Prompting 1.1

Prompt structure, levels of prompting, meta/reverse meta prompting, and foundational tactics with examples.  
[Heads up!](https://docs.lovable.dev/tips-tricks/prompting-one#heads-up)  
What is Prompting?  
Why Prompting Matters  
Understanding How AI Thinks

Core Prompting Principles: The C.L.E.A.R. Framework

The Four Levels of Prompting

1. Structured “Training Wheels” Prompting (Explicit Format)

[2. Conversational Prompting (No Training Wheels)](https://docs.lovable.dev/tips-tricks/prompting-one#2-conversational-prompting-no-training-wheels)

3. Meta Prompting (AI-Assisted Prompt Improvement)

4. Reverse Meta Prompting (AI as a Documentation Tool)

Advanced Prompting Techniques

Zero-Shot vs. Few-Shot Prompting

Managing Hallucinations and Ensuring Accuracy

Leveraging Model Insights (Know Your AI Tools)

Additional Prompting tips

Start with a Solid Knowledge Base

Be specific, avoid vagueness

Incremental prompting

Include Constraints and Requirements

Avoid ambiguity in wording

Mind your tone and courtesy

Use LLMs’s Modes Intentionally

Use formatting to your advantage

[Leverage examples or references](https://docs.lovable.dev/tips-tricks/prompting-one#leverage-examples-or-references)

Using image prompts

Feedback integration

Emphasizing Accessibility

Predefined Components and Libraries

Multilingual Prompting

Defining Project Structure and File Management

Provide Precise Edit Instructions (Focus the AI)

Locking Files (Workaround)

Design and UI Tweaks

Refactoring and Optimizing Code

Debugging with AI Assistance

When (and When Not) to Involve the AI

[Applying These Strategies in Different Tools](https://docs.lovable.dev/tips-tricks/prompting-one#applying-these-strategies-in-different-tools)

In LLMs’s Builder

With make.com or n8n (workflow automation)

Edge cases and external integrations

Summary

Conclusion

[​](https://docs.lovable.dev/tips-tricks/prompting-one#heads-up)Heads up!

To help you make the most out of LLMs, I compiled a list of prompting strategies and approaches. Some of these were collected from my experience, and others were shared in communities. Since prompting relies on large language models (LLMs), effective prompting strategies can significantly improve its efficiency and accuracy.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "what-is-prompting%3F)

What is Prompting?

Prompting refers to a textual instructions you give an AI system to perform a task​. In LLMs (an AI-powered app builder), prompts are how you “tell” the AI what to do – from creating a UI to writing backend logic. Effective prompting is critical because LLMs uses large language models (LLMs), so clear, well-crafted prompts can greatly improve the AI’s efficiency and accuracy with their output​. In short, better prompts lead to better results.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "why-prompting-matters)

Why Prompting Matters

Most people think prompting is just typing a request into an AI and hoping for the best – not so. The difference between a mediocre AI response and having AI build entire workflows for you, comes down to *how you prompt*. Whether you’re a developer or non-technical, mastering prompt engineering in LLMs can help you:

Automate repetitive tasks by instructing the AI precisely what to do.

Debug faster with AI-generated insights and solutions.

Build and optimize workflows effortlessly, letting AI handle the heavy lifting once properly guided.

And the best part? You don’t need to be an expert programmer. With the right prompting techniques, you can unlock AI’s full potential in LLMs without wasted trial-and-error. This playbook will take you from foundational concepts to advanced prompt strategies so you can communicate with AI effectively and build faster.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "understanding-how-ai-thinks)

Understanding How AI Thinks

Unlike traditional coding, working with AI is about *communicating* your intentions clearly. Large Language Models (LLMs) like the ones powering LLMs don’t “understand” in a human sense – they predict outputs based on patterns in their training data. This has important implications for how you should prompt:

For consistent outcomes, it helps to structure your prompt into clear sections. A recommended format (like *“training wheels”* for prompting) uses labeled sections for Context, Task, Guidelines, and Constraints​:

Provide Context and Details: AI models have no common sense or implicit context beyond what you give them. Always supply relevant background or requirements. For example, instead of just saying “Build a login page,” specify details: *“Create a login page using React, with email/password authentication and JWT handling.”* Include any tech stack or tools (e.g. “using Supabase for auth”) explicitly.

Be Explicit with Instructions and Constraints: Never assume the AI will infer your goals. If you have constraints or preferences, state them. For instance, if an output should use a specific library or remain within certain scope, tell the model up front. The AI will follow your instructions literally – ambiguities can lead to unwanted results or AI “hallucinations” (made-up information).

