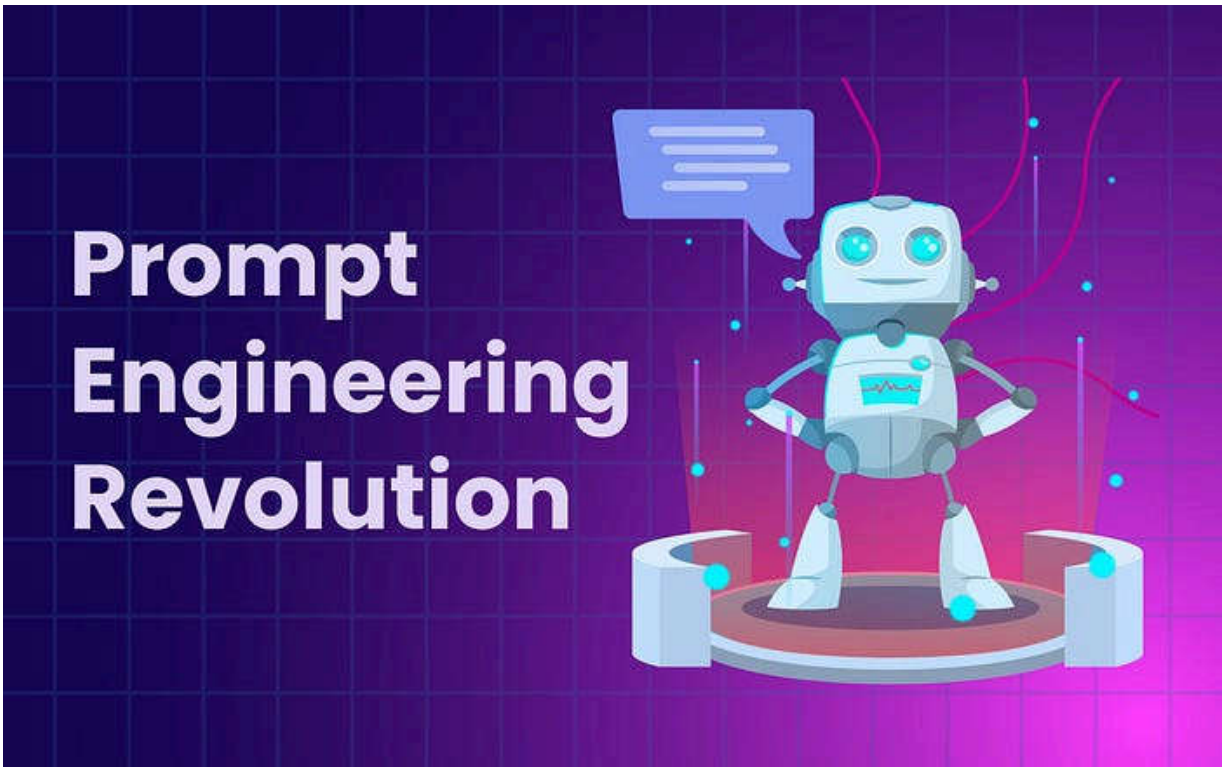
< Go to the original



Prompt Engineering Revolution: How Metaprompting and New AI Workflows Are Changing Startups

Explore how AI startups use prompt engineering & metaprompting to build smarter agents. Learn real strategies, tools, and best practices.

Arslan Ahmad

Follow

Artificial Intelligence in Plain English a11y-light ~22 min read ·
June 19, 2025 (Updated: June 19, 2025) · Free: Yes

an overnight AI prototype.

You would immediately ask how.

Not by months of coding, but by crafting an ingenious prompt for an AI model that addresses the client's exact problem.

This scenario isn't science fiction — it's happening now.

AI **prompt engineering** has rapidly evolved from a quirky idea into a secret sauce for the most innovative AI startups. Founders are closing seven-figure deals by showing up, writing some code *and prompts*, and building perfect demos at breakneck speed.

In other words, knowing *how to talk to AI* effectively — and even getting AI to help write its own instructions — is becoming just as important as writing the underlying software.

