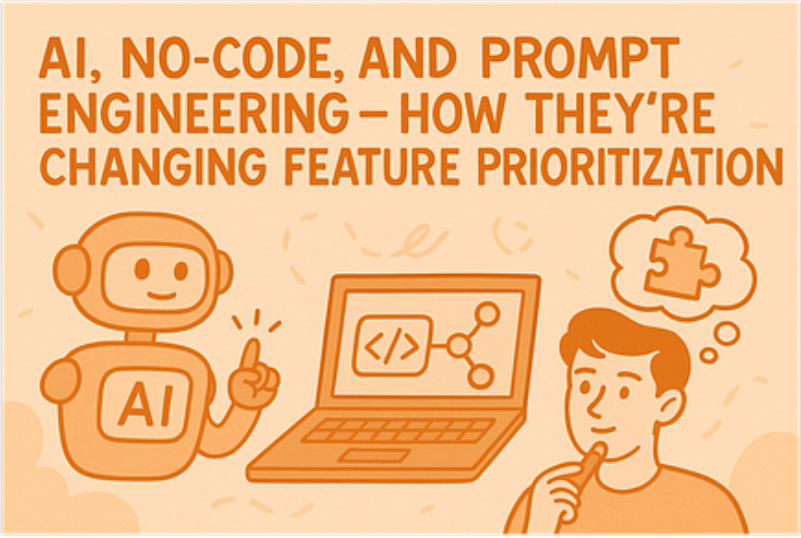
How AI, No Code, and Prompt Engineering are Changing Feature Prioritization

In today's tech scene, product teams are juggling a new toolkit from smart AI agents to "vibe coding" prototypes built without writing a line. This is transforming how startups and product teams decide what to build next.

Kartscrut • 14 min read • 2025-05-07

https://ai.gopubby.com/how-ai-no-code-and-prompt-engineering-are-changing-feature-prioritization-391e628abc41

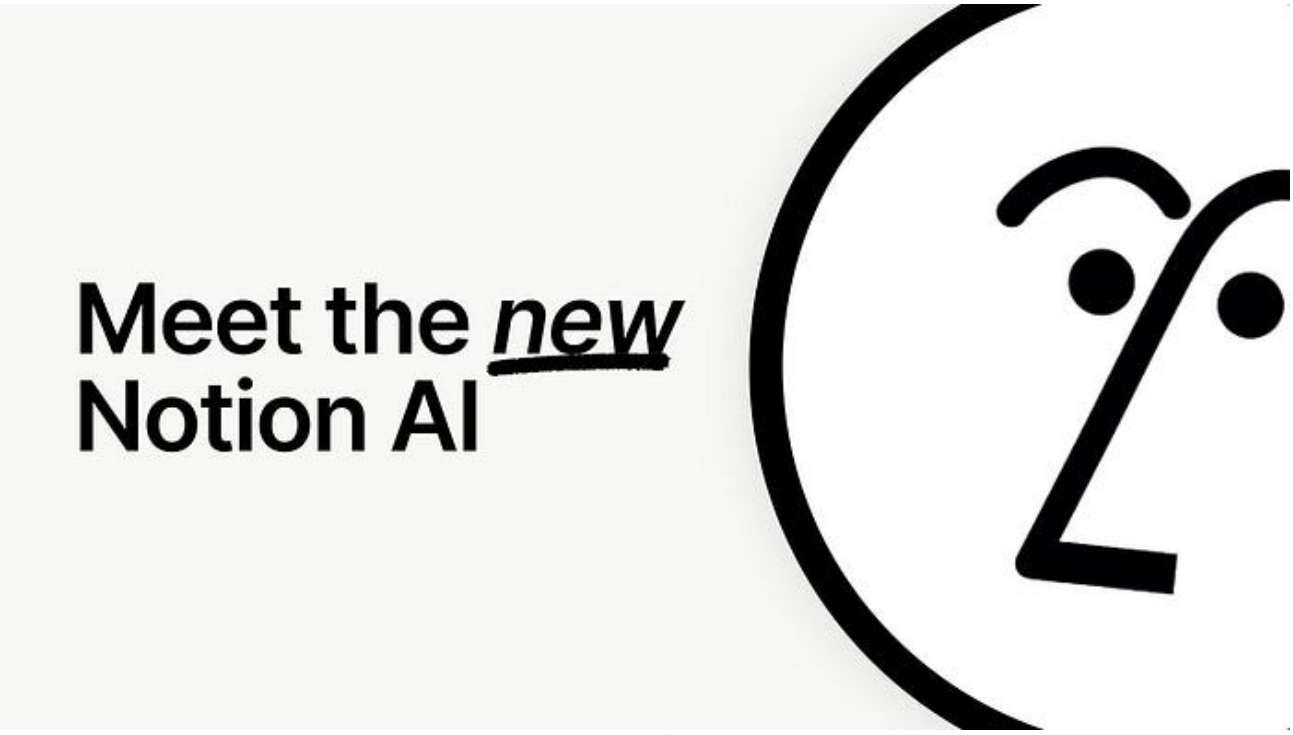


AI as a Feature, Not a Gimmick

Everyone's talking about AI, but how do you tell meaningful innovation from buzzword hype? The key is **focusing on real user value**. The best AI-powered features start with a clear problem to solve and not just adding AI for AI's sake. As one product strategy guide puts it, *"AI belongs when it clearly contributes to business and user value."* In other words, an AI feature should reduce friction, save time, improve accuracy, or otherwise make the user's life better. If it doesn't serve the product's core purpose, it's likely just novelty or just noise.

Take **Notion's AI writing assistant** for example. Notion didn't add AI because it was trendy - they identified specific pain points their users face (writer's block, summarizing notes, catching errors) and targeted those. In their launch, they literally listed how Notion AI helps *"tackle that first draft,""spur ideas and creativity,"* act as an *"editor"* for grammar and tone, and *"summarize a long meeting or document."* Each of those is a real-world task that teams and individuals struggle with. By focusing on these genuine needs, Notion ensured their AI wasn't just a flashy add-on,

but a feature that **truly improves the user experience**.



New Notion AI

Here are some examples of how teams and individuals can use Notion AI:

- **Let it tackle that first draft** — the first word can often be the hardest one to write. Instead, ask Notion AI to handle your first draft about a topic and get some ideas for you to turn into something great.
- **Spur ideas and creativity** — instantly get a list of ideas about anything. This can help you be more creative, surfacing ideas as a jumping off point (or some you wouldn't even have thought of).
- **Act as your hawk-eyed editor** — whether it's spelling, grammar, or even translation, Notion AI catches mistakes or translates whole posts to help ensure writing is accurate and actionable.
- **Summarize a long meeting or document** — instead of sifting through a garble of meeting notes, let Notion AI pull out the most important points and action items.

Examples of pain points Notion AI addresses, as highlighted in Notion's AI announcement. The feature was designed to help with starting drafts, generating ideas, editing text, and summarizing content - all real productivity boosters, not just "because AI."

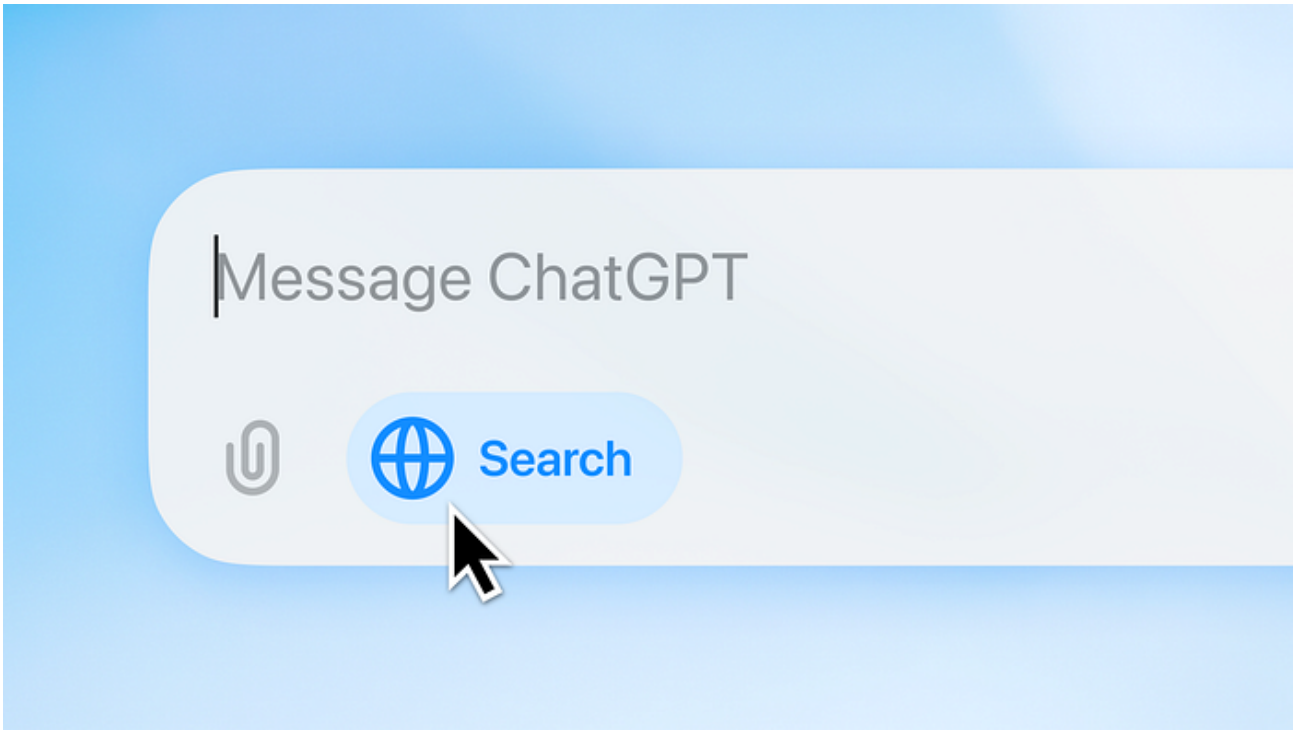
Contrast that with products that slapped an "AI inside!" sticker on features that didn't need it. We've seen chatbots launched with great fanfare only to fizzle because they didn't solve a real

problem, or “AI-powered” recommendations that felt random and gimmicky. The lesson for product teams is clear: **start with the outcome, not the technology**. If an AI technique can deliver that outcome better or faster, great - prioritize it. If not, think twice. As a guide for product owners notes, always ask *“What outcome are we enabling?”* and only pursue AI if it’s the most effective way to achieve it. In practice, a simple, well-targeted automation often beats a complex AI that doesn’t fit the user’s workflow.

In short, **value earns priority**. An AI feature that quietly makes your product 10x better is worth far more than one that just generates headlines. Users will quickly tell the difference. Product teams should use that as a north star when weighing AI ideas: does this make the experience feel like magic (in a useful way), or is it just adding confusion?

The Rise of Natural Language Interfaces (Blame/Thank ChatGPT)

A major shift in product design coming from the AI boom is the rise of **natural language interfaces** (letting users interact with software by simply talking or typing requests in ordinary language). The poster child here is **ChatGPT**. When OpenAI released ChatGPT in late 2022 as a free chatbot, it stunned the world by showing how *conversational* an AI assistant could be. Within weeks it hit millions of users, and by February 2025 ChatGPT had soared to over **400 million weekly active users** *(This might even have doubled by now as per Sam the Altman thanks to Ghibil Art and new releases)* becoming the fastest-growing consumer app in history. Crucially, this adoption wasn’t because of a fancy GUI or killer app integration; it was because the **interface was just conversation**. Anyone could open the chat box and ask anything.



OpenAI’s team, to their own surprise, discovered that a simple chat UI unlocked countless use cases. People were using ChatGPT for everything from writing code and debugging, to drafting emails, to getting therapy-like advice. This success has not only been sustained, OpenAI has kept rapidly improving the product (adding new features **continuously to feed demand**) but it’s influenced nearly every tech roadmap out there. Suddenly, users expect to “ask” software for

what they want, not hunt through menus. Products from search engines to productivity tools have rushed to add ChatGPT-like assistants or chat features.

For example, Notion's AI, beyond just writing help, now lets you **chat with your workspace content** - you can ask a question and get answers sourced from your notes and docs. Similarly, many apps (Slack, Zoom, Canva, to name a few) have introduced conversational assistants inside their UI. This trend forces product teams to rethink design: some are even asking *"Do we need a button for this, or can the user just **tell** the app what they need?"* Natural language interfaces can simplify complex workflows by removing steps. A user might type *"find and highlight any out-of-budget items in this spreadsheet"* instead of manually applying filters and formatting. That's powerful!

But it also raises new challenges. For one, **discoverability**, users won't know everything your AI can do unless you guide them. This is why products often provide example prompts or have the AI proactively suggest, "Hey, I can help you with that." And then there's the matter of accuracy and scope. A chat interface is open-ended: users might ask things you never anticipated. Product peoples have to prioritize what knowledge or functionality the AI assistant should handle first (and what is out of scope). For instance, when **ChatGPT launched**, it was based on a general model (GPT-3.5) with no internet access. Over time, OpenAI prioritized features to extend its utility: web browsing to bring in up-to-date info, Code Interpreter to handle data and programming tasks, a transition to the more capable GPT-4 model, and even voice input for spoken conversations. Each of these features was chosen because of how people were actually using (or wanted to use) ChatGPT. Users were writing code and asking for data analysis, so Code Interpreter (which lets the AI run code and return results) became a killer feature. Users wanted to talk to ChatGPT hands-free, so voice input was added. In other words, **natural language changed the roadmap** instead of pre-defining narrow features, the team looked at what people asked for and then built capabilities to fulfill those requests.

The takeaway here is that thanks to ChatGPT and its kin, **talking to our tools** is becoming normalized. Product teams should consider where a conversational or natural language feature might delight users - and where it might frustrate them - and adjust their priorities accordingly. In some cases, the best new "feature" you can add is letting users access *existing* features by simply asking for them in plain English.

The Trend of Vibe Coding

Building software used to be slow. In startups, we're all familiar with the cycle: come up with an idea, write specs, have engineers code it up over days or weeks, and *then* see if it actually works for the user. Today, that cycle is speeding up dramatically thanks to **no-code/low-code tools** and a new wave of AI-assisted development sometimes dubbed **"vibe coding."** In essence, if you can describe what you want, these tools will try to build it. This is a game-changer for prototyping and feature exploration.

Traditional no-code platforms like Webflow, Bubble, or Framer have been letting non-engineers create apps and workflows via drag-and-drop or simple configurations. That trend has been growing for years (the no-code/low-code market is huge, projected to reach $50 billion by 2028 Src. nucamp.co). But *vibe coding* takes it a step further: you literally type what you want in natural language, and the AI does the coding behind the scenes. As one analysis explains, *"vibe*

coding leverages AI, Eg: ChatGPT or GitHub Copilot turning natural language into code, making development even more accessible and creative." It's like having a junior developer who never sleeps, ready to scaffold your idea at a moment's notice.

Image Source Jama

All this speed comes with a double-edged sword: if **anything can be built in minutes, how do you decide what *should* be built?** It's every PM's dream and nightmare: an explosion of prototypes. On one hand, you can try lots of ideas cheaply - A/B test different features before fully investing. On the other hand, it's easy to get a "prototype sprawl" where the team chases too many concepts. **Stronger filtering and prioritization are more important than ever.** The fundamentals of product-market fit don't change just because it was easy to build something. In fact, the low effort to create means we have to be extra disciplined in evaluating which prototypes truly solve a problem or have traction.

Another consideration is quality and feasibility. Code or designs generated by AI can be hit-or-miss. They often require cleanup, polish, and performance tuning by humans. An industry analysis of AI-generated code pointed out pitfalls like security vulnerabilities, inconsistent coding styles, and even logical errors creeping in if you don't carefully review the AI's output. As Uncle Ben said, "with great power comes great responsibility" *treat AI as an assistant, not a replacement* . For product folks, this means involving engineers in the loop to vet any AI-created prototypes before considering them "ready" and budgeting time to refactor or rebuild parts of them properly if you decide to move forward.

Think of these tools as **idea accelerators**. They let you get the gist of a feature or product quickly, which is fantastic for early feedback and internal demos. But you still need your experts to assess scalability, security, UX consistency, etc. One framework describes vibe coding as *"ideal for quick prototypes or small-scale apps, but large or critical projects still require hands-on programming for flexibility and safety."* In practical terms: generating a basic CRUD app with AI is fine, but you wouldn't auto-generate the core banking software for a finance app and ship it without rigorous review.

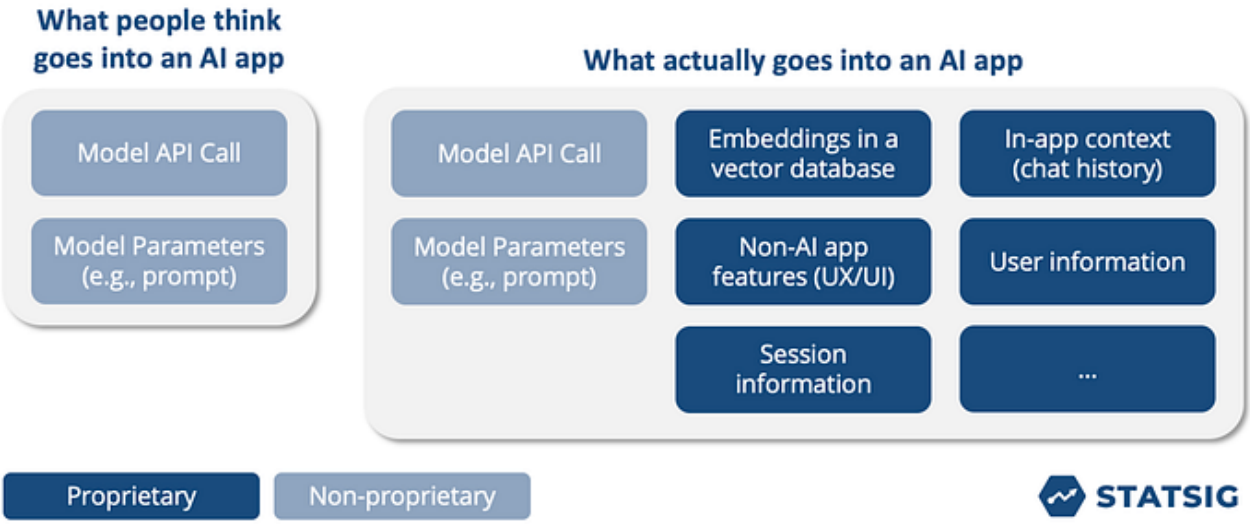
For prioritization, this rapid prototyping ability means teams might shift how they decide on features. Instead of lengthy debate or analysis paralysis, you can build a **minimum viable experiment** with AI help and see real user reactions. The features that stick or show promise can then be prioritized for full development. Those that flop, you discard early without having lost much time. It encourages a more iterative, experimental culture. Just beware of the temptation to greenlight a project solely because "well, we already have a prototype of it." The question to keep asking is "Does this align with our strategy and deliver value to users?" If yes, having a quick

prototype is a bonus that can accelerate execution. If no, then even the coolest AI-generated demo should be deprioritized or shelved.

Prompt Engineering: Surely the Final Boss

Alongside AI and no-code tools comes an interesting new discipline: **prompt engineering**. This is the art and science of crafting the right inputs or instructions for AI systems to get the desired output. Why is this relevant to product folks and feature prioritization? Because increasingly, designing a feature means designing how you "talk" to the AI behind it.

When you integrate an AI model into your product (whether it's for customer support, content generation, recommendations, etc.), you don't just plug it in and hope for the best. You have to **configure its behavior**, often through prompts or examples. For instance, if you're building an AI writing assistant like Notion did, you need to figure out how to prompt the model to continue a user's writing in a helpful way. Notion revealed that one of their first AI commands was "continue writing" which basically, given the user's existing text, the AI should keep going in the same style/flow. To make that happen reliably, the team likely had to experiment with different ways of feeding the user's content and the instruction into the model (perhaps something like: *"Here is the user's draft. Continue writing in the same tone and voice."* behind the scenes). They also augmented the AI with additional context, like the page topic or the user's own notes, to improve quality of it. All of that is prompt engineering work i.e. figuring out *what* to send to the AI model to get a useful result.



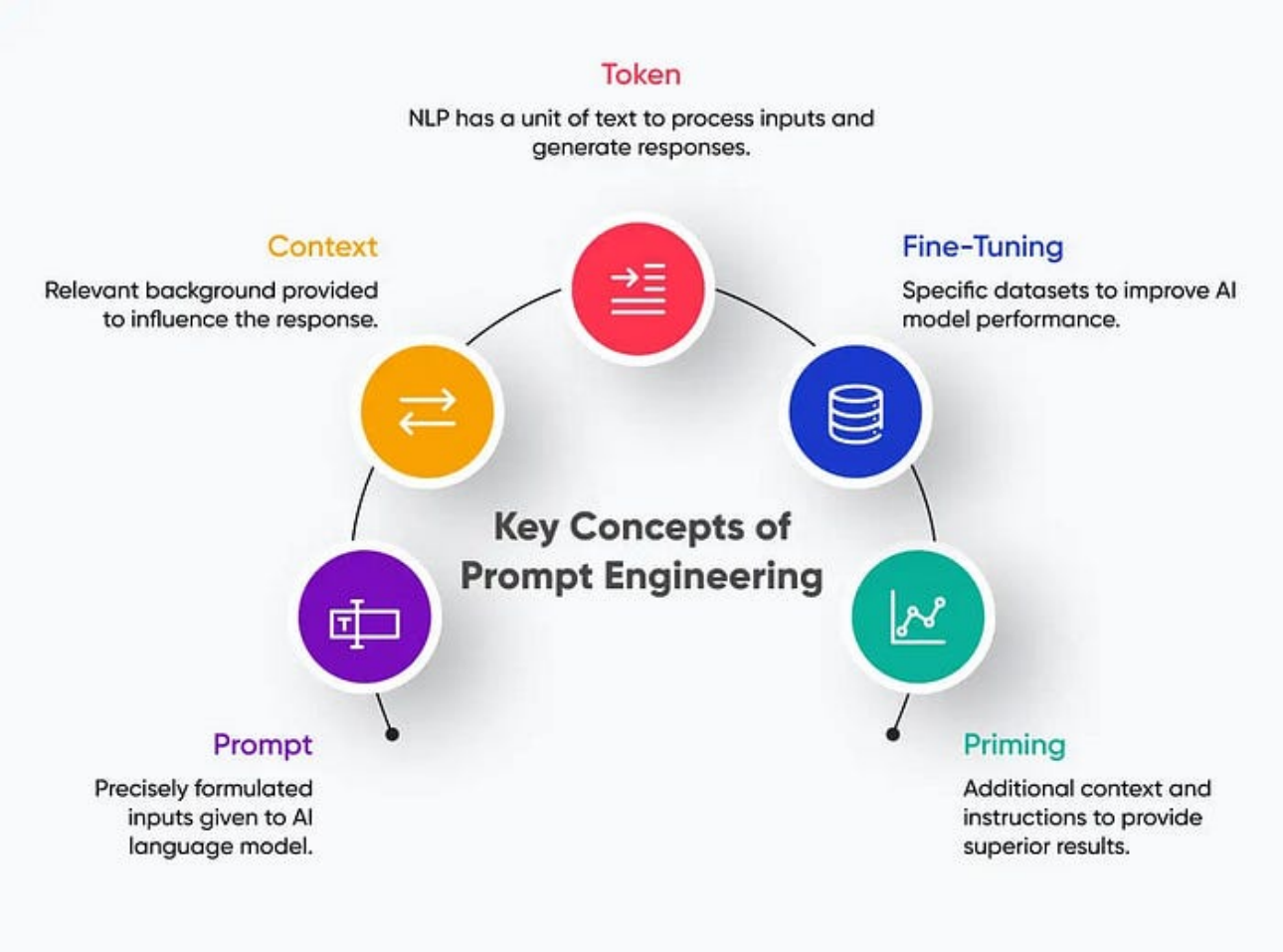
Building a successful AI-powered feature involves more than plugging into an AI API. It requires surrounding the model with the right prompts, data context, and UI/UX support. (Illustration: what people think goes into an AI app vs. what actually goes into it, including proprietary data, context from the application, and other non-AI components) [Image Src. Statsig]

Prompt engineering isn't just for writing assistants. If you have a chatbot in your app, you'll have to create a system prompt that sets its persona and guidelines (e.g., "You are an expert financial advisor helping users budget..."). If you use AI to generate UI designs from descriptions, you have to prompt it with the right design language. Even for those AI coding tools we talked about, the better you phrase your request, the better the code that comes out. This has led to the somewhat humorous job title "Prompt Engineer" emerging at some companies. Yeah I even saw

an opening for this position on LinkedIn Jobs.

It's worth noting that over time, some of this prompt engineering burden might lessen as AI models improve or allow more direct ways to customize (like fine-tuning, or learning from user behavior). But right now, in 2025, it's a crucial piece. When prioritizing an AI-driven feature, you might also need to prioritize time for prompt engineering and testing in your development process. It's not as straightforward as building a deterministic algorithm; you need to allocate time to play with the AI's outputs, maybe run beta tests, and refine how you ask the AI to ensure the feature works for the diverse scenarios users might throw at it.

One practical tip for teams is to **treat AI behavior design as part of the UX**. For example, if your app's AI is supposed to give financial advice, define the tone (friendly but professional), the reading level (no jargon), how it should handle uncertainty ("if I'm not sure, do I make a guess or ask for clarification?"). These considerations can be addressed in the prompt or system settings of the AI. By being deliberate about this, you avoid a lot of the "AI said something weird" issues that could derail user trust.



Key Concepts of Prompt Engineering (Image Src. BotPenguin)

In summary, product folks don't necessarily have to become expert prompt engineers themselves, but they should understand the importance of this step. It might influence hiring (maybe you need a conversational designer or prompt specialist on the team) and it certainly influences timelines (leave room for iteration). When prioritizing features that involve AI, factor in the prompt engineering as part of the feature's development effort and success criteria. After all,

a feature is only as good as its output and for AI, the output is heavily influenced by how you ask for it.

Responsible Design Is Product Excellence

In the end it is all about trust and trust is essential for adoption. Regardless of performance, users need to understand what the system is doing and why. Transparency, fairness, and control should be baked into your feature design from the beginning.

Ai generated :p

Be clear about how the AI makes decisions. Give users appropriate visibility and options for control. Ensure the feature complies with privacy regulations and ethical expectations. Trust doesn't just protect your product, it elevates it.

Ultimately, the mission hasn't changed: understand user needs, and deliver solutions that address those needs in a viable way. What **has** changed is the toolkit at our disposal. We can delegate more to AI (from coding to content generation), we can interact with our users in more fluid ways (conversationally and dynamically), and we can try out ideas with far less friction than before.

So, whether you're a founder sketching out a roadmap, a PM triaging feature requests or a Product Designer just innovating, remember that you have an expanded palette now. **Think big**, **start small** (maybe prompt an AI for a prototype tonight!), and stay user-centered. The companies that get this right are not only going to build faster but they're going to build better products, period. And that is something worth prioritizing because

> Only constant in life is change.

> **Reference**

1. The Product Owner's Guide to AI Features - Balancing Innovation with Value Deliver by cprime.
2. Lessons from Notion: How to build a great AI product, if you're not an AI company by Sky Scofield for STATSIG.
3. No-Code, Low-Code, Vibe Code: Comparing the New AI Coding Trend to Its Predecessors by By Ludo Fourrage for nucamp.
4. Must Watch Video: NVIDIA CEO Jensen Huang's Vision for the Future.

> **Thank you for reading!** *For more of me, visit https://kartscrut.com Find me on X(Twitter) and LinkedIn*

Other mentions by Author

- ai.gopubby.com | Published in AI Advances
- medium.com | Written by Kartscrut