Structure Matters (Order and Emphasis): Thanks to transformer architecture, models pay special attention to the *beginning and end* of your prompt. Leverage this by putting the most crucial details or requests at the start, and reiterating any absolute requirements at the end if needed. Also remember models have a fixed context window – overly long prompts or very long conversations may cause the AI to forget earlier details. Keep prompts focused and refresh context when necessary (e.g. remind the model of key points if a session is long).

Know the Model’s Limits: The AI’s knowledge comes from training data (e.g. Sonnet 3.7’s data cutoff, or LLMs’s provided knowledge base). It can’t know about recent events or proprietary info you haven’t given it. It will try to *sound* confident even if it’s guessing (which leads to hallucinations). Always provide reference text or data for factual queries, or be prepared to verify its output. Also, complex instructions might be handled better by more advanced models (LLMs’s Chat mode likely uses a powerful model ideal for reasoning), whereas simple tasks can be done with default modes quickly. Understanding that LLMs generate text *one token at a time, predicting likely continuations* can help you craft prompts that set them on the right path from the start.

Think of prompting as telling a very literal-minded intern *exactly* what you need. The clearer and more structured your guidance, the better the results. Next, we’ll dive into core principles that make a prompt effective.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "core-prompting-principles%3A-the-c-l-e-a-r-framework)

Core Prompting Principles: The C.L.E.A.R. Framework

Great prompts follow a set of simple principles. A handy way to remember them is CLEAR: Concise, Logical, Explicit, Adaptive, Reflective. Use these as a checklist when crafting your instructions:

Concise: Be clear and get to the point. Extra fluff or vague language can confuse the model. Use direct language: for example, BAD: “Could you maybe write something about a science topic?” GOOD: “Write a 200-word summary of the effects of climate change on coastal cities.” Avoid filler words – if a detail isn’t instructive, it’s distracting. Aim for precision and brevity in describing what you want.

Logical: Organize your prompt in a step-by-step or well-structured manner. Break complex requests into ordered steps or bullet points so the AI can follow easily. Rather than a single run-on request, separate concerns. BAD: “Build me a user signup feature and also show some stats on usage.” GOOD: *“First, implement a user sign-up form with email and password using Supabase. Then, after successful signup, display a dashboard showing user count statistics.”* A logical flow ensures the model addresses each part of your request systematically.

Explicit: State exactly what you want and don’t want. If something is important, spell it out. Provide examples of format or content if possible. The model has a vast knowledge, but it won’t read your mind about specifics. BAD: “Tell me about dogs.” (Too open-ended.) GOOD: “List 5 unique facts about Golden Retrievers, in bullet points.” Likewise, if you have a desired output style, say so (e.g. “Respond in JSON format” or “Use a casual tone”). Treat the AI like a beginner: assume nothing is obvious to it.

Adaptive: Don’t settle for the first answer if it’s not perfect – prompts can be refined iteratively. A big advantage of LLMs’s AI (and LLMs in general) is that you can have a dialogue. If the initial output misses the mark, *adapt your approach*: clarify instructions or point out errors in a follow-up prompt. For example, *“The solution you gave is missing the authentication step. Please include user auth in the code.”* By iterating, you guide the model to better results. You can even ask the AI how to improve the prompt itself (this is Meta Prompting, covered later).

Reflective: Take time to review what worked and what didn’t after each AI interaction. This is more about *you* than the model – as a prompt engineer, note which prompt phrasing got a good result and which led to confusion. After a complex session, you might even ask the AI to summarize the final solution or reasoning (we’ll discuss Reverse Meta Prompting shortly). Being reflective helps you craft better prompts in the future, building a cycle of continuous improvement in your AI communication.

Keep these CLEAR principles in mind as you develop prompts. Next, we’ll look at specific prompting techniques from basic to advanced, including how to structure prompts and leverage the AI as a collaborator.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "the-four-levels-of-prompting)

The Four Levels of Prompting

Effective prompting is a skill that grows with practice. Here we outline four levels of prompting mastery, from structured “training wheels” to advanced meta techniques. Each level has its use-case – combine them as needed:

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "1-structured-%E2%80%9Ctraining-wheels%E2%80%9D-prompting-explicit-format)

1. Structured “Training Wheels” Prompting (Explicit Format)

When you’re just starting or tackling a very complex task, it helps to use a labeled structure in your prompt. This acts as *training wheels* to ensure you provide all necessary information. A proven format in LLMs is to break the prompt into sections like:

Context: Background or role setup for the AI. (E.g. “You are a world-class LLMs AI coding assistant.”)

Task: The specific goal you want to achieve. (E.g. “Build a full-stack to-do list app with user login and real-time sync.”)

Guidelines: Preferred approach or style. (E.g. “Use React for frontend, Tailwind for styling, and Supabase for auth and database.”)

Constraints: Hard limits or must-not-dos. (E.g. “Do not use any paid APIs. The app should work on mobile and desktop.”)

By clearly labeling each part, you leave little room for misunderstanding. For example, a prompt might look like:

Context: You are an expert full-stack developer using LLMs.  
Task: Create a secure login page in React using Supabase (email/password auth).  
Guidelines: The UI should be minimalistic, and follow Tailwind CSS conventions. Provide clear code comments for each step.  
Constraints: Only modify the LoginPage component; do not change other pages. Ensure the final output is a working page in the LLMs editor.

This level of detail guides the AI step-by-step. *Training Wheels prompting* is excellent for novices or complex multi-part tasks – it forces you to think through exactly what you need, and it helps the model by structuring the request.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "2-conversational-prompting-no-training-wheels)

2. Conversational Prompting (No Training Wheels)

As you get comfortable, you won’t always need such rigid structure. Conversational prompting means you can write to the AI more naturally, similar to how you’d explain a task to a colleague, while still being clear. The key is to maintain clarity and completeness without the formal labels. For instance:

Let’s build a feature to upload a profile picture. It should include a form with an image file input and a submit button. When submitted, it should store the image in Supabase storage and update the user profile. Please write the necessary React component and any backend function needed for this, and ensure to handle errors (like file too large) gracefully.

This is a more free-form prompt but still logically ordered and explicit about the requirements. No training wheels, yet it’s effective. Conversational prompts work well once you trust yourself not to forget important details. They keep interactions more natural, especially in ongoing chat where you’re iterating on results.

Even in conversational style, you can simulate structure by breaking into paragraphs or bullet points for different aspects of the request. The goal is the same: clear communication. You might use this style for quicker tasks or once the AI has already been primed with context.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "3-meta-prompting-ai-assisted-prompt-improvement)

3. Meta Prompting (AI-Assisted Prompt Improvement)

This is an advanced technique where you literally ask the AI to help you improve your prompt or plan. Since LLMs’s AI (like ChatGPT) can reason about language, you can use it to refine your instructions. This is especially useful if you get an output that’s off-base – it could be a sign your prompt was unclear. For example:

Review my last prompt and identify any ambiguity or missing info. How can I rewrite it to be more concise and precise?

Rewrite this prompt to be more specific and detailed: ‘Create a secure login page in React using Supabase, ensuring role-based authentication.

The AI might respond with a better-structured or more detailed version of your request. This can reveal what was unclear. Essentially, you’re letting the AI act as a *prompt editor*. In LLMs, you can do this in Chat mode safely (since Chat mode won’t directly edit your project). Meta prompting turns the AI into a collaborator that helps you ask for what you really want. It’s a powerful way to bootstrap your prompt engineering skills – the AI can suggest improvements you hadn’t considered.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "4-reverse-meta-prompting-ai-as-a-documentation-tool)

4. Reverse Meta Prompting (AI as a Documentation Tool)

Reverse meta prompting means using the AI to summarize or document what happened *after* a task, so you can learn or reuse it later. Think of it as asking the AI to reflect on the process and give you a prompt or explanation for next time. This is great for debugging and knowledge capture. For example, after you troubleshoot a tricky issue with LLMs, you might prompt:

Summarize the errors we encountered setting up JWT authentication and explain how we resolved them. Then, draft a prompt I could use in the future to avoid those mistakes when setting up auth.

The AI might produce a concise recap of the problem and solution, followed by a template prompt like *“Context: building auth… Task: avoid X error by doing Y…”*. This reverse meta approach helps you build a personal library of reusable prompts and lessons learned. In LLMs, this can be gold: the next time you face a similar task, you have a tried-and-true prompt ready to go (or at least a clear checklist to follow).

Suppose you spent an hour debugging why an API call failed. Once it’s fixed, ask the AI to document that. You’ll not only reinforce your understanding, but also create material to feed into the Knowledge Base or future projects so the AI doesn’t repeat the same mistakes.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "advanced-prompting-techniques)

Advanced Prompting Techniques

Once you’ve got the basics, it’s time to leverage more advanced strategies to get the most out of LLMs’s AI. These techniques help handle complex scenarios, reduce errors (like hallucinations), and tailor the AI’s output to your needs.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "zero-shot-vs-few-shot-prompting)

Zero-Shot vs. Few-Shot Prompting

Zero-Shot Prompting means you ask the model to perform a task with *no examples*. You rely on the model’s general training to know what to do. This is the default for most prompts: you state the request, and the AI generates an answer purely from what it “knows” and understands from your prompt. Zero-shot is efficient and works well if the task is common or clearly described. For instance: *“Translate the following sentence to Spanish: ‘I am learning to code.’”* is a zero-shot prompt – straightforward command, and the AI uses its knowledge to respond (no examples needed).

Few-Shot Prompting means you provide a couple of examples or demonstrations in your prompt to show the AI exactly the format or style you want. Essentially, you’re teaching by example in the prompt itself. This can dramatically improve output quality for specific formats or when the task is unusual. In a few-shot prompt, you might say:

Correct the grammar in these sentences:\nInput: “the code not working good” → Output: “The code is not working well.”\nInput: “API give error in login” → Output: “The API gives an error during login.”\nNow Input: “user not found in database” → Output:

By giving two examples of input-output, the AI is primed to continue with a similar pattern for the third. Few-shot prompting is useful in LLMs when you need a specific style of response (e.g., code comments in a certain format, or commit message examples). It does consume more prompt tokens (because you’re including those examples), but often yields more consistent results.

When to use which:

Try zero-shot first for simple tasks or when you trust the model’s built-in ability. If the results aren’t in the format or depth you want, switch to few-shot by adding an example. For instance, if you ask for a function and the output isn’t following your preferred style, show an example function with the style you like and prompt again. Few-shots shine for complex output (like writing tests cases – provide one sample test, then ask it to write more). In summary, zero-shot for quick direct answers, few-shot for controlled style or complex instructions.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "managing-hallucinations-and-ensuring-accuracy)

Managing Hallucinations and Ensuring Accuracy

AI “hallucinations” are moments when the model confidently invents information or code that isn’t correct. In a coding platform like LLMs, hallucinations might mean the AI uses a nonexistent function, calls an API that doesn’t exist, or fabricates details in a summary. While we can’t eliminate this completely (it’s an AI limitation), we can prompt in ways that reduce hallucinations:

Provide Grounding Data: The more *reliable context* you give, the less the AI has to guess. In LLMs, always leverage the Knowledge Base for your project. Include your Project Requirements Document (PRD), user flows, tech stack, etc., in the project’s context. That way, the AI’s answers will be “grounded” in the specifics of *your* app. For example, if your app uses a certain library or has a defined data model, put that in the Knowledge Base so the AI won’t make up different ones.

In-Prompt References: When asking factual questions or code that interacts with external systems, include relevant documentation snippets or data. E.g., *“Using the API response format given below, parse the user object… [then include a small JSON example].”* By showing the AI real data or docs, it’s less likely to fabricate functions or fields.

Ask for Step-by-Step Reasoning: Sometimes you suspect the AI might be winging it. In those cases, prompt it to show its reasoning or verification. For instance, in Chat mode you could say: *“Explain your solution approach before giving the final code. If there are any uncertainties, state them.”* This chain-of-thought prompting makes the AI slow down and check itself. It can catch errors or at least reveal them in the reasoning, which you can correct.

Instruct Honesty: You can include a guideline in your prompt like *“If you are not sure of a fact or the correct code, do not fabricate it – instead, explain what would be needed or ask for clarification.”* Advanced models often follow such instructions (they might respond with, “I’m not certain, but I assume X…” rather than just giving a wrong answer). It’s not foolproof, but it can mitigate confidently incorrect outputs.

Iterative Verification: After the AI gives an answer, especially for critical things (like calculations, or important facts, or complex code), do a verification step. You can ask the AI, or use another tool, to double-check the output. For example: *“Confirm that the above code follows the requirements and explain any part that might not meet the spec.”* This prompt makes the AI review its work and often it will catch if it deviated from your instructions.

In LLMs, hallucinations might also mean the AI creates a file or component you didn’t ask for, or takes some creative liberty that wasn’t intended. Always review AI-generated code for sanity. If something looks too “magical” or unexpected, question it. By managing hallucinations with these strategies, you maintain control over your project and ensure accuracy.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "leveraging-model-insights-know-your-ai-tools)

Leveraging Model Insights (Know Your AI Tools)

Not all AI models are the same, and even the same model can behave differently depending on settings. To get master-level results, it helps to understand the tools at your disposal in LLMs:

Chat Mode vs Default Mode: LLMs provides (as of this writing) a Chat mode (conversational AI assistant) and a Default/Editor mode (which directly applies changes). Use them intentionally. Chat Mode is excellent for brainstorming, discussing design decisions, or debugging – the AI can freely generate ideas or analysis without immediately coding. For example, you might describe an error and in Chat mode say, *“Let’s analyze this error log and figure out what went wrong.”* The AI can then walk through potential causes. Default Mode, on the other hand, is for executing changes (writing code, creating components). A typical workflow might be: outline or troubleshoot in Chat mode, and once you have a plan, switch to Default mode to implement it with a straightforward prompt (since default mode will modify your project files). Knowing when to use each mode keeps your development flow efficient and safe.

Model Capabilities and Limits: If you have options of models (e.g., GPT-4 vs GPT-3.5), use the one appropriate for the task. GPT-4 (often behind Chat mode) can handle more complex instructions and long reasoning chains, but it’s slower and may have higher cost. GPT-3.5 is faster/cheaper but sometimes needs more explicit guidance and may be more prone to errors in complex tasks. If LLMs Labs features allow it, enabling chain-of-thought or higher reasoning can be useful for debugging (e.g., you might literally instruct: “Use chain-of-thought reasoning to identify the root cause of this issue before fixing it.”). The AI will then output a reasoning process which you can follow.

Token Length and Responses: Be aware of the response length. If you ask for a very large output (like a whole module of code), the AI might cut off or lose coherence if it exceeds the token limit. In such cases, break the task into smaller prompts (e.g., generate code for one function at a time). LLMs’s chat or prompt UI might show a warning if output is truncated – that’s a sign to request the remaining part or divide the work.

Formatting and Code Preferences: The AI can adapt to your formatting preferences if you state them. For example, tell it “output code in markdown format” or “follow the project’s ESLint rules” if you have them. It won’t magically know your style guide unless you include it in the context. If you prefer certain naming conventions or patterns, you can mention that in the prompt (this is part of being Explicit). Over time, as the AI sees consistent style in your project, it will mimic it – but giving gentle reminders in prompts can accelerate that alignment.

In summary, treat the AI as a powerful but literal tool. Understand the modes and models you’re interacting with, and always frame your prompts to play to their strengths (structured, detailed input) while guarding against their weaknesses (forgetfulness, verbosity, hallucinations). Now, let’s translate these principles into concrete best practices for using LLMs effectively.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "additional-prompting-tips)

Additional Prompting tips

Finally, let’s cover specific tips and techniques when working in the LLMs platform. These best practices combine the general prompt engineering concepts with LLMs’s features to help you get the best outcome.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "start-with-a-solid-knowledge-base)

Start with a Solid Knowledge Base

Before you even write a prompt, set up your project’s Knowledge Base (in LLMs’s project settings). Include the Project Requirements (PRD), user flows, tech stack details, UI design guidelines, and any backend specifics. This acts as persistent context the AI will always have. For example, if your PRD clearly lists “Out of scope: social login”, the AI is less likely to randomly add a Google login feature. You can also explicitly prompt at the start:

*Before writing any code, read the project Knowledge Base and confirm you understand the app’s purpose and constraints.*

This ensures the AI internalizes your project’s context and reduces irrelevant suggestions or hallucinated features.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "be-specific%2C-avoid-vagueness)

Be specific, avoid vagueness

Vague prompts lead to vague results. Always clarify *what* you want and *how*.

DON’T:

Copy

Make this app better.

Another example:

Copy

Create a form for user input

DO:

The latter gives clear direction on scope and expected outcome.

Copy

Refactor the app to clean up unused components and improve performance, without changing UI or functionality.

Another example:

Copy

Create a user registration form with fields for username, email, and password and include a submit button.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "incremental-prompting)

Incremental prompting

Resist the urge to ask for an entire complex app in one prompt. Break your development process into logical steps and prompt for one at a time.

DON’T:

Copy

Build a CRM app with Supabase, auth, Google Sheets export, and data enrichment.

Build my entire e-commerce app with authentication, product listings, and checkout.

DO:

This step-by-step progression​​ helps the AI stay focused and accurate, and you can catch issues early:

Copy

Set up a Supabase-connected CRM backend.

Copy

Great! Could you please add a secure authentication flow with user roles?

Copy

Thank you! The next step is to integrate Google Sheets to export records.

Another example:

Copy

Set up a database schema for user information.

Copy

Develop an API endpoint to retrieve user data please

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "include-constraints-and-requirements)

Include Constraints and Requirements

Don’t shy away from spelling out constraints. If something *must* or *must not* be done, say so.

Adding constraints

Copy

Create a simple to-do app with a maximum of 3 tasks visible at a time.

Include the ability to add, edit, and delete tasks.

Copy

Optimize this code, but ensure the UI and core functionality remain unchanged. Document each change you make.

Copy

Use at most 3 API calls for this, and ensure no external library is required.

Copy

The page should display a maximum of 3 tasks at a time.

Such limits keep the AI from over-engineering. Adding a constraint like