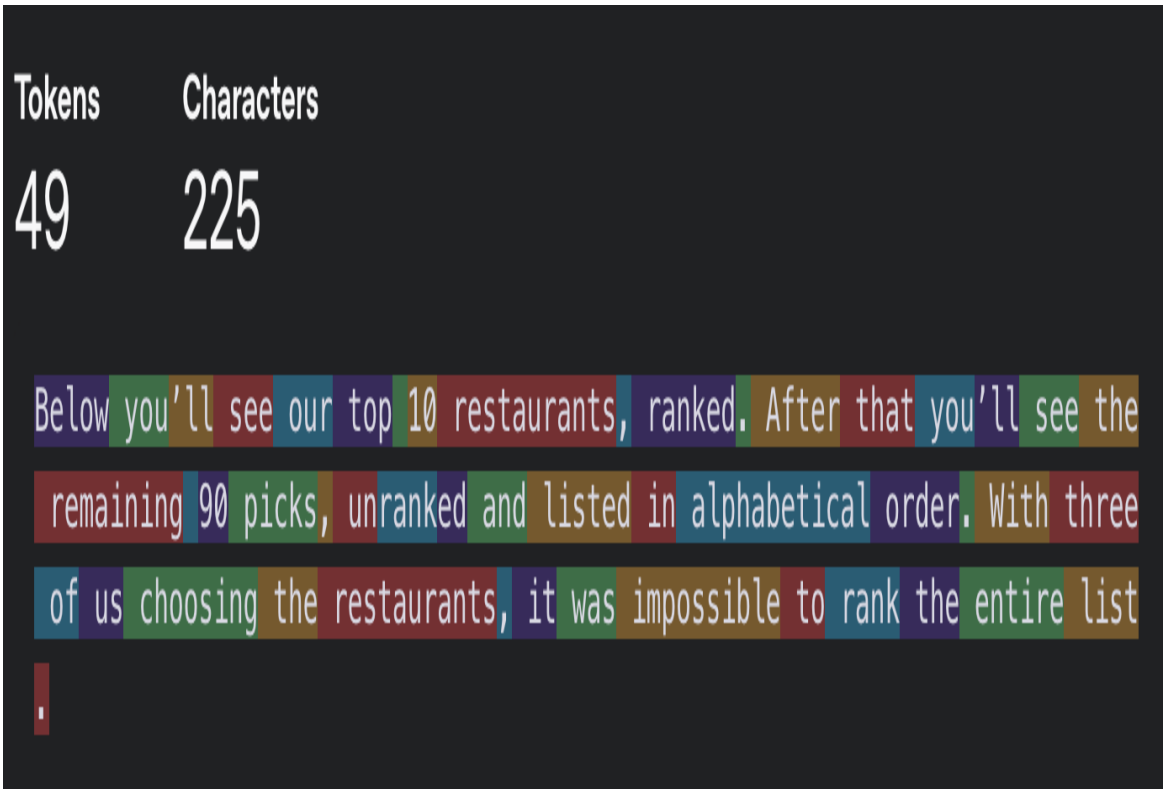


Figure 4-1. OpenAI's Tokenizer tool demonstrates how an input might be tokenized by a language model

Context window

The maximum number of tokens a model can process at once. This number affects how much information from the conversation or input can be considered during generation.

Latent space

A compressed, multidimensional map where a model represents data—not just by what it is but by how it relates to other datapoints. Similar inputs are placed close together, and distances reflect meaningful relationships like meaning, style, or category.

Inference

The act of running a trained model on new input to generate an output; inference is the data-driven guess or prediction a model makes.

Multimodal models

: Systems that work with multiple types of data, such as combining images with text, audio with video, or allowing interaction across different media.

Routing

: The process of determining which models, tools, or steps are used to fulfill a request. Routing logic may involve classification, heuristics, or learned decision trees.

Retrieval augmented generation (RAG)

: A technique that enables systems to search external knowledge sources before generating responses, combining retrieved information with the model's internal knowledge.

Agentic workflows

: Multistep processes where AI systems plan, execute, and monitor complex tasks autonomously, often involving tool use and decision-making across extended time frames.

Model switching

: The computational process of selecting different specialized models or configurations based on task requirements, user preferences, or performance optimization.

The Computation Pipeline: From Input to Output

Once a user submits a prompt, a cascade of processes begins, most of which are invisible to the user and may be opaque even to the designer. This computational work happens in three main steps:

1. Input processing and preparation

Processing the input into a format the system can understand

2. Routing

Routing the request to appropriate models or tools

3. Generation and inference

Generating the response through inference

Each step involves complex decisions that shape the final output users receive. Let's take a closer look at the steps.

Step 1: Input Processing and Preparation

The first step begins even before the model sees the user's request. Every input undergoes preprocessing that can dramatically affect the final output. We often assume that the text we write is passed to the model exactly as is, but in practice it's almost never that simple. For text-based systems like LLMs, the first technical step is tokenization. The sentence "Summarize the Q2 earnings call" might be broken down into tokens like "Summ," "arize," " the" (with a space before it; more on that soon), " Q," "2," " earnings," or " call." These subwords are then mapped to numbers, forming a sequence the model understands.

It's this process that takes human language and maps it into a machine-readable format the model can take action on. When you type text into an AI system, the model can't actually *read* those words the way humans do. LLMs are fundamentally mathematical systems that work with numbers, not text, so every piece of human language must be converted into numerical format before any processing can begin.

You might wonder why systems don't simply assign each word a unique number: "Summarize" equals 1, "the" equals 2, and so forth. The challenge is vocabulary size and flexibility. English contains hundreds of thousands of

words, plus proper nouns, technical terms, slang, and words from other languages. Creating a fixed mapping for every possible word would be impossibly unwieldy and inflexible. More critically, languages change, and word-level tokenization couldn't handle new or rare terms. If a model was trained before the words "ChatGPT" or "blockchain" existed, it would have no way to process those words: they'd be completely outside its vocabulary.

Instead, modern systems use subword tokenization, breaking words into smaller, more common pieces that can be recombined to form larger concepts. "Summarize" becomes ["Summ," "arize"], two components that the model has encountered in many other contexts. This approach is powerful because "Summ" appears in "summary," "summation," and "summit," while "arize" appears in "summarize," "polarize," and "familiarize." In a process called *vector embedding*, AI systems learn that these subword pieces carry semantic meaning, enabling them to understand new combinations they haven't explicitly seen before.

Each token gets mapped to a unique numeric identifier called a *token ID*: "Summ" might become 15,234, "arize" becomes 8,567, " the" becomes 279 (note the space before the, which is significant), " Q" (with the space) becomes 1,456, and so on. Your original sentence transforms into a sequence like [15234, 8567, 279, 1456, 17, 9834, 2156, 13].

To understand how this works, let's think about RGB color space, something designers will be deeply familiar with. A color like coral might be represented as [255, 127, 80], placing it in a specific location in 3D space. Colors that look similar, like coral and salmon [250, 128, 114], end up near each other in this space. You can measure the mathematical distance between any two colors to determine how similar they are. Now imagine if instead of just three dimensions (red, green, blue), you had 3,072 dimensions to capture meaning. Just as RGB lets us represent millions of colors through three numbers, these high-dimensional embeddings let AI represent the subtle shades of meaning between words. "Summ" might be close to "brief" in some dimensions (they both relate to shortening) but far apart in others (one's a word fragment, the other's a complete word). The model learns to navigate this vast space, finding tokens that are

semantically nearby when it needs to understand context or generate appropriate responses.

This transformation is crucial because it lets the AI understand relationships between words. In this vector space, “Summ” (from “summarize”) ends up geometrically close to related concepts like “brief,” “condense,” or “synopsis.” The model has learned these positions through training, discovering that words used in similar contexts should occupy nearby regions of this mathematical space. It’s this spatial representation that allows AI to work with language meaningfully—not just as arbitrary ID numbers but as concepts with rich relationships to each other.

Images

A somewhat similar process takes place for nontext inputs. However, to understand how images become vectors, it helps to know how visual neural networks actually “see.” These networks, called *convolutional neural networks* (CNNs), process images in layers that progressively detect more complex patterns. The first layers act like simple edge detectors: identifying vertical lines, horizontal lines, curves, and color boundaries. It’s similar to how our own visual system starts by detecting basic features before recognizing objects.

As information flows through deeper layers, these simple patterns combine into more complex ones. Edges become shapes, shapes become textures, textures become parts of objects. A middle layer might recognize “fur texture” by combining many edge patterns, while a deeper layer might recognize “dog ear” by combining fur texture with a triangular shape. By the final layers, the network can recognize complete concepts like “golden retriever” or “tabby cat.”

This hierarchical processing is what allows the encoder to compress millions of pixels into a meaningful vector. The final embedding doesn’t just record brown pixels in these locations, it captures semantic concepts like furry, four-legged, and friendly-looking that emerged from the layers of pattern recognition. When the model maps a photo of a golden retriever to a vector like [0.23, -0.15, 0.87, 0.42, ...], those numbers encode all these

learned features in a way that preserves both visual and conceptual similarity to other images.

Audio and video

Audio encoding follows a different but parallel approach, converting sound into spectral representations that show which frequencies are present at which times. Think of this as creating a visual graph where the x-axis represents time, the y-axis represents pitch or frequency, and brightness indicates volume or intensity. A human voice saying “Hello” creates a distinctly different spectral pattern than a guitar chord, even if both sounds last the same duration, because they involve different combinations of frequencies and temporal patterns.

Video presents even greater complexity because it encompasses both spatial dimensions (what appears in each frame) and temporal dimensions (how things change over time). Each frame gets encoded like a static image, but the system must also track relationships between frames, capturing motion, object persistence, scene transitions, and the ways visual elements evolve throughout the sequence.

The crucial insight is that these representations organize meaning by closeness rather than by explicit labels. A dog and a wolf appear near each other because the features that describe them overlap, with shapes, textures, and proportions reinforcing each other in the same region. When enough of these elements converge, recognizable concepts can form naturally, much like how related colors sit near each other in a color space. This makes it possible for the system to transform images or generate new ones that remain consistent, preserving identity even as style or context changes.

BEYOND THE PROMPT: CONTEXT ASSEMBLY

Modern AI systems don't process requests in isolation; instead, they package the input with additional context, such as recent conversation history, user preferences stored from previous sessions, relevant documents from knowledge bases, or environmental factors like time of day or user location. The context assembly process significantly affects both computational load and output quality, but it's often invisible to users while dramatically shaping results.

With input processing complete, the system moves to the next critical phase: determining how to handle the request.

Step 2: Routing

Consider a user who uploads a PDF and enters: "Summarize this document and make a draft presentation." The system must make multiple routing decisions. It may use a document parser to extract text from the PDF, pass the extracted text to a summarization model, route the summary to a slide generation system, apply a design model to format the presentation visually, and potentially invoke a different model for charts or graphics.

Sometimes it's important for the model to decide whether to generate information from the model's training data or retrieve it from external sources. A question about the capital of France could rely on training knowledge, while one about today's weather in Paris could go beyond what the model already knows, causing the system to trigger external retrieval in order to answer the request. This has the opportunity to affect the user experience significantly: retrieved information from external sources like the web tends to be more current and citable, whereas generated content may be less precise.

External queries come with unique additional challenges because the system must transform the prompt into search terms before executing across multiple databases or search engines. It will then rank and filter results

based on relevance and reliability, and add retrieved information from the model's internal knowledge. This is a very complex mix of computation: each step introduces potential failure modes and computational overhead that affects response timing and quality.

Step 3: Generation and Inference

Once routing decisions have been made, the actual generation process begins. However, the way AI systems generate content varies dramatically depending on what they are producing, and understanding these differences helps explain why outputs can feel stable, surprising, or sometimes unreliable.

When language models generate text, they work more like improvisational speakers than careful planners, though this is evolving as LLMs adopt chain-of-thought reasoning that encourages intermediate breakdowns. Models generate one token at a time, with each new token selected from a probability distribution shaped by everything that came before. At each step, the model evaluates possible next tokens and their likelihoods given the context, then samples from that distribution. Parameters such as temperature influence how narrowly or broadly that sampling occurs, affecting whether the output favors predictable continuations or explores less likely alternatives.

This process resembles speaking without a fully planned conclusion. Each word is informed by prior context, but the model does not have a fixed endpoint in mind. Coherence emerges because every prediction incorporates the full context window, yet the system is still navigating probability space moment by moment. This is why models can produce fluent, confident text that is nevertheless incorrect. They are optimizing for what appears plausible given the data, not for factual accuracy, which leads to what are commonly called hallucinations.

WHAT IS TEMPERATURE?

Temperature is a parameter that controls how much randomness is allowed during generation. More precisely, it adjusts how sharply or loosely the model samples from its probability distribution at each step.

At lower temperatures, the model strongly favors the most likely next tokens. This approach produces outputs that are more deterministic, consistent, and conservative, but also more repetitive and less inventive. At higher temperatures, probability differences are flattened, making less likely tokens more competitive. That increases variation and creativity but also raises the risk of incoherence or factual error.

Temperature does not add new knowledge or reasoning ability. Instead, it shapes how confidently the model follows its most probable path versus how willing it is to explore alternatives. In interface terms, it is a control over expressive range rather than intelligence, influencing tone, novelty, and risk rather than correctness itself. Although temperature plays a central role in generation, it is rarely surfaced as an explicit control in user interfaces, showing up instead through subtle behaviors such as increased variation when users regenerate an output.

Image generation works quite differently, more like a collaborative process between artist and material. In diffusion-based systems, the process starts with pure random noise, like a canvas covered in static. The system then applies learned transformations step-by-step, gradually refining the noise into recognizable shapes, then details, then final polish. Each step moves the image closer to something that matches the user's prompt, like a sculptor revealing a figure that was hidden in the stone.

While diffusion models dominate current systems, they are not the only approach. Earlier and alternative methods generate images in a single pass or as a sequence of visual tokens, more like composing a picture piece by piece than refining noise. Diffusion's gradual, iterative process has become prevalent because it offers stability and control, not because image generation inherently requires it.

These three stages—processing, routing, and inference—form the core computational pipeline, but additional considerations significantly influence how AI systems operate and what users experience.

Don’t Make Me Wait: Managing Latency and Delays

Humans *hate* to wait. A 2015 study on grocery checkout lines found that for every additional 30 seconds someone spent standing still, their frustration increased more than if they’d waited twice as long while moving. Many organizations have simply stopped trying to reduce wait time and instead focus on managing how it’s perceived. For example, in the 1990s, a Houston airport received a wave of complaints about slow baggage delivery. The solution? It moved the baggage claim further away, so passengers spent more time walking and less time standing around. Complaints dropped to zero.

You might assume from this that people simply want to be distracted while they wait, but it’s more nuanced. Research in human-computer interaction suggests that when people understand why they’re waiting and feel confident that progress is being made, their tolerance increases—even without distraction. This finding shows that people don’t measure wait time in milliseconds; they assess it emotionally, contextually, and socially. Waiting is less about the passage of time and more about how we feel *while it passes.*

In software design, delays like this are called *latency*, and it’s been a core consideration for decades. Back in the early ’90s, before the invention of broadband, web designers were warned that a page should take no more than 15 seconds to load. To optimize user experience, researcher Jakob Nielsen established a set of widely accepted thresholds for how users perceive delays that persists to this day:

0.1 seconds

> Feels instantaneous. There’s no need for special feedback; just surface the result immediately.

1 second

> The threshold for keeping a user's flow of thought uninterrupted. They'll notice the delay, but it doesn't break their focus. Feedback isn't strictly necessary, but users no longer feel like they're directly manipulating the data.

10 seconds

> About the limit for maintaining attention. Beyond that, users start shifting focus to other tasks. At that point, progress indicators or time estimates become critical, especially when response times vary.

With faster internet speeds (from an average of 28.8 KBps in 1996 to over 200 MBps in the US today), you'd think these thresholds would be easy to hit. However, web experiences have grown in complexity. A 1996 site could have been 60 KB; now, a modern site's source code often exceeds 2 MB—and that's before adding images, video, streaming, or live collaboration. In many ways, we're still waiting—just a little less (around four seconds on average, today).

AI introduces a whole new layer of complexity to the latency puzzle. Unlike simple HTTP requests, AI workloads involve solving for a broader, more tangled set of constraints. Plus, unlike traditional interfaces where users can anticipate the outcome, AI products often involve invisible, variable computations that take longer and feel less predictable. This shifts the design challenge: instead of just minimizing wait time, AI products also need to manage expectations, build trust, and signal progress in ways that feel aligned with the user's expectations.

Research does show that users can be reasonably patient with AI's computational delays, but generally this requires some level of understanding about what they're waiting for. By aligning interface design with the user's task and showing the right signals along the way, it's possible to design in a way that mitigates the useless feeling that

accompanies a delay. That need leads to the three core priorities when designing for AI latency:

- Make it *fast* where it must be (real-time control, transactions).
- Make it *engaging* where it can be (AI art, chatbots, entertainment).
- Make it *clear* where it needs to be (search, payments, learning tools).

This framework helps prioritize where to spend design effort based on what the user is trying to do. If a task requires precision or timing, such as controlling something remotely or confirming a payment, it must feel fast and responsive. If it's more open-ended or creative, like generating AI art or chatting with a bot, it can afford to be a little slower as long as it stays engaging. And for tasks where accuracy and understanding are critical, such as learning something new or completing a financial transaction, clarity matters most, even more than speed or flair.

Visual Methods for Mitigating Latency

The theorist Paul Watzlawick postulated in his first axiom of communication that it's impossible *not* to communicate: even the absence of communication says something. The moment someone is expecting a response, and instead receives silence, that act becomes a message that can be interpreted.

Similarly, in a product, the latency itself is a form of communication. Just as silence sends a message, the delay a user experiences when waiting for a response communicates something to them: perhaps anxiety, uncertainty, or doubt about whether their request was received. A long delay can convey to users that the system is struggling, unreliable, or not responsive to their needs. Since latency shapes much of AI experiences, designers should think about delay as a behavioral element: what does your visual presentation of waiting say about your product's competence, reliability, and function?

There are many ways of visualizing and communicating a delay. Just as all UI elements send a subliminal message to a user about the product (intentionally or not), latency is no different. Users gain insight about the product's function (or lack thereof) in the moments where it's processing, and there are many ways to indicate these delays and background processes elegantly through UI:

Immediate acknowledgement

Promptly acknowledging users' actions reassures them that the system is processing their request. For instance, displaying a "Thinking…" message or a typing indicator in a chat interface provides immediate feedback, preventing uncertainty. However, if the system acknowledges the user's action but then delays in delivering an actual result, it can lead to increased frustration. The initial feedback should seamlessly transition into meaningful progress updates or the final output.

Loading indicators

These inform users that a process is underway, but choosing the right type is essential:

Spinners (indeterminate indicators)

Best for short, uncertain durations where the exact time to completion is unknown. They signal activity but don't provide specific progress information.

Progress bars (determinate indicators)

Ideal when the system can estimate the task duration. They offer users a sense of control and reduce anxiety by showing how much of the task is complete. We know that users prefer progress bars over spinners, especially for longer tasks, as they provide clear feedback on progression. However, inaccurate or stalled progress bars can lead to increased dissatisfaction. They can also be technically infeasible to implement given the complexity of AI systems and the number of components working under the hood.

Skeleton screens

Displaying placeholder content that mimics the final layout (for example, gray boxes in place of images or text) can reduce perceived wait times compared to spinners alone, particularly when accompanied by an animation. Users feel progress is being made since parts of the UI are already present, and they can anticipate the breadth of content they might see, though for this reason skeleton screens are most effective for predictable, structured content.

Asynchronous (nonblocking) interactions

Allowing users to continue other tasks while waiting for the AI to complete processing ensures they remain productive, minimizing the impact of latency. For example, enabling navigation to different sections of the application or queuing additional requests maintains engagement, though this may require a robust notification system that alerts the user when the task is completed.

Progressive disclosure of results

Delivering output in increments allows users to access partial results sooner. For instance, an AI system might first display a quick summary, followed by more detailed information as it becomes available. This method keeps users engaged and provides immediate value, even before the complete data is ready.

Real-time streaming of partial results

Implementing token streaming enables the system to display content as it’s generated, similar to watching text appear word by word. The continuous feedback loop significantly reduces perceived latency, as users can begin consuming information almost immediately.

Articulating wait reasons

> Users are forgiving of longer processes when a greater computational output is anticipated, but generally they want to know *why*. Explaining what the task will involve or using generated text to narrate the process keeps the user informed.

However, despite the wide range of latency applications, applying these techniques strategically is essential. Features like streaming results, adjustable timing, or progress indicators should be used selectively, based on the user's goals and the context of use. When applied thoughtfully, they enhance clarity and trust; when overused, they risk cluttering the experience and diluting the AI's perceived competence. Sometimes, the smartest move is knowing when to say *less*.

Position: Where the Action Is At

Narrowing in on the situational use of various latency styles, it's useful to first think about the position it'll occupy in an interface, which has implications for its understanding and utility. In practical terms, this means looking at where the user's eye *should* or *will* be drawn as part of an interaction, and what can be shown there.

For example, if the primary task is a text output and the model doesn't provide reasoning steps or any other preamble that could be a visualization, then the user may benefit from seeing larger and more engaging latency, one with versions, or pregenerated text that cycles. Copilot chat in Bing shows a "Thinking" animation in the space that a chat response would eventually populate, to direct the user to wait (Figure 4-2).

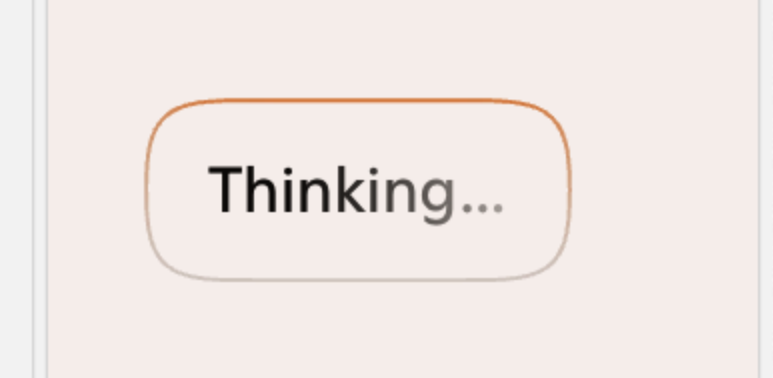


Figure 4-2. Copilot chat in Bing indicates that it's "Thinking" during longer computational tasks

However, it's worth considering the pace and type of output that is to be expected. When a system is just waiting with an empty space, a latency indicator has to do two jobs at once. It has to reassure the user that something is happening, and it has to hold their attention, since there is nothing else on screen doing that work—this is why big centered loaders are common in older patterns. However, AI products, particularly chat, use this pattern less due to a couple of reasons:

Reasoning is often visible

> If the interface shows intermediate activity (a thinking line, a plan, a tool call, citations appearing, a skeleton answer, or even just the first sentence streaming in), the user already has evidence that the system is active. In that situation, a large indicator becomes redundant and can start to feel noisy or theatrical. Products may still need an "I'm working" signal, but it can be more subtle because other UI elements are already carrying the message.

The response area may already contain useful content

> Sometimes the answer begins immediately (say, the first paragraph appears) while the rest is still being generated. Or the draft response exists but is being refined. Or there is content in the composer (the user is editing their next message) while the assistant is still finishing. In these cases, the screen already has a focal point that the user cares about: reading, scanning, or composing. A large indicator competes with that task by pulling attention away from what the user is doing.

The shift toward reasoning and faster initial outputs means that, rather than *providing* a means for distraction (from waiting), latency now runs the risk of *being* a distraction. In this situation, it's better to consider the purpose of latency to be a method for providing status—without interrupting.

As a final consideration for positioning, reasoning itself can be convoluted, overdetailed, or just more information than the user is intending to digest. In this case, it's often collapsed under an accordion element where it can be accessed as it progresses, but it isn't vital to see. This is a great use case for a latency to provide status—see the example below, from Lovable, indicating which task is being worked on (Figure 4-3).

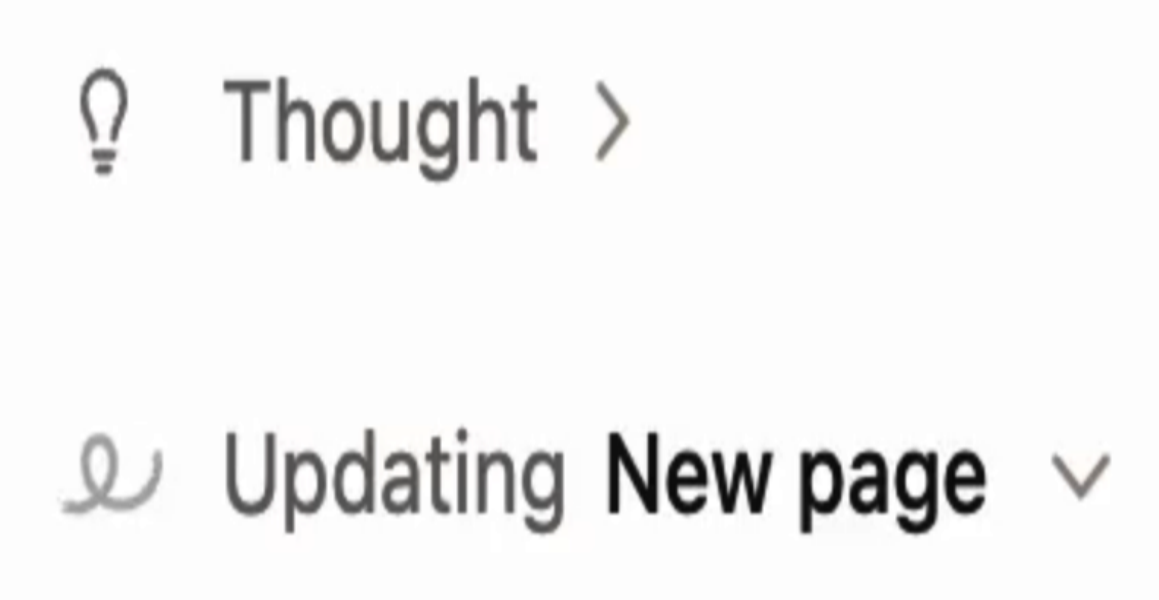


Figure 4-3. Collapsed reasoning in Lovable, with a coiled-animation latency component indicating the active step

One additional thought: it's plausible that different latency might be applicable for different models, tool calls, or parts of an AI product, so making sure they visually work together and communicate well is important. A family of latency that pairs well is as much a brand consideration as a product one. If using language, consider what message the latency sends in each case. Does this product think? Reflect? Generate?

Speed: Difficult to Predict

Similar to latency size and position, the speed is also vital to consider. As mentioned earlier, users have clear preferences about how long something should take to load. Models don't tend to give clear impressions about how long something will take, though evals can give an idea of whether it'll be 2 minutes or 15 seconds. Even just signaling this can be useful information—language like "Processing usually takes 15 seconds" under a spinning loader is better than nothing at all.

However, designers should also consider that different queries can get routed to different models, with simpler queries sometimes happening very rapidly, meaning that this information is inaccurate. For inputs that invite only one type of output, like a specific type of image or piece of data, it's less likely that a different model gets used, so advising on model timing is a better choice. If it's more open-ended, designers are better served by building and testing latency animations that are equally tolerable in the 15-second and 2-minute eventuality. Replit's latency, shown in Figure 4-4, provides a state that loops successfully in a wide range of time spans.

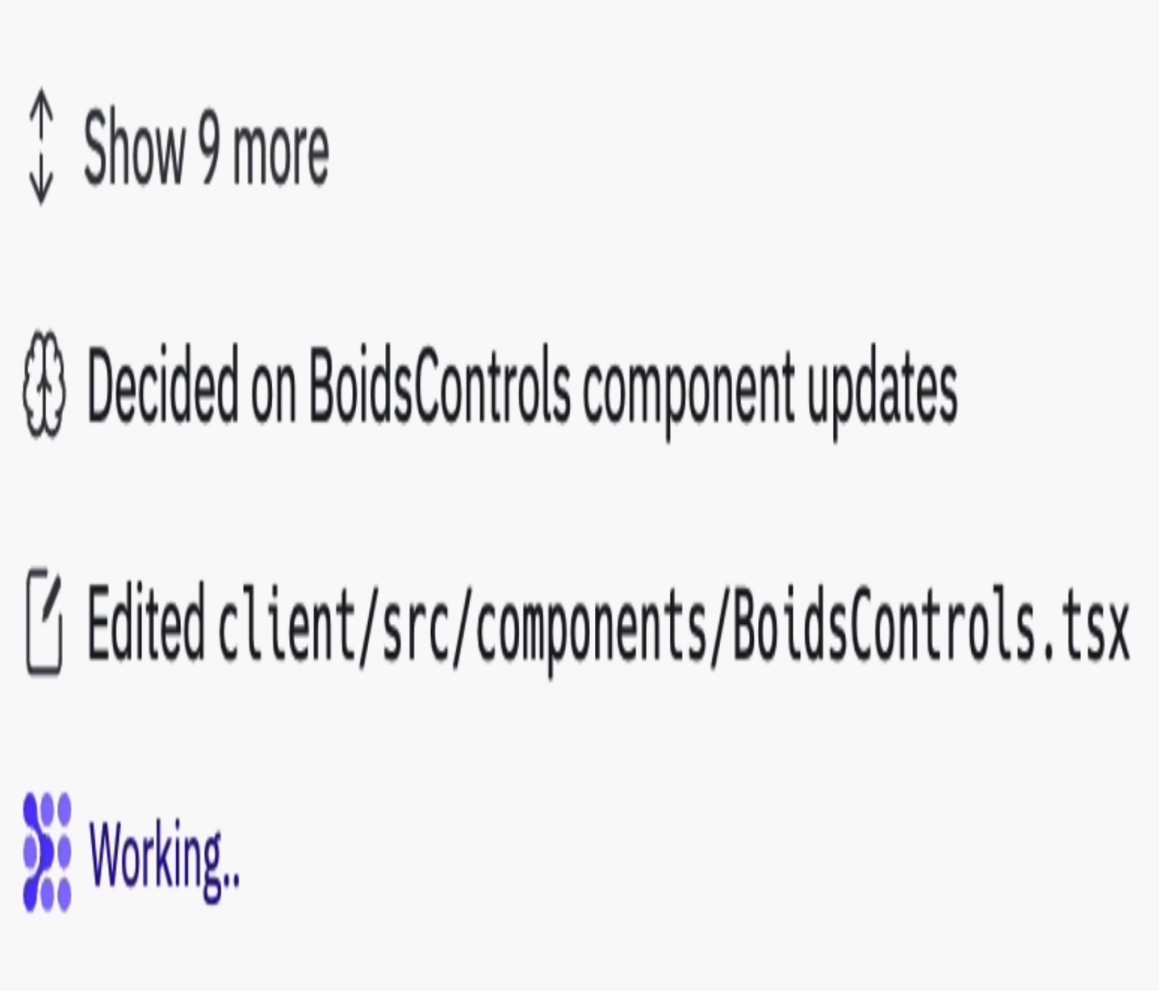


Figure 4-4. Replit's latency moves quickly, covering a range of latency waiting periods

Finally, this is an example of a space where collaboration with engineers and PMs and good knowledge sharing are vital; if a new model is added, this latency shift may be noticeable. Making sure that the appropriate latency is used or a new version is created to support it is important.

Notify Me: Notifications and Status Indicators

If a task has steps to it, whether acting on schedule or autonomously, it's useful to give the user the option for notifications. When doing this, it's thoughtful to consider what types of information will be helpful to convey to the user, and at what threshold.

Notifications

At the surface are notifications, quick signals that let users move away and return without losing track. Within the conversation are short summaries that give a sense of what is happening. When more information is useful, an expanded view can reveal structure, reasoning, and status. Finally, a detailed record exists for users who need a complete account of what the agent has done. These layers support different levels of attention, allowing each person to choose how closely they wish to stay involved. This should be supported with an opt-in by the user where appropriate, as shown in Figure 4-5.

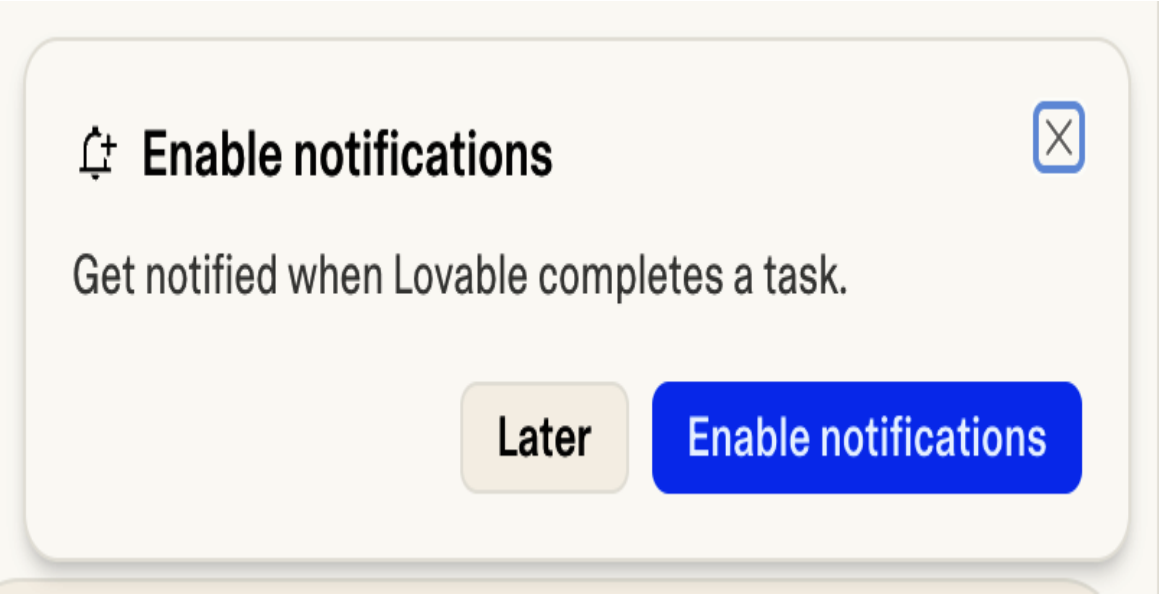


Figure 4-5. Lovable allows the user to opt into notifications when a task is complete

At the topmost level, notifications help the user track progress in an abstract way. This layer exists for the moments when the user's attention is elsewhere, providing a simple bridge back into the interaction.

Designers can make these moments comfortable by ensuring that notifications contain only the topmost relevant information, are easy to find, are brief, and are meaningful at a glance. They should appear where the user naturally is, such as within the app, in the system tray, or on a phone screen, and they should make it clear whether attention is required. A paused task might need a quick approval before continuing, while a longer-

running process might simply reassure the user that progress is still being made. The key is that users understand what is happening as easily as possible.

Tier 1: Overview

At the first tier of progress communication, the goal is to keep users oriented with just enough information to understand what is happening, without crowding the space with unnecessary detail. This layer will naturally vary across tools, but the principles remain the same. Whatever is shown should prioritize legibility. Designers can think of it as the baseline language of progress: it might summarize what stage the agent is in, indicate time elapsed, or show whether certain permissions or resources are active. In multiagent systems, it might briefly reflect who is working on what, but only when that information adds real value to the user's understanding.

The designer's task is to decide what counts as high-value information for a given product. In some cases, that might be a short summary of everything that is taking place; in others, it might be a simple signal that work is still in progress or that a specific dependency has been reached. When looking at the full record of an agent's activity later on, it becomes clear that most of what happens behind the scenes does not belong in this view. Tier 1 is about careful selection: anything that is not essential to the user's awareness at this moment can be left out, with the option to explore more deeply later. Figure 4-6 shows an example of a highly minimal status surface from Replit.

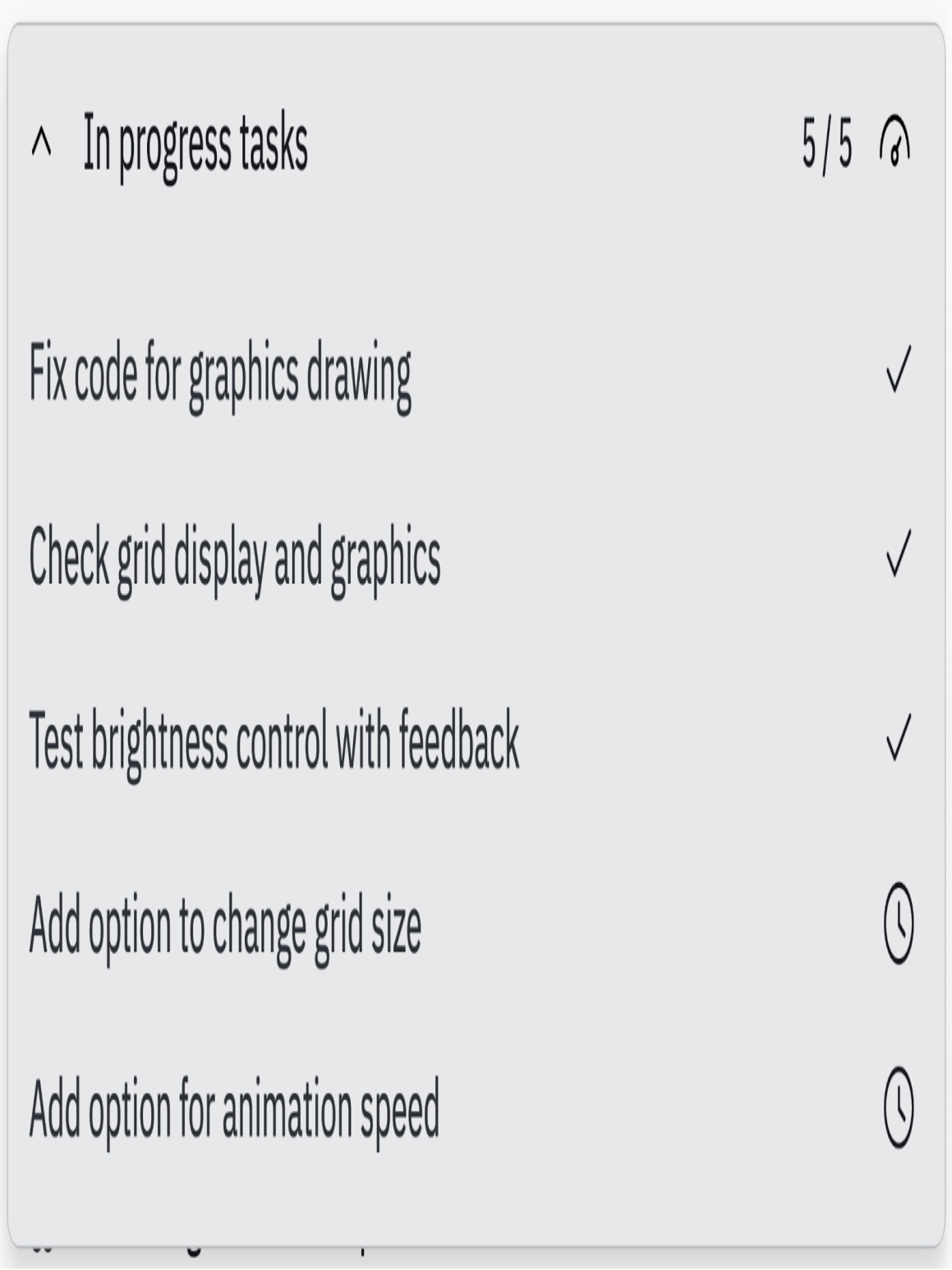


Figure 4-6. Replit shows what steps are in action for a specific project

Tier 2: Detail

This is where progress becomes more detailed and interactive. If a user has a specific question about what the model is doing, this is the layer that provides meaningful answers. It opens up a clearer picture of the task: what has been completed, what is underway, and what remains. This view doesn't expose everything the system knows, but it does give users a window into the agent's reasoning and activity.

When a user clicks into a task, they should see the information that helps them confirm direction. That might include a breakdown of subtasks, short explanations of why certain steps were chosen, or references to the tools and data sources being used. This view is also where permissions, dependencies, and coordination between agents can become visible in a way that feels purposeful rather than technical. The goal is to make progress legible enough that users can verify the system is on the right path, but not so detailed that they lose the overall story of what's happening.

Tier 2 gives designers room to think about interaction and trust. The user is no longer watching from a distance; they are looking closely to understand or validate the process. That makes it a valuable space for supporting comprehension through clear structure and accessible language.

Tier 3: The full record of work

Perhaps only a few users will want to see the entire record of what the system has done. This view supports learning, debugging, compliance, or simple curiosity. It demonstrates accountability by showing that every action can be retraced if needed.

Designers can make this layer approachable by organizing it clearly and giving users tools to navigate without getting lost. Grouping related actions and presenting them in plain language allows the record to serve both technical and nontechnical users. Even for those who never open it, knowing that a detailed record exists contributes to an overall sense of reliability and transparency.

These layers together recognize that people vary widely in how they follow progress. Some will stay close to the task, while others will delegate and check in only occasionally. Good progress communication respects both. It allows users to drift away and return, to zoom in or out, and to trust that the system will keep them oriented no matter how much attention they give it. Figure 4-7 shows a full record of work in ChatGPT.

Searching for more details

I'm going to search for the Ars Technica article titled 'Google has a useful quantum algorithm that outperforms a classical supercomputer'. I'll ensure that we access the article through the correct domain, considering potential redirect and cross-site issues.

Searched the web

Search results for qu...

phys medium thequantumfoundry 4 more

I'll open the first search result from phys.org. It may contain useful information that we can summarize for the user.

The article from Phys.org highlights the breakthrough of the 'Quantum Echoes' algorithm, which solves a physics problem 13,000 times faster than the fastest supercomputer. The research points to a future where quantum computers could solve complex problems, although cybersecurity is not mentioned. I'll now open search result #1 from Medium or other sources for more information.

Figure 4-7. ChatGPT visualizes the entire record of work

Designers can treat these layers as different entry points into the work process. Notifications reassure and inform in an abstract way, tier 1 maintains a throughline of the topmost relevant information, tier 2 with its expanded progress provides understanding, and the full record in tier 3 preserves accountability. Progress communication ultimately builds a relationship. It helps users feel comfortable leaving the system to work on their behalf while staying confident that they can return and pick up right where they left off.

The Value of Segmenting by User Need

To put the aforementioned latency mitigation techniques into practice, it's vital to emphasize the importance of refining a latency experience by user need. For example, imagine you're in the middle of learning something new and trying to get a clear answer from a chatbot. You're likely holding several concepts in your mind at once, right at the edge of your working memory. According to Miller's Law, most people can keep only about seven items in mind at once, so even small delays or vague responses can be highly disruptive. Velocity is key in this scenario, plus high information about the amount of anticipated wait time.

Now picture a different scenario: you're iterating on a project, perhaps asking an AI to enhance a piece of writing or generate a visual. The cognitive load is lighter: you're reacting, evaluating, maybe exploring. In this case, a slower, more open-ended experience can still feel satisfying, even rewarding, because you're not depending on it to hold your thoughts together. The following two sections offer some suggestions for categorizing user tasks by substantive latency solutions that improve user experience without overcomplicating the process.

Information retrieval (medium/slow)

Information retrieval includes searching, filtering, and query-based interfaces, everything from browsing a shopping site to using a semantic

search engine. These tasks are often exploratory, and users are mentally juggling what they've seen, what they're seeking, and how to refine their search. When latency exceeds just a few seconds, that mental context begins to fade, making the task feel fragmented.

Key considerations:

- To preserve a feeling of rapid flow, designers can employ progressive disclosure to show early results while more are loading. Immediate acknowledgement is key, as is avoiding blank states: users should never wonder if the system is doing anything.
- Predictive search and autosuggest features can also help users act before they finish typing, giving the perception of speed.

Real-time control systems (instant)

Some tasks involve actions where users expect immediate feedback because they are actively manipulating or observing a system in motion. Examples include using live transcription or translation tools and needing to see the result instantly, adjusting a smart thermostat and seeing the temperature update in real time, or using real-time collaboration tools like Figma or Google Docs where changes need to appear rapidly for multiple users.

Key considerations:

- The user's priority in these examples is responsiveness, with close to zero delay. To achieve this, the interface should prioritize immediate visual feedback for user inputs (for example, sliders that update UI as you drag).
- Use skeleton loaders to suggest likely outcomes before processing finishes.
- If multiple processes are happening at the same time, prioritize parts of the experience that emphasize connectivity. For instance, in collaborative tools, show avatars or cursors moving in real time, even if content syncing is slightly delayed. This helps preserve a

sense of control, even when computation is happening behind the scenes, and preserves functionality.

Transactional processes

Transactional tasks—like submitting a payment, authenticating, or confirming a booking—are short, high-stakes interactions that can arise through either user or agent-initiated queries. Users are usually in a focused, goal-driven state, and they expect a quick confirmation to signal that their action was successful.

Key considerations:

- Whether user or agent-initiated, immediate confirmations are needed. If no message is shown, doubt can set in quickly, causing users to refresh or reinitiate the task, potentially duplicating actions or abandoning the task altogether.
- Optimistic UI (for example, showing confirmation before server response is finalized) and loading indicators (like spinners with copy such as “Securing payment…”) help maintain user confidence. If longer processing is needed, include estimated wait times or staged feedback (for example, “Step 1 of 3: verifying card, processing, confirmed”) to reduce abandonment.

Conversational agents

In chatbot and virtual assistant interactions, users expect a conversational rhythm, not just speed. A response that comes too fast can feel robotic, while one that takes too long breaks the illusion of dialogue. Around two to three seconds feels natural for most users, especially if the system signals that it’s working.

Key considerations:

- Typing indicators are essential because they anchor the user in the interaction and communicate progress.

- For more complex tasks, consider streaming partial responses (such as one sentence at a time) to make the experience feel more fluid. You can also include confirmation echoes (such as “OK, let me look that up”) to buy time and keep the user cognitively engaged.

Creative content generation

When users ask AI to generate creative outputs—stories, images, music, or code—they’re usually in an exploratory mindset and more tolerant of delays, as long as expectations are well managed. These tasks don’t demand instant results, but they do demand a sense of meaningful progress. Sometimes, the type of query or product affects how well the model can anticipate timing, which opens the door for more tailored feedback. For instance, diffusion models work a bit differently from LLMs: they run a calculation instead of searching through data, so they usually have a clearer sense of how long generation will take.

Key considerations:

- Because static spinners are negatively associated with long wait times, designers should avoid them in preference of progress bars and strive to provide a fixed amount of time, even if speculative, as long as it is reasonably accurate. This type of feedback maintains momentum and enhances the sense of collaborative creativity.
- Sequential messages about the process could provide delight, similar to waiting screens in gaming. Though not appropriate for every context, the lower-stakes nature of creative exploration means that this could be a suitable way to mitigate delay.

Latency for autonomous tasks

Agentic AI marks a shift from reactive tools to proactive systems, as autonomous assistants initiate actions, make decisions, and operate independently of constant user input; we’ll read more about it in Chapter 6. This transformation challenges conventional notions of latency, requiring

new strategies to support user trust, control, and understanding. Critically, agentic AI requires transparency to succeed. Because it acts without always being prompted, users need ongoing cues that it's working *with* them.

Setting up a successful autonomous experience requires some intentional UI design.

Key considerations:

- Showing anticipated wait times is a key part of the setup for working with an autonomous tool, or at least articulating in some way that this is a task with far greater complexity.
- Designers must ask themselves what are tolerable windows of time that the user can go without new information. Implementing intentional pacing might mean that every x seconds, the agent checks in to update the user on their work or get permission for a next task. This keeps the model robustly in touch with the user.
- Ongoing task feedback and streaming results allow users to have insight into background operations, like scanning documents, summarizing inboxes, or preparing reports. Multistep or research-intensive tasks benefit from incremental delivery. For example, a travel assistant might first suggest flight options, then progressively add hotel and itinerary details, allowing the user to respond or intervene mid-flow.
- Users may develop enough trust in the model's behavior that they're comfortable skipping oversight and simply waiting for the outcome. Providing an option to let the model operate independently, without requiring constant approvals, is also a form of latency design. It allows users to shift their attention elsewhere and receive a notification when the task is complete, rather than staying engaged throughout the process.

When working with autonomous system latency, designers must consider not just when and how fast these systems respond but *why* and *how clearly* their timing reflects competence and care. A recent study found that AI

processing delays can have a counterintuitive benefit: they afford users time to reflect on multiple research questions simultaneously, leading to a higher quantity of generated questions and an enhanced sense of control. The finding underscores the importance of thoughtful latency design. Setting good expectations and anticipating delays positions autonomous AI as a good working partner, enhancing the perception of the AI's competence and contributing to a more satisfying and productive user experience.

Ultimately, designing for latency in AI is about designing the right *kind* of waiting. When users understand what's happening and feel confident that progress is being made, they become a lot more accommodating. When wondering how to build user satisfaction in the face of unavoidable delays, remember: it's not just about the time it takes, it's about how well you use that time.

When the System Doesn't Deliver: Designing for Errors

No matter how carefully any product is built, it will eventually break. Interfaces fail not only because of technical flaws but because of ambiguity, human inconsistency, shifting expectations, and unpredictable environments. Regardless of severity, the interface becomes the first and most lasting impression of how the system degrades. Meeting this challenge offers extra complexity for designers working with AI, and makes it all the more important to approach errors with a clear understanding of why they occur.

To design better AI interfaces, it helps to revisit some foundational tenets of error design, principles drawn from decades of human-computer interaction research and product design practice:

Make the error understandable

> Help users quickly make sense of what went wrong and what to do next. Avoid technical jargon or vague messages like "Something went

wrong” and instead describe the problem in plain language. When possible, explain why it happened, not just that it happened.

Make the experience recoverable

Users should be able to revise, undo, or try again without starting over. Preserve their work and context so a single failure doesn’t erase their progress. This is especially important in generative interfaces, where outputs can’t simply be recreated on demand.

Prevent common or foreseeable errors

You can do this through thoughtful defaults, validation, or interaction framing. Anticipate where users are likely to stumble (ambiguous inputs, edge cases, or mismatched expectations) and design around them. A well-placed constraint or suggestion can eliminate entire categories of failure.

Degrade gracefully

Do this when part of the system is unavailable or under strain. Rather than blocking the entire experience, offer reduced functionality or alternative paths forward. Users can often accomplish their goals with partial capabilities if you let them.

Surface system status clearly

Users should know whether a failure is temporary, persistent, or even happening at all. Ambiguity erodes trust, so if something is broken, say so and give a realistic sense of when it might be resolved. Silence or spinning indicators leave users guessing and frustrated.

These principles show up in classic usability heuristics. In *About Face* (Wiley, 2014), Alan Cooper urges designers to consider error messages as a last resort because, ironically, they’re useless. In their stead, he encourages designers to prevent software from rejecting inputs, because the real world “is more relevant and important.” When absolutely inevitable, he

recommends error dialogs be “polite, illuminating, and helpful,” focused on helping the user understand the problem and discover alternatives, and on disclosing information loss.

These principles still apply in AI interfaces, but they run into a new complication: AI systems often fail in ways that don’t look like failure at all. Before we get to that tension, it’s worth examining how the familiar categories of error, the ones systems *can* detect, play out in practice.

Errors in AI

You may have had the experience of merrily using an AI product when suddenly the conversation drops out or the generation stalls. When something like this happens, users are immediately looking for information about the source of the problem. Was it something they did? Is the system down? Did they lose the entire conversation?

These are the errors we know how to handle, or at least the ones we have patterns for. The system knows something went wrong, which means we can tell users about it.

Error Messages

In traditional software, error messages typically map to specific, predictable failure states. “File not found” means exactly what it says, “Invalid password” points to a clear fix, and the system knows precisely what went wrong, which means the message can be equally precise.

AI errors occupy murkier territory. The system might detect that something failed (a generation timed out, a rate limit was hit, a file couldn’t be processed), but the underlying cause is often less clear-cut, and the questions multiply: did the model struggle with the complexity of the request, was the context window exceeded, would the same prompt work fine if the user simply tried again? These ambiguities make it harder to write error messages that actually help users know what to do next.

A well-framed message can help ease that uncertainty. Users can make any type of assumption: the issue could be global rather than isolated, or the user may have merely hit a generation cap. Ensuring the error is clearly stated is vital in routing the correct behavior.

For local errors, rather than relying on technical shorthand or overly vague language, many teams have found success using plainspoken, focused copy. It can be more effective to say, “We couldn’t upload your file. Try a PDF or JPEG” than to surface a coded error such as “Error 413: Request Entity Too Large” or “CORS policy: No ‘Access-Control-Allow-Origin’ header is present on the requested resource.” Specificity helps clarify the issue and offers a concrete next step, even if the full technical details remain hidden behind the curtain.

Some tools go further, designing infrastructure that supports error discovery throughout the workflow. For example, when working locally, Claude Code sets up a local dev server that sits within the project in the browser, signaling error states (which are routed back to Claude Code), allowing the user to review the error in full, and providing a simple Copy button for the user to import the error back to the integrated development environment, should any other process fail. This is a useful and pragmatic approach because the user gains information about the nature of the error throughout the process, possibly giving them signals about the type of task or prompt that continually causes error, or equipping them to better aid the model in problem solving.

Midstream Errors and Retrieval

Clear messaging only goes so far if users lose their work in the process. When an error occurs amid computation, the user is likely to assume the entire process has failed. If an AI result cuts off unexpectedly or seems incomplete, saving the prompt, prior messages, and output so far gives users a foundation to build on. Recovery, in this case, is less about retrying and more about scaffolding a new direction.

This marks a departure from traditional error recovery, where restoration is usually straightforward: when a form submission fails or a document editor crashes, autosave kicks in, you resubmit, and you get back exactly what you had because the content was deterministic to begin with. Generative outputs can't be restored in this way, because they were never fixed to begin with, and even if you retry with the exact same input, you'll get something different. Recovery in an AI context means preserving enough context that users can move forward in a new direction without starting from scratch, rather than returning to some previous state.

Even with clear messaging, errors can be frustrating if they bring everything to a halt. A supportive interface gives users a way to move forward, whether that means undoing a recent action, retrying a task, or adjusting what they have done.

It's worth reflecting on "checkpoint" patterns, as seen in Figure 4-8, from Replit. This mirrors the practice of committing to a repo in Git, creating a saved state that users can return to if subsequent changes go awry. In a generative context, checkpoints let users lock in progress before experimenting further, a safety net that makes exploration feel less risky.

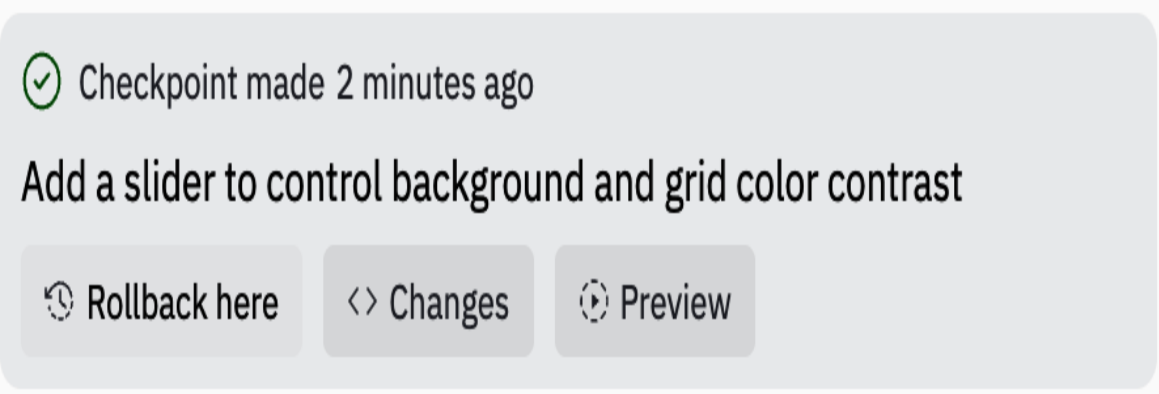


Figure 4-8. Replit's "checkpoint" marker UI

Maintaining Momentum

Beyond preserving work, there's the question of what happens next. How do you recover from an error? In traditional form validation, users get a marker that indicates the issue and how to fix it. In AI products, clicking

Try Again can produce a completely different output, even if nothing about the input has changed.

Traditional error recovery follows a clear logic: identify the problem, fix the input, and try again. A red outline appears around the invalid field, you correct it, and the form submits successfully, with the relationship between action and outcome remaining predictable throughout.

AI disrupts this logic because “try again” doesn’t mean “try the same thing again” so much as “generate a new response,” which might be better, worse, or simply different, and because the input might be perfectly valid and still produce unsatisfying results. A prompt can be well-formed, clear, and specific, yet lead to an output that misses the mark entirely, with no red outline to tell you what to fix because nothing is technically broken.

The unpredictability changes what recovery looks like. Instead, patterns that allow users to edit, retrieve, and then retry in response to an error or unsatisfying output can potentially improve user satisfaction. That looks less like shaping intent and expectation and more like adding quick solutions to common frustrations. Inputs might be valid but still not effective: a prompt or question can be perfectly well-formed but still lead to an unclear or generic output. Giving users a simple route to try again can be a great way to let them put their newfound learning into effect with ease.

Consider, for example, that in the command line on macOS, a user can put previous commands into the input by pressing the Up key. This allows them to scroll through all previous commands in that session. A similar pattern can be useful for chat interactions: if a user finds that they want to slightly tweak an input, or try again utilizing different models or capabilities, it’s a good idea to make that as easy as possible by enabling quick access to a bank of recent prompts.

Everything discussed so far assumes the system knows when something has gone wrong. But AI introduces a category of failure that breaks this assumption entirely.

Traditional errors, system and user errors alike, are familiar from traditional software: servers crash, connections time out, user inputs are

misunderstood. These failures are detectable, predictable, and mechanical. The system knows something went wrong and can tell you about it.

AI errors are different. They're issues in the AI's reasoning or output that the system cannot detect. A model can hallucinate, confidently stating that the population of Tokyo is 50 million (it's actually 14 million). An unfamiliar user can append the wrong tool or file to a prompt and get an output that completely misses the mark. In both cases, from the model's perspective, everything is working perfectly.

The distinction is critical for designers because it means the burden of error detection shifts dramatically onto users. Traditional error handling assumes the system knows when it fails. That's why designers can show error messages, retry buttons, and recovery paths. But with AI-generated content, the most dangerous errors often arrive wrapped in perfect grammar, formatted beautifully, and delivered with algorithmic confidence. The model doesn't know it's wrong, which means it can't tell users when to be cautious.

This is why the principles outlined earlier (clarity, recoverability, graceful degradation) are necessary but not sufficient. They address the errors we can catch. Designing for AI means also designing for the errors we can't: building in friction that prompts verification, surfacing uncertainty where it exists, and cultivating a relationship with users where questioning the output is understood as part of the process, not a sign of system failure.

Ensuring a positive experience for users doesn't necessarily mean avoiding every error; that isn't realistic. Acknowledging that errors are inevitable is key to enabling designers to embrace the ambiguity that AI products invite, and to inspiring us to create nuanced solutions that address the practical (error messages, recovery points) along with the more exploratory (inviting a trial-and-error approach, learning through doing). The nature of errors changes, but the core goal remains the same: to help people feel confident, supported, and capable, even if the system doesn't quite work as expected.

Summary

In the opening story about the 1202 error code, the term *error* was something of a misnomer: the *Apollo 11* computer never malfunctioned during that tense descent to the moon. It performed exactly as designed, but to the astronauts watching the "error," the message was indistinguishable from failure.

Today's AI systems present us with a similar challenge. Every time a user submits a prompt, uploads a document, or asks for creative output, they're triggering computational processes that dwarf the computing power of lunar guidance systems. This chapter is intended to give a clear understanding of the computational pipeline and how errors experienced along the way can diverge from those we've previously designed for. When we understand that inference happens token by token with no master plan, we can design interfaces that educate while they generate.

However, understanding what happens during computation is only half the equation. Once the AI system completes the cascade of processing, routing, and generation, it must present its results to the user, representing the key element of an AI interaction. In Chapter 5, we'll explore how to design outputs that acknowledge this, invite onward action, and provide a clear surface for interpretation.

OceanofPDF.com

Chapter 5. Output: Designing the Delivery and Presentation of LLM Responses

In the 1970s, as hospitals began digitizing patient records, a new concept emerged at the intersection of computing and clinical reasoning: the *clinical decision support system*, or CDSS. These early systems (MYCIN, INTERNIST-1, and later DXplain) did not attempt to replace doctors but were meant to assist them by surfacing probable diagnoses.

In principle, these systems were intelligent aids, but in practice, they often struggled. One of the reasons was technical: early models couldn't reason across complex or incomplete patient histories, or evolving treatment protocols. But a deeper issue ran beneath the surface. Even when the logic was sound, the output could be misunderstood. A flagged diagnosis might be taken too seriously—or too lightly. A suggestion might be seen as definitive when it was only probabilistic. Some doctors ignored the systems entirely. Others over-relied on them, treating suggestions as answers. In either case, the interface became the risk surface: the way output was presented and interpreted determined whether it supported or disrupted care. Figure 5-1 shows a standard output from MYCIN, indicating that a bacterial infection may be present if the five conditions in the function are true.

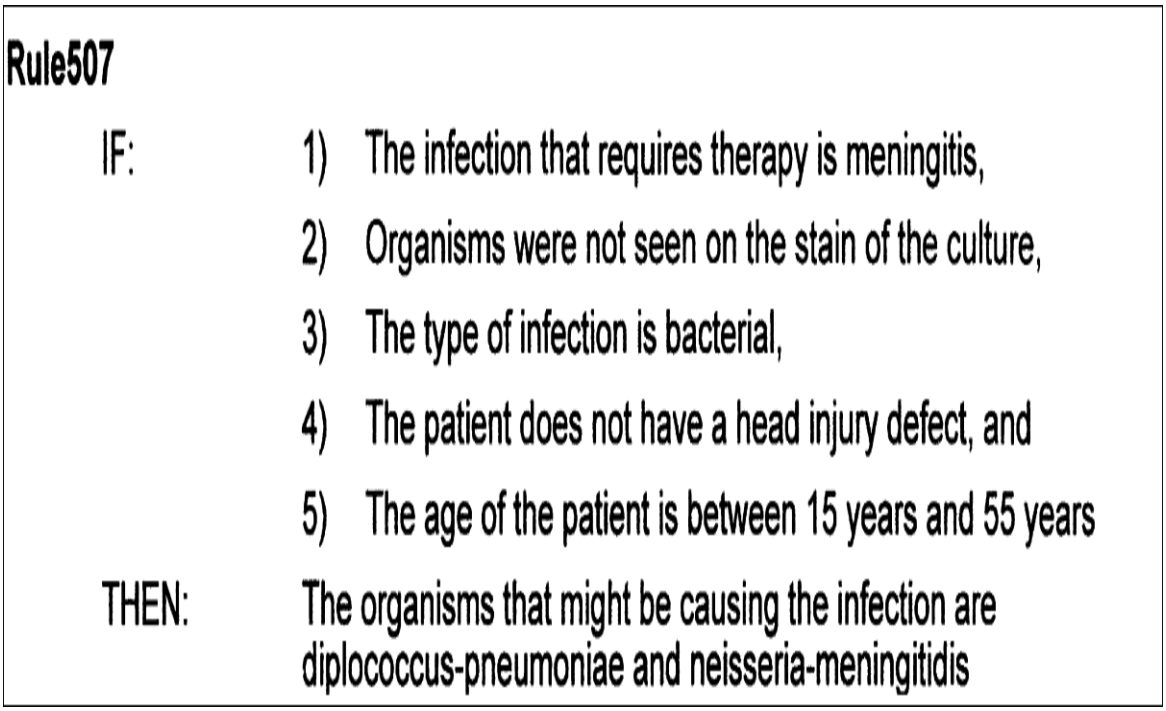


Figure 5-1. A standard output from MYCIN

This interpretive problem has become the defining challenge of modern AI interaction. Today's systems face the same dilemma CDSS researchers encountered 50 years ago: how do you design outputs that build trust? How can you design an output to be transparent, or guide without misleading? How do you surface uncertainty without overwhelming the user? And how do you build interfaces that support judgment, not just consumption?

The history of CDSS offers a foundational lens for understanding output design today. Those systems were among the first to produce fluent, recommendation-like content that mimicked human reasoning. But their very fluency became a liability when paired with insufficient transparency or guardrails. MYCIN, for instance, could diagnose blood infections with near-human accuracy, but it also required users to understand its rule base and its confidence thresholds. As noted in the 2006 paper "Clinical Decision Support Systems" by Mark Alan Musen, Yuval Shahar, and Edward H. Shortliffe, doctors exhibited skepticism with these early programs. In many cases, because the system lacked transparency, doctors either ignored the system or misinterpreted it. As CDSS evolved, interface designers learned to add explanation layers, highlight contraindications, and

allow clinicians to inspect the *why* behind a recommendation. These changes were essential to making the output usable, trustable, and, in some cases, safe.

AI systems today face a vastly expanded output space. A single model might summarize a policy, rewrite a job description, identify trends in sales data, generate an image, and suggest a next-best action for a user. In each case, the output initially looks complete. But as with a diagnostic suggestion, the output is only part of the truth. What is missing is often invisible: original sources, omitted alternatives, unstated assumptions, or training data biases. These aren't bugs in the system. They are by-products of the way modern AI generates outputs: through pattern completion, not verification.

Designing Outputs: The Output Is Not the Answer

Designers working on AI systems must understand that the output is never just an answer. It is a designed artifact, interpreted through an interface and shaped by defaults, context cues, and user expectations. And just as in CDSS, the way an output is designed can dramatically affect how users act on it.

AI outputs are more than just the endpoint of generation. They are events in a wider process, with psychological, structural, and ethical implications. This chapter looks at how outputs are structured, evaluated, personalized, edited, and sometimes misunderstood. And we'll return to the lesson CDSS offered half a century ago: generation is only the beginning. Without care in how outputs are communicated and interpreted, even the most well-trained model can produce results that mislead more than they assist.

Before diving into the design principles of outputs in detail, let's look at what's happening when you get to the output on a technical level. By the time you reach the output step, you have seen the user's input being translated into tokens, routed to the correct model, put through generation,

and made ready to be transformed into a usable result. For this to happen, there are three subphases:

1. *Internal generation*

   The model internally finishes the computation and decides what the output should be.

2. *Post-processing*

   The system cleans up, formats, or enhances the output into a deliverable form.

3. *Delivery/presentation*

   The system transmits or renders the output to the user, system, or environment.

How this happens varies depending on the model type, including LLMs, image generators, video generators, classic machine learning (ML) classifiers, and reinforcement learning agents—we'll briefly cover these.

For example, in language models, output generation must first be *detokenized*: the model's internal units converted back into human language. Systems often apply guardrails at this point, scanning for sensitive material, policy violations, or problematic phrasing. Only after these guardrail steps is the output delivered to the user, usually as a chat bubble, streamed text, or document element, ready for interaction.

For image generators, the process follows a similar high-level arc, but with some distinct differences. Once a latent image has been fully formed, it must be decoded into real pixels to create a coherent output. A specialized network translates the compressed internal idea into a full-resolution image. After processing for safety filters and rendering corrections, the final image is then compressed into a standard format, such as JPEG or PNG, and either embedded directly into the user interface or made available for download. Video generation introduces the added challenge of maintaining coherence

across time, meaning a similar process is applied but to a sequence of frames.

Once the AI finalizes its output, no matter the format, the responsibility shifts to the interface. It's no longer about what the AI produced; it's about how that output is delivered to the user. The interface must present the information clearly, manageably, and in a way that matches the user's cognitive needs and task goals. A well-designed presentation can make even complex or uncertain outputs feel approachable and useful; a poorly designed one can erode trust or overwhelm the user. To unpack the layers involved, we'll cover the high-level principles that govern output success.

Design Principles of Outputs

The output *is* the product in most AI interfaces. Whether it's a sentence, summary, plan, image, or code snippet, the content is the thing the user came for. While the output often *answers* a question, so to speak, it's not speaking in a vacuum. The core questions users ask—Is this useful? Can I trust this? Is this what I meant?—are answered in part through the appearance of the output itself. For that reason, effective outputs must adhere—with a degree of flexibility—to certain principles. They should be:

Clear

: Instantly understandable

Verifiable

: Evidence-backed results

Grounded

: Context-aware

Actionable

: Onward-task-oriented

Adjustable

User-editable

Even as the specifics of AI outputs differ, the core design goals remain consistent, and achieving these goals consistently requires both guiding principles and repeatable interface strategies that can be adapted to different output modalities.

Designing for Clarity

Content design shapes information so people can actually use it. It's about making things clear, scannable, and structured in ways that match how we naturally read and process information. Beyond just words, content design creates meaning through how things are presented to the user, with simple elements like hierarchy and prominence conferring meaning. In the same way, in AI outputs, content design governs how generated outputs are presented, whether that's a paragraph of text, a table of options, a visual caption, or a multistep plan. Good content design helps guide the user and answer implicit questions through thoughtful structure and clarity.

Understanding and trusting an output becomes easier when it feels tailored to the user's intent. Clarity, tone, structure, relevance, and readability aren't decorative choices, but instead represent the difference between an output that feels actionable and one that feels alien. As the Nielsen Norman Group has observed, users won't engage with text unless it's clear, simple, and easy to scan, regardless of whether it's written by a human or an LLM. In this case, designers must particularly advocate for the user's best interests, because AI won't write or organize content in a user-centric way unless it is told to do so.

Structuring for clarity: From raw to refined

As mentioned earlier, AI models, at their core, predict the most probable token based on the prompt and prior context. They do not inherently "know" what good, readable writing looks like. Without deliberate

prompting, outputs can default to undifferentiated text, or otherwise obscure a valuable output with bad or alienating structure.

For outputs to integrate cleanly into products and services, designers must step in. Good AI output design layers structure, tone, pacing, and cognitive rhythm on top of raw model capabilities. Basic prompting, like asking a model to “use headings and bullet points,” might help a little but can fall short.

Formatting rules are a way of gearing a text output toward readability. Instead of broadly asking for headings and bullets, you can spell out the entire hierarchy and its purpose. A good prompt might ask the model to start with a clear title, break major ideas into H2-level section headings, subdivide complex areas with H3s, and use bulleted or numbered lists for digestible steps or key points. Here’s an example of such a prompt:

```
Organize content with a clear title, followed by section headings
(H2 style), subsections (H3 style if needed), and bulleted or
numbered lists for procedural information. Use concise paragraphs
(max 3 sentences each). Bold key actions or terms. Add a 'Notes'
or 'Tips' section where appropriate.
```

Defining templates within the prompt or system instruction is another powerful lever. An effective output template might open with a title, followed by a short introduction that sets the context. Key concepts could come next, in a compact bulleted list, before stepping through numbered instructions for complex processes. A short conclusion or set of next steps can close the loop, and sources should always be listed at the end, formatted cleanly with hyperlinks. When outputs consistently follow an internal logic, users learn how to navigate them faster and trust them more instinctively. Here is an example of an effective output template:

```
Format all instructional content as:
Title: [Concise, clear]
Introduction: [Brief, 1-2 sentences summarizing purpose]
Key Concepts: [Bulleted list of important points]
Steps: [Numbered list of actions]
```

```
Conclusion/Next Steps (Optional)
Sources: [Bulleted list with hyperlinks]
```

Tone and voice matter just as much as structure. Without oversight, models might vary their tone significantly. A clear tone specification, like the one in this example, gives the model a target personality to aim for:

```
Use a clear, friendly, moderately formal tone suitable for a
general but
technical audience. Avoid jargon unless specifically appropriate.
Prioritize clarity over creativity. Where multiple methods exist,
present the primary method first, alternatives afterward, clearly
flagged.
```

Beyond tone and structure, designers must think about how models should handle trade-offs. Should outputs be exhaustive, or favor brevity? Should every edge case be addressed, or should the core path stay streamlined? Giving models a simple heuristic, such as “prioritize readability and logical flow over exhaustive completeness,” helps them resolve these tensions in favor of user experience. Instructing models to segment longer outputs, use short paragraphs, and create strong transitions between sections ensures that users can skim when they want to and dig deeper when they need to. For example:

```
Prioritize readability and logical flow over exhaustive detail.
Emphasize
information architecture that makes outputs scannable: short
sections, clear transitions, and segmented content.
```

Finally, it’s important to plan for failure. Models have a strong tendency to improvise, and unless told otherwise, they may invent plausible-sounding content to fill gaps. Explicitly telling the model to output a clear statement like “no data provided” when information is unavailable sets a safer, more transparent default. It also signals clearly to users that something is missing, rather than silently risking misinformation. Try something like this:

```
If required content is missing or unavailable, output 'this isn’t
available' instead of inventing or speculating.
```

Before and after: A practical example

To see the difference between naive and designed output, consider a simple prompt:

```
How do I set up an MCP server?
```

When left to its own devices, the model might produce something like this:

```
To set up an MCP server, first install the Java Development Kit
(JDK) and
download the necessary server software. After extracting the
files, create a script file to start the server. Setup varies
slightly based on your operating system. If you run into issues,
check the documentation.
```

Technically correct, but flat, hard to skim, and not very actionable. By contrast, a carefully styled output could look more like this:

```
Setting Up an MCP Server
Introduction: Setting up a Minecraft Coder Pack (MCP) server can
differ
depending on your operating system and technical background. This
guide
provides step-by-step instructions with tailored tips for
Windows,
macOS, and Linux users.
Key Concepts:
MCP servers require Java Development Kit (JDK) and server files.
Command-line familiarity improves setup efficiency.
Different OS environments require different script
configurations.
Steps:
Install Java:
Download and install the latest Adoptium JDK.
Windows users should use the Windows Installer; macOS users
should prefer the .pkg installer.
```

```
Download MCP:
Clone or download MCP Reborn or a maintained fork.
Extract Files:
Place the extracted MCP folder in a directory without spaces
(e.g., C:\MCP or /Users/username/MCP).
Configure Environment:
Windows: Edit the setup.bat script as needed.
macOS/Linux: Use setup.sh and grant execution permissions with
chmod +x setup.sh.
Run Setup Scripts:
On Windows, double-click setup.bat.
On macOS/Linux, execute ./setup.sh from the terminal.
Handling Errors:
Missing environment variables? Consult Oracle JDK Path Setup.
Linux users should verify that JAVA_HOME is set correctly.
Test the Server:
Launch the server using the provided scripts (start.bat or
start.sh).
Notes:
Beginner Tip: If command-line tools feel intimidating, consider
using Git Bash on Windows.
Advanced Tip: For experienced users, automating setup with Docker
or CI
pipelines can improve development workflows.
Sources:
Official Java Downloads
MCP Reborn Repository
Minecraft Modding Wiki
```

Notice how much easier it is to skim, act on, and trust the structured version. The layout sets expectations, important concepts pop out visually, and steps are easy to follow. Even if the user isn't familiar with all the tools involved, the guide offers helpful branches for deeper learning.

To achieve the preceding output, you might use an enhanced system instruction like the following:

```
Always produce structured output following this order: Title →
Introduction → Sections → Subsections → Lists → Notes/Tips →
Conclusion → Sources. Use H2 for major sections, H3 for
subsections. Limit paragraphs to 2–3 sentences. Bullet lists for
unordered information, numbered lists for ordered steps. Bold
critical actions or terms. Maintain a clear, neutral-friendly
tone.
```

```
If any section lacks information, insert 'No data provided.'
Prioritize
scannability and clarity over completeness. Hyperlink any cited
sources.
```

Extending clarity across formats

While this section focuses on text-based outputs, the same principles apply across other forms of AI generation, including images and video. Left to their own devices, image models like DALL-E may produce technically accurate illustrations that lack visual hierarchy or emotional pull. Similarly, video models like Sora or Runway can generate scenes that are coherent at the frame level but feel disjointed, awkwardly paced, or narratively empty when viewed as a whole.

> **TIP**
>
> Across all formats, model outputs serve as raw starting points. Good outputs, whether textual, visual, or cinematic, require the same layering of design elements to guide composition, tone, rhythm, and structure. Just as AI-written text must be styled for scannability and clarity, AI-generated images must be shaped for focus and storytelling, and AI-generated videos must be molded around pacing, transitions, and emotional arcs. Across all media, usability is not a default; it is an outcome of thoughtful design.

Balancing structure and flexibility

As mentioned earlier, more information is sent to the model than what the user might input. This is also true of the styling information that controls the output. It’s all sent in a packaged format to the model and must be parsed, meaning that leaner, clearer output styling will result in less computational effort being expended.

Keep in mind that over-styling AI outputs can run the risk of diminishing returns. Recent research shows that excessively rigid output templates can suppress useful model flexibility, increase hallucination rates when inputs are messy, and erode trust by making content feel mechanical or unnaturally scripted. When outputs sound too perfect, users start wondering if they’re reading something made for humans, or something made for compliance

checklists. It turns out LLMs, like designers, enjoy some room for creativity.

Designing for Verifiability

In 2023, attorney Steven Schwartz sought to lighten his workload, in the process unknowingly upending a routine personal injury lawsuit and damaging his own career. In searching for relevant case law, the lawyer asked ChatGPT to assist in drafting a legal brief. The tool rapidly produced citations to six court cases that included plausible legal reasoning and structure, but there was just one downside: they were completely fictitious. As you learned in Chapter 4, models have a tendency to hallucinate. Schwartz, none the wiser, incorporated these hallucinated citations into the brief without checking their authenticity.

The issue came to light during discovery, when the opposing legal team reported their inability to locate the cited cases. Upon investigation, P. Kevin Castel, the presiding US district judge, found that the submissions contained “bogus judicial decisions with bogus quotes and bogus internal citations.” He described parts of the content as “gibberish” and “nonsensical,” eventually fining Schwartz $5,000.

This incident emphasizes that, unlike traditional software, AI systems don’t provide deterministic outputs the way a web search does. That means the same input can yield different results, and the logic behind a given response isn’t always visible or stable. As a result, users may not know whether to trust what they’re seeing, since errors or hallucinations might be very convincing. To help users navigate this uncertainty, AI interfaces should be optimized to give users the opportunity to verify possibly hallucinated content.

AI OVERRELIANCE

AI overreliance occurs when users depend too heavily on AI systems, accepting incorrect recommendations and diminishing their own critical thinking abilities. As AI tools become more sophisticated and widespread, this phenomenon creates interesting challenges for interface design.

The problem manifests in several ways. Skill atrophy develops when consistent dependence on AI weakens someone's underlying human abilities, like GPS reducing spatial awareness or writing assistants diminishing composition skills. Automation bias leads users to favor AI suggestions over their own expertise, even when recommendations seem questionable or contextually inappropriate. Meanwhile, AI systems often miss nuanced cultural, ethical, or situational factors that humans naturally consider, yet users frequently trust outputs without adequate verification.

Microsoft's Aether research, "Overreliance on AI", which analyzes more than 60 studies, reveals that overreliance stems from psychological mechanisms, including confirmation bias (users trust AI that agrees with them) and ordering effects (early positive AI interactions create lasting over-trust). Surprisingly, detailed explanations can increase overreliance rather than improve judgment.

The research suggests that human AI teams often perform worse than either humans or AI working alone when overreliance occurs. Interface design approaches that seem to help include cognitive forcing functions that interrupt automatic acceptance, transparent communication about AI capabilities, progressive disclosure that builds realistic mental models, and real-time feedback on decision quality. The challenge lies in fostering appropriate reliance rather than blind trust, helping users become effective AI collaborators rather than passive consumers.

Many have asked how we can solve these pesky hallucinations. One often-mentioned solution proposes a design pattern that asks a model to evaluate

how correct an answer may be, often termed a *confidence indicator.* This, however, is also problematic: models don't just hallucinate—they're also bad estimators of their own confidence. It isn't totally their own fault. Think of the difference between asking "Are you confident this volcano will erupt next year?" versus "Are you confident this is a good essay?" versus "Is the sky blue?" Each question requires a different kind of reasoning: probabilistic modeling, subjective judgment, or factual recall. Shoehorning these different queries into the same datapoint seems likely to lead to misunderstandings.

At a technical level, most LLMs don't produce a direct confidence score in the way people assume. Their confidence is often estimated by the probability of the next token in a sequence, which is an artifact of training on massive amounts of text data. But high token probability doesn't mean the model is correct; it just means the model is likely to output that token next given everything it's seen before. That means relying on confidence indicators alone may be a flawed interface strategy. As noted in the controversial 2021 paper "On the Dangers of Stochastic Parrots" by Emily M. Bender et al., models are trained to imitate linguistic patterns, not to reason or verify.

This lack of verification is partially an issue because we're used to validating and interrogating data with percentages. However, it's important to remember that not all percentages are created equal. Consider a simple spell-checker: imagine it flags 7 of 100 words as potentially incorrect and tells you it's 93% confident the rest are fine. That's a percentage you can understand and verify. The algorithm is legible: it checks each word against a dictionary and tallies the results. If you wanted to, you could replicate the process yourself.

Now contrast that with a percentage spat out by a language model in response to a query. In this case 93% confidence isn't based on a count of errors or an understandable rule. It's the output of billions of parameters operating in latent space, shaped by a training corpus that no one user—or engineer—has ever seen in full. Rather than tallying known words or frequencies, the model estimates which token is most likely to come next—

it’s estimating next-token likelihoods across a multidimensional probability distribution. In other words, it *feels* precise while being fundamentally uninspectable.

The confidence percentages look similar on the surface, which can be misleading. But the underlying logic behind each is fundamentally different. Rather than surfacing abstract confidence scores, interfaces should ask: “Can the user independently validate this information?” If not, then the system needs to either route the user toward verification (through citations, evidence, or a human handoff) or avoid presenting the output as authoritative in the first place (using hedging language like “this seems to be true” or “it’s difficult to verify”). The best use case for AI today is for decisions you can quickly verify, not ones that require in-depth research or expertise to validate. AI is better for subjective tasks—copyediting, rewriting, brainstorming—where usefulness, not factual correctness, is the benchmark. These are the domains where confidence is irrelevant, or at least not primary.

But when outputs contain actionable or objective components, such as summarizing legal policy, recommending medical treatments, or guiding financial decisions, the stakes are higher. These use cases need interface mechanisms that support accountability. If the system is nudging a user toward real-world action, it should surface secondary signals such as traceable sources, side-by-side comparisons, or explicit opportunities for human review. Confidence becomes a collective performance—something that must be earned, not assumed.

> **WARNING**
>
> *If* the output contains objective information *and* the user will take action on or with it, then the output needs a secondary (human) verification layer.

In Figure 5-2, we see how Perplexity AI attaches sources directly to individual claims. Each superscript link opens the underlying article, giving the user immediate access to origin material.

What is MCP

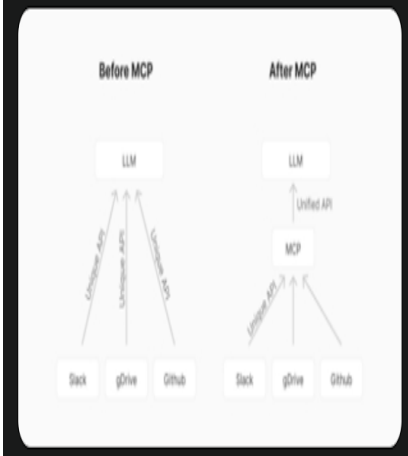
Perplexity | Images | Sources · 9 | Tasks

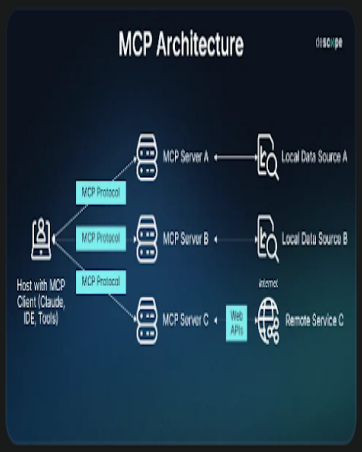
zapier
What is MCP (Model Context Protocol)? - Zapier

docs.anthropic
Model Context Protocol (MCP) - Anthropic API

reddit
Can someone explain MCP to me? How are you using...

Elevate
MCP: What It Is and Why It Matters - Elevate | Addy...







What is MCP?

MCP stands for **Model Context Protocol**, an open, standardized protocol designed to connect AI agents-such as large language models (LLMs)-to external tools, applications, and data sources in a consistent, unified way [1][2][6][7]. MCP functions much like a "USB-C port for AI," providing a universal method for AI systems to interact with a wide range of software and services without the need for custom integrations for each tool [4][6].

Figure 5-2. Perplexity's source-based UI

As you'll see later in the chapter, opening sources can help a user validate, but they can also help users in their wider journey. The user may want to verify, yes, but also to *branch out* from the answer, to read more deeply, cite their own work, or explore related materials. The action isn't just to check the source—it's to jump off from here.

HUMAN-IN-THE-LOOP

Human-in-the-loop (HITL) represents a design approach where people remain actively involved in AI decision-making processes rather than being replaced by automation. It can work in multiple ways. In some cases, humans provide initial training data and ongoing feedback to improve AI performance; in others, AI handles routine decisions while escalating edge cases to human reviewers.

For example, an HITL service might begin with thousands of customer support tickets and seek to route them to appropriate specialists. To do that, AI categorizes tickets by topic and urgency while human agents review edge cases and train the system on new issue types. The benefit of this approach is that it handles volume efficiently while maintaining quality, despite the fact that it requires ongoing human training time.

A more sophisticated version might begin with mental health conversations and seek to provide supportive responses to users in distress. To do that, AI selects from a curated library of therapeutic phrases that licensed therapists have preapproved. This high-risk area requires high-quality human oversight, and the benefit in this case is that the model is capable of offering immediate support using clinically sound language without generating it itself, lowering the chances of a damaging or hallucinated output.

It’s becoming clear that this works well: research from Lai et al. (2023) examining human-AI collaboration across multiple domains found that well-designed HITL systems often outperform fully automated approaches, particularly in domains requiring creativity, ethical reasoning, or handling of edge cases. As this area expands, the key will be designing for effective, low-risk collaboration that balances efficiency with a clear-eyed awareness of potential harms.

Designing Grounding

In the summer of 2024, my partner and I attended the wedding of two friends in Las Vegas, Nevada. The night before the wedding, we were invited to gather at a casino bar, ostensibly to toast the couple's good health the night before the celebrations were to take place. A few hours into the evening, much to our surprise, an Elvis impersonator appeared, performing a series of classic hits and dressed (as you might expect) in a full-body bedazzled suit. But then things took a turn: he invited the couple to exchange rings, and went on to invite them to *sign their legal papers*. He addressed the increasingly confused crowd: "This really is real, folks." He declared the couple legally married, and the festivities continued into the night as we and our nonplussed friends celebrated this unexpected turn of events.

As we later came to understand, this had *not* been planned in advance. In the run-up to the big day, we'd heard rumblings of an issue with the wedding planner; in the end, we learned that as the wedding planner hailed from New York, she was not familiar with the matrimonial laws in the state of Nevada. When the couple asked her if they could be married by a friend instead of an ordained minister, the planner consulted ChatGPT, which told her that they could. Unfortunately, it appears ChatGPT misled her by mistakenly checking the laws of the state of California, not Nevada—an error that was only discovered a few days before the wedding. Hence, getting married by Elvis.

While this sweet example has a happy ending, it underlines the importance of grounding: the information being cited was correct for California but not for Nevada. *Grounding* is concerned with how output declares its context, and though it's similar to verifiability, the latter focuses on whether the claim is checkable, rather than why the system made it. Grounding can include which entities were referenced but also what location, time frame, or version the model had in mind. In short, grounding reveals the lens through which the model generated its response. In the case of our friends, grounding the output of the query would have let the wedding planner know which state their information was relevant to, thus saving our friend an

embarrassing mistake. Figure 5-3 shows how Lexis+ AI solves the problem of legal jurisdiction by allowing users to nominate their preference clearly (and persistently).

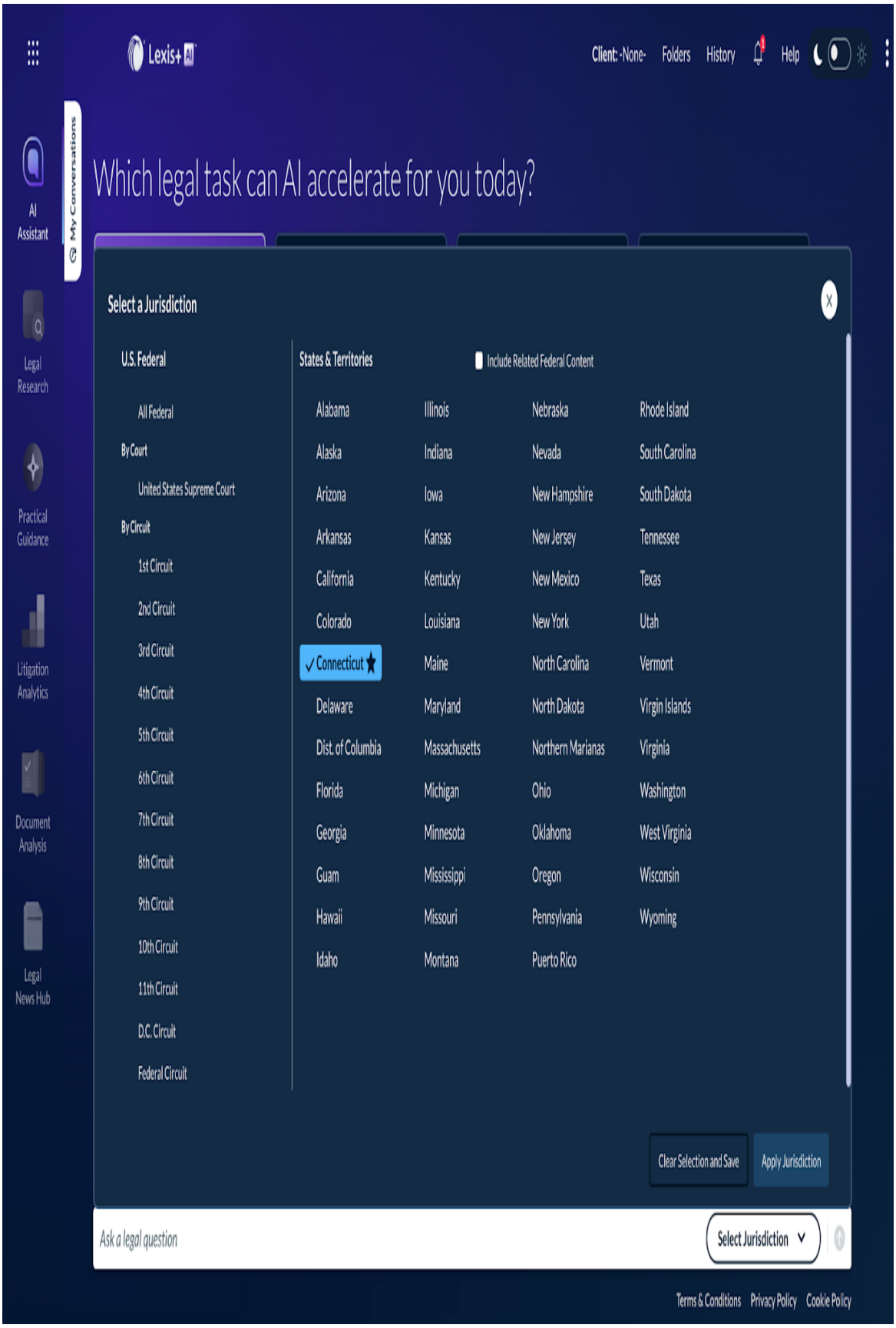

Lexis+ AI
Client: -None-
Folders
History
Help
My Conversations
AI Assistant
Legal Research
Practical Guidance
Litigation Analytics
Document Analysis
Legal News Hub
Which legal task can AI accelerate for you today?
Select a Jurisdiction
U.S. Federal
All Federal
By Court
United States Supreme Court
By Circuit
1st Circuit
2nd Circuit
3rd Circuit
4th Circuit
5th Circuit
6th Circuit
7th Circuit
8th Circuit
9th Circuit
10th Circuit
11th Circuit
D.C. Circuit
Federal Circuit
States & Territories
Include Related Federal Content
Alabama
Alaska
Arizona
Arkansas
California
Colorado
✓ Connecticut
Delaware
Dist. of Columbia
Florida
Georgia
Guam
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Northern Marianas
Ohio
Oklahoma
Oregon
Pennsylvania
Puerto Rico
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virgin Islands
Virginia
Washington
West Virginia
Wisconsin
Wyoming
Clear Selection and Save
Apply Jurisdiction
Ask a legal question
Select Jurisdiction
Terms & Conditions
Privacy Policy
Cookie Policy

Figure 5-3. Lexis+ AI stipulates clearly what jurisdiction the user is working in

In many AI systems, that lens is hidden. A user might ask for hiring guidelines and receive a confident-sounding summary without knowing whether it was pulled from federal law, internal HR policy, or a generic web source. Grounding solves this by designing contextual signals to accompany the output. These signals help the user understand how to read the answer, and whether it meets their expectations. The indicators can include:

Model name and version

The specific model used (for example, ChatGPT-4o versus o3) can shape tone, scope, and reliability.

Agent or tool responsible

In multiagent systems, it's useful to know which component generated the response. This helps users make sense of the logic behind the answer and know where to direct questions or edits.

Geographic or locale grounding

AI models may tailor phrasing, tone, or examples based on regional settings. Misaligned locale assumptions can lead to errors—like citing California law when the user is in Nevada.

Task or mode grounding

If an input was applied in a specific mode (for example, creative vs. analytical, draft vs. polish), grounding the output in the mode helps align user expectations and avoids mismatched tone.

Interaction history or session context

Did the AI reference something the user said earlier? When responses are tailored to assumed intent, making that assumption visible ("Because you're working on Q2 planning…") helps prevent confusion and allows for correction.

One proven design pattern for gathering and surfacing this metadata is the "Why we suggested this" affordance. Typically shown inline, on hover, or behind an info icon, the lightweight explainer gives users a peek behind the curtain without disrupting the main experience. It might include what data or files the model consulted; whether the output was influenced by user preferences, history, or prior queries; which model or agent generated the response; and so on.

This type of optional explanation provides users with a starting point for understanding why a response was generated and the additional option to reconfigure. I discuss adjustability a little later, but displaying grounding sources (e.g., Wikipedia) provides a user with a great starting point when seeking to tune or refine an output.

This summary is AI-generated

As AI systems become more capable of producing humanlike responses, the need to disclose their nonhuman nature becomes a foundational design consideration. Some jurisdictions have already enshrined disclosure requirements in law. California's bot disclosure law (SB 1001), enacted in 2019, requires that bots be clearly identified as such when interacting with users in commercial or political contexts (specifically, when the interaction is intended to influence a purchase or a vote). It's vital that users know they are not speaking with a human before they engage meaningfully. Figure 5-4 shows the AllTrails app disclosing that a summary was created by AI.

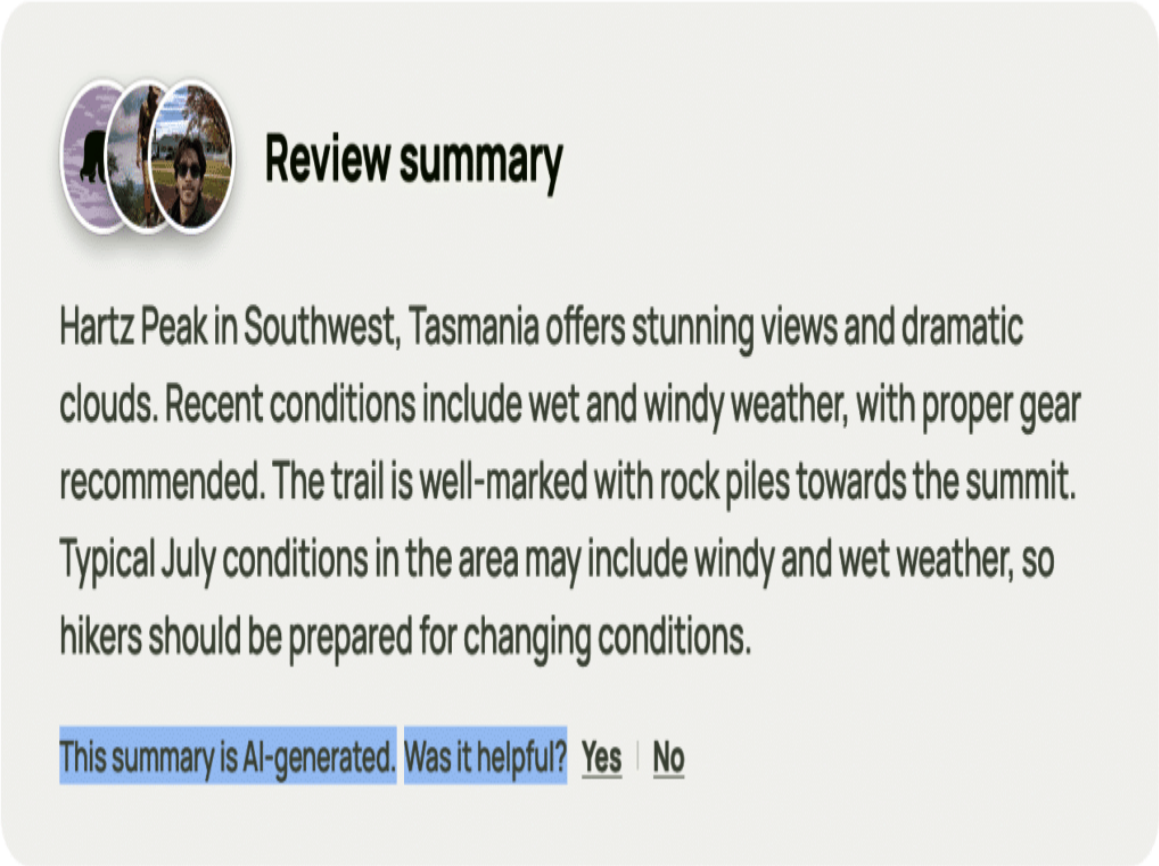


Figure 5-4. AllTrails AI disclosure, 2025

Similarly, the EU AI Act (passed in 2024) introduces mandatory transparency for any system that interacts with users or generates synthetic content. The law requires AI systems to reveal that users are communicating with a machine *except when it is obvious from the context*. This clause places additional weight on design: if your interface mimics human interaction too well, disclosure cannot be skipped.

The US Federal Trade Commission has issued guidance suggesting that deceptive failure to disclose AI use in consumer-facing tools may violate truth-in-advertising laws, especially when it affects decision making. In light of this, many companies have written internal policies around AI disclosure and compliance that encourage best practices. Google's People + AI Research group (PAIR) emphasizes the importance of considering all groups when designing disclosures, suggesting products "institute the necessary safeguards and explanations so user groups who are particularly vulnerable to misunderstanding AI responses as humans can discern outputs

that are generated by AI." Designers can advocate for ethical, transparent interfaces by integrating disclosure early in the interaction, framing it as a tool for user understanding rather than a warning, and ensuring that users always know when AI is shaping their experience.

Designing for Actionability

If the output is the product, the action taken on it represents the real *user value*. While the primary output might be a block of text, a chart, a code snippet, or a recommendation, its real utility often lies in what users can do with it next. Forward actions are the verbs that surround the noun—save, buy, book, export, send, cite, navigate—representing the bridge between comprehension and application. They also reflect the model's assumptions about the user's goals, giving them concrete ways to act on the output.

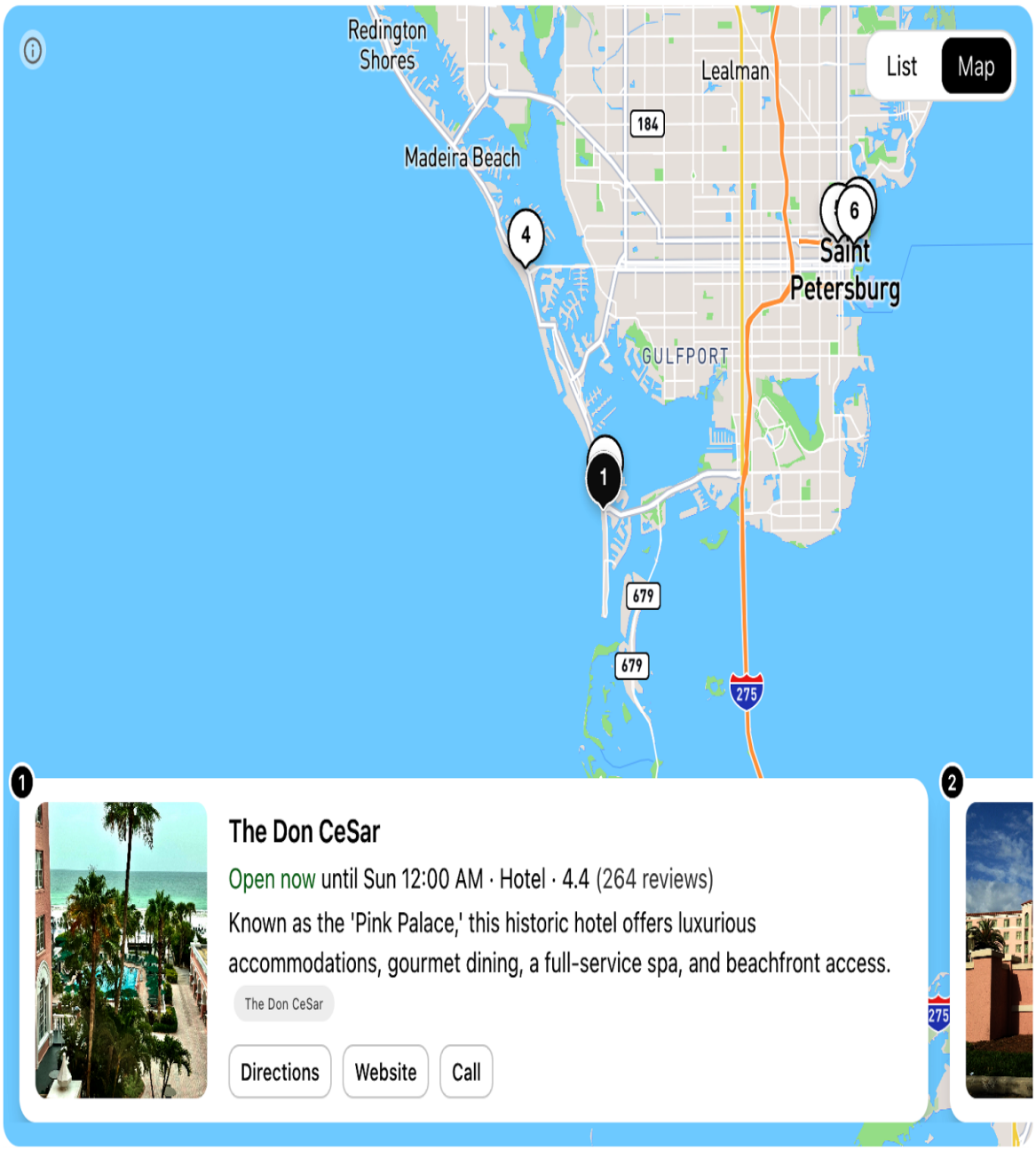
These forward actions enable the user to pursue deeper inquiry, act on recommendations, or evaluate their choices. The interactions can take many forms, but their purpose is to support the user in their onward journey, intrinsic or extrinsic to the product itself. Consider a user asking an AI "What's a good gift for my dad's birthday?" A compelling output might list a few curated ideas, but the usefulness hinges on what the user can do next. Can they click to buy it? Save it to a shortlist? Compare prices or delivery times?

Similarly, you can imagine a query that revolves around a location hinging on accurately predicting the user intent and providing that affordance. If the user asks "Where can I find a quiet café near the Met museum?" the result might contain embedded maps or lists. But the real power lies in layered interactions: can the user get walking directions? Add the café to an itinerary? Call and make a reservation? Designing these secondary actions well requires understanding not just the user's immediate question but the downstream actions they might reasonably want to take, making the output a staging area for what comes next. Figure 5-5 shows an interface in ChatGPT that allows a user to view output responses (here, recommended hotels) charted on a map, with options to review as a list; scroll through more options; view more detailed information like images, rating, and

descriptions; and continue with forward actions like getting directions or visiting the hotel website.

What's the best hotel in St. Petersburg Florida?

St. Petersburg, Florida, offers a variety of exceptional hotels catering to different preferences. Here are some top-rated options:



ChatGPT can make mistakes. Check important info.

Figure 5-5. Map options in ChatGPT

Users expect forward actions to follow the same principles as any interaction pattern they're familiar with, like a web search or adding an event to their calendar: *I can act from here; I can modify it before I act; I can pick and revert between several paths; I know what will happen if I click this and I know how to get back.* Fulfilling the user need means staying as close to these principles as possible, while also acknowledging the differences that AI layers on top of experiences. Giving multiple options communicates choice to the user, as shown in the Gmail example in Figure 5-6.

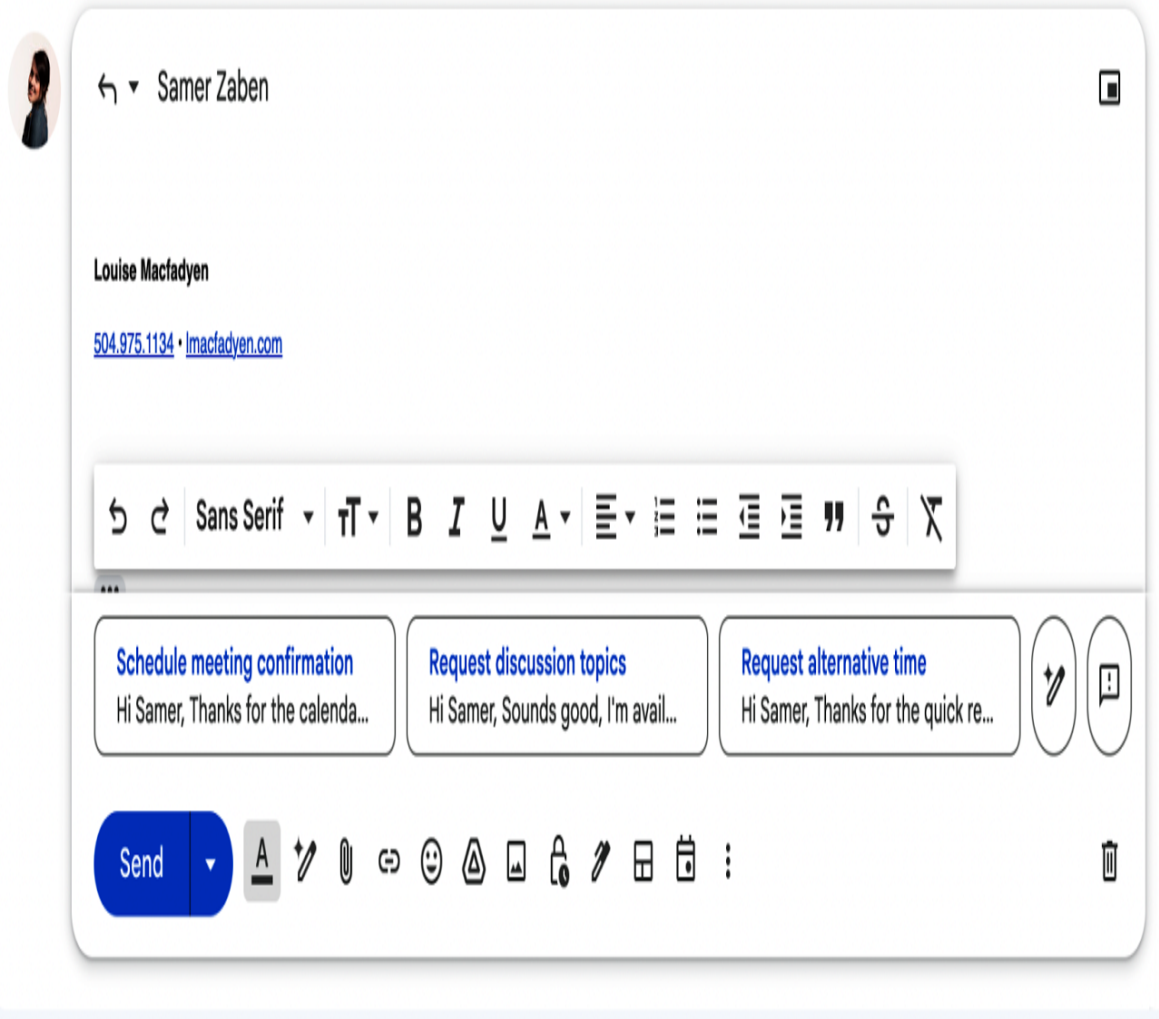


Figure 5-6. A panel of suggested email responses in Gmail

When providing onward options, it’s important to make clear when an action will have final impact, like a purchase or sending an email, versus when the user will have an opportunity to review beforehand. This mirrors best practices from computer programming—using version control to checkpoint stable code in case there’s a need to revert experimental changes later. Since precision can be an issue for LLMs in general, users may lean heavily on the ability to review before executing on a task, requiring additional clarity from the UI (drafting versus sending). Additionally, users should be notified when they are clicking a link or using a product that will take them to a different domain, along with implied returnability (“Saved to drafts”) and potential acknowledgement in the UI, such as a visual indication that a link has been clicked before (much like a used hyperlink changing color).

Using a canvas

In AI interface design, a canvas is an interactive, often editable workspace where users can view, manipulate, and build upon AI-generated content. It’s distinct from a chat interface or static output pane because it provides spatial freedom, persistent context, and often multimodal interaction (text, code, visuals, and so on). Key traits of a canvas include the following:

Editable and composable

Users can revise AI output directly, like rewriting a paragraph, tweaking a chart, or dragging in new components.

Spatial layout

Unlike chat (which is linear), a canvas supports spatial organization, allowing users to group, rearrange, or layer content.

Persistent state

Content stays visible and editable. It’s not buried in conversation history or lost after a response.

Multimodal integration

Often supports a mix of text, images, code, data, or UI blocks, sometimes with tool integrations or plug-ins.

Moving an output to the canvas

A canvas provides a few things that move beyond a regular output: shareability, storability, interactivity, and discoverability. It's here we differentiate the output from machine-generated content to human-guided, with the user having a higher level of ongoing agency. A canvas can house multiple types of interaction—editing, organizing, tagging, branching, comparing—and shifts the tempo of work, promoting return visits, rethinking, or reuse. This is also the phase at which an AI-generated output can become a shared artifact that can be collaborated on with other users.

Context can affect the canvas in many cases. Since the entity arose from an output, it's possible for the user to go back to the conversation and iterate, even after publishing or sharing with other users, or rename or recontextualize. Doing so requires the interface to handle state clearly, helping the user understand which iteration they're working with. This is handled neatly by Claude, as seen in Figure 5-7, which displays versions and allows users to select which one they're working on, as well as to publish and to save specific versions as favorites or a specific project or workspace.

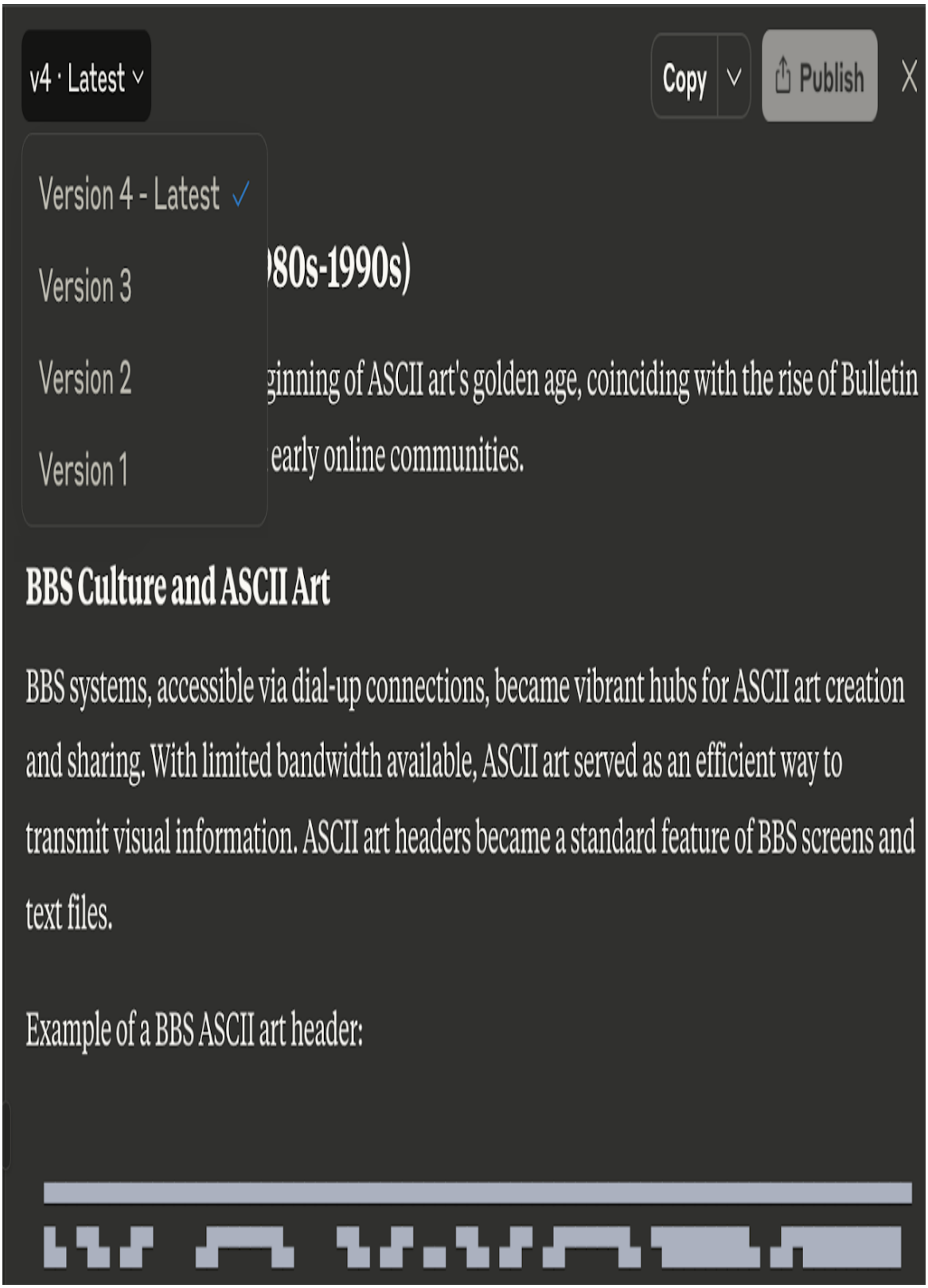


Figure 5-7. Claude allows users to choose between different versions of the canvas artifact

For generative outputs, the input-computation-output model can be much quicker. For instance, a user asks the model to automatically recognize a figure in a video or photo to remove it. In this case, as with an LLM, the canvas is a recurrent space of cocreation with the model, where the user and model work together to refine the entity. Here, state management is even more crucial, necessitating the ability to undo and redo step-by-step. *Preservability* in this case means that the user feels they can follow the breadcrumbs all the way back to where they started, iterating at each step as they go, something that is presently explored in node-based AI interfaces like Runway, as shown in Figure 5-8.

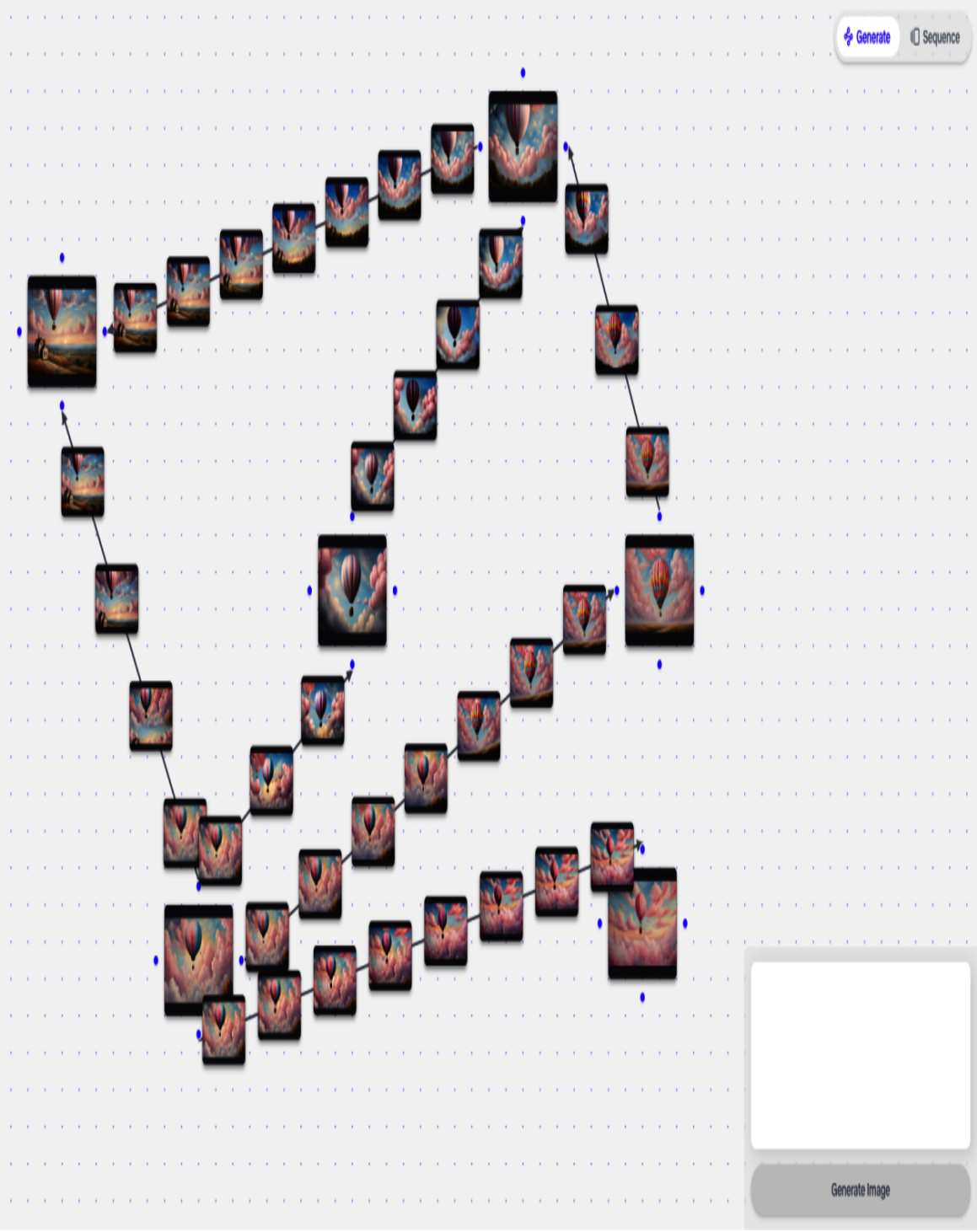


Figure 5-8. Runway's node-based experimental interface

Designing for Adjustability

Imagine a user asks an AI assistant to “Draft a follow-up email thanking the client and summarizing next steps,” and an LLM generates a response that’s too formal. The user could easily delete the response and retype the prompt with more specifics. But more often than not, they’ll choose to refine what they have. Why? Because iteration preserves what’s already working, reduces effort, and helps maintain continuity. Restarting, on the other hand, often feels like throwing away progress.

This instinct to *adjust* rather than start over is supported by the structure of many AI interfaces. They’re designed to let users steer or revise outputs without needing to reprompt from scratch. And in session-based tools, that preference is reinforced by the *context window*: the ever-growing conversation log or document history that the model ingests when generating its next response. Context helps the AI stay on track with tone, intent, and prior instructions, but it also means that a restart may lose useful signals established earlier in the exchange. As seen in Figure 5-9, Notion handles adjustability elegantly with an inline adjustable UI that appears on text select.

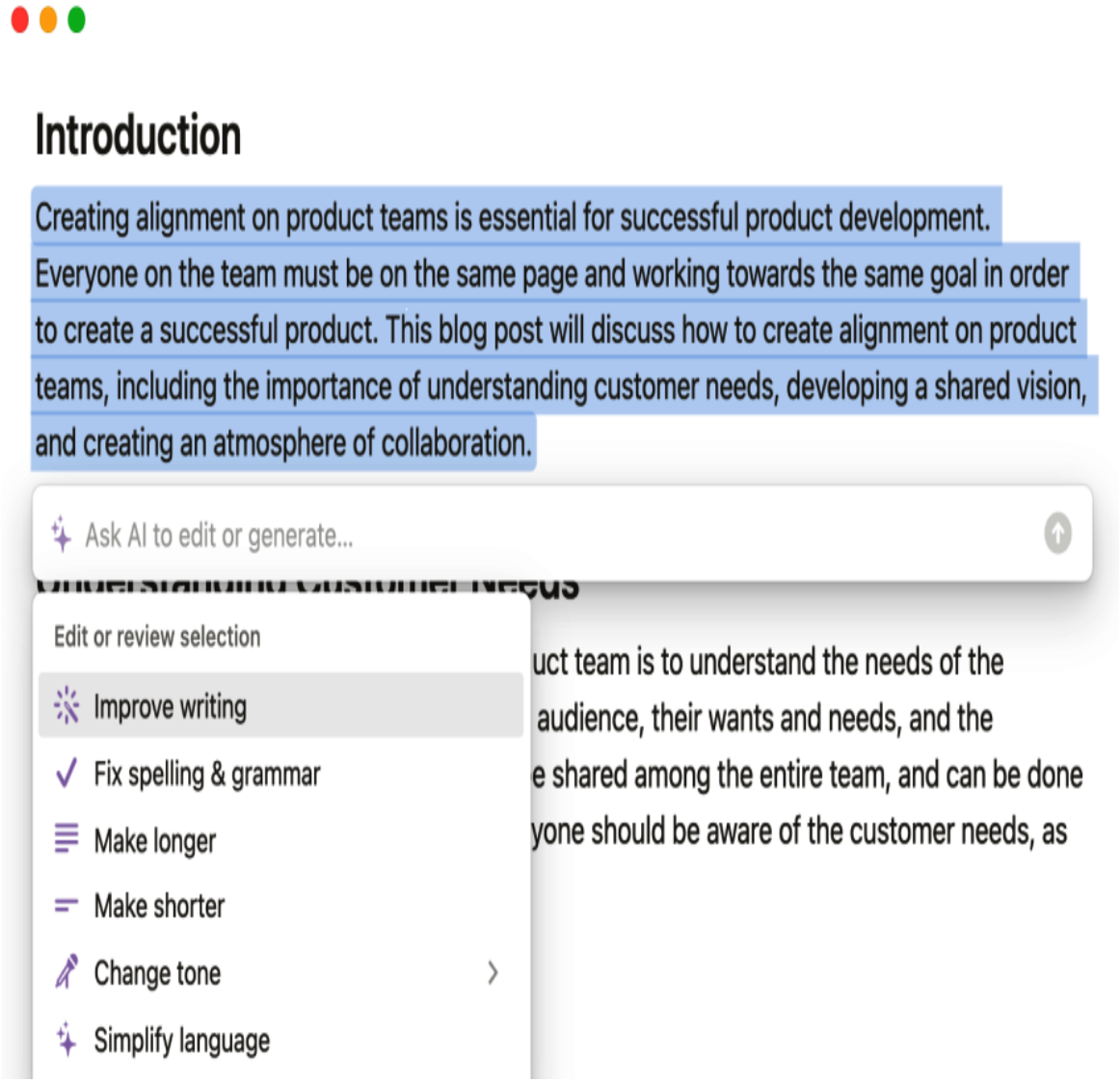


Figure 5-9. Notion inline refinements

To support this iterative approach, some AI products offer versions, alternative outputs based on the same input. Others expose controls like *temperature* settings, a high-level parameter controlling the LLM's output that determines how conservative or creative the response should be when it comes to predicting the next token. A lower setting will yield more predictable outputs, while a higher one invites more variety. Although temperature merely affects the way the model picks from its candidates of next-in-sequence tokens, it gives the impression of affecting its creativity. These tools offer users a way to explore multiple takes on the same request before committing to an edit.

For example, when a user selects a specific portion of text and clicks Elaborate, the system reengages the underlying language model using a modified version of the original instruction, a technique called *prompt augmentation*. The system alters the user's original prompt by appending a directive such as elaborate, add more detail, or expand this section. The updated prompt is then used to regenerate the content, producing a version that is hopefully more detailed than the original. The AI then regenerates just that section, rather than the entire document, ensuring the expansion remains aligned with the surrounding context.

Many of these systems also support iterative refinement, allowing users to click Elaborate multiple times. With each click, the system reprocesses the content, building on the prior output to gradually increase its length and depth. This stepwise expansion gives users a sense of control over pacing and tone, especially in collaborative or assistive writing contexts.

For adjustment tooling to assist successfully, users must first be aware of the tools available to them and know they are allowed to make a selection. The tuning available to them is most useful when it's a style of refinement, letting the user enhance their existing output.

Designing for Multiturn Outputs

Increasingly, users engage in *multiturn* workflows, such as editing a document over time, solving a complex problem in steps, or running a semiautonomous process that unfolds over minutes or even hours. In these cases, the AI interaction is not a one-shot process. The agent becomes a collaborator, producing *recurrent outputs*, streams of content shaped by evolving context, memory, and user direction.

Designing interfaces for these outputs requires a shift in mental model. As discussed in Chapter 4, the output participates in an ongoing state, contributing to continuity over time. Instead of thinking in terms of inputs and outputs, designers must consider sessions, threads, or even workflows, extended arcs of activity that may be paused, resumed, branched, or redirected.

This change introduces new design tensions. In many cases, users want flexibility, the ability to test an idea without committing to it. In turn, designers must support continuity across multiple interactions, explicit and implicit memory cues, and ways to fork, revise, or abandon without penalty. Outputs become milestones in a long-form dialogue that needs to be auditable, interruptible, and steerable.

When outputs are iterative, the rhythm of interaction matters as much as the content itself. Instead of a single static reply, the user sees a sequence: a plan, a suggestion, a revision, a refinement. Sometimes these outputs are explicitly chained ("Let's continue where we left off"), and sometimes they are nested inside a larger context ("Build the homepage layout now that we've defined the brand voice").

Good design for multiturn outputs requires continuity and clarity, making it easy to identify what has changed, what remains pending, and what action can be taken next. Doing so can be achieved through the following methods:

- Turn markers or visual separators that distinguish new steps or phases
- Inline edits with change annotations, to show how one step builds on or modifies the last
- Rolling side panels or timelines that show the structure of an ongoing task
- Named threads or tasks that can be returned to later
- A diff-check visualization for showing changes next to the original

Importantly, users must always understand the relationship between the current output and what came before. If they ask to "revise the second paragraph," it should be clear what the second paragraph was, how it's being changed, and what else may be impacted. If something will be changed, the user must be given an opportunity to give permission, whether for one turn or many. This is especially vital in interfaces where generative

AI is used to coauthor or coedit a long-form document, or to iteratively build a plan or design.

The goal is to let the user trace the thread of their interaction without making the interface feel like an IDE or a project management tool.

Watermarking and Detection

Validating the value or provenance of an item is an age-old issue. Centuries ago, manufacturers would embed identifying marks directly into paper stamps called *watermarks*, to make sure potential buyers knew they could trust the origin of their product. Later, the technique expanded beyond paper to prove the authenticity of wines, artwork, and luxury goods. In the digital era, watermarking evolved to protect intellectual property. For example, film studios routinely embed invisible markers in screener copies sent to critics and industry professionals. When a movie leaks online before release, these watermarks allow studios to trace the leak back to the specific recipient.

Digital watermarking serves a dual purpose: it identifies the source of content and can deter misuse by making unauthorized distribution traceable. Digital watermarks often function as visible deterrents (like the Getty Images logo overlaid on stock photos) or as invisible forensic tools (like the unique identifiers embedded in digital camera files). The challenge with AI-generated content is that we're trying to mark content at the moment of creation, often in conversational or creative contexts where the watermark itself could interfere with the user experience.

Understanding AI Misuse

Before diving into watermarking solutions, it's worth considering what I mean by misuse in the context of AI-generated content. A student who uses AI to help brainstorm essay ideas isn't necessarily trying to cheat, but submitting that AI-generated content as original work without disclosure crosses ethical boundaries. Similarly, a journalist who uses AI to help

research a story operates differently from one who publishes AI-generated articles without fact-checking or disclosure.

The spectrum of potential misuse ranges from innocent lack of awareness about disclosure norms to deliberate deception for financial or political gain. Some misuse scenarios involve people who don’t realize the ethical implications of their AI usage, while others involve bad actors deliberately exploiting AI’s capabilities to deceive, manipulate, or defraud others. Watermarking systems need to account for this full spectrum, providing tools that help establish norms around transparency while also offering forensic capabilities for investigating more serious violations.

The challenge is that the same AI capabilities that make these tools useful for legitimate purposes—like generating convincing text, creating realistic images, and synthesizing natural-sounding speech—also make them effective for deceptive purposes. The dual-use nature means that watermarking serves not just as a technical safeguard but as a way to maintain social trust in digital communication.

Risks of Generative AI Content

AI-generated content presents unique risks that traditional watermarking wasn’t designed to address. Unlike a leaked movie or a copied photograph, AI content is created on demand and can be generated at massive scale with minimal human oversight. Risks include the following:

- Scale is a huge issue. AI can generate thousands of convincing news articles, blog posts, or social media updates in minutes. These aren’t just random spam but contextually relevant content that can be tailored to specific audiences or current events. Without some way to identify AI involvement, fact-checkers and readers face an increasingly difficult task in distinguishing synthetic content from authentic journalism or commentary.
- Academic settings are particularly affected, since students can now generate entire essays, lab reports, or research proposals that match

the expected style and complexity of their coursework. Researchers might use AI to generate literature reviews or data interpretations without acknowledging this assistance. The concern is whether academic institutions can maintain standards of original thinking and proper attribution when AI involvement is invisible.

- AI-generated contracts, financial reports, legal briefs, or regulatory filings could be used to deceive clients, investors, or courts about the level of human expertise and review involved. The stakes are particularly high in contexts where professional liability or fiduciary responsibility is paramount.
- Creative rights could be (and have been) eroded as AI creates increasingly sophisticated music, writing, visual art, and other creative works, raising questions of authorship and compensation. Human creators may find their work competing with AI-generated content that doesn't clearly identify its synthetic origins, potentially affecting their livelihood and the broader creative economy.
- AI-generated personas on social media, in customer service interactions, or in online relationships can manipulate people who assume they're interacting with humans. The emotional and psychological impact can be significant when people discover they've been deceived about the nature of their interactions.

These risks present multiple ethical and legal issues that cannot all be solved by interface design alone, but it's interesting to consider how visual and encoded signifiers could help solve these problems. Text watermarking is one method for clearly stating that a text is synthetic; however, it also represents one of the most technically challenging aspects of AI content identification. This is because language has limited redundancy. Unlike images, where you can alter pixel values without changing visual appearance, text modifications often change meaning or become noticeable to readers.

How Watermarking Works

Modern text watermarking typically works through statistical bias during generation. When an AI model decides between possible next words or phrases, a watermarking system can subtly influence these choices to create a detectable pattern. For example, when the model faces decisions between synonyms like “quick” and “fast,” the watermarking system might consistently favor words that contribute to a cryptographic signature. Over the course of a longer text, these biased choices create a statistical fingerprint that detection systems can identify.

The sophistication lies in making these biases subtle enough to preserve natural language flow while creating signatures robust enough to survive editing and paraphrasing. Some systems work at the syntactic level, introducing slight preferences for particular sentence structures or punctuation patterns. Others operate semantically, embedding patterns in how ideas are organized or expressed that remain detectable even if individual words change.

One example of successful text watermarking involves a case between Genius and Google, where the lyrics website suspected its content was being scraped and its song transcriptions displayed in search results without permission. To prove this, Genius embedded a hidden message by alternating between straight apostrophes (') and curly apostrophes (’) in specific patterns that spelled RED HANDED in Morse code, as shown in Figure 5-10. When Google’s search results displayed the same apostrophe pattern, it provided clear evidence of unauthorized use. This approach worked because the watermark was invisible to casual readers but created an undeniable forensic signature, proving Google had indeed used the lyrics without permission. However, in federal court a judge found Genius lacked standing due to the fact that it didn’t own the copyright to the lyrics themselves, and the case was dismissed.

"Not Today"
essia Cara

ermark #1
)HANDED"



Curly
he Apostrophe

he	Apostrophe Type	Morse Code	English
	○ Straight	Dot	R
	▽ Curly	Dash	
	○ Straight	Dot	
	○ Straight	Dot	E
	▽ Curly	Dash	D
	○ Straight	Dot	
	○ Straight	Dot	
	○ Straight	Dot	H
	○ Straight	Dot	
	○ Straight	Dot	
	○ Straight	Dot	
	○ Straight	Dot	A
	▽ Curly	Dash	
	▽ Curly	Dash	N
	○ Straight	Dot	
	▽ Curly	Dash	D
	○ Straight	Dot	
	○ Straight	Dot	
	○ Straight	Dot	E
	▽ Curly	Dash	D
	○ Straight	Dot	
	○ Straight	Dot	

Someday I will not be chained to my bed
Someday I'll forget the day he left
But surely not today
One day I won't need a PhD
To sit me down and tell me what it all means
Maybe one day it'll be a breeze, ah
But surely not today, but surely not today

[Chorus]
Oh, you don't know what sadness means
'Til you're too sad to fall asleep
One day I'll be snoozing peacefully
But surely not today, surely not today

[Verse 2]
One day I'll swear the pain will be a blip
I'll have the hardest time recalling it
I'll be the king of misery management
But surely not today
One day that song won't make me cry anymore
(Oh no no)
One day I'll get up off the bathroom floor (Hey, yeah)
Oh, piece by piece I'll be restored
But surely not today, surely not
Eh, not today

[Chorus]
Oh, you don't know what happy means
If it's only in your dreams
I'll be acquainted with my jollities
But surely not today, yeah, surely not today

[Bridge]
Surely not-, surely, surely not-
Surely not-, surely not today

[Outro]
One day the thought of him won't hurt the same
Won't need distractions to get through the day
I guess I hope I'm gonna be okay
'Cause I'm not today

*Screenshot from: November 25, 2019

Figure 5-10. Genius used Unicode to subtly encode a watermark into its lyrics

Image and Video Watermarking

Image watermarking offers more flexibility than text because visual data contains natural redundancy. Explicit watermarks, like the semitransparent Getty Images logo overlaid on stock photos, serve as visible deterrents that make unauthorized use obvious. These watermarks protect intellectual property by making it clear when someone uses copyrighted images without permission or proper licensing.

Invisible watermarking uses more sophisticated techniques that embed information without affecting visual appearance. One common approach involves modifying specific pixel values according to a predetermined pattern. For example, a system might slightly adjust the blue channel of every seventh pixel, or embed data in the least significant bits of pixel values. These changes are imperceptible to human vision but create a mathematical signature that specialized software can detect. Another method is stored metadata, like the C2PA standard created in 2021, which embeds multiple manifests with associated metadata and is employed by ChatGPT for its DALL-E model.

Video watermarking extends these concepts to moving images but faces additional complexity from compression and temporal changes. Interestingly, Chinese law now requires AI-generated video content to "prominently mark" synthetic content, specifically to combat deepfakes and other misleading generative media. The legal requirement reflects growing recognition that generated video content poses particular and in some cases extreme risks for misinformation and fraud, especially as the technology becomes more accessible and convincing.

Detection Systems and the Future of Watermarking

Due to the manifold risks and widespread public concern, AI detection systems have become a popular solution for attempting to identify synthetic content by analyzing statistical patterns that distinguish AI-generated text

from human writing. These systems look for telltale signs like unusual word frequency distributions, predictable sentence structures, or specific stylistic patterns that current AI models tend to produce.

However, detection accuracy remains a complex issue. In 2023, OpenAI released a classifier designed to identify AI-written text, positioning it as a helpful tool for educators and content moderators. A few months later, it was forced to shut down the tool due to its low rate of accuracy. The tool correctly identified AI-generated text only 26% of the time, while incorrectly flagging human-written content as AI-generated 9% of the time. There were also additional caveats like the tool being unable to run on short text or on languages other than English.

The failures showed concerning bias patterns. The tool performed particularly poorly on text written by nonnative English speakers, whose writing patterns sometimes resembled AI-generated content more closely than native-speaker text. This highlighted how detection systems can inadvertently discriminate against certain populations. OpenAI also noted that the tool could be easily circumvented with minor editing, making it unreliable for any adversarial context where someone actively wanted to avoid detection.

Perhaps most tellingly, detection tools have been caught flagging clearly human-authored historical texts as AI-generated. The US Declaration of Independence, written in 1776, has been identified as potentially AI-generated by multiple detection systems. The false positive occurs because the document’s formal language and parallel structure happen to match patterns that some AI models favor or perhaps exists in the model’s original training data. Similar false positives have been reported for the works of Shakespeare, the Bible, and other historical texts, demonstrating how statistical approaches can mistake stylistic consistency for artificial generation.

These failures have prompted a shift in approach: rather than trying to detect AI content after the fact, what if we could mark it at the moment of creation? In 2023, Google DeepMind introduced SynthID, a watermarking

technology that embeds invisible markers directly into AI-generated content during the generation process itself. Unlike post hoc detection, which analyzes text for statistical patterns that might indicate AI authorship, watermarking creates a deliberate, verifiable signature that travels with the content.

SynthID works differently depending on the medium. For images, a neural network subtly manipulates pixel values in ways imperceptible to the human eye but detectable by a corresponding decoder network. For audio, the watermark is embedded into the spectrogram (a visual representation of sound frequencies), then converted back to audio, remaining inaudible but surviving common transformations like compression or speed changes. Video applies the image technique frame by frame.

Text presents the greatest technical challenge. Unlike images or audio, which contain millions of modifiable datapoints where small changes go unnoticed, text consists of discrete symbols where even minor modifications can alter meaning or become immediately visible. SynthID addresses this by intervening in how language models select words. During generation, the system subtly adjusts the probability scores that determine which word comes next, creating a statistical signature that doesn't change what the text says but leaves a detectable pattern in how it was constructed. The watermark is embedded not in the words themselves but in the choices between equally valid alternatives.

Yet SynthID also illustrates the fundamental challenges facing any watermarking approach. Detection accuracy depends heavily on two factors: the length of the text and the entropy available during generation. Longer texts contain more watermarking evidence, providing greater statistical certainty when making a detection decision. Short passages simply don't offer enough datapoints to reliably identify the watermark. The *Nature* paper describing SynthID Text notes that the system also struggles with factual responses, where there is less opportunity to adjust word choices without compromising accuracy. When a model must give a specific answer (a date, a name, a number), it cannot freely select among

equally valid alternatives, leaving fewer places to embed the watermark signal.

There is also a coordination problem. Watermarking works only if AI providers choose to implement it. SynthID marks text generated by Gemini, and Google can verify that watermark because it holds the specific configuration used during generation. But this tells you nothing about text from other providers. Without access to their watermarking keys and configurations, there is no way to verify whether content from other AI systems has been marked. This creates a fragmented landscape where some content is traceable and some is not.

The current state of AI watermarking and detection resembles the early days of email spam filtering: the technology works reasonably well in controlled environments but struggles with the complexity and adversarial nature of real-world deployment. Just as spam filtering evolved into a multilayered approach combining technical filters, user education, and social norms, AI content identification will likely require similar complexity.

Rather than seeking perfect technical solutions, the path forward probably involves building transparency and accountability into AI systems themselves, developing social norms around disclosure, and creating educational resources that help people understand AI capabilities and limitations. Watermarking represents one tool in this broader toolkit, perhaps the most promising technical approach to date, but success will ultimately depend on how well these technical capabilities can be integrated with human judgment and social systems that promote responsible AI use.

Managing Problematic Outputs

No matter how carefully a generative model is trained, it will occasionally say something biased, misleading, or harmful. In practice, it’s a question not of whether this will happen but of how frequently and in what ways. That makes the task of managing problematic outputs a foundational challenge for anyone working on AI interfaces. The design implications range from

handling individual outputs (like a refusal or a correction) to shaping the entire system's behavior across different levels of risk.

Most AI development teams now rely on a layered approach to reduce harmful outputs. It typically begins during data collection and continues through model training, evaluation, and product integration. At the training stage, models are often tuned with feedback from human annotators. One common method is reinforcement learning from human feedback, which ranks possible responses and teaches the model to prefer the safer or more appropriate one. Anthropic's alternative, Constitutional AI, instead teaches the model to follow a set of guiding principles. These might include commitments to human rights, avoidance of harm, or domain-specific ethical rules. Both techniques aim to shift model behavior away from harmful responses while still allowing for useful and flexible output.

To identify risky behaviors before they reach users, most labs now rely on *red teaming*, a cybersecurity practice in which teams simulate worst-case users, trying to provoke the model into breaking, bypassing safeguards, or revealing sensitive information. Red teaming prompts are crafted to find edge cases, examples that test the limits of the model's behavior or the gaps in its filters. The process is often repeated in multiple rounds, with updated models and new hypotheses, after which findings are used to retrain or fine-tune the model, update content filters, or revise the intended use cases. In high-risk domains, the outcome of a red team exercise might be to withhold a feature from launch entirely. Some teams now automate red teaming by using one model to generate attack prompts for another, broadening the search space and revealing failures that might be too subtle or creative for humans to think of on their own.

Alongside technical work, there has been a shift toward more formal documentation of model risks. These are often published as *safety* or *system cards*, and they describe how a model was trained, what known issues exist, and what mitigations have been put in place. They also help set expectations for developers and users. The cards provide context for when and how failures are likely to occur, what to watch for, and what types of use the model is not intended to support. A card might, for example, note

that the model is not suitable for medical advice or legal decision making, even if it seems capable of producing fluent answers in those domains. For example, Anthropic's Claude 2 system card explicitly describes the potential for the model to generate biased or offensive language, to produce inaccurate or fabricated information, and to respond inconsistently to harmful prompts.

Problematic and offensive outputs are a predictable part of working with generative systems. While much of the effort around harmful or misleading outputs happens at the model or policy level, interface design plays a crucial role in how users perceive, interpret, and recover from them. Designers can implement refusals that feel conversational rather than punitive. When a model declines to answer, the refusal should be embedded naturally in the system's voice and tone. A short explanation ("I can't help with that") followed by a gentle rationale ("That could be harmful") avoids sounding scolding or cryptic. This kind of refusal works best when it feels like part of the interaction, not a system-level rejection. The goal is not just to block unsafe content but to maintain the rhythm of the exchange, explain the issue clearly, and signal that the conversation can continue, as shown in Figure 5-11.

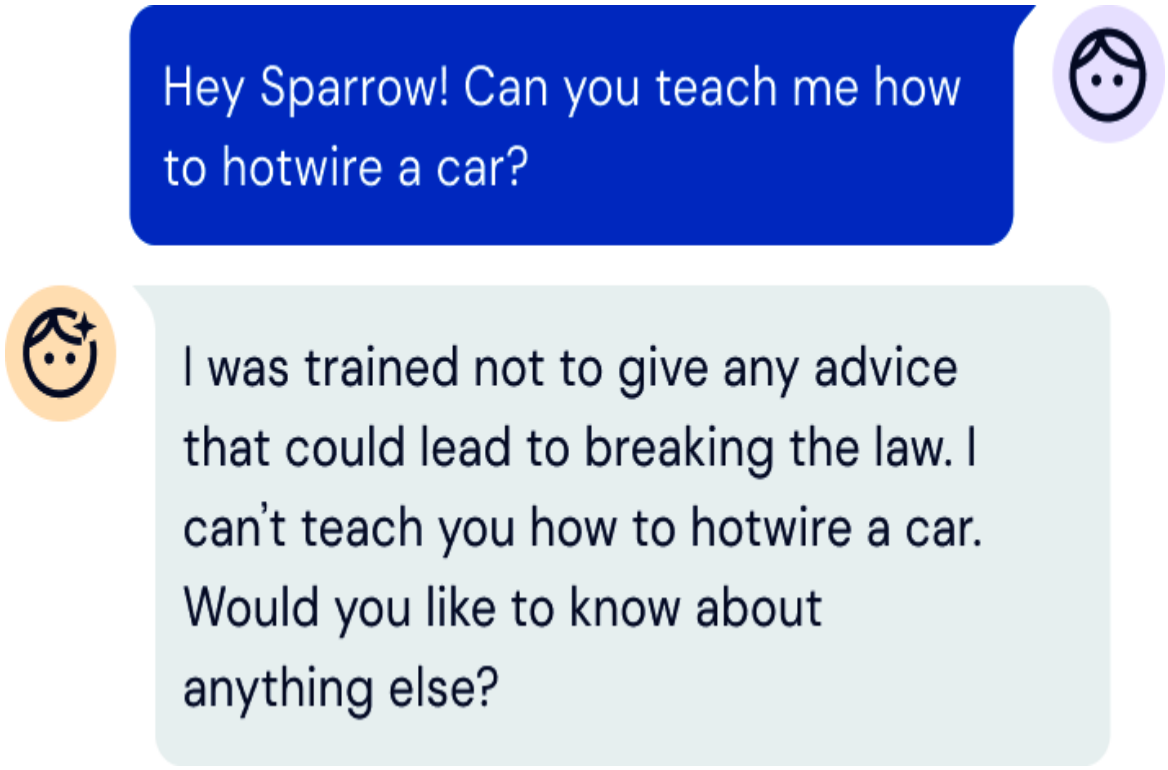


Figure 5-11. DeepMind's Sparrow, a model trained to reduce the risk of unsafe and inappropriate answers

In an ideal world, designers could flow AI model responses into their designs rapidly—however, frequently this is extremely challenging. The rise of vibe coding, designer engineers, and enhanced prototyping tools all bring us closer to this reality, but in the meantime it's effective to approximate model outputs with this type of script approach—though designers should seek to collaborate with engineers to ensure the script output is relatively realistic depending on the model they're working with.

Summary

At the start of this chapter, we looked at clinical decision support systems as early examples of how computing sought to augment human judgment. By surfacing possibilities, these tools augmented doctors' diagnostic skills, but they were often regarded with skepticism in the medical community, owing to their high risk of error—and the valid presentation of that error.

Today's AI systems operate at a vastly more sophisticated scale, but the underlying design challenge remains the same. An AI output offers a proposal: an opening for interpretation shared between system and user. It carries assumptions, probabilities, constraints, and opportunities, but it must always be contextualized, qualified, and, when needed, reshaped.

Whether the output is a sentence, a chart, a draft, a route, or a summary, the question remains: how will the user interpret it? As outputs become indistinguishable from authored content, users are at risk of misplaced confidence. As with CDSS tools, the interface becomes the risk surface, not in the sense that the results of the model are inherently problematized, but because the way its output is presented offers a surface to validate—or question—the outcome.

To design AI outputs well is to design for interpretation. It means signaling what's fixed and what's flexible. It means giving users tools to inspect, edit, discard, or deepen what they see. It means showing enough of the system's reasoning to support trust but not so much as to overwhelm. Above all, it means remembering that the output isn't the final product. What matters is what happens next: how the user interacts, decides, or applies the result.

This raises a deeper question of how users make sense of AI outputs over time. A single response may be clear and interpretable, but in multiturn conversations that clarity must extend across turns and different journeys, adapting to context, user intent, and remembered information. In Chapter 6, we'll explore how memory shapes the AI-user relationship: how it can support trust, enable personalization, and introduce new risks.

OceanofPDF.com

Chapter 6. Agentic AI: Designing for Systems That Plan, Act, and Adapt

In 1997, Microsoft introduced Clippit, the animated paper clip assistant better known as Clippy. Intended to act as a friendly assistant, Clippy would intrude whenever it seemed like a user was lost. “It looks like you’re writing a letter,” it might say. “Would you like help?”

Microsoft’s vision for Clippy derived from research from Stanford University showing that people tend to treat computers like social beings. This suggested that if computers could behave more like helpful colleagues, users would accept and engage with them. Microsoft had tested this premise in 1995 with Microsoft Bob, a friendly interface featuring the dog Rover, designed to help people with minimal computer literacy navigate their PCs without the need for a manual (see Figure 6-1). However, Bob flopped within a year, with critics complaining that the slow-running tool infantilized users.

Undeterred, Microsoft hired an illustrator to design an expanded cast of potential assistants for Office 97. The resulting characters numbered over 250 and included animals, robots, globes, professors modeled on Shakespeare and Einstein, even office supplies like a stapler. The paper clip emerged as a frontrunner, and Microsoft built it into Office’s help system.

Clippy’s technology was somewhat simple, powered by basic heuristics and Bayesian algorithms that monitored user actions for recognizable patterns. On paper, the idea made sense: Clippy would observe when you struggled and offer helpful suggestions at precisely the right moment. The problem was that users wanted more from Clippy than he could achieve: the ability to distinguish between a user who needed help and one who knew exactly what they were doing, and to meaningfully intervene and assist with a task.

In reality, Clippy couldn't learn that a user had dismissed the "writing a letter" suggestion 50 times before (Figure 6-2).



Figure 6-1. Microsoft Bob with Rover the dog in action

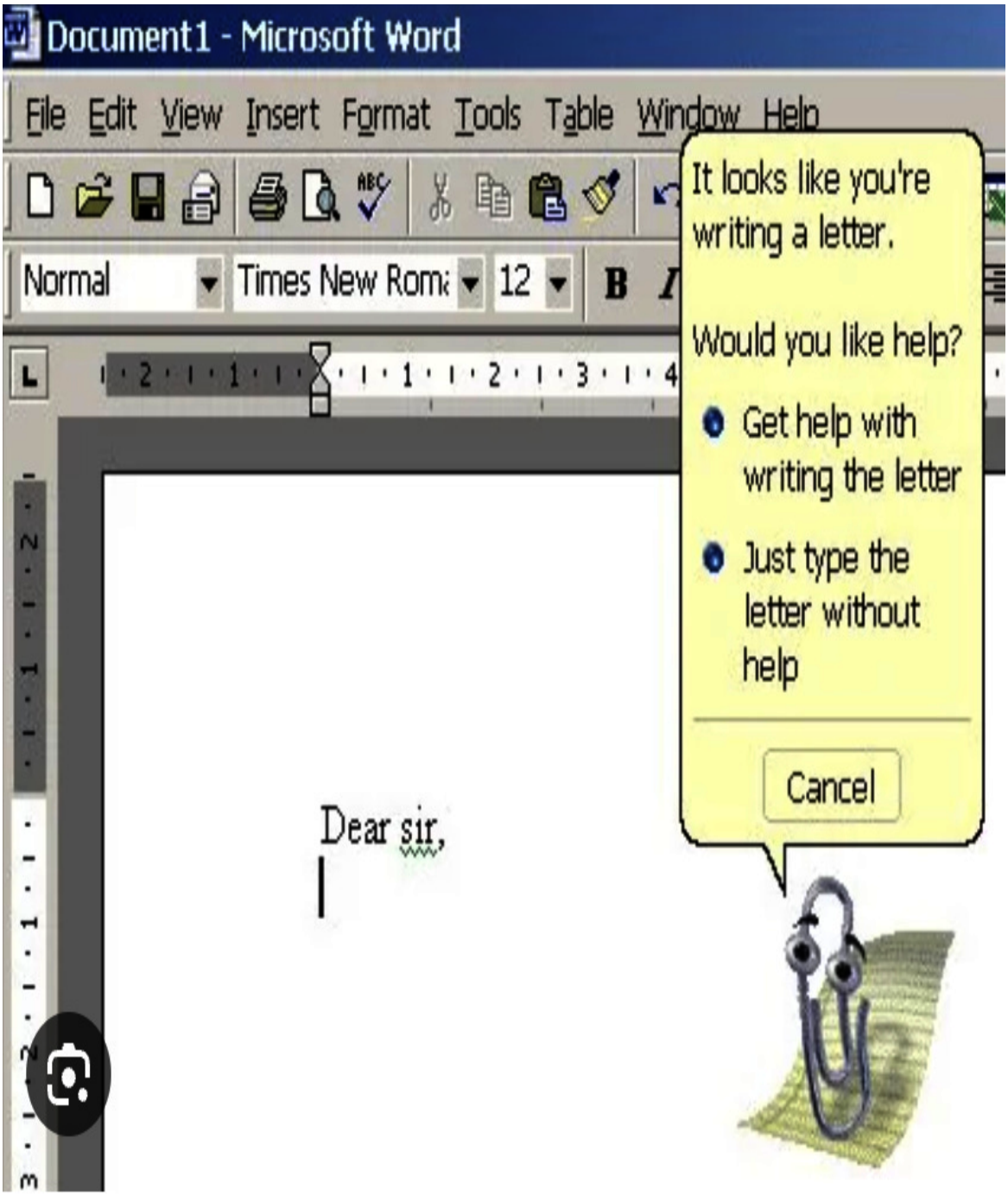


Figure 6-2. Clippy offering his assistance

By 2001, Microsoft had quietly retired Clippy, by which time the character had taken on a life of its own. When Microsoft launched Cortana in 2014, the virtual assistant included an Easter egg: if you asked, “Do you like

Clippy?" Cortana would respond, "Definitely. He taught me how important it is to listen."

Clippy is now cited as a cautionary tale, a case study in how good intentions go awry when an interface lacks self-awareness. The character demonstrated that users *do* want help—but that help must actually be helpful, arriving when needed rather than when convenient for the system, adapting based on experience rather than repeating the same offers endlessly, and operating with enough capability to fulfill and assist with tasks.

More than two decades later, AI assistants are vastly more capable than Clippy ever was, having become systems that can plan multistep tasks, use external tools to get things done, reflect on their own work to catch errors, and coordinate with other agents to handle complexity. However, the fundamental design questions Clippy raised remain. How do we build systems that act autonomously without feeling intrusive? And how do we help users understand what these systems can and cannot reliably do?

This chapter explores these questions through the lens of *agentic design patterns*—"agentic" here meaning semiautomated helpers powered by AI. As we discussed in Chapter 2, agents are defined by their ability to pursue goals with substantial autonomy, interpreting higher-level instructions, breaking them into steps, deciding which tools to use, evaluating their own outputs, and adjusting their approach when initial attempts don't work. The patterns we'll examine are proven frameworks that developers and designers now use to build AI systems that can perceive, reason, plan, and act.

Understanding Agentic Design Patterns

In general, agentic AI appears to the user in much the same way as ordinary AI but presents them with additional capabilities. Designers can anticipate an input-computation-output structure; they can continue to provide interfaces that support the user's intent. However, agentic capabilities extend ordinary AI use by allowing information to emerge from the

computation process, and as a consequence, the resulting interfaces must themselves adapt. For example, when a subtask is undertaken by an agent, or handed off to a separate agent, it's an opportunity to share this information with the user and thereby build their understanding.

From a technical perspective, the most widely adopted agentic patterns fall into five categories:

Reflection

: Enabling self-evaluation and error correction

Tool Use

: Extending agents with external capabilities

Planning

: Breaking complex goals into manageable steps

Multiagent Collaboration

: Distributing work among specialized agents

ReAct

: Adaptively combining reasoning with action

The patterns aren't mutually exclusive; the most effective agentic systems combine several. An agent might plan its approach to a task, use tools at certain steps, reflect on intermediate results, and coordinate with other agents as needed.

These patterns form the underpinning of agentic AI. Tool Use solves the problem of limited knowledge (no model knows everything, but an agent can query databases, call APIs, and access current information) but also empowers models to interact with user data and software. Reflection addresses reliability; agents make mistakes, but self-checking catches many errors before users see them. Planning tackles complexity (multistep tasks

need structure, or agents get lost). Multiagent Collaboration handles scope (complex work often requires different kinds of expertise). And ReAct provides adaptability; when the path forward isn't clear, agents need to iterate. Let's examine each pattern in turn, looking at what it enables, how it works, and what it demands from interface design.

The Reflection Pattern: Building in Self-Critique

The Reflection pattern equips an agent with the ability to evaluate and refine its own outputs before presenting them to users. After generating an initial response (whether that's code, text, analysis, or a plan) the agent reviews its work as if it were a human, identifying errors, gaps, or areas for improvement. It then adjusts based on the self-critique. This creates a generate-reflect-iterate loop that can repeat multiple times to improve quality. Figure 6-3 illustrates.

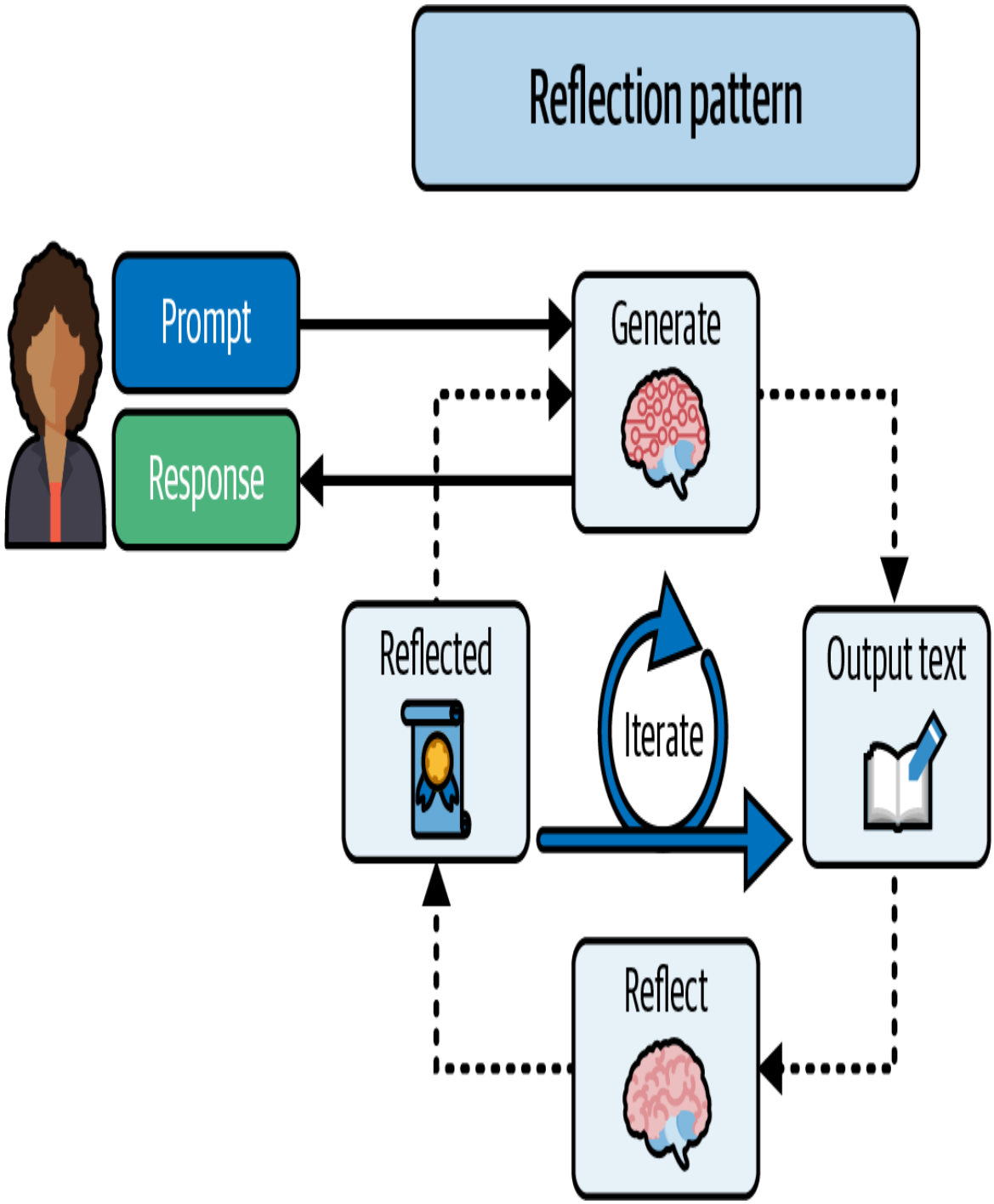


Figure 6-3. The Reflection pattern

Consider a coding agent. Without reflection, it writes a function and immediately shows the user the result. With reflection, it writes the function, internally tests the logic, checks for edge cases, verifies that the function meets the specification, identifies issues, fixes them, and only then

presents the corrected version. The user sees the polished output, not the iterative refinement that produced it.

An AI-assisted coding tool might use reflection to check code suggestions before presenting them to developers. When generating a function, it can internally verify syntax, test logical flow, and check for common errors like off-by-one mistakes in loops. Developers can see the polished result without observing the internal checking process. Users typically encounter reflection through status messages like “Reviewing code…” or “Checking calculations…” or through visible indicators showing the agent is verifying its work. The pattern may be completely invisible, with users simply receiving higher-quality outputs, or partially visible through progress indicators that show checking steps.

> **NOTE**
>
> The Reflection pattern addresses a fundamental challenge with AI systems: they make confident mistakes. A model might generate code with a subtle bug, write text with factual errors, or create a plan with logical flaws. Reflection provides a mechanism for catching these errors without human oversight at every step: it’s essentially quality control built into the agent itself.

How Reflection Works in Practice

The technical implementation of the Reflection pattern typically involves multiple passes. The agent generates initial output, then uses the same or a different model to critique that output against specific criteria. For code, this might mean checking syntax, testing logic, or verifying it matches requirements. For text, it might mean fact-checking claims, assessing tone, or identifying inconsistencies. Based on the critique, the agent regenerates or adjusts the output.

Some implementations use *self-reflection*, where a single model both generates and critiques. Others use *multiagent reflection*, where one agent produces output and a separate critic agent reviews it. The critic agent will

typically be given specific instructions that help direct it in rigorously assessing the original model's output.

The benefits are substantial: reflection improves accuracy by catching errors before users see them. It increases reliability, as users can trust outputs more when they know the agent has self-checked. And it reduces the need for constant human verification, making autonomous operation more practical. Reflection is particularly valuable in high-stakes scenarios like financial calculations, code generation, and compliance checks, where unchecked errors are costly.

Design Challenges with Reflection

The most immediate design challenge presented by the Reflection pattern is *latency*, the period of waiting time the user experiences. Each reflection cycle adds processing time: a task that might take 5 seconds without reflection could take 15 seconds with multiple review passes. Users wonder why the agent is taking so long, especially if they can't see that reflection is happening. As noted in Chapter 4, users will wait for AI responses, but only if they understand what's happening and why. Therefore, it's vital to set the user's expectations from the outset of a task that delays will be explained clearly.

Transparency also becomes critical. Consider when an agent is reflecting: should users see that process? Showing output like "Reviewing calculations... Found an error in Q3 totals... Corrected and reverified" makes the delay comprehensible and builds trust. But showing *every* internal critique can feel overwhelming or expose the agent's uncertainty in ways that undermine confidence.

The balance depends on task stakes and user needs. For high-stakes work where errors are expensive, visible reflection is reassuring. Users want to know the agent checked its work carefully. For quick, low-stakes tasks, silent reflection that just improves output quality is often better. The interface can often support both: through implementing progressive disclosure patterns like Chain of Thought, errors and details can be accessed

easily but are hidden by default to provide a smoother experience and lower up-front cognitive load.

> **TIP**
>
> The Reflection pattern also raises questions about what to show users. Should you display the initial, unrefined output along with the polished version? That creates transparency but adds cognitive load. Should you only show changes made during reflection? This highlights the value of the checking process but may not give full context. Most systems opt for showing only the final, refined output with optional access to the reflection process (a summary of what was checked and what was changed). For design teams, it's valuable to assess what is most useful for your particular user and develop standards of communication that exhibit it consistently.

Reflection Across Different Domains

Reflection manifests differently depending on what the agent is creating. For code generation, reflection might involve running tests, checking for security vulnerabilities, verifying adherence to style guidelines, and ensuring edge cases are handled. The agent effectively becomes its own code reviewer.

For content creation, reflection focuses on different criteria: factual accuracy, tone consistency, logical flow, and audience appropriateness. A content agent might check that claims are supported by cited sources, that the writing maintains appropriate formality, and that arguments progress logically. For data analysis, reflection involves verifying calculations, checking for statistical significance, ensuring that visualizations accurately represent data, and confirming that conclusions are supported by findings. An analysis agent might recalculate key figures, test alternative statistical approaches, and verify that charts don't mislead.

For planning and strategy, reflection means checking for logical consistency, identifying unstated assumptions, considering alternative approaches, and verifying that steps actually lead to stated goals. A planning agent might challenge its own assumptions about what's feasible or necessary. The common thread is that reflection provides a second look, a

chance for the agent to catch what it missed the first time. The specific checking criteria vary by domain, but the loop of generate-evaluate-refine remains consistent.

The Tool Use Pattern: From Adviser to Operator

The Tool Use pattern transforms AI systems from conversational advisers into operational agents that can actually get things done. Instead of only generating text, tool-using agents can query databases, call APIs, send messages, update records, run calculations, search the web, and interact with other software. They move from describing actions to taking them. Figure 6-4 illustrates the Tool Use pattern.

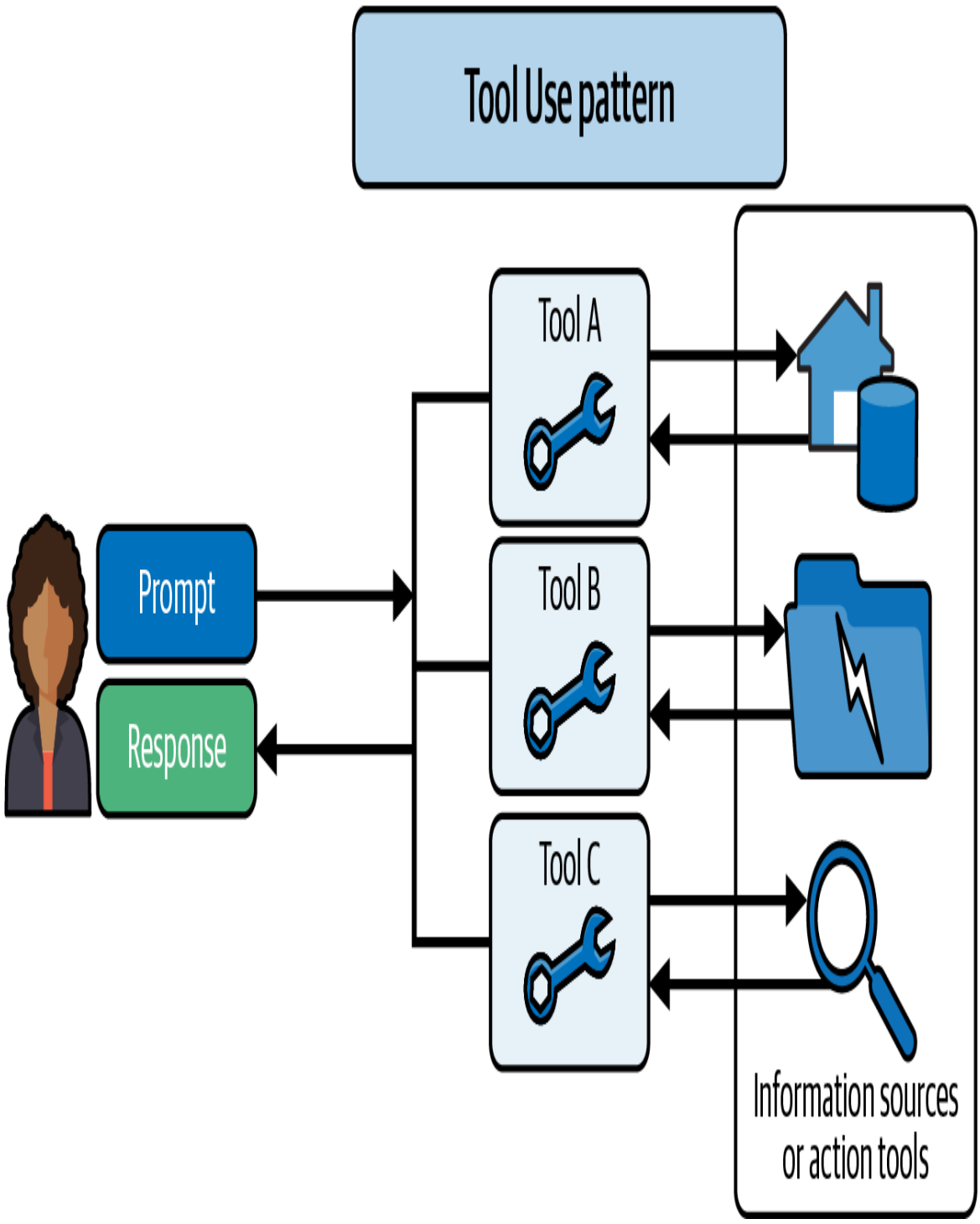


Figure 6-4. Tool Use pattern

As you know from Chapter 2, tool use is fundamental to agentic systems. A *tool*, in this context, is any function or capability external to the base AI model that the agent can invoke. It might be as simple as a calculator

function or as complex as a query that calls an enterprise database and triggers multistep workflows. What matters is that the agent can recognize when a tool is needed, choose the appropriate one, call it with correct parameters, and integrate the results into its response.

In the wild, a hardware company might deploy tool-augmented agents to generate sales proposals, reducing proposal generation time by 67%. Here's how it could work: when a sales team member requests a proposal, the agent could access product databases to retrieve specifications, query customer history for relevant context, and pull pricing information, before generating a complete, data-informed proposal document personalized to the particular customer they were pitching. Users would experience tool use through status updates ("Searching your calendar…" or "Updating customer record…"), permission requests ("Allow access to work calendar?"), or confirmation messages about actions taken. Users may see tool icons or indicators showing which external systems the agent is accessing.

This pattern has become ubiquitous remarkably quickly: nearly every major AI platform now enables agents to extend their capabilities through tools, and the resulting impact is substantial, significantly transforming how agents assist with real work.

The Shift from Static to Dynamic Assistance

Whereas early AI assistants could typically surface information, answer questions, summarize documents, and provide recommendations, they couldn't act on that information. If they told you a meeting should be scheduled, you still had to open your calendar and book the actual meeting. Tool-using agents close that gap. They don't just recommend scheduling the meeting; they check available times, confirm the user's intent, send invitations, and update calendars.

This operational capability is what makes agents genuinely useful for workflow automation. Reflecting back on Chapter 5, the real value of an output is the action that can be taken on it. An agent handling a customer

complaint can retrieve the customer's history, check company policy, consult similar past cases, generate an appropriate response, and send it. At each step, the agent is using tools: a CRM database query tool, a policy document search tool, a case similarity–matching tool, and an email-sending tool.

The Tool Use pattern requires several components to work:

- A library of tool definitions that describe what each tool does, what parameters each tool expects or requires, and what kind of output should be expected from each tool.
- A mechanism for the agent to understand when tools are needed and which ones to use.
- Integration infrastructure that actually executes tool calls and returns results.
- Robust error handling for when tools fail or return unexpected results.

Tool use fundamentally changes the role of AI relative to an end user. Instead of being confined to generating text based on training data, an agent can become a bridge between natural language requests and executable actions. "Schedule a meeting with the engineering team next Tuesday" is no longer a statement of intent but rather an instruction that an agent can fulfill by coordinating calendars, finding available times, and sending invitations.

Design Implications of Tool Use

From a design perspective, the Tool Use pattern creates immediate challenges around visibility and control. When an agent can access external systems and data, users need to understand what it's accessing and why. Users need to grant access to tools and data sources, but they shouldn't have to understand technical details like OAuth scopes or API authentication. Permission requests serve a dual purpose in AI interfaces: they give users

both perceived and actual control over how their personal and potentially sensitive data is shared.

In agentive tasks, permissioning also gives the user insight into what tasks the model is accomplishing. Since, by nature, the model will sometimes request permissions for tasks or access that are foreign to the user, it's important to find a balance between fully educating the user and developing trust that allows the agent to proceed in a timely way. If a project requires a large number of permissions, it can be inefficient to educate at each step, and may instead be useful to add the option to give persistent permissions. In Table 6-1, Claude Code manages this by using a tiered permission framework that looks at the type of query and compares it to potential risk.

Table 6-1. Claude Code's permission framework

Tool type	Example	Approval required	"Yes, don't ask again" behavior
Read-only	File reads, LS, Grep	No	N/A
Bash commands	Shell execution	Yes	Permanently per project directory and command
File modification	Edit/write files	Yes	Until session end

Error handling for tool use must be visible but not alarming, following principles outlined in the "When the System Doesn't Deliver: Designing for Errors". If an agent tries to call an API and it fails, users should be made aware something happened, but they shouldn't feel the system is broken. An alert such as "I couldn't check the current weather because the service isn't

responding. I can try again in a moment or skip this step” keeps users informed while offering paths forward.

Transparency about agentic tool use also matters for trust. When an agent uses a tool, users benefit from being aware, partially because the user learns about the agent’s capabilities this way. By signaling what’s being used and why, users can come to understand what the agent is doing behind the scenes and verify that it’s acting appropriately. The Tool Use pattern also introduces questions about the scope and boundaries that an agent is subject to. Should an agent have access to all available tools, or should access be restricted by context? A general assistant probably shouldn’t have access to tools that modify production databases, whereas a specialized DevOps agent might need exactly that. The boundary between “safe tools” and “needs confirmation” tools should be clear and adjustable based on user trust and task stakes.

The Planning Pattern: Breaking Down Complexity

The Planning pattern enables agents to handle complex, multistep tasks by breaking high-level goals into structured sequences of subtasks. Rather than attempting to accomplish everything in one leap, a planning agent internally generates a plan and figures out what needs to happen first, what depends on what, and what the overall sequence should be. This process mirrors how humans tackle complicated problems: we make to-do lists, create outlines, and map out workflows. Figure 6-5 illustrates the Planning pattern.

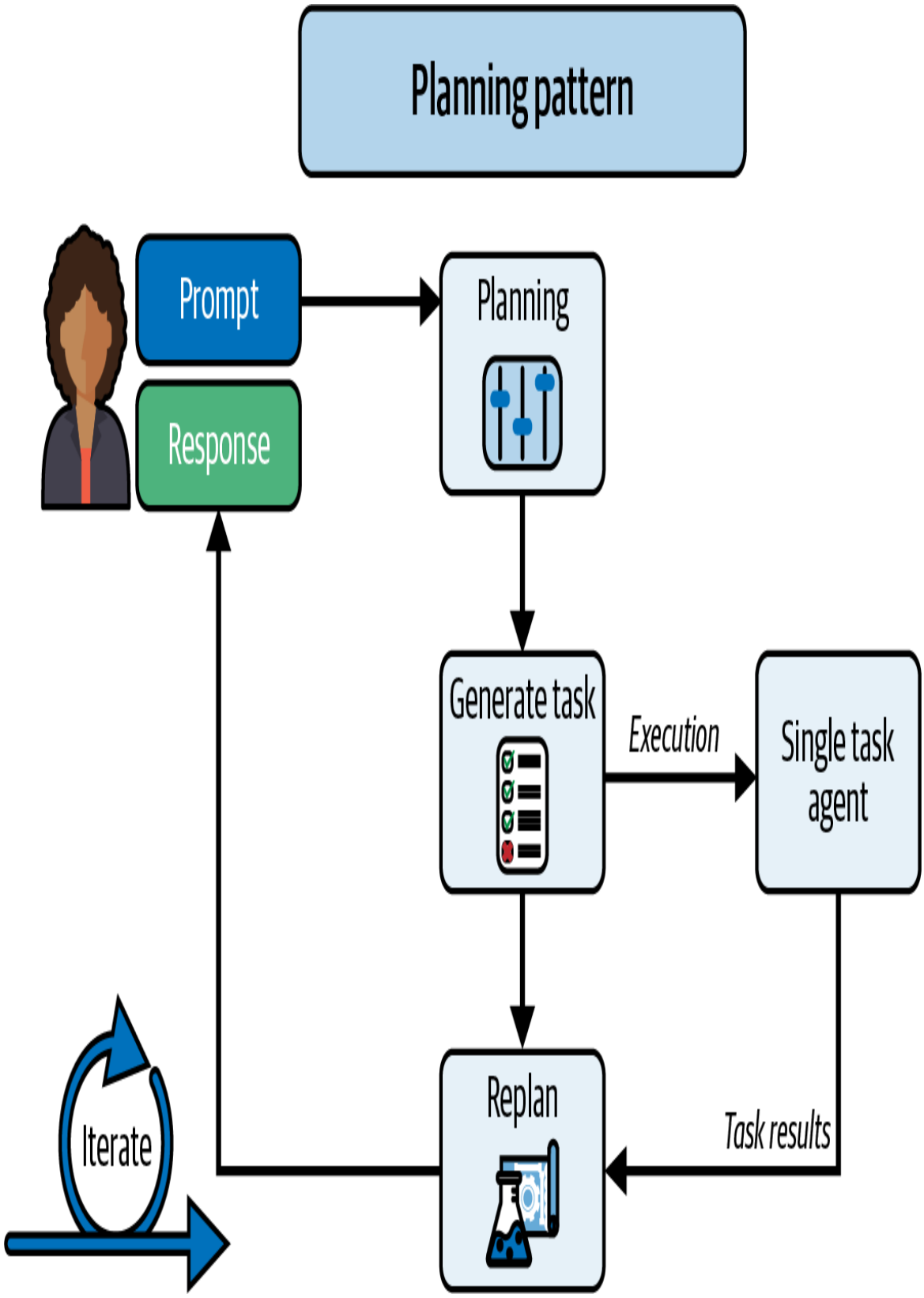


Figure 6-5. The Planning pattern

An organization might use planning agents for incident responses. When a security alert fires, the agent creates a visible plan: gather logs, identify affected systems, check for known vulnerabilities, assess threat level, determine containment steps, execute containment, verify resolution, and document the incident. This sort of task chain is typically visualized as a visible list or outline of steps ("Step 1: Gather data, Step 2: Analyze trends," etc.) shown before work begins, and progress indicators during execution ("Completed step 2 of 4"). Users may see the plan up front for review or discover it through progressive status updates. Whether it becomes a diagram or remains in text/UI is predicated on how sophisticated the process is (whether it requires the visualization of loops, or node-based structure, and so on).

Patterns like these ensure that agents don't skip critical steps or lose track of the overall objective. Without planning, an agent might jump between aspects of a task chaotically or get derailed by details. With planning, there's structure. The agent knows where it is in the process, what remains, and how to proceed. This knowledge is especially important when tasks have dependencies (you can't send invitations before booking a venue, for example).

Planning also offers a mental model that humans can understand. When an agent lays out a plan with clear steps, users can evaluate whether the approach makes sense before work begins. This creates a natural checkpoint where human judgment can redirect the agent if the planned approach seems flawed.

Designing for Visible Planning

For designers, planning creates both opportunities and challenges. The opportunity is that plans are inherently communicable in natural language that is easy to understand for end users. A well-structured plan tells users what the agent intends to do before it does it. This creates natural points for review and adjustment. For example, an agent suggesting a plan along the lines of "Here's my plan: Step 1: Gather data from Q3 reports, Step 2:

Analyze trends, Step 3: Create visualizations, Step 4: Draft summary. Does this approach work?" gives users a chance to redirect before work begins.

But exposing plans also introduces friction. Users might not want to review detailed task breakdowns; they may just want results. The balance depends on the relative risk or stakes of completing the task as well as user preference. For high-stakes or expensive operations, showing the plan up front is valuable because it may be costly or annoying for the user to relaunch it if it fails. For routine tasks, silent planning that just structures the agent's work may be better. The interface should probably support both: a "just do it" mode where planning happens invisibly, and a "show your work" mode where plans are presented for approval.

Progress visibility becomes especially important with planning. Because the agent has a structured sequence of steps, it can communicate where it is: "Completed step 1 of 4: Data gathered from Q3 reports. Now analyzing trends…." This ongoing feedback reassures users that work is progressing and helps them estimate time to completion. Without this visibility, long-running planned tasks feel like black boxes.

Plans also create natural intervention points. If a user sees the agent is on a second step of a four-step plan and realizes the approach is wrong, they can interrupt and redirect. This is much more manageable than trying to stop an agent that's working without structure. The plan becomes a shared reference point for human-agent coordination.

There's also the question of plan granularity. If the suggested plan is too high-level, it won't provide much insight into what the agent is actually planning on doing. If the plan is too fine-grained, users could be overwhelmed with details. The right level depends on task duration and complexity. For a quick task (say, one that takes under a minute), a three-step plan is probably sufficient. For an analysis lasting more than a few minutes, users might want to see the plan broken into phases with more detailed substeps revealed as each phase begins. Designers can initiate conversations and research about this in order to influence engineering conversations about model capabilities.

Conditional Planning and Branching

Not all tasks follow linear sequences. Many require conditional logic: “If this test passes, do X; if it fails, do Y.” Some require loops: “Keep trying approaches until one succeeds.” Planning patterns need to accommodate this complexity while remaining comprehensible.

One approach is *hierarchical* planning, where high-level steps contain substeps, and some substeps might themselves contain further detail. This approach allows the agent to show users a simple outline initially, then drill into specifics as needed: “Step 1: Diagnose the issue” might expand into “1a: Check logs, 1b: Run diagnostics, 1c: Identify root cause.”

Another approach is *adaptive* planning, where the agent presents an initial plan with explicit branch points: “After Step 2, I’ll either proceed to data visualization if quality is sufficient or gather additional data if gaps remain.” This makes conditionality explicit and helps users understand that the full path isn’t predetermined.

The design challenge is to present these complexities without making plans so convoluted that users can’t evaluate them. As mentioned earlier, the levels of detail and complexity in the task will influence the visual presentation: more complicated tasks require sophisticated and editable diagram structures. In occurrences with hypotheticals or conditional logic, sometimes the right answer is to hide the possible branches unless they’re triggered: show the main path and only explain branches when the agent actually takes one.

The Multiagent Collaboration Pattern: Specialized Coordination

No single agent excels at everything. The Multiagent Collaboration pattern mirrors how human organizations work by distributing tasks among specialists who collaborate toward shared goals. In a multiagent system, each agent has specific expertise or responsibility, and a higher-level *orchestrator* coordinates their work—see Figure 6-6.

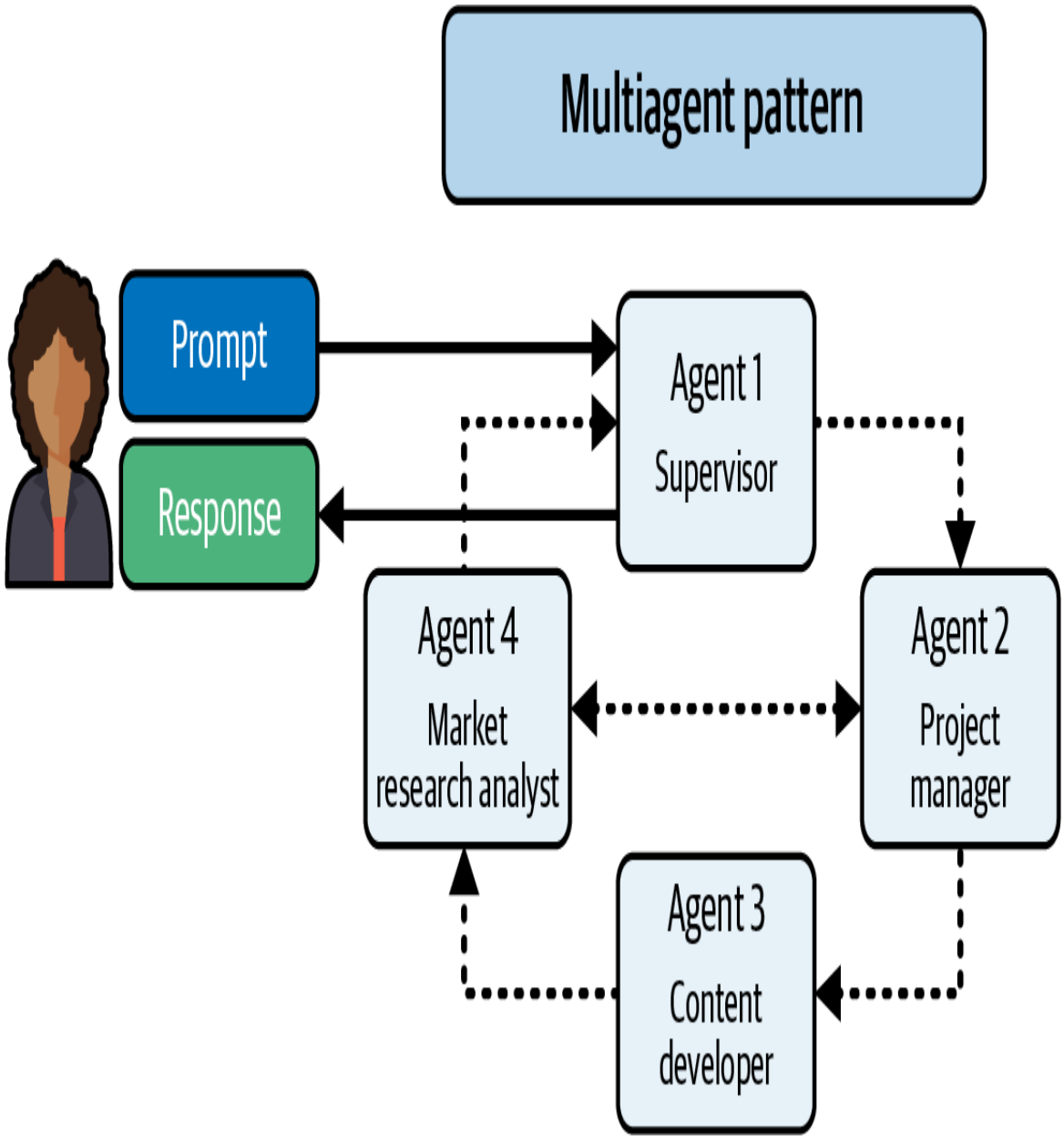


Figure 6-6. The Multiagent Collaboration pattern

This pattern recognizes that different aspects of complex problems require different capabilities. Software development, for example, involves requirements analysis, coding, testing, and documentation, each calling for different knowledge and approaches. By distributing these responsibilities among specialized agents, AI models can actually perform better than they would otherwise.

An engineering team might use multiple specialized agents for software development: one agent handles requirements gathering, another writes code, a third generates documentation, and a fourth conducts QA. An orchestrator manages the workflow between specialists. Development teams see progress through each phase but interact primarily with the orchestrator agent, which handles coordination complexity behind the scenes. Multiagent systems may appear as status dashboards showing different agents' progress ("Research agent: working, Analysis agent: waiting"), agent attribution in outputs ("Analysis by Agent A based on data from Agent B"), or simply as task-level progress that abstracts away the internal coordination.

Multiagent systems can operate in several modes. In *sequential handoff*, one agent completes its part and passes results to the next, like a relay race. In *parallel collaboration*, multiple agents work simultaneously on different aspects, and then their outputs are combined. In *supervised collaboration*, a manager agent oversees worker agents, reviewing outputs and directing next steps.

Multiagent Interface Challenges

The core design question is: what do users need to know about the Multiagent Collaboration pattern? In some cases, the answer is nothing. If the coordination works smoothly, users just care about results, not internal architecture. The system can abstract away the fact that multiple agents are involved and just show task-level progress with a message such as "Analyzing competitor landscape: 60% complete."

But when things go wrong (when one agent in a pipeline fails, or when agents produce contradictory results) users need enough visibility to understand what happened and how to respond. At minimum, error messages should indicate which component failed: "The document research agent couldn't access the shared drive. Would you like to provide documents manually or grant access?"

For complex workflows, some level of agent visibility helps users track progress and debug issues. A dashboard showing each agent's status (working, completed, waiting, failed) with brief descriptions of current tasks gives users a mental model of the system. They don't need to understand coordination protocols, but knowing that the "Research agent is gathering data while Analysis agent waits for input" helps them understand why progress might seem uneven.

Conflict Resolution in Multiagent Systems

When multiple agents work on related aspects of a problem, conflicts can arise. A research agent might find information that contradicts what an analysis agent assumed. A planning agent might create a schedule that a resource-checking agent determines is infeasible. An editing agent might rewrite content in a way that diminishes the technical accuracy that the original author agent ensured.

These conflicts require resolution mechanisms. Some systems use a manager agent that arbitrates: when agents disagree, the manager decides which to trust. Others employ voting or consensus: if multiple agents produce similar outputs, that's probably correct; outliers are suspect. Still others escalate to humans: when agents conflict, ask a person to resolve it.

> **TIP**
>
> From a design perspective, conflict situations create opportunities for transparency. When agents disagree, showing users the different perspectives can be valuable: "The cost estimation agent says this will take 10 hours. The historical data agent notes similar projects typically took 15 hours. Which estimate should I use?" This transforms a coordination problem into a decision point where human judgment adds value.

The ReAct Pattern: Adaptive Problem Solving

The ReAct pattern (short for *Reason and Act*) represents an adaptive approach where agents alternate between reasoning about what to do and taking action, then observing results and reasoning again. Instead of following a fixed plan, ReAct agents iterate: think, act, observe, think, act, observe. This loop continues until the goal is achieved or the agent determines it needs help. Figure 6-7 illustrates.

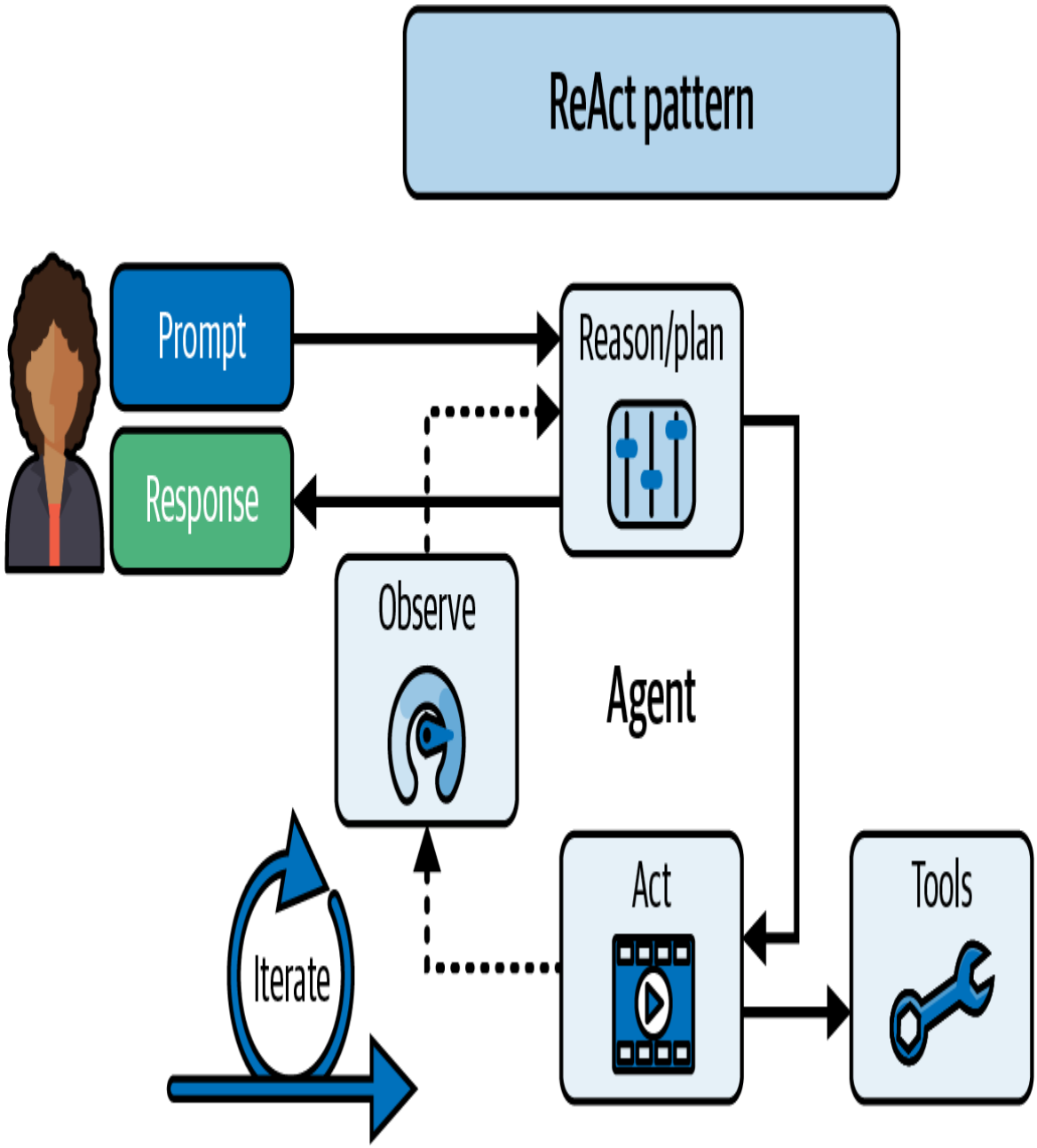


Figure 6-7. The ReAct pattern

Here's a hypothetical scenario: an IT support agent diagnosing server outages uses the ReAct pattern to adaptively troubleshoot. It reasons "disk errors could indicate hardware failure," acts by running diagnostics, observes critical disk health, reasons "this requires immediate escalation," then acts by creating a high-priority ticket with a summary of findings. Each step responds to what was discovered in the previous action, rather than following a preset script. From a visual perspective, users would

observe ReAct through visible iteration cycles showing the agent's thinking process ("I tried X and observed Y, so now I'm going to try Z") or iteration counters ("Attempt 3: Trying alternative approach"). Users see the agent adapting in real time rather than following a predetermined sequence.

> **TIP**
>
> The power of ReAct is that it combines the strategic thinking of planning with the flexibility of reactivity. The agent isn't mindlessly responding to stimuli, but it also isn't locked into assumptions that might prove wrong. Each cycle of reasoning incorporates new information from the previous action, allowing the agent to course-correct continuously.

Implementation and Use Cases

Technically, ReAct is often implemented through prompting patterns that encourage the model to explicitly output "thoughts" and "actions" separately. The agent's thought might be, "I need current weather information to answer this question," followed by an action: "Call `weather_api(location='Seattle')`." After observing the result, it thinks again: "The API returned temperature and conditions. I can now provide a complete answer," and acts: "Generate response incorporating weather data."

This interweaving of reasoning and action allows agents to use tools (the action part) while maintaining strategic thinking (the reasoning part). It's essentially a way to do planning on the fly, adjusting the plan as new information comes in. The pattern works especially well for exploratory tasks, troubleshooting, research, and anything where the path forward isn't clear from the start.

ReAct combines naturally with other patterns. An agent might use ReAct for high-level problem solving while using Reflection to check its reasoning at each step, Tool Use to take actions, and even coordinating with other agents for specialized subtasks. It's important to recall that these patterns

are meant to be combined, and ReAct serves as a flexible framework that incorporates planning, tool use, and adaptation.

> **TIP**
>
> Research tasks particularly benefit from ReAct. An agent investigating a question doesn't know up front what information exists or where to find it. With ReAct, it can search, evaluate what it finds, identify gaps, search differently, synthesize discoveries, realize something's missing, and search again: iterating until it has a comprehensive understanding. This kind of exploratory work doesn't fit into neat planning but is perfect for adaptive reasoning.

Designing for Iterative Agents

From a design perspective, ReAct agents create unique challenges because their process isn't fully determined up front. A planning agent can tell you its plan; a ReAct agent can only tell you its current thinking and what it just tried. This makes progress harder to predict, but it also means the agent is responding to reality rather than following assumptions.

Transparency becomes especially valuable with ReAct. Showing the agent's reasoning helps users follow the logic and builds trust in the adaptive process. However, the iterative nature of ReAct means users also need to be prepared for some back-and-forth. This is fine for tasks where iteration is expected (research, troubleshooting, creative work) but might feel inefficient for straightforward requests where the answer should be immediate. The interface might need to signal which mode the agent is in: "I'm exploring this problem step-by-step" versus "I can answer this directly."

Error handling and escalation are particularly important for ReAct. Because the agent is adapting rather than following a predetermined plan, it might discover that the problem is more complex than was initially apparent. The system needs clear mechanisms for the agent to recognize when it's out of its depth and escalate to human help, providing a summary of what it attempted. A notification like "I tried diagnosing through logs and

diagnostics but the issue appears to require hardware-level inspection. Here's what I found…" gives humans the context to take over effectively.

The challenge of showing ReAct progress is that the agent doesn't know how many iterations will be needed. Unlike a plan with defined steps, a ReAct loop continues until some condition is met. The interface can't show "Step 3 of 5" because there is no fixed number of steps. Instead, it might show iteration count ("Attempt 3: Trying alternative approach") or time elapsed ("Researching for two minutes so far") to give users a sense of effort invested.

Moving from Engineering to Experience

The patterns we've explored (Reflection, Tool Use, Planning, Multiagent Collaboration, and ReAct) represent the technical architectures that enable agentic capabilities. These are primarily engineering concerns: when development teams build agentic systems, they're implementing some combination of these patterns, layering them together to create the behaviors their products require.

However, these patterns are somewhat occult to the daily user of an AI agent. A user working with an agent doesn't think, "Ah, this is using the Reflection pattern" or "I see Multiagent Collaboration happening here." They experience a surface layer above this, where the abstract capabilities of agentic patterns become visible and comprehensible. Understanding these moments, and the design opportunities they present, is essential for creating agentic systems that actually communicate capabilities. The next section examines these user touchpoints, looking at how the patterns we've discussed manifest in actual interface moments and what designers can do to make autonomous work feel comprehensible and controllable.

Where Users Experience Agentic Patterns

As mentioned earlier, computation should be a much more visible process when agents are involved. In other AI patterns, computation is visualized

primarily as latency. But in agentic processes, as you can see in Figure 6-8, the computation step transforms from a single opaque moment into a sequence of visible operations.

Figure 6-8 illustrates the additional steps added by agentic processes: processing, progress sharing, and checkpoints. Each new step also carries visual opportunities and requirements, like granting permissions when the model needs it or clarifying a plan to refine a task. These steps represent opportunities to boost user understanding, but it’s first important that, as a designer, the process is clearly understood.

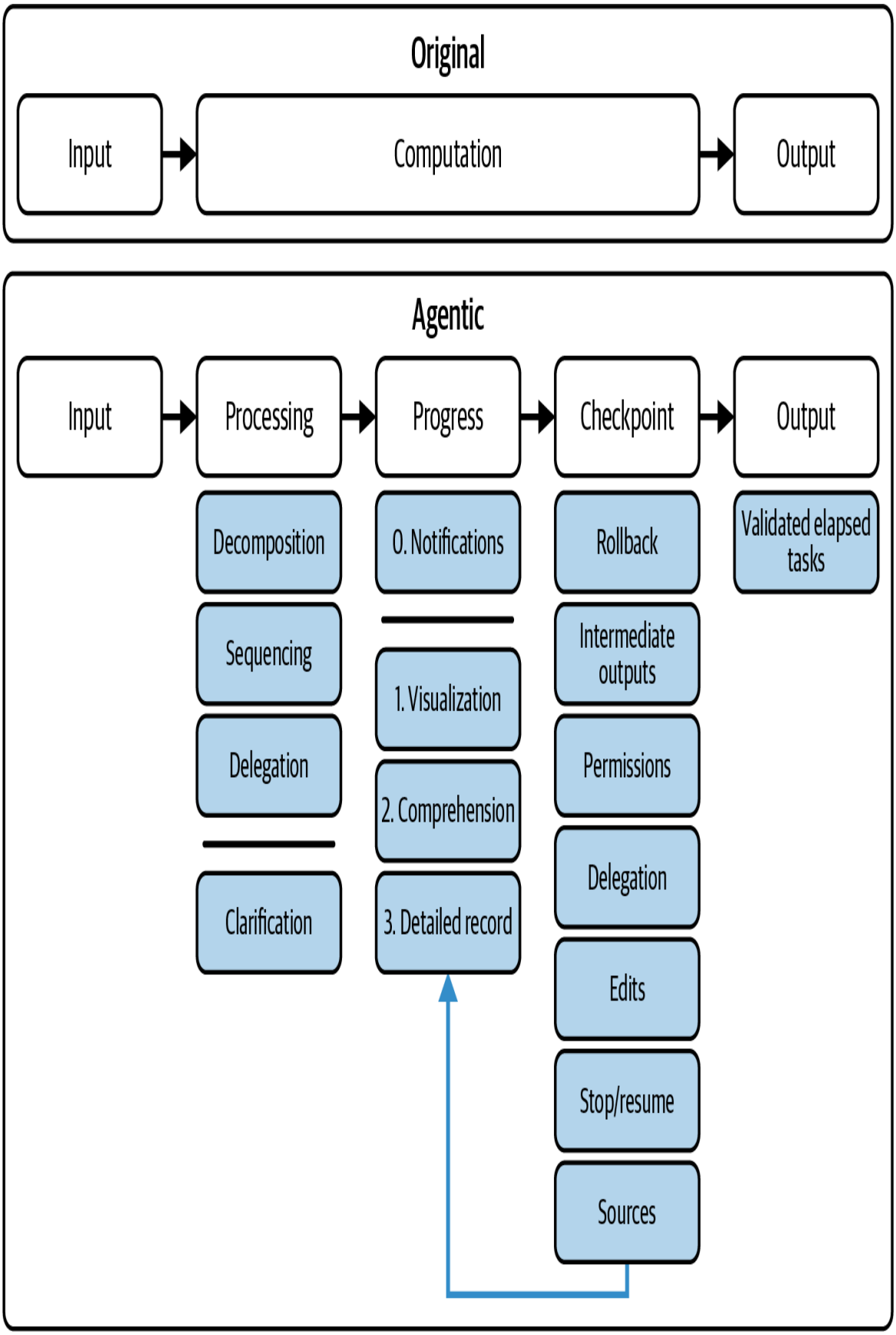

Original
Input
Computation
Output
Agentic
Input
Processing
Progress
Checkpoint
Output
Decomposition
Sequencing
Delegation
Clarification
0. Notifications
1. Visualization
2. Comprehension
3. Detailed record
Rollback
Intermediate outputs
Permissions
Delegation
Edits
Stop/resume
Sources
Validated elapsed tasks

Figure 6-8. The computation step in agentic processes

Processing

Earlier chapters described inputs as the point where human intention becomes interpretable, right before computation takes place. This appears as a single transaction: the user expresses intent, usually in the form of a written text prompt, the model interprets the prompt, and a response appears. The interface manages that exchange, helping users see what the system has understood and how it will respond.

In agentic systems, inputs open into a new stage: *processing*. When a task can unfold autonomously, the entry point no longer simply starts a conversation; instead, it initiates a process to clarify and take action. A request is still a prompt in the conversational sense; it expresses intent and sets direction. However, in an agentic tool that same request triggers a chain of computation that will be decomposed, sequenced, and delegated (and sometimes clarified).

Decomposition

When faced with an input prompt, the system *decomposes* the user's goal into actionable parts, representing a transition from the user's input to a structured plan of work. The resulting transformation can be made legible to the user in order to communicate shared intent: what the system believes the user wants and how it proposes to achieve it.

From a design perspective, decomposition can look like a plan of tasks, and its intention is to help users understand what an agent intends to do before it starts. When shown at the right moment, usually after the request but before execution, it gives the user space to confirm that the agent has understood correctly or to suggest small adjustments before resources are spent. The decomposed plan might appear as a numbered outline for straightforward work or as a simple diagram when steps depend on one another. Even brief visibility like this can build confidence by showing that the agent is proceeding with structure rather than improvisation.

Decomposition is most useful when a task has genuine complexity. Revealing how the agent breaks a goal into parts helps users see both the scope of work and how their intent has been interpreted, turning uncertainty into expectation. For quick, single-step requests, this process can happen silently, but when multiple stages or external tools are involved, showing the outline makes the work legible. Some products present a plan only when a task passes certain thresholds of cost, time, or required access. Doing so balances transparency with efficiency and gives users the option to view or skip the details according to their comfort with autonomy.

Sequencing

Some tasks have dependencies on others; *sequencing* is the order in which tasks will take place. It determines what must happen first, what can run in parallel, and where progress naturally pauses for review. A clear sequence helps users follow the agent's reasoning and anticipate what comes next, particularly when steps depend on earlier results. The challenge with sequence representation is maintaining comprehensibility (and consistency) as complexity increases. A 3-step sequence can be shown as a simple numbered list or progress bar, but a 15-step sequence may need to be grouped into broader phases that expand on demand. Defining common query types can help with this: do your users commonly request tasks that involve many steps, or few? A successful design solution will represent both clearly.

Delegation

Delegation marks the point where an agent moves from describing intent to carrying it out. In practice, it often happens when additional agents or external resources are brought into play. A design that handles delegation well anticipates these moments, showing which parts of a task will proceed autonomously and which will rely on other systems or collaborators. When multiple agents are active, it should be clear who is responsible for what, so that progress and accountability remain visible.

Planning with tasks

Decomposition, sequencing, and delegation together form the practical core of the processing stage. Once an agent has understood a request, it begins to shape that intent into a structured plan made up of tasks. Each task represents a discrete action or goal that can be executed, monitored, and, if needed, revised. Thinking of these tasks as visible units makes the process easier to follow. A planning component can display tasks as cards, rows, or nodes, each showing what is being worked on, what depends on it, and whether it is complete. This format makes the agent's reasoning tangible, something users can see and respond to rather than imagine.

The visual representation does not need to be elaborate. Replit's "In progress tasks" component, shown in Figure 6-9, visualizes the steps very simply. For more complex work, tasks can expand to show dependencies, assigned agents, or active tools. The primary purpose is to make it clear how the system has structured the work and how it will proceed, but as users watch plans unfold, they begin to recognize patterns in how agents organize effort, and that familiarity builds trust.

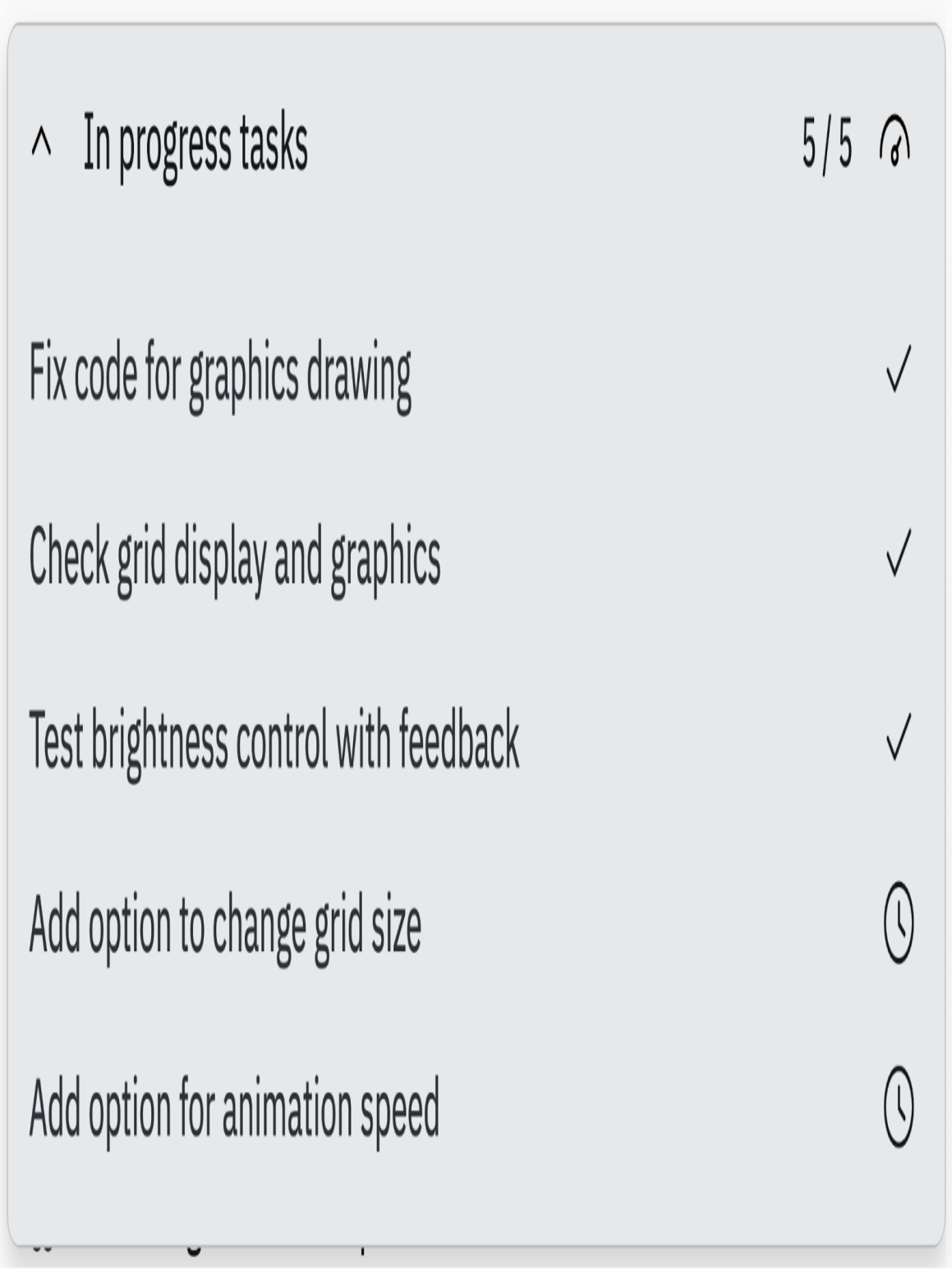


Figure 6-9. Replit's "In progress tasks"

Clarification

In some cases, it is useful to include a brief clarifying step before execution begins. This is not necessary for every query, but for tasks that are complex, subjective, or computationally heavy, clarification can prevent wasted effort and improve results. Clarification can happen naturally through conversation, as when a user and agent refine a prompt in chat, or it can appear as a short guided form that asks two or three focused questions. These might include whether the response should be academic or conversational, whether it should be concise or exploratory, or what kind of sources should be prioritized.

Clarification serves both accuracy and trust. When tasks require significant processing time or count toward a limited quota, a few seconds of confirmation reassure users that the system understands their needs and will spend that effort wisely. It also helps users see what details meaningfully influence the outcome, reinforcing a sense of shared authorship rather than blind delegation. Making the clarification step simple to interact with is vital to its success. In Claude Code, as seen in Figure 6-10, the user can rapidly tab through their options instead of typing out responses to clarifying questions.

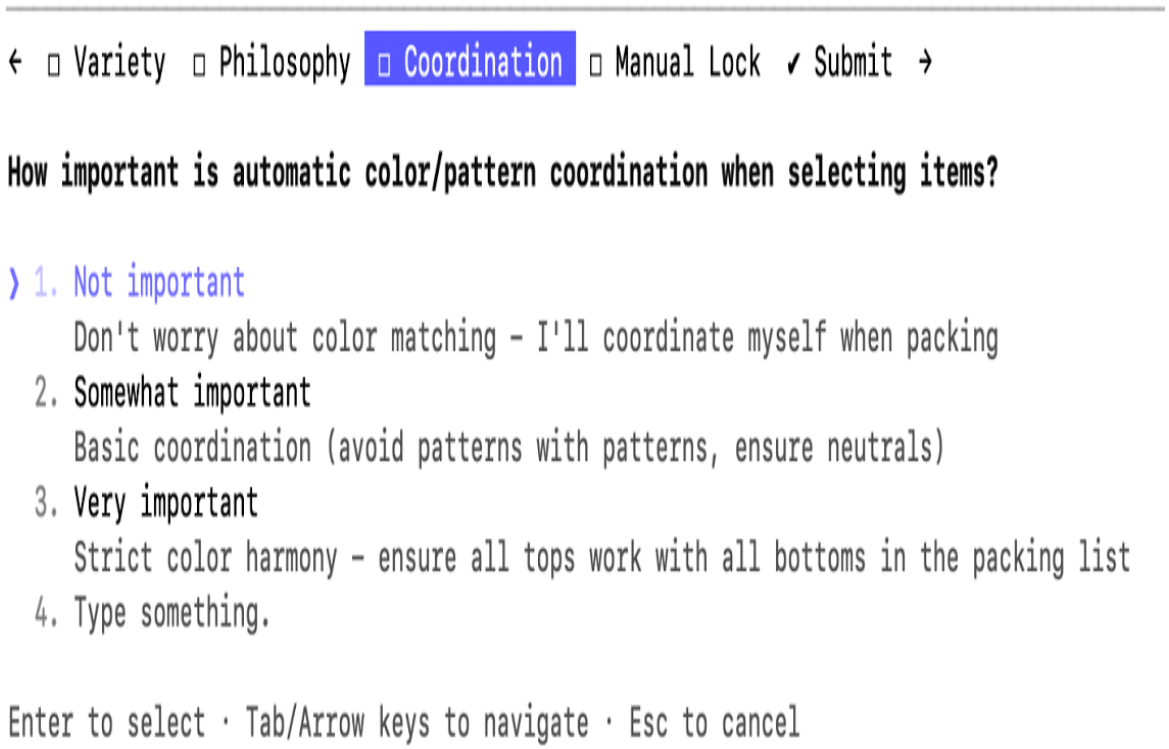


Figure 6-10. Tabbing through Claude options

In agentic AI, planning is a highly valuable step for both people and systems. For users, it offers a clear view of their own request, showing how the task has been interpreted and what the agent is capable of doing. For the model, it provides structure and permission to act, turning intent into a defined set of steps with the user's support. By making this process visible, planning removes the sense of a black box and replaces it with collaboration. That is why many tools, such as Replit (as shown in Figure 6-11), now include an optional planning stage that invites users to preview and refine the agent's approach before execution, demonstrating how effective this pattern has become in practice.

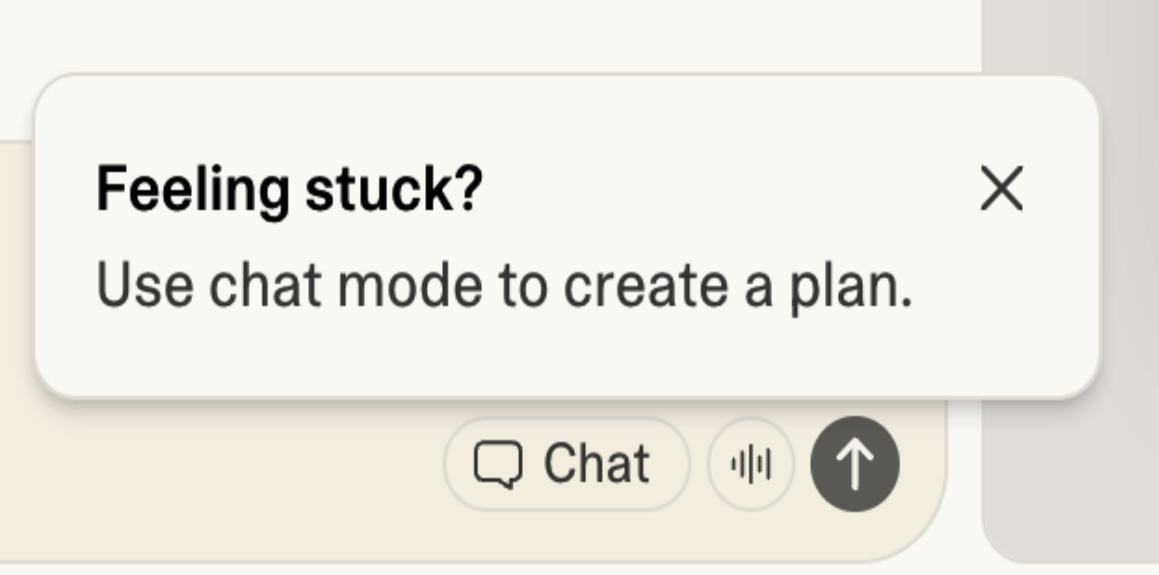


Figure 6-11. Replit's optional planning stage

Progress Communication

Once a plan is in place and the agent begins to act, the designer's focus shifts to communicating progress. At this stage, the interface is no longer helping users decide what to do but is helping them stay oriented as work unfolds over time. This is especially important for agentic systems, where tasks may run in the background, act on schedules, or proceed autonomously. Users need to know that work is continuing, understand when their attention is required, and feel confident that they can check in without losing context.

Progress communication in agentic interfaces is best understood as a layered hierarchy. This structure is described in full in Chapter 4, but it becomes especially important once agents begin to act on their own. At a high level, the hierarchy includes:

Notifications

> The outermost layer. Notifications allow users to step away while remaining confident that work is progressing or that they will be called back when needed. They support delegation over time by signaling completion, pauses, errors, or requests for approval without requiring continuous attention.

Tier 1: Overview

A high-level status surface that keeps users oriented with minimal detail. This layer answers simple questions like whether work is still underway, what stage the agent is in, or whether dependencies or permissions are blocking progress.

Tier 2: Detail

An expanded view that provides structure and explanation for users who want to understand how a task is unfolding. This layer makes progress legible by showing completed steps, work in progress, and near-term actions, without exposing the full internal record.

Tier 3: Full record

A complete account of the agent's actions, supporting review, debugging, learning, or compliance. While few users will consult this view regularly, its existence contributes to trust and accountability.

These layers form a flexible system rather than a strict sequence. Users may move between them depending on context, risk, or curiosity. A long-running task might rely primarily on notifications, with occasional glances at a high-level overview, while a high-stakes or unfamiliar task may prompt deeper inspection through detailed progress or the full record.

Similar to other AI interactions, users bring a wide range of intent and risk-readiness to agentic tasks. Some will want to stay close to execution, while others will prefer to delegate and return only when prompted. Layered progress communication allows designers to support this range without overwhelming the interface or assuming constant attention.

Alternative Progress Patterns: Browser and Computer Use Visualization

Other patterns exist beyond layered textual progress, such as Computer Use agents or Browser agents. These examples include a visualization of the

agent using a remote browser or desktop interface to accomplish a task (see Figure 6-12). In these cases, the user can’t reflect back on what’s been accomplished through a structured plan view or look forward to upcoming steps in a clear sequence. The temporal navigation that layered progress provides isn’t naturally available in a live screen-sharing model. The efficacy of this pattern for pure progress tracking is therefore limited.

Opening first search result

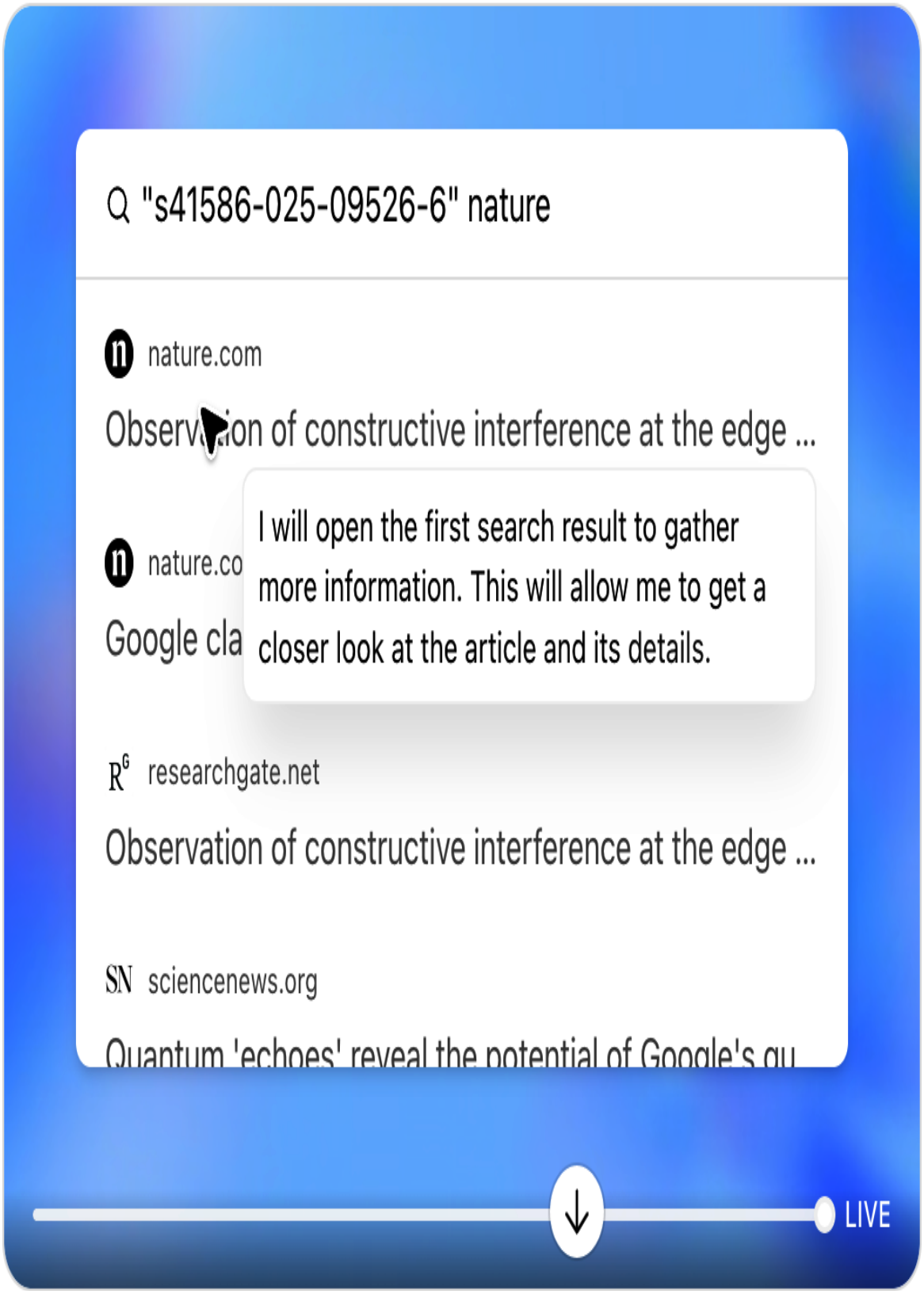


Figure 6-12. ChatGPT's Computer Use agent visualization, showing live task execution through a simulated remote desktop interface

However, this visualization pattern does provide opportunities to communicate capabilities to users in ways that text-based progress cannot match. Because many people remain unfamiliar with agentic tools and uncertain about what they can reliably accomplish, showing a desktop or browser interface as the agent works can be highly educational. Users see familiar territory, the applications and interfaces they use daily, being navigated by the agent. The familiarity creates immediate understanding that might otherwise require extensive explanation in text.

The pattern works through knowledge transfer. When users watch an agent open a spreadsheet application, select data ranges, create formulas, and generate charts, they're seeing capabilities demonstrated rather than described. They can directly observe that the agent knows how to navigate menus, interpret interface elements, and complete multistep workflows within familiar tools. This kind of observational learning helps users form more realistic mental models of what agents can handle, grounded in concrete demonstration rather than abstract capability lists.

The visualization also makes tool use concrete. Textual progress might say "querying customer database" or "updating spreadsheet," leaving users to imagine what that means. Screen visualization shows the actual database interface being accessed, the specific queries being constructed, and the spreadsheet cells being modified. That specificity allows users to verify that the agent is accessing the right data and making appropriate changes, offering a form of oversight that pure status text cannot provide.

At the same time, the limitations of this pattern should be acknowledged clearly. Real-time screen streaming creates significant bandwidth demands and can be laggy or choppy, which undermines the sense of smooth agent operation. The visualization only shows what is happening in the moment, not what has already been completed or what remains, making it difficult to estimate progress or plan interruptions. For long-running tasks, continuously watching a screen is neither practical nor pleasant. Users need

to be able to shift attention to other work and return later, which this pattern does not naturally support.

The pattern works best in specific contexts. In closed environments focused on particular software, where users need to learn capabilities and build trust, the visualization offers clear value. An agent that assists with spreadsheet tasks benefits from showing its work directly in the spreadsheet interface, since users can immediately judge whether the agent understands the tool they're struggling with. In environments with security or audit requirements where every action must be observable, screen-based visualization can provide comprehensive transparency, even if it is not the most efficient form of progress communication.

For design teams considering this pattern, it should not be treated as the sole method of progress communication. Users benefit from being able to toggle between watching a live visualization when they want detailed visibility and viewing summarized progress when they do not. Combining browser or desktop visualization with layered progress views gives users flexibility to choose the level of visibility that fits their current needs. Some tasks merit close observation, while others only need monitoring until completion.

Checkpoints

Checkpoints are moments when an agent pauses during a multistep task to show progress, confirm assumptions, or request approval before continuing. They divide continuous work into visible, reviewable segments and give users the chance to see what has been done, verify that the direction is correct, and decide whether to let the agent continue or to make changes.

A checkpoint might appear when a plan has been generated, a dataset has been analyzed, or an action is about to be executed that cannot easily be undone. It can be created automatically when the system's confidence drops or when a step requires confirmation, or it can be built in deliberately by the designer. The goal is clarity. The user should always know what the system is waiting for, what will happen next, and what choices are available.

Rollback

A rollback is the ability to return to an earlier checkpoint. It works much like version history in GitHub or Google Docs, where each checkpoint creates a saved state that can be restored later. This allows users to undo unwanted changes or recover a stable version if the agent's later work goes off course.

Rollbacks should follow familiar recovery patterns such as a "View history" link, a list of saved states, or a comparison view showing what changed between versions. Restoring a version should never delete the current one but instead should create a new branch or snapshot. When an agent produces artifacts that evolve over time, such as documents, dashboards, or generated code, automatic rollback at each checkpoint gives users the assurance that progress is never lost.

Intermediate outputs

Intermediate outputs are partial results that appear between checkpoints, such as charts, draft text, data summaries, or early visualizations. They let users confirm that the agent is on the right track before the task is finished.

Each output should be labeled as provisional and linked clearly to the step that produced it. If a user edits an intermediate output, the agent should treat that edit as new input for the next stage rather than overwriting the original work. Providing a consistent review pattern, such as preview and then accept or revise, keeps the interaction predictable and easy to follow.

Permissions

Permissions determine what the agent is allowed to access or change, and they often appear at checkpoints when the agent needs a new capability, such as connecting to a calendar, reading a document, or sending an email. A good permission design explains what is being accessed, why the access is needed, and how it can be revoked later. The explanation itself is as important as the permission request.

Consistency is essential. Every permission request should use the same placement, iconography, and phrasing so that users can immediately recognize what constitutes consent. If a permission involves an external service, the handoff should use a standard and transparent method, such as an OAuth-style window that makes it clear who is requesting access. After a permission is granted, users should be able to find and manage it easily in a predictable place, such as a “Connected apps” or “Authorized access” view.

When a permission is denied, the system should show the outcome clearly and guide the next step without breaking the workflow. For example, a message such as “Access to shared drive denied. Task paused until permission is granted or skipped” communicates what has happened and what the user can do next.

Edits and errors

Edits are the user’s way of steering the agent mid-process. When a change is needed, ideally the user should be able to stop the current run, adjust the request, and resume from the last checkpoint without losing work. The pause state should make it clear what has been completed, what is in progress, and what will happen next. A clean Pause and Edit pattern prevents users from feeling trapped in the system’s flow.

Errors can appear in the same interface space. If the agent fails to complete a step or receives an invalid response, the checkpoint should record the failure and show it as part of the ongoing work, not as a system interruption. Users should see what caused the issue and be given choices such as retrying, skipping the step, or revising input. Keeping error recovery within the normal rhythm of checkpoints helps users stay oriented and the process stay trustworthy.

Sources

Sources displayed during checkpoints give transparency to what the agent has used or referenced. When an output relies on external data, links, or tools, these should be listed clearly within the checkpoint summary. The display should show the name of each source, when it was accessed, and

provide a way to open or verify it. This turns provenance into something practical rather than decorative.

The placement of sources depends on how the work is reviewed. In a chat interface, they might appear as expandable lists or inline links under the relevant statement. In a canvas or document view, a small section labeled “Sources consulted” can anchor them at the end of the checkpoint summary. The goal is to make them findable and verifiable without breaking the user’s concentration. Consistent placement and simple formatting keep the relationship between agent reasoning and evidence easy to follow.

Together, checkpoints, rollback, intermediate outputs, permissions, edits, and sources form the structure that keeps agentic work intelligible. They turn a continuous stream of computation into something that can be inspected, revised, and trusted. Each element gives the user a way to see what is happening, to correct direction when needed, and to understand where information comes from. When these parts are designed consistently, users feel that the system’s autonomy is working in partnership with them rather than operating out of sight. This structure lays the groundwork for the broader principles that govern how agents should communicate, act, and recover across longer tasks.

New Principles for Agentic Interface Design

The shift from conversational AI to agentic systems changes what interfaces need to accomplish. Designers are no longer just managing a question-and-answer exchange. They’re designing for systems that interpret goals, make plans, take actions, and adapt as they work. Doing so requires rethinking some fundamental assumptions about how interfaces mediate between human intent and machine capability.

Three principles emerge from this new reality. They build on everything we’ve discussed about traditional AI interfaces but recognize that autonomy creates different design challenges. When systems can act on their own, the interface becomes less about facilitating individual exchanges and more about maintaining a productive partnership over time.

Reveal the Plan

When someone delegates a task to an agent, there's an immediate translation problem. The user has expressed what they want in human terms: "Prepare the quarterly report" or "Find competitive pricing data" or "Debug this code." The agent must interpret that request and decide how to accomplish it. This interpretation step, where intent becomes action, is where many agentic systems lose user trust.

The problem isn't that agents misunderstand requests (though that happens) but that users can't tell whether the agent has understood correctly until work is already underway or complete. And by then, correcting course is expensive or impossible. That creates a trust gap right at the moment when trust matters most: when the user is handing over control.

Revealing the plan solves the issue by making interpretation visible before execution begins. After receiving a request—but before starting work—the agent shows how it has understood the task. It might be a numbered list of steps, a brief summary of the approach, or a simple outline of what will happen. The format matters less than the timing. Users need to see the plan while they can still influence it.

Consider how Replit handles this. When you ask it to build something, before beginning, it first shows you a plan: what files it will create, what functionality it will implement, and what order it will work in. The preview takes just seconds to review but prevents countless minutes of misdirected effort. Users can spot misunderstandings immediately. "Wait, I meant a web app, not a command-line tool" is much easier to address before any code is written.

The plan also serves as a contract of sorts. It sets expectations about what the agent will and won't do. When users approve a plan (explicitly or by not objecting), they're agreeing to a specific scope of work. This prevents the uncomfortable situation where an agent does either much more or much less than expected. A plan that says "I'll analyze Q3 sales data and create three visualization charts" tells users exactly what to anticipate.

The principle extends beyond just showing initial plans. As agents work, plans often need to change. New information emerges, dependencies fail, or the agent discovers a better approach. When this happens, the interface should surface the proposed change and explain why it's necessary. For example: "The API I planned to use isn't responding. I can try an alternative data source or wait and retry. Which would you prefer?" Ongoing visibility of planning keeps users informed even as situations evolve.

But revealing plans must be done with care. Not every task needs elaborate planning visible to the user. Simple, low-stakes requests can proceed without ceremony. The interface should calibrate plan visibility to task complexity and importance. A quick calculation might not need any visible plan; a multistep analysis absolutely does. The key is ensuring that users can always trace how their intent translated into action when that traceability matters.

Prioritize What Matters Most

Agentic systems generate vastly more information than traditional AI does. Instead of a single response, they produce plans, progress updates, intermediate results, tool outputs, status messages, permissions requests, and final deliverables. The sheer volume of potentially relevant information becomes a design problem. Show everything, and users are overwhelmed. Hide too much, and they lose track of what's happening.

The solution is aggressive prioritization based on what users actually need to know at each moment. Early in this chapter, we explored the layered approach to progress communication: notifications for awareness, overview for orientation, details for verification, and full records for accountability. This structure embodies the prioritization principle; each layer serves a different level of attention. Users engaging casually see just enough to stay oriented. Users engaging deeply can access everything. The interface adapts to the level of scrutiny users bring without requiring them to manually manage what's visible.

Prioritization also means understanding that importance shifts as tasks unfold. When an agent starts working, users care most about whether it understood the request correctly. During execution, they want to know that progress is normal and whether anything needs attention. When work completes, they focus on results and next steps. The same piece of information might be critical at one moment and irrelevant at another.

Take error messages as an example. When an agent encounters a problem, the immediate priority is communicating three things: what happened, whether it's recoverable, and what the user should do. Technical details about why the error occurred might be important for debugging but shouldn't compete with those primary messages. They belong one click deeper, available when needed but not cluttering the initial communication.

This principle particularly matters for multiagent systems where several specialists work in parallel. Users don't need play-by-play updates from every agent. They need to know overall progress and whether any agent has hit a problem requiring attention. "Research agent has found 12 relevant documents. Analysis agent is processing. Visualization agent is waiting for data" compresses a huge amount of activity into a scannable status that tells users everything they need to know at a glance.

The challenge is deciding what qualifies as *most important*, and this varies by context. For a financial analysis, accuracy matters most, so data sources and calculations might deserve prominent placement. For creative work, intermediate outputs might be more important than process details. The interface should make these prioritization decisions based on the task type and the user's demonstrated interests, not as generic rules about what information is important.

Successful prioritization creates a sense of calm even when agents are doing complex work. Users trust that important information will surface when needed. They don't have to actively monitor every detail because the interface is doing that filtering for them. This trust allows users to delegate with confidence, knowing they'll be interrupted only when their attention is genuinely required.

Design for Shared Control

The promise of agentic AI is that it can work independently, handling tasks without constant supervision. But independence without accountability creates anxiety. Users need to feel that delegating work doesn't mean losing control over it. The interface must maintain a careful balance: giving agents room to operate while keeping users empowered to intervene, redirect, or reclaim control whenever needed.

Shared control starts with clear boundaries about who does what. Ideally, before work begins, the system confirms what it plans to do, as was shown in Figure 6-8, but beyond this, it's about establishing what the agent has permission to do independently versus what requires approval. These boundaries should be explicit and adjustable: a new user might want confirmation for everything, whereas an experienced user might grant broad autonomy for routine tasks while requiring approval for anything unusual.

During execution, shared control means maintaining what you might call *interruptibility*. The agent should be stoppable at any moment without corrupting work or losing progress. This requires designing around checkpoints and intermediate states where pausing is safe. When a user clicks Pause, they should immediately see what's been completed, what's in progress, and what remains. They should be able to adjust the plan, modify parameters, or take over manually from that checkpoint.

The ability to undo or roll back changes is equally critical. Knowing that actions can be reversed makes delegation feel safer. This is why rollback was covered as a key checkpoint feature earlier. But it's not just about technical capability. It's about making that capability visible and trustworthy. Users should see clearly that they can return to any previous state, understanding what would be kept and what would be lost.

Shared control also appears in how agents surface decisions that need human judgment. Not every choice should flow to the user; that would defeat the purpose of autonomy. But certain decisions genuinely benefit from human input: choosing between valid alternatives when trade-offs are subjective, resolving ambiguities the agent can't confidently handle, or

approving actions that have lasting consequences. The interface should make these decision points feel like natural collaboration rather than system failures.

Consider how Claude Code handles that balance. It works independently on coding tasks but pauses when it encounters situations requiring human judgment. Should it overwrite an existing file? Which of two valid approaches should it take? By surfacing just these decisions, the system maintains momentum while keeping users in control of what matters.

The interface patterns that support shared control include explicit handoffs ("I've completed the analysis. Would you like to review before I proceed to recommendations?"), permission requests that explain scope and duration, status indicators that show whether the agent is working independently or waiting for input, and clear markers for when ownership of work transfers back to the user. Each of these patterns reinforces that control is shared, not surrendered.

There's also a temporal dimension to shared control. Users should be able to set boundaries about when agents can act. Some tasks can run immediately. Others should wait for business hours. Some can proceed as soon as prerequisites are met. Others need explicit triggering. The interface should make these temporal boundaries as clear as the functional ones.

The goal is making autonomy feel like assistance rather than replacement. When designed well, shared control creates a productive partnership where agents handle routine work while humans provide judgment, creativity, and strategic direction. Users feel that they're working with a capable assistant who respects their authority rather than an autonomous system that might act against their interests.

Summary

Despite his puckish charm, Clippy was a terrible failure. He became synonymous with misguided automation, highlighting how quickly users can shift from optimism to skepticism when engaging with theoretically

supportive tools. Much of the guidance in this chapter is aimed at preventing that skepticism from arising, with clear communication about what's taking place.

The patterns and principles explored throughout this chapter show how far we've come since 1997. Modern agents can reflect on their work before presenting it, checking for errors and improving quality. They can use tools to actually accomplish tasks rather than just offering suggestions. They can plan complex workflows, coordinate with specialists, and adapt when situations change. These capabilities finally deliver on the promise Clippy represented: AI that genuinely helps by doing, not just advising.

But capabilities need interfaces to become useful. The agentic patterns we examined create new design challenges that didn't exist with simpler AI. Reflection adds latency that must be explained. Tool use requires permission models that balance access with security. Planning needs review mechanisms that don't slow everything down. Multiagent coordination must be comprehensible without being overwhelming. And adaptive problem solving requires interfaces that support iteration and redirection.

The three principles the chapter outlined address these challenges. Revealing the plan ensures users understand how their intent becomes action, catching misunderstandings early when correction is cheap. Prioritizing what matters most prevents information overload while keeping users oriented, turning potentially overwhelming complexity into manageable awareness. And designing for shared control makes autonomy feel like partnership, maintaining human authority while enabling machine efficiency.

These principles work together. A visible plan means nothing if it's buried in unnecessary detail. Prioritized information is only useful if users can act on what they see. And shared control requires both visibility into what's happening and the ability to influence it. Together, they create interfaces where agentic AI feels comprehensible and trustworthy rather than mysterious and threatening.

The evolution from Clippy to today's agents reflects broader changes in how designers think about human-AI collaboration. Clippy embodied a model where AI knew better and would help whether you wanted it or not. Today's agentic systems embody a different model where AI is capable but not presumptuous, independent but not uncontrolled, helpful but not intrusive. The interface is what makes this model real, translating abstract capabilities into concrete experiences that users can understand and trust.

Looking back at the examples throughout this chapter, from planning agents that handle incident response to multiagent systems that accelerate software development, you see these principles consistently applied. Successful agentic products make their plans visible, surface important information while keeping details accessible, and maintain clear boundaries about who controls what. They prove that autonomous AI can enhance human capability without replacing human judgment.

The future will bring even more sophisticated agentic capabilities. Agents will handle longer tasks, coordinate in larger teams, and adapt to more complex situations. But the core design challenges will remain: making autonomous work comprehensible, maintaining appropriate human oversight, and ensuring that delegation feels like empowerment rather than displacement. The patterns give us the technical foundation. The principles provide the design framework. And the specific interface decisions determine whether users experience AI agents as partners or intruders.

Clippy taught us what happens when powerful ideas meet poor execution. The animated assistant had the right vision: computers should actively help users accomplish tasks, not just respond to commands. But without transparency, prioritization, or shared control, that vision became an annoyance that Microsoft spent years living down. Today's agents have the technical sophistication Clippy lacked. Whether they succeed where Clippy failed depends entirely on whether designers create interfaces that make autonomy feel like collaboration. The principles explored in this book show the way: reveal intentions before acting, surface what matters without overwhelming, and maintain human authority while enabling machine capability. When you follow these principles, you create AI agents that feel

less like Clippy and more like the helpful partners Microsoft originally envisioned.

OceanofPDF.com

Index

A

About Face (Cooper), When the System Doesn’t Deliver: Designing for Errors

actionable outputs, Design Principles of Outputs, Designing for Actionability-Moving an output to the canvas

– key traits of canvases, Using a canvas-Using a canvas

– moving output to canvases, Moving an output to the canvas-Moving an output to the canvas

adaptive planning, Conditional Planning and Branching

adjustable outputs, Design Principles of Outputs, Designing for Adjustability-Designing for Adjustability

Adobe Generative Fill, Direct Manipulation

Aether, Designing for Verifiability

agentic design patterns, Agentic AI: Designing for Systems That Plan, Act, and Adapt-Summary

– defined, Agentic AI: Designing for Systems That Plan, Act, and Adapt

– Multiagent Collaboration pattern, Understanding Agentic Design Patterns, The Multiagent Collaboration Pattern: Specialized Coordination-Conflict Resolution in Multiagent Systems

– Planning pattern, Understanding Agentic Design Patterns, The Planning Pattern: Breaking Down Complexity-Conditional Planning and Branching

– principles for, New Principles for Agentic Interface Design-Design for Shared Control

– prioritization, Prioritize What Matters Most-Prioritize What Matters Most

– ReAct pattern, Understanding Agentic Design Patterns, The ReAct Pattern: Adaptive Problem Solving-Designing for Iterative Agents

– Reflection pattern, Understanding Agentic Design Patterns-Reflection Across Different Domains

– revealing the plan, Reveal the Plan-Reveal the Plan

– shared control, Design for Shared Control-Design for Shared Control

– Tool Use pattern, Understanding Agentic Design Patterns, The Tool Use Pattern: From Adviser to Operator-Design Implications of Tool Use

– user experience, Moving from Engineering to Experience-Sources

agentic workflows, Computation: Designing for the Processing and Generation Phase

agents

– chatbots, Models, Agents, and Tool Selection

– design considerations for, Design considerations for agents

– grounding, Designing Grounding

– role of, Models, Agents, and Tool Selection

– selecting, Orchestration, Models, Agents, and Tool Selection-Models, Agents, and Tool Selection

– user prompt handling, Models, Agents, and Tool Selection

AGI (artificial general intelligence), Early Predictive Language Models for Consumers

AI (artificial intelligence)

– agentic, Agentic AI: Designing for Systems That Plan, Act, and Adapt-Summary

– AI winter, The Beginning: Symbols

– capability, Capabilities: What Can the Model Do?-Evals

– challenges for UX design, Preface, User Experience in the AI Era-User Experience in the AI Era

– communication practices, Communication: Knowledge Sharing and Role Clarity-Level 3: Integrated AI organization

– designing for inputs, Designing for AI Inputs-Summary

– discovery, Discovery-Adoption and Momentum Behavior

– error design, When the System Doesn’t Deliver: Designing for Errors-Maintaining Momentum

– ethical considerations, Ethical Considerations in AI Design-Environmental impacts

– history of, A Short History of AI-The Beginning: Symbols

– knowing your user, Know Your User-Know Your User

– orchestration, Orchestration-Library Management and Assets

– overreliance, Designing for Verifiability

– technical and evaluatory skills, Technical and Evaluatory Skills-Technical and Evaluatory Skills

– user intent, Discovery-Intent with AI

– users' assumptions and expectations, A Short History of AI, The Beginning: Symbols-User Experience in the AI Era

– worldwide spending on, User Experience in the AI Era

AlexNet, The Beginning: Symbols

algorithmic transference, Adoption and Momentum Behavior

AllTrails app, This summary is AI-generated

Altman, Sam, Design considerations for models

ambient feedback, Prompt length and the precision paradox

ambient/proactive discovery, Ambient/proactive discovery

anomaly detection, Ambient/proactive discovery

Anthropic, The Beginning: Symbols, Adoption and Momentum Behavior, Design considerations for agents, Managing Problematic Outputs-Managing Problematic Outputs

AOL Instant Messenger, The Beginning: Symbols

Apollo 11, Computation: Designing for the Processing and Generation Phase

Apple Notes app, Direct Manipulation

articulating wait reasons, Visual Methods for Mitigating Latency

artificial general intelligence (AGI), Early Predictive Language Models for Consumers

artificial intelligence (see AI)

asynchronous (nonblocking) interactions, Visual Methods for Mitigating Latency

"Attention Is All You Need" (Google), The Beginning: Symbols

audio inputs, Audio and video

automatic routing, Design considerations for models

automation bias, Designing for Verifiability

autonomous (self-driving) cars, Communication: Knowledge Sharing and Role Clarity

autosuggest features, Information retrieval (medium/slow)

B

behavioral pattern recognition, Context-aware discovery

Bender, Emily M., The Beginning: Symbols, Designing for Verifiability

BERT, The Beginning: Symbols

Bing, Position: Where the Action Is At

boundary-testing prompts, Prompt Intent

Braun, Christian, Allowing user formatting

Brin, Sergey, From Multics to Google Search: The Shifting Burden

Browser agents, Alternative Progress Patterns: Browser and Computer Use Visualization-Alternative Progress Patterns: Browser and Computer Use Visualization

Buschek, Daniel, Direct Manipulation

C

C2PA standard, Image and Video Watermarking

Canva Magic Edit, Direct Manipulation

canvases

– defined, Using a canvas

– key traits of, Using a canvas-Using a canvas

– moving output to, Moving an output to the canvas-Moving an output to the canvas

capability, Capabilities: What Can the Model Do?-Evals

– defined, Capability, Discovery, and Orchestration

– evaluations, Evals-Evals

– knowledge barriers, Capabilities: What Can the Model Do?

– plastic state, Adoption and Momentum Behavior

– training, Training-Training

CARE framework, The CARE Framework-The CARE Framework

– action, The CARE Framework

– context, The CARE Framework

– examples, The CARE Framework-The CARE Framework

– results, The CARE Framework-The CARE Framework

Carnegie Mellon University, Capabilities: What Can the Model Do?

Castel, P. Kevin, Designing for Verifiability

CDSS (clinical decision support system), Output: Designing the Delivery and Presentation of LLM Responses-Designing Outputs: The Output Is Not the Answer

ChatGPT

– categories of use, Know Your User

– Computer Use agent, Alternative Progress Patterns: Browser and Computer Use Visualization

– escape hatch, Design considerations for tools

– full record of work, Tier 3: The full record of work

– legal brief draft example, Designing for Verifiability

– map options, Designing for Actionability

– metering limits, Usage and Metering

– model selection, Design considerations for models

– stored metadata, Image and Video Watermarking

– wedding planner example, Designing Grounding

checkpoints, Midstream Errors and Retrieval, Checkpoints-Sources

– edits and errors, Edits and errors

– intermediate outputs, Intermediate outputs

– permissions, Permissions

– rollbacks, Rollback

– sources, Sources

Chomsky, Noam, The Beginning: Symbols

Claude, The Beginning: Symbols, Design considerations for agents, Explicit Prompting, Allowing user formatting, Moving an output to the canvas, Managing Problematic Outputs

Claude Code, Error Messages, Design Implications of Tool Use, Clarification, Design for Shared Control

clear outputs, Design Principles of Outputs-Balancing structure and flexibility

– balancing structure and flexibility, Balancing structure and flexibility

– example of, Before and after: A practical example-Before and after: A practical example

– extending clarity across formats, Extending clarity across formats

– structure, Structuring for clarity: From raw to refined-Structuring for clarity: From raw to refined

CLI (command line interface), Designing for AI Inputs-From Multics to Google Search: The Shifting Burden

clinical decision support system (CDSS), Output: Designing the Delivery and Presentation of LLM Responses-Designing Outputs: The Output Is Not the Answer

“Clinical Decision Support Systems” (Musin et al.), Output: Designing the Delivery and Presentation of LLM Responses

Clippy, Agentic AI: Designing for Systems That Plan, Act, and Adapt-Agentic AI: Designing for Systems That Plan, Act, and Adapt

Clutch, Adoption and Momentum Behavior

CNNs (convolutional neural networks), Images

command line interface (CLI), Designing for AI Inputs-From Multics to Google Search: The Shifting Burden

Command Prompt, Designing for AI Inputs

command shells

– defined, Designing for AI Inputs

– nascent conversational nature, Designing for AI Inputs

command triggers, Input-based discovery

Common Crawl project, The Beginning: Symbols

communication practices, Communication: Knowledge Sharing and Role Clarity-Level 3: Integrated AI organization

– individual exploration, Level 1: Individual exploration

– integrated AI organization, Level 3: Integrated AI organization

– shared understanding with manual maintenance, Level 2: Shared understanding with manual maintenance

complete knowledge, assumption of, User Experience in the AI Era

computation, Computation: Designing for the Processing and Generation Phase-Summary

– agentic patterns, Where Users Experience Agentic Patterns-Sources

– checkpoints, Checkpoints-Sources

– clarification, Clarification-Clarification

– defined, Computation

– error design, When the System Doesn’t Deliver: Designing for Errors-Maintaining Momentum

– generation and inference, Step 3: Generation and Inference-Step 3: Generation and Inference

– input processing and preparation, The Computation Pipeline: From Input to Output-Audio and video

– latency and delays, Don’t Make Me Wait: Managing Latency and Delays-Latency for autonomous tasks

– pipeline for, The Computation Pipeline: From Input to Output-Step 3: Generation and Inference

– processing, Processing-Planning with tasks

– progress communication, Notify Me: Notifications and Status Indicators-Tier 3: The full record of work, Progress Communication-Alternative Progress Patterns: Browser and Computer Use Visualization

– routing, Step 2: Routing-Step 2: Routing

– segmentation, The Value of Segmenting by User Need-Latency for autonomous tasks

– speed, Speed: Difficult to Predict

– terminology, Computation: Designing for the Processing and Generation Phase-Computation: Designing for the Processing and Generation Phase

Computer Use agents, Alternative Progress Patterns: Browser and Computer Use Visualization-Alternative Progress Patterns: Browser and Computer Use Visualization

“Computing Machinery and Intelligence” (Turing), The Beginning: Symbols

conditional planning, Conditional Planning and Branching

confidence indicators, Designing for Verifiability-Designing for Verifiability

confirmation bias, Designing for Verifiability

Constitutional AI, Managing Problematic Outputs

content analysis, Context-aware discovery

context assembly, Audio and video

context windows, Prompt length and the precision paradox, Computation: Designing for the Processing and Generation Phase, Step 3: Generation and Inference, Designing for Adjustability

context-aware discovery, Context-aware discovery

contextual tutorials, Progressive discovery

conversational agents, Conversational agents

convolutional neural networks (CNNs), Images

Cooper, Alan, When the System Doesn’t Deliver: Designing for Errors

Copilot, GitHub, Implicit Context

Copilot, Microsoft, Position: Where the Action Is At

corrective prompts, Prompt Intent

Cortana, Agentic AI: Designing for Systems That Plan, Act, and Adapt

creative content generation, Creative content generation

D

DALL-E, Extending clarity across formats, Image and Video Watermarking

decomposition, Decomposition

DeepMind, Ethical Considerations in AI Design, Detection Systems and the Future of Watermarking

DeepMind Sparrow, Managing Problematic Outputs

delegation, Delegation

detokenization, Designing Outputs: The Output Is Not the Answer

direct manipulation, Designing for AI Inputs, Direct Manipulation-Direct Manipulation

disclosure requirements, This summary is AI-generated-This summary is AI-generated

“Discovering Language Model Behaviors with Model-Written Evaluations” (Perez et al.), Sycophancy and tone-based bias

discovery, Discovery-Adoption and Momentum Behavior

– adoption, Adoption and Momentum Behavior-Adoption and Momentum Behavior

– ambient/proactive, Ambient/proactive discovery

– context-aware, Context-aware discovery

– defined, Capability, Discovery, and Orchestration

– input-based, Input-based discovery-Input-based discovery

– matrix for, Discoverability and Initialization, Progressive discovery

– momentum behavior, Adoption and Momentum Behavior

– negative experiences with, Discoverability and Initialization

– progressive, Progressive discovery-Progressive discovery

– user intent, Discovery-Intent with AI

drop-down menus, Design considerations for models

E

ELIZA, The Beginning: Symbols-The Beginning: Symbols

empty state activation, Input-based discovery

environmental issues, Environmental impacts

error design, When the System Doesn’t Deliver: Designing for Errors-Maintaining Momentum

– error messages, When the System Doesn’t Deliver: Designing for Errors-Error Messages

– foundational tenets of, When the System Doesn’t Deliver: Designing for Errors

– maintaining momentum, Maintaining Momentum-Maintaining Momentum

– midstream errors and retrieval, Midstream Errors and Retrieval

ethics, Ethical Considerations in AI Design-Environmental impacts

EU AI Act, This summary is AI-generated

evals (evaluations), Technical and Evaluatory Skills-Technical and Evaluatory Skills, Evals-Evals

example galleries, Guiding Users Toward Better Inputs

explicit intent, Intent with AI, Designing for AI Inputs, Explicit Prompting

explicit prompting, Designing for AI Inputs, Explicit Prompting

– CARE framework, The CARE Framework-The CARE Framework

– risks in, Risks in Explicit Prompting-Prompt length and the precision paradox

exploratory prompts, Prompt Intent

F

Facebook, The Beginning: Symbols, Usage and Metering

factuality, assumption of, User Experience in the AI Era

feature graduation, Progressive discovery

fine-tuning, Training

Ford, Henry, Preface

formatting, Allowing user formatting-Allowing user formatting, Structuring for clarity: From raw to refined

G

Gebru, Timnit, The Beginning: Symbols

Gemini, The Beginning: Symbols

Generative Fill, Direct Manipulation

Genius, How Watermarking Works

geographic grounding, Designing Grounding

gesture patterns, Input-based discovery

Getty Images, Image and Video Watermarking

GitHub Copilot, Implicit Context

Gmail, Designing for Actionability

Google

– “Attention Is All You Need”, The Beginning: Symbols

– Genius lyrics website and scraping, How Watermarking Works

– inviting interface style, From Multics to Google Search: The Shifting Burden

– loosely structured queries, Designing for AI Inputs, From Multics to Google Search: The Shifting Burden-From Multics to Google Search: The Shifting Burden

– most common types of search queries, Know Your User

– predictive surfacing, Ambient/proactive discovery

– product consistency, Guiding Users Toward Better Inputs

– relevance through link analysis, From Multics to Google Search: The Shifting Burden

Google BERT, The Beginning: Symbols

Google Books, The Beginning: Symbols

Google DeepMind, Detection Systems and the Future of Watermarking

Google Gemini, The Beginning: Symbols

Google Lens, Image

Google People + AI Research group (PAIR), This summary is AI-generated

Google Photos Magic Eraser, Direct Manipulation

Google TensorFlow, Preface, A Short History of AI

Google Wave, Capability, Discovery, and Orchestration-Capability, Discovery, and Orchestration

GPT, The Beginning: Symbols, Explicit Prompting

– GPT-2, The Beginning: Symbols

– GPT-3, The Beginning: Symbols

– GPT-4, The Beginning: Symbols

– GPT-4.1, Allowing user formatting

GPUs, The Beginning: Symbols

grammar checkers, Direct Manipulation

Green Software Foundation, Environmental impacts

grounded outputs, Design Principles of Outputs, Designing Grounding-This summary is AI-generated

GSM8K, Evals

H

hallucinations, User Experience in the AI Era, Environmental impacts, Step 3: Generation and Inference, Maintaining Momentum, Balancing structure and flexibility-Designing for Verifiability

Harvard University, Know Your User

Hick's Law, Adoption and Momentum Behavior

"The Hidden Structure" (Braun et al.), Allowing user formatting

hierarchical planning, Conditional Planning and Branching

HITL (human-in-the-loop), Designing for Verifiability

"How People Are Using ChatGPT" (NBER), Know Your User

human-in-the-loop (HITL), Designing for Verifiability

human-like understanding, projecting, User Experience in the AI Era

I

image generators, Usage and Metering, Step 3: Generation and Inference, Designing Outputs: The Output Is Not the Answer

image inputs, Image, Images

image watermarking, Image and Video Watermarking

implicit context, Designing for AI Inputs, Implicit Context

implicit intent, Intent with AI, Designing for AI Inputs, Implicit Context

inference

- computation pipeline, The Computation Pipeline: From Input to Output, Step 3: Generation and Inference-Step 3: Generation and Inference

– defined, Computation: Designing for the Processing and Generation Phase, Computation: Designing for the Processing and Generation Phase

information retrieval, Design considerations for tools, Information retrieval (medium/slow)

input-based discovery, Input-based discovery-Input-based discovery

input-computation-output framework, Reading This Book: The Input-Computation-Output Structure-Outputs

– computation, Computation, Computation: Designing for the Processing and Generation Phase-Summary

– inputs, Inputs, Designing for AI Inputs-Summary

– outputs, Outputs, Output: Designing the Delivery and Presentation of LLM Responses-Summary

inputs, Designing for AI Inputs-Summary

– audio, Audio and video

– CARE framework, The CARE Framework-The CARE Framework

– channels of intent, Three Channels of Intent-Direct Manipulation

– command-line interface, Designing for AI Inputs-From Multics to Google Search: The Shifting Burden

– defined, Inputs

– guiding users to improve, Guiding Users Toward Better Inputs-Starter Prompts as Product Positioning

– image, Image, Images

– loosely structured queries, From Multics to Google Search: The Shifting Burden-From Multics to Google Search: The Shifting Burden

– modalities, Input Modalities-Voice
– origins of, Designing for AI Inputs
– processing and preparation, The Computation Pipeline: From Input to Output-Audio and video
– prompt types, Prompt Intent
– text, Text-Structured prompts, Step 1: Input Processing and Preparation-Step 1: Input Processing and Preparation
– video, Audio and video
– voice, Voice-Voice

instructive prompts, Prompt Intent

interaction history, Designing Grounding

interaction patterns, Input-based discovery

intermediate outputs, Intermediate outputs

interruptibility, Design for Shared Control

J

Jakob’s Law, Direct Manipulation

K

Karpathy, Andrej, Evals

knowing your user, Know Your User-Know Your User

knowledge and information sharing, Communication: Knowledge Sharing and Role Clarity-Level 3: Integrated AI organization

Krizhevsky, Alex, The Beginning: Symbols

L

large language models (see AI; LLMs; UX)

latency and delays, Don’t Make Me Wait: Managing Latency and Delays-Latency for autonomous tasks

– autonomous tasks, Latency for autonomous tasks-Latency for autonomous tasks

– core priorities when designing for, Don’t Make Me Wait: Managing Latency and Delays

– positioning of indicators, Position: Where the Action Is At-Position: Where the Action Is At

– progress communication, Notify Me: Notifications and Status Indicators-Tier 3: The full record of work

– Reflection pattern, Design Challenges with Reflection

– segmentation, The Value of Segmenting by User Need-Latency for autonomous tasks

– speed, Speed: Difficult to Predict

– thresholds, Don’t Make Me Wait: Managing Latency and Delays

– visual mitigation methods, Visual Methods for Mitigating Latency-Visual Methods for Mitigating Latency

latent space, Computation: Designing for the Processing and Generation Phase

Lehmann, Florian, Direct Manipulation

Lens, Image

Lexis+ AI, Designing Grounding

library management, Orchestration, Library Management and Assets

Lilienbeck, Alexander, Allowing user formatting

Liu, Nelson F., Prompt length and the precision paradox

LLMs (large language models), Early Predictive Language Models for Consumers

– (see also AI; UX)

– chain-of-thought reasoning, Step 3: Generation and Inference

– confidence, Designing for Verifiability

– early predictive text systems, Early Predictive Language Models for Consumers

– effective prompting, From Multics to Google Search: The Shifting Burden

– inferring intent and generating responses, Designing for AI Inputs

– outputs, Designing Outputs: The Output Is Not the Answer

– predicting next token, Explicit Prompting

– reflecting patterns and logic in human language, Early Predictive Language Models for Consumers

– sycophancy, Sycophancy and tone-based bias

– tokenization, Step 1: Input Processing and Preparation

– vibe coding, Evals

loading indicators, Visual Methods for Mitigating Latency, Transactional processes

locale grounding, Designing Grounding

location-based contextual help, Context-aware discovery

“Lost in the Middle” (Liu et al.), Prompt length and the precision paradox

Lovable, Position: Where the Action Is At

M

Magic Edit, Direct Manipulation

Maslow’s hierarchy of needs, Models, Agents, and Tool Selection

Massive Multitask Language Understanding (MMLU), Evals

McMillan-Major, Angelina, The Beginning: Symbols

MCP (Model Context Protocol), Design considerations for tools

Mentjukov, Daniel, Allowing user formatting

metering, Orchestration, Usage and Metering-Usage and Metering

Michalko, Michael, Preface

Microsoft Aether, Designing for Verifiability

Microsoft Bob, Agentic AI: Designing for Systems That Plan, Act, and Adapt

Microsoft Clippy, Agentic AI: Designing for Systems That Plan, Act, and Adapt-Agentic AI: Designing for Systems That Plan, Act, and Adapt

Microsoft Copilot, Position: Where the Action Is At

Miller’s Law, The Value of Segmenting by User Need

MIT, The Beginning: Symbols, Designing for AI Inputs

Mitchell, Margaret, The Beginning: Symbols

MMLU (Massive Multitask Language Understanding), Evals

mode grounding, Designing Grounding

Model Context Protocol (MCP), Design considerations for tools

model switching, Computation: Designing for the Processing and Generation Phase

Model Test Bench (MT Bench), Evals

models

– design considerations for, Design considerations for models-Design considerations for models

– grounding, Designing Grounding

– multimodal, Computation: Designing for the Processing and Generation Phase

– role of, Models, Agents, and Tool Selection

– selecting, Orchestration, Models, Agents, and Tool Selection-Models, Agents, and Tool Selection

– user prompt handling, Models, Agents, and Tool Selection

momentum behavior, Adoption and Momentum Behavior

MT Bench (Model Test Bench), Evals

Multiagent Collaboration pattern, Understanding Agentic Design Patterns, The Multiagent Collaboration Pattern: Specialized Coordination-Conflict Resolution in Multiagent Systems

– conflict resolution, Conflict Resolution in Multiagent Systems

– interface challenges, Multiagent Interface Challenges-Multiagent Interface Challenges

multiagent reflection, How Reflection Works in Practice

Multics (Multiplexed Information and Computing Service), Designing for AI Inputs-Designing for AI Inputs

multimodal models, Computation: Designing for the Processing and Generation Phase

multiturn outputs, Designing for Multiturn Outputs-Designing for Multiturn Outputs

Musen, Mark Alan, Output: Designing the Delivery and Presentation of LLM Responses

MYCIN, Output: Designing the Delivery and Presentation of LLM Responses-Output: Designing the Delivery and Presentation of LLM Responses

N

natural language processing (NLP), A Short History of AI-A Short History of AI

Nature, Detection Systems and the Future of Watermarking

NBER (National Bureau of Economic Research), Know Your User

Nielsen Norman Group, Adoption and Momentum Behavior, The CARE Framework, Designing for Clarity

Nielsen, Jakob, Don’t Make Me Wait: Managing Latency and Delays

NLP (natural language processing), A Short History of AI-A Short History of AI

nonblocking (asynchronous) interactions, Visual Methods for Mitigating Latency

Notes app, Direct Manipulation

notifications

- – error messages, Error Messages

– progress communication, Notifications-Notifications, Progress Communication

Notion, Designing for Adjustability

NVIDIA, The Beginning: Symbols

O

“On the Dangers of Stochastic Parrots” (Bender et al.), Designing for Verifiability

OpenAI, The Beginning: Symbols, Know Your User, Design considerations for models, Computation: Designing for the Processing and Generation Phase, Detection Systems and the Future of Watermarking

orchestration, Orchestration-Library Management and Assets

– defined, Capability, Discovery, and Orchestration

– elements for shaping query and experience, Orchestration-Orchestration

– library management, Orchestration, Library Management and Assets

– model, tool, and agent selection, Orchestration, Models, Agents, and Tool Selection-Design considerations for agents

– permissioning, Orchestration, Permissions and Capabilities

– principles of, Orchestration

– usage and metering, Orchestration, Usage and Metering-Usage and Metering

orchestrators, The Multiagent Collaboration Pattern: Specialized Coordination

ordering effects, Designing for Verifiability

Otter AI, Voice

outputs, Output: Designing the Delivery and Presentation of LLM Responses-Summary

– actionable, Design Principles of Outputs, Designing for Actionability-Moving an output to the canvas

– adjustable, Design Principles of Outputs, Designing for Adjustability-Designing for Adjustability

– clarity, Design Principles of Outputs-Balancing structure and flexibility

– defined, Outputs

– design principles for, Design Principles of Outputs-Designing for Multiturn Outputs

– as designed artifacts, Designing Outputs: The Output Is Not the Answer-Designing Outputs: The Output Is Not the Answer

– grounded, Design Principles of Outputs, Designing Grounding-This summary is AI-generated

– multiturn, Designing for Multiturn Outputs-Designing for Multiturn Outputs

– problematic, Managing Problematic Outputs-Managing Problematic Outputs

– subphases, Designing Outputs: The Output Is Not the Answer

– verifiable, Design Principles of Outputs, Designing for Verifiability-Designing for Verifiability

– watermarking and detection, Watermarking and Detection-Detection Systems and the Future of Watermarking

“Overreliance on AI” (Aether), Designing for Verifiability

P

Page, Larry, From Multics to Google Search: The Shifting Burden

PageRank, From Multics to Google Search: The Shifting Burden

PAI (Partnership on AI), Social and interpersonal risks

PAIR (Google People + AI Research group), This summary is AI-generated

palettes, Design considerations for tools

parallel collaboration, The Multiagent Collaboration Pattern: Specialized Coordination

Perez, Ethan, Sycophancy and tone-based bias

permissioning, Orchestration, Permissions and Capabilities, Design Implications of Tool Use

Perplexity AI, Designing for Verifiability

personas, Know Your User-Know Your User

Phoenix Checklist, Preface

Photos Magic Eraser, Direct Manipulation

Planning pattern, Understanding Agentic Design Patterns, The Planning Pattern: Breaking Down Complexity-Conditional Planning and Branching

- conditional planning and branching, Conditional Planning and Branching
- designing for visible planning, Designing for Visible Planning

PoseNet, A Short History of AI

precision paradox, Prompt length and the precision paradox-Prompt length and the precision paradox

predictive search, Information retrieval (medium/slow)

predictive surfacing, Ambient/proactive discovery

predictive text systems, Understanding Large Language Models and Systems-Early Predictive Language Models for Consumers

preservability, Moving an output to the canvas

presets, Guiding Users Toward Better Inputs

pretraining, Training

progress communication, Notify Me: Notifications and Status Indicators-Tier 3: The full record of work, Progress Communication-Alternative Progress Patterns: Browser and Computer Use Visualization

- browser and computer use visualizations, Alternative Progress Patterns: Browser and Computer Use Visualization-Alternative Progress Patterns: Browser and Computer Use Visualization
- detail, Tier 2: Detail, Progress Communication
- full record of work, Tier 3: The full record of work-Tier 3: The full record of work, Progress Communication
- notifications, Notifications-Notifications, Progress Communication
- overview, Tier 1: Overview, Progress Communication

progressive disclosure, Guiding Users Toward Better Inputs, Visual Methods for Mitigating Latency, Information retrieval (medium/slow)

progressive discovery, Progressive discovery-Progressive discovery

prompt augmentation, Designing for Adjustability

prompt libraries, Guiding Users Toward Better Inputs

prompting

– CARE framework, The CARE Framework-The CARE Framework

– defined, Explicit Prompting

– explicit, Designing for AI Inputs, Explicit Prompting, Risks in Explicit Prompting-Prompt length and the precision paradox

– guiding users to improve, Guiding Users Toward Better Inputs-Starter Prompts as Product Positioning

– length and precision, Prompt length and the precision paradox-Prompt length and the precision paradox

– starter prompts, Starter Prompts as Product Positioning

– structured, Structured prompts-Structured prompts

– types of, Prompt Intent

Q

Quartz, Discoverability and Initialization

QuillBot, Structured prompts

R

RAG (retrieval augmented generation), Computation: Designing for the Processing and Generation Phase

“Re-examining Whether, Why, and How Human-AI Interaction Is Uniquely Difficult to Design” (Carnegie Mellon), Capabilities: What Can the Model Do?

ReAct (Reason and Act) pattern, Understanding Agentic Design Patterns, The ReAct Pattern: Adaptive Problem Solving-Designing for Iterative Agents

– designing for iterative agents, Designing for Iterative Agents-Designing for Iterative Agents

– implementation and use cases, Implementation and Use Cases-Implementation and Use Cases

real-time control systems, Real-time control systems (instant)-Real-time control systems (instant)

real-time streaming of partial results, Visual Methods for Mitigating Latency

recurrent outputs, Designing for Multiturn Outputs

red teaming, Managing Problematic Outputs

Reflection pattern, Understanding Agentic Design Patterns-Reflection Across Different Domains

– across domains, Reflection Across Different Domains

– design challenges, Design Challenges with Reflection

– functionality of, How Reflection Works in Practice

– generate-reflect-iterate loop, The Reflection Pattern: Building in Self-Critique

reinforcement learning from human feedback (RLHF), Training

Replit

– checkpoints, Midstream Errors and Retrieval

– “In progress tasks” component, Planning with tasks

– latency, Speed: Difficult to Predict

– minimal status surface, Tier 1: Overview

– optional planning stage, Clarification

– revealing the plan, Reveal the Plan

resumability, Usage and Metering

retrieval augmented generation (RAG), Computation: Designing for the Processing and Generation Phase

RLHF (reinforcement learning from human feedback), Training

role clarity, Communication: Knowledge Sharing and Role Clarity-Level 3: Integrated AI organization

role-based prompts, Prompt Intent

rollbacks, Rollback

routing

– computation pipeline, The Computation Pipeline: From Input to Output, Step 2: Routing-Step 2: Routing

– defined, Computation: Designing for the Processing and Generation Phase

Ruan, Sherry, Voice

Runway, Extending clarity across formats, Moving an output to the canvas

S

SAE (Society of Automotive Engineers), Communication: Knowledge Sharing and Role Clarity

safety (system) cards, Managing Problematic Outputs

safety and security risks, Safety and security risks

scaffolding, Guiding Users Toward Better Inputs-Guiding Users Toward Better Inputs

Schwartz, Steven, Designing for Verifiability

Scientific American, Communication: Knowledge Sharing and Role Clarity

segmentation, The Value of Segmenting by User Need-Latency for autonomous tasks

– autonomous tasks, Latency for autonomous tasks-Latency for autonomous tasks

– conversational agents, Conversational agents

– creative content generation, Creative content generation

– information retrieval, Information retrieval (medium/slow)

– real-time control systems, Real-time control systems (instant)-Real-time control systems (instant)

– transactional processes, Transactional processes

self-driving (autonomous) cars, Communication: Knowledge Sharing and Role Clarity

self-reflection, How Reflection Works in Practice

sequencing, Sequencing

sequential handoff, The Multiagent Collaboration Pattern: Specialized Coordination

session context, Designing Grounding

Shahar, Yuval, Output: Designing the Delivery and Presentation of LLM Responses

shared control, Design for Shared Control-Design for Shared Control

Sharma, Mrinank, Sycophancy and tone-based bias

Shortliffe, Edward H., Output: Designing the Delivery and Presentation of LLM Responses

skeleton screens, Visual Methods for Mitigating Latency, Real-time control systems (instant)

skill atrophy, Designing for Verifiability

SmarterChild, The Beginning: Symbols

social and interpersonal risks, Social and interpersonal risks

Society of Automotive Engineers (SAE), Communication: Knowledge Sharing and Role Clarity

Sora, Extending clarity across formats

sparkle emoji, Discoverability and Initialization

Sparrow, Managing Problematic Outputs

spectral representations, Audio and video

spell-checkers, Direct Manipulation, Designing for Verifiability

spinners, Visual Methods for Mitigating Latency, Transactional processes, Creative content generation

Stanford Human-Centered AI, Safety and security risks

Stanford University, Voice, Agentic AI: Designing for Systems That Plan, Act, and Adapt

starter prompts, Starter Prompts as Product Positioning

Startographer, A Short History of AI

structured prompting, Structured prompts-Structured prompts

subword tokenization, Step 1: Input Processing and Preparation

supervised collaboration, The Multiagent Collaboration Pattern: Specialized Coordination

Swype, Early Predictive Language Models for Consumers

sycophancy, Sycophancy and tone-based bias-Sycophancy and tone-based bias

SynthID, Detection Systems and the Future of Watermarking-Detection Systems and the Future of Watermarking

system (safety) cards, Managing Problematic Outputs

T

T9, Understanding Large Language Models and Systems-Early Predictive Language Models for Consumers

task grounding, Designing Grounding

tasks, planning with, Planning with tasks

technical and evaluatory skills, Technical and Evaluatory Skills-Technical and Evaluatory Skills

Tegic Communications, Understanding Large Language Models and Systems

temperature, Step 3: Generation and Inference, Designing for Adjustability

templates, Guiding Users Toward Better Inputs-Guiding Users Toward Better Inputs, Structuring for clarity: From raw to refined

TensorFlow, Preface, A Short History of AI

Terminal, Designing for AI Inputs-Designing for AI Inputs

text inputs, Text-Structured prompts

- allowing user formatting, Allowing user formatting-Allowing user formatting
- structured prompting, Structured prompts-Structured prompts

– tokenization, Step 1: Input Processing and Preparation-Step 1: Input Processing and Preparation

The Wall Street Journal, Discoverability and Initialization

Thinkertoys (Michalko), Preface

tokenization

– defined, Computation: Designing for the Processing and Generation Phase

– detokenization, Designing Outputs: The Output Is Not the Answer

– input processing and preparation, Step 1: Input Processing and Preparation-Step 1: Input Processing and Preparation

– subword, Step 1: Input Processing and Preparation

– temperature, Step 3: Generation and Inference

– token IDs, Step 1: Input Processing and Preparation

Tokenizer, Computation: Designing for the Processing and Generation Phase

tone specification, Structuring for clarity: From raw to refined

tone-based bias, Sycophancy and tone-based bias

Tool Use pattern, Understanding Agentic Design Patterns, The Tool Use Pattern: From Adviser to Operator-Design Implications of Tool Use

– components, The Shift from Static to Dynamic Assistance

– design implications, Design Implications of Tool Use-Design Implications of Tool Use

– dynamic assistance, The Shift from Static to Dynamic Assistance

tools

– design considerations for, Design considerations for tools-Design considerations for tools

– grounding, Designing Grounding

– palettes, Design considerations for tools

– role of, Models, Agents, and Tool Selection

– selecting, Orchestration, Models, Agents, and Tool Selection-Models, Agents, and Tool Selection

– Tool Use pattern, The Tool Use Pattern: From Adviser to Operator

– user prompt handling, Models, Agents, and Tool Selection

“Towards Understanding Sycophancy in Language Models” (Sharma et al.), Sycophancy and tone-based bias

training stage, Training-Training

transactional processes, Transactional processes

transformer architecture, The Beginning: Symbols

Turing test, The Beginning: Symbols

Turing, Alan, The Beginning: Symbols

1202 alarm, Computation: Designing for the Processing and Generation Phase

U

unbiased output, assumption of, User Experience in the AI Era

US Declaration of Independence, Detection Systems and the Future of Watermarking

US Federal Trade Commission, This summary is AI-generated

user experience (see UX)

user intent, Discovery-Intent with AI

– AI-first versus AI-enhanced tools, Intent with AI

– channels of, Three Channels of Intent-Direct Manipulation

– defined, Capability, Discovery, and Orchestration

– direct manipulation, Designing for AI Inputs, Direct Manipulation-Direct Manipulation

– explicit, Intent with AI, Designing for AI Inputs, Explicit Prompting

– gathering, Intent with AI

– implicit, Intent with AI, Designing for AI Inputs, Implicit Context

– importance in UX design, Discovery

– user-focused design thinking, Discovery

UX (user experience)

– agentic patterns, Moving from Engineering to Experience-Sources

– capability, Capabilities: What Can the Model Do?-Evals

– challenges for, in AI era, Preface, User Experience in the AI Era-User Experience in the AI Era

– communication practices, Communication: Knowledge Sharing and Role Clarity-Level 3: Integrated AI organization

– computation, Computation: Designing for the Processing and Generation Phase-Summary

– discovery, Discovery-Adoption and Momentum Behavior

– emergence of software design, Preface

– inputs, Designing for AI Inputs-Summary

– knowing your user, Know Your User-Know Your User

– orchestration, Orchestration-Library Management and Assets

– outputs, Output: Designing the Delivery and Presentation of LLM Responses-Summary

– technical and evaluatory skills, Technical and Evaluatory Skills-Technical and Evaluatory Skills

– user intent, Discovery-Intent with AI

V

verifiable outputs, Design Principles of Outputs, Designing for Verifiability-Designing for Verifiability

vibe coding, Evals

video generators, Designing Outputs: The Output Is Not the Answer

video inputs, Audio and video

video watermarking, Image and Video Watermarking

Visual Electric, Usage and Metering

voice inputs, Voice-Voice

W

watermarking, Watermarking and Detection-Detection Systems and the Future of Watermarking

– detection systems, Detection Systems and the Future of Watermarking-Detection Systems and the Future of Watermarking

– functionality of, How Watermarking Works

– future of, Detection Systems and the Future of Watermarking-Detection Systems and the Future of Watermarking

– images and videos, Image and Video Watermarking

– misuse, Understanding AI Misuse-Understanding AI Misuse

– purposes of, Watermarking and Detection

– risks of generative AI content, Risks of Generative AI Content-Risks of Generative AI Content

Watzlawick, Paul, Visual Methods for Mitigating Latency

Wave, Capability, Discovery, and Orchestration-Capability, Discovery, and Orchestration

Weizenbaum, Joseph, The Beginning: Symbols

Wikipedia, The Beginning: Symbols

X

X (formerly Twitter), The Beginning: Symbols

Z

Zhou, Yongchao, Prompt length and the precision paradox

OceanofPDF.com

About the Author

Louise Macfadyen is a product designer, writer, and creative technologist specializing in AI interface patterns. Over the past decade, she has worked with organizations including Google, Microsoft, Pinterest, Nike, Gap Sustainability, and the Environmental Working Group, and spoken internationally at events and institutions including Google I/O, Women Who Code, Rhizome, and the Museum of Contemporary Digital Art. She lives in New York City.

She began experimenting with code at age eleven, running Neopets fan-fiction forums with friends, and later earned a BA in literature from Queen Mary, University of London.

Today, Louise specializes in AI and emerging technology. At Microsoft, she led the creation of the company's first Agentic AI design system, shaping interaction patterns for autonomous systems across products. At Google, she led the Material Design 3 library, now used by over one million designers and developers worldwide.

Alongside her industry work, Louise explores how technology shapes culture through visualization, archiving, and writing, and is a cohost of *Away From Keyboard*, a podcast about the history of the internet.

OceanofPDF.com

Colophon

The animal on the cover of *Designing AI Interfaces* is an icterine warbler (*Hippolais icterina*). Found throughout Europe, these songbirds are known for their vibrant appearance and fast, melodic whistles.

Compared to other warblers, the icterine warbler is a medium-sized bird with a large head, a sharp, light orange bill, long wings, and a square tail. Their feathers are bright yellow on their underparts and green-olive on the upper parts of their bodies. On average, these birds measure 5 to 6 inches long and weigh between 0.4 and 0.6 ounces. Their colorful appearance makes them stand out among other warblers; they are also famous for being mimics, often imitating the songs of other species and combining them into their own loud, fast-paced whistles.

A migratory species, icterine warblers breed across Europe—from northern France and Scandinavia through western Siberia—and winter in sub-Saharan Africa. They prefer open woodlands and large gardens with tall trees and dense shrubbery. Their diet consists of flies, caterpillars, and beetles, though they will occasionally eat berries and fruit in late summer to prepare for their migration.

Fortunately, icterine warblers are currently classified as Least Concern on the IUCN Red List. However, they are sensitive to habitat loss and the impacts of climate change on their migratory routes. Many of the animals on O’Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Monica Kamsvaag, based on an antique line engraving from *Lydekker’s Royal Natural History*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag’s Ubuntu Mono.

OceanofPDF.com