In this post, we'll understand the new art of prompt engineering and the rise of **"metaprompting."** We'll explore how AI startups are using these techniques to build powerful AI agents (like customer support bots that can handle thousands of tickets), how they structure prompts into layers, and the best practices (from *prompt folding* to *escape hatches*) that have emerged.

We'll also look at why tech founders are now acting as "forward deployed engineers" working directly with customers, and share the tools, tips, and mental models that modern AI developers swear by.

Prompt Engineering: The New Secret Sauce in AI Startups

So, **what is prompt engineering** anyway?

At its core, it's the craft of writing the instructions or questions (prompts) that we feed to large language models (like GPT-4 or ChatGPT) to get useful results.

It might sound simple — just ask the AI a question, right? — but doing it **well** is both an art and a science.

A poorly phrased prompt can give you nonsense, while a carefully engineered prompt can unlock amazingly detailed, accurate answers.

Prompt engineering is about finding that *just right* way to ask.

Not long ago, the term "prompt engineer" might have raised eyebrows as a gimmicky title. But fast forward to today: prompt engineering has *"rapidly evolved into the critical, often secret, weapon behind some of the most successful and innovative AI startups"*.

In AI startups, everyone from engineers to founders is spending hundreds of hours tweaking prompts, testing them, and optimizing them — because the quality of these prompts can make or break their product's performance.

In fact, one startup even open-sourced a **six-page prompt** that powers its AI support agent — essentially treating the prompt like code — because that prompt was so central to their solution.

Why the obsession?

Because in AI applications, **the prompt *is* the user interface** for controlling a model's behavior. It's like writing a spec for an AI helper: you have to clearly define the AI's role, its tone, the step-by-step approach to tasks, what to do and *not* do, and even how to format its answers.

The better you articulate all of that in the prompt, the better the AI performs.

The *really* good news?

Anyone can learn prompt engineering with practice and the right guidance — it's not some mystical gift reserved for AI PhDs. It just requires a shift in thinking: you're not writing traditional code, you're teaching or coaching an AI through language.

Meet Metaprompting: AI that Writes (and Improves) Its Own Instructions

One of the most exciting developments in this field is **metaprompting** — essentially using AI to help build better AI prompts.

Sounds wild, but here's how it works: you take a draft of your current prompt (maybe it's not getting the results you want), and you ask a more powerful AI model to **critique it or improve it.**

The AI might come back with suggestions: clearer wording here, an example there, a warning to avoid a certain pitfall, and so on.

Metaprompting is reshaping AI workflows because it adds a whole new feedback loop.

Instead of a human prompt engineer being solely responsible for refining the prompt, you have an AI pair-editor of sorts.

For example, you could feed your underperforming prompt along with some examples of the AI's mistakes into GPT-4 and say, *"You are an expert prompt engineer — how can I make this prompt better?"*

The large model will analyze it and propose improvements, effectively acting as a prompt consultant. This technique **"uses AI itself to write and refine your prompts"**, sparing developers countless hours of manual trial-and-error.

It's a fascinatingly *recursive* approach — the AI helping to program itself — and many top teams are using it to squeeze more performance out of their systems.

In fact, metaprompting is sometimes called **prompt folding** when used in iterative loops, because one prompt can fold into another; for instance, a high-level prompt might generate more specialized sub-prompts for the next stage of a task.

The result of metaprompting?

It taps into the AI's own knowledge of its strengths and weaknesses. After all, who better to tell you how to talk to an AI than the AI itself?

This doesn't mean the human is out of the loop — rather, the human becomes a kind of prompt *editor*, steering the process and vetting the AI's suggestions. It's a bit like pair programming, but your partner is an AI with encyclopedic knowledge.

For AI startups, this means they can iterate on prompt designs rapidly and even **standardize prompt quality** by letting the AI suggest the optimal wording or structure.

Real-World AI Agents: Customer Support and Beyond

These techniques aren't just academic — they're powering **real-world AI agents** that are already in production.

One notable example is AI-powered **customer support agents**.

Imagine a chatbot that can handle a flood of customer tickets for a software company, resolving issues end-to-end without human intervention in most cases.

Sounds like every support team's dream! One YC-backed startup did exactly this: they built an AI support agent whose prompt was so comprehensive (remember that six-page prompt?) that it could reliably handle a large share of customer inquiries.

The result is an agent that doesn't just spit out generic answers, but actually follows company-specific procedures and **resolves around 50–60% of support tickets automatically** (with the rest escalated to humans as needed).

The prompt was even open-sourced, showcasing how detailed these instructions get — it's essentially a **policy and workflow manual** written for the AI.

To give you a sense of how robust these systems are: that support agent's prompt literally acts like an internal manager overseeing the AI's actions.

It defines rules like, *"If the user says X and we have policy Y, do Z; if unsure, escalate to a human."* It even uses an **internal tagging system** (think <manager_verify>accept</manager_verify> vs <manager_feedback>reject</manager_feedback> tags) to decide whether the AI should proceed with an action or not.

In other words, the prompt doesn't just ask the AI to answer a question — it guides the AI through a decision tree of checks and balances before responding. This is prompt engineering at its finest: **programming the AI's behavior via natural language instructions**.

And customer support is just one use case. Similar **AI agents** are being deployed in other business areas as well:

- **Sales and Lead Qualification:** AI chatbots that engage incoming leads on a website, ask the right questions, and determine if

and budget (without feeling like a boring form) and then score the lead on a scale, e.g. using BANT criteria. Such an agent can then route hot leads to a human salesperson ASAP or nurture the cooler ones — all guided by a prompt that defines how to ask questions naturally and how to score the responses. The prompt essentially turns an LLM into a diligent **sales development rep** that never sleeps.

- **Technical Support and Troubleshooting:** Beyond basic customer service, some AI agents act as tier-1 tech support. They diagnose user problems, walk through solutions step-by-step, and know when to hand off to a human if it's too complex. A prompt for an **empathetic technical support agent** might include instructions like: *"You are a friendly senior support engineer. Always start by acknowledging the user's frustration. Then gather details (like software version, error logs), check these against known issues, and provide step-by-step fixes. If the issue involves security or is beyond your knowledge base, escalate it to level 2 support."* This kind of prompt ensures the AI is both **effective and comforting** to the user, following best practices that a top human support rep would. The AI can even confirm if a solution worked and only then conclude the chat. Users end up feeling heard and helped, often not realizing an AI handled their case.
- **Sales Objection Handling:** Some startups are arming their sales teams with AI assistants that can jump into emails or chats to handle customer objections in real-time. Imagine an AI that listens on a sales call or in a live chat and, when a customer says, "I'm not sure it's worth the cost," the AI (following its prompt playbook) instantly provides a data-backed response highlighting ROI, tailored to that industry — all while sounding like a helpful

like having the *ultimate sales playbook* distilled into an AI that is always by your side.

These examples show how **prompt engineering enables AI to perform specialized, high-value tasks.**

Each use case has a carefully crafted prompt or set of prompts functioning like an AI rulebook or script.

Notice a pattern?

The prompt always defines a **role** (e.g. lead qualifier, support engineer, sales rep persona), provides **context** or knowledge, sets **guidelines** (how to respond, when to escalate), and sometimes even dictates the **output format** (like a JSON with scores, or a step-by-step list). We're essentially *designing an AI agent's brain* through the prompt.

(Interested in trying something fun yourself? You can apply similar principles on a smaller scale — for example, using ChatGPT prompts to learn new skills. Check out the top 3 ChatGPT prompts to learn LeetCode patterns. You might be surprised how a well-engineered prompt can turn ChatGPT into a personal tutor for coding and system design!)

Building with Layers: The 3-Layer Prompt Architecture

As AI startups scaled their solutions to many customers and tasks, they discovered that using **one giant prompt for everything** wasn't efficient.

modular and scalable.

Here are the three layers:

Layer 1 — The System Prompt (AI's Operating System)

This is the foundational instruction set that applies *globally* to your AI agent.

Think of it as the company-wide policy or the AI's overall personality and ground rules.

The system prompt defines the AI's core identity (e.g. *"You are an expert customer service AI for XYZ Corp, friendly and professional..."*) and fundamental principles it must always follow.

It might include universal dos and don'ts, like **never violating company guidelines**, always maintaining a certain tone, and when to escalate to a human.

This layer is the same for every interaction, ensuring consistency in the AI's behavior across the board. In essence, it's the AI's built-in **constitution**.

Layer 2 — The Developer Prompt (Contextual Configurations)

On top of the system prompt comes a flexible layer that developers can change for specific contexts or customers.

For example, if your AI agent is serving different customers or different scenarios, the developer prompt will feed in that context: "This user is on *Premium Plan*, and their company is in healthcare, here are their past issues, and by the way, use a formal tone with them."

It might also contain *customer-specific rules*, like "If this particular client asks about billing above $500, escalate to their account manager John Doe."

Essentially, the developer prompt **tailors the AI's knowledge and behavior for a given client or use case**. It's usually prepended dynamically based on context.

This allows one AI system to serve many customers with each getting a personalized touch, without having to retrain the model for each case.

Layer 3 — The User Prompt (Live Input)

Finally, the top layer is the actual **user's query or message** — the real-time input from the end-user that kicks off the interaction.

This could be a customer asking, "Hey, I got error 503, what do I do?" or a lead saying, "I'm interested in your product, can you tell me more?" The user prompt gets combined with the other two layers underneath so that the AI model receives *one big assembled prompt* when it's time to respond. The user's question plus the developer

The user prompt is, of course, different every time — it's the only part provided by the user, while layers 1 and 2 come from the developers' side.

This layered approach is incredibly powerful. It means you don't have to stuff every possible rule and detail into one prompt (which could be inflexible and huge).

Instead, you have a **stable foundation (system level)** and a **configurable middle layer** for each scenario, and then the live input on top.

The AI effectively gets a structured packet of instructions: general rules + specific context + actual question.

According to experts, this architecture lets a single AI model serve many different needs with *tailored, context-aware behavior* for each, while still maintaining a consistent core voice and policy.

It's how AI startups can offer personalized solutions at scale — one "brain" serving many masters, by wearing different hats as needed.

Learn 5 prompts to practice system design questions.

Prompt Design Best Practices (From Prompt Folding to Escape Hatches)

Crafting great prompts is part science, part art.

Here are some of the most important ones to know:

- **Define Clear Roles and Personas:** Always start by telling the AI *who it is in this context*. Setting a clear role (e.g. *"You are a senior cloud support engineer"* or *"You are an empathetic career coach"*) provides immediate context and influences the tone and style of the AI's answers. This is often called **role-setting** or persona prompting. By giving the model a defined identity and perspective, you anchor its responses in the frame you want. It's amazing how differently an AI will answer the same question when it's instructed to act as a mathematician versus a marketer! So don't be shy about explicitly saying "You are **this**, and your goal is **that**."
- **Be Extremely Specific and Detailed:** A great prompt leaves little to guesswork. Outline the task clearly, step by step if needed, including any constraints or criteria. For complex tasks, literally break down what the AI should do first, second, and so on. Think of the AI as a new junior employee who needs thorough guidance. If there are certain formats or formulas to follow, spell them out. For instance, if you expect an answer in JSON or a list of bullet points, mention that format. The best AI prompts often read like a mini-spec or SOP (Standard Operating Procedure). This "manager approach" of over-communicating the details can significantly improve reliability and consistency of the outputs.
- **Use Prompt Folding (Multi-Step Prompts):** Some problems are too complex for a single prompt. **Prompt folding** is a technique where you break a task into multiple stages and have the AI generate intermediate prompts or plans for each stage. In practice, this might mean your initial prompt says, "First,

first output could be a new prompt tailored to that sub-task, which you then feed back in. This dynamic prompt generation makes your AI agent more adaptive, essentially **creating sub-prompts on the fly based on context.** It's like the AI is writing parts of its own game plan as it goes. This approach shines in complex workflows, ensuring the AI tackles one piece at a time with a specialized prompt for each piece.

- **Provide Reasoning Scaffolding (Think Step-by-Step):** A big challenge with AI models is getting them to "think through" a problem rather than jumping to a flawed answer. One trick is to build **reasoning scaffolds** into your prompt — basically, encourage the AI to show its work. You can explicitly instruct, *"List out the steps or reasoning before giving the final answer,"* or even request a brief *"thoughts"* section in the output. Some advanced prompts create a hidden or internal reasoning trace that isn't shown to the end-user but helps the AI stay on track. By asking the model to explain *why* it's doing something (even if just to itself), you reduce magical guessing and improve accuracy. For example, prompt: *"Take a minute to consider all relevant info and outline your thought process, then answer."* This method is akin to a **debug mode** for the AI. In fact, some modern AI systems return a "thinking trace" by design, which can be invaluable for debugging. The bottom line: if the task is complex, let the AI reason it out step-by-step — you can always edit out the reasoning in the final output if needed.
- **Include Worked Examples:** Few-shot prompting (a.k.a. giving **examples in the prompt**) is a proven way to improve performance. If you want the AI to follow a certain style or solve a type of problem, show it one or two *model examples* right in the

question]\nA: [well-crafted answer]." This serves as a template. The AI will infer the pattern and try to mimic it for new inputs. *Worked examples* basically demonstrate the desired behavior. They are extremely useful when the task has a format that's hard to explain abstractly. By seeing an example, the AI gets a concrete sense of what you expect. Just be sure your examples are high-quality and truly representative of what you want — garbage in, garbage out.

- **Set Format and Style Guidelines:** Don't hesitate to tell the AI exactly how to format its output. If you need an answer as a JSON object, include an example JSON snippet in the prompt and say "Answer in this JSON format only." If you want a paragraph followed by bullet points, say so explicitly. Structured output not only makes it easier for you to parse the AI's response later, but it also reduces ambiguity for the model (it doesn't have to guess *how* to present the answer). Some startups even embed XML-like tags in prompts to enforce structure, as seen in the earlier example (<manager_verify>accept</manager_verify> etc.). Structure is your friend — it turns a free-form AI into a predictable, formatted output generator.
- **Implement "Escape Hatches":** Sometimes the best answer is admitting not having an answer. A well-engineered prompt will instruct the AI on what to do if it's unsure or if a question falls outside its knowledge. This is often called an **escape hatch**. For example, you can add a line: *"If you don't know or don't have enough information, **do not** fabricate an answer. Instead, respond with: 'I'm not sure about that, let me escalate this to a human.'"* By giving the AI permission to say "I don't know" or defer, you prevent it from hallucinating false information. This greatly

also involve defining when to hand off to a human (like "if the user asks to reset their password, flag it for human support"). Think of escape hatches as safety valves in your prompt design.

These best practices barely scratch the surface, but they highlight a key theme: **good prompt engineering is about anticipating what could go wrong or be unclear, and baking in instructions to handle it.**

You're essentially coding logic and guardrails using natural language.

As you iterate on prompts, you'll keep refining these elements — adding a new example when the AI misunderstands something, or tightening a rule when you see an odd output.

Find out the top 3 prompts that help you learn Python quickly.

Testing and Iteration: Worked Examples & Evaluation Frameworks

Writing a prompt is only half the battle — the other half is **testing it thoroughly and iterating** on it.

In fact, many AI builders will tell you that a prompt isn't "finished" until it's been put through an evaluation wringer. This is where **worked examples** and **evaluation frameworks** come in as critical tools.

Earlier we discussed adding examples *inside* the prompt to guide the AI.

But you should also test your prompt on a variety of example inputs *from outside* to see how it performs. Essentially, you create a **suite of test cases** (just like unit tests for code) for your prompt. These might be real user queries or simulated edge cases that you think your AI should handle.

By running these through your prompt and examining the outputs, you can identify where the AI might be going off track.

Are there certain phrasing of questions where it fails?

Does it follow the format consistently?

Each problematic case is a clue, and you can tweak the prompt to address it or even add that scenario as another example in the prompt.

The best AI startups treat their **evaluation set as gold.**

One expert said, *"The prompts are important, but the evaluation suite — the set of test cases to measure prompt quality and performance — is your most valuable IP."*

In other words, the knowledge of *how* you test the prompt and what criteria you expect is incredibly important. It's what lets you improve reliably and know when you're making progress.

Every time you adjust the prompt, you run your eval tests to see if things improved or if any previously good behavior broke (regressions). This approach brings scientific rigor to what might otherwise feel like guesswork.

An **evaluation framework** can be simple or sophisticated.

On the simple end, it might be a spreadsheet of example queries with a column for "Did the AI do the right thing? (Y/N)" and notes.

On the advanced end, companies are investing in automated eval tools that score the AI's output on various dimensions (accuracy, helpfulness, adherence to format, etc.).

Some even integrate these evals into continuous integration pipelines — every new prompt update gets automatically tested.

The goal is to catch issues early and quantitatively track improvements. For instance, you might have 100 test prompts and see that after a revision, the AI went from passing 85/100 to 92/100 cases.

That's a clear win.

Crucially, **iteration is key**.

Prompt engineering is rarely one-and-done. You'll go through many cycles of *"prompt -> test -> analyze -> refine -> repeat."*

can quickly recalibrate your prompts if needed. It's a bit like gardening — you're constantly pruning and training the AI's behavior with these prompts and tests.

The upside is that over time you build a very resilient system that you trust, because you've seen it perform across all the tricky scenarios you care about.

One more thing: keep in mind that as models evolve, prompt strategies might need updating too.

A trick that worked wonders on GPT-3 might be less needed on GPT-4 (which might understand nuance better), or vice versa. That's another reason to stay disciplined with testing. It also pays to **stay informed and keep learning** — the field of prompt engineering is moving fast.

(If you want to learn in more detail or get hands-on practice with prompt design and evaluation, consider structured resources like our prompt engineering course for building a professional portfolio or even a broader AI fundamentals course like Grokking Modern AI Fundamentals to strengthen your foundation.)

Founders as Forward-Deployed Engineers: A New Dev Workflow

Beyond the technical tactics, prompt engineering is also changing the **culture and workflow** of AI startups.

Perhaps nowhere is this more evident than in how founders and early engineers are approaching customer problems.

Y Combinator's CEO recently pointed out that today's AI startup founders often act as **"forward-deployed engineers"** — a term borrowed from Palantir — meaning they embed themselves with clients, understand their needs up close, and rapidly craft AI solutions for those specific problems.

In plain terms, the founder (or a lead developer) is out there on the front lines, not just selling the software but actually *implementing* it hand-in-hand with the customer.

This is a big shift from the old-school enterprise software sales model.

Traditionally, you'd have a salesperson promise the moon, then a solutions engineer might do a custom demo, and months of negotiations later you close a deal, then spend more months integrating.

With the forward-deployed engineer approach, it's almost the opposite: a highly technical founder meets the client on day one, **identifies a pain point**, and then perhaps by the next day, delivers a

The pitch isn't just "Our AI can do customer support," it's "I noticed your support team spends 3 hours a day triaging tickets — so I went ahead and configured an AI agent last night using some of your data, and look, it already handles 90% of routine tickets. Imagine what it could do with a bit more tuning."

The customer sees their exact problem being solved in real time, and the effect is powerful: **deals close faster** (sometimes in days instead of months) because the value is immediately proven.

For developers and prompt engineers, this means the lines between *sales*, *product development*, and *deployment* are blurring.

The prompt you engineer might be part of a live demo the day after a client meeting. It's a high-feedback, high-speed cycle.

Founders are coding the AI's behavior (via prompts and some glue code) almost in the meeting room with the client.

This strategy has been so effective that many small AI startups are winning contracts over big incumbents; they can adapt and deliver a custom AI solution *so quickly* that larger competitors (who have longer sales cycles and less flexible products) get left in the dust.

In short, **startup agility + prompt engineering know-how = a new kind of competitive moat.**

user and solving their actual problem with whatever it takes — often that's clever prompt design and rapid iteration.

It's not about building a one-size-fits-all AI and hoping it matches everyone's needs; it's about **tuning and tailoring the AI on the fly**.

Prompts are the vehicle for much of that tailoring, since you can often adjust an AI's behavior dramatically without changing the underlying model — just by rewriting some instructions.

This approach does require technical folks who are also comfortable in customer-facing situations, but it's proving to be a superpower in the AI startup scene.

Tools, Tips, and Mental Models for Modern AI Builders

As you venture into prompt engineering and building AI solutions, it helps to arm yourself with the right **tools and mental models**.

Here are some parting tips that developers and AI builders are using to stay ahead:

1. Think of AI as a Collaborator (or Intern)

One useful mental model is to treat your AI model as if it's a new member of your team.

Sometimes you're the mentor (writing detailed prompts to guide it), and sometimes the AI can mentor you (as in metaprompting, where it gives you ideas). This perspective reminds you that communication needs to be clear and unambiguous.

Just like a human teammate, the AI will do better if you set the right context, instructions, and feedback.

And if the AI is particularly capable (say GPT-4), you might even treat it like a specialist you consult — e.g. *"Here's my problem, how would you solve it?"* — and then you integrate its advice back into your code or prompts.

2. Leverage Tools for Prompt Management

As prompt libraries grow, keeping track of different versions and variants can get tricky.

Developers are turning to tools to help manage prompts — from simple version control using Git (treat prompts as code!) to specialized prompt design platforms.

Some frameworks like LangChain, PromptLayer, or other prompt orchestration libraries help assemble those multi-layer prompts or chain multiple AI calls together.

good for.

Think of it as your personal prompt cookbook.

3. Understand Model Differences

Not all AI models are the same — each has its own "personality" and quirks.

A prompt that works perfectly on OpenAI's GPT-4 might need tweaking to work on an open-source Llama model, for example.

Larger models can usually follow complex instructions better and might allow more creative or open-ended prompts, whereas smaller models might need more explicit guidance.

Savvy AI builders keep these differences in mind.

In practice, you might use a powerful model during development to **fine-tune your prompt** (since it's more forgiving and can generate ideal outputs), and then **distill** those insights to a smaller, cheaper model for production.

It's a bit like first having the senior expert solve the problem, then training a junior on the exact procedure to repeat. This "prompt distillation" approach helps balance cost and performance.

4. Stay Curious and Keep Learning

Make it a habit to read about new prompting techniques, share insights with the community, and even take courses to sharpen your skills.

(If you're serious, you might enjoy the structured learning in Grokking Prompt Engineering or our Modern AI Fundamentals course, which cover a lot of real-world techniques and projects.)

The more prompts you write and test, the more intuition you'll build about how these models "think."

Over time, you'll develop a gut feel for what a prompt needs to include to get the desired outcome — that's when prompt engineering becomes more art than science.

5. Embrace the Iterative Mindset

Perhaps the most important tip is psychological: embrace iteration and don't be discouraged by failures.

Prompt engineering often involves *flailing* a bit — you'll try a prompt that you think is brilliant only to get a bizarre answer.

That's okay.

Treat each failure as a data point.

Why did the model respond that way?

Every tweak is getting you closer to the goal.

In a way, building an AI agent via prompts is like having a dialogue with the model — you refine your instructions, it refines its output.

Maintain that conversational, iterative mindset, and you'll find that even the toughest AI challenges can be solved step by step.

Conclusion: The New AI Playbook

We're witnessing a fundamental shift in **how software is built**.

In the era of AI, writing good instructions for machines (in natural language) is as crucial as writing good code.

This new playbook — prompt engineering, metaprompting, layered prompts, continuous evaluation — is enabling tiny startups to create AI products that compete with tech giants, simply by being more agile and clever in how they harness large language models.

Founders are jumping in as forward-deployed engineers, and developers are adopting the role of prompt crafters and AI orchestrators.

The exciting part is that this field is highly accessible. You don't need a PhD in machine learning to start crafting effective prompts and building useful AI agents.

is all about clear communication.

The future of AI development will belong to those who can blend traditional coding with prompt crafting, who can collaborate with AI systems as partners, and who remain flexible and user-focused in solving problems.

Thank you for being a part of the community

Before you go:

- Be sure to **clap** and **follow** the writer 👏
- Follow us: X | LinkedIn | YouTube | Newsletter | Podcast | Twitch
- Start your own free AI-powered blog on Differ 🚀
- Join our content creators community on Discord 👩‍💻
- For more content, visit plainenglish.io + stackademic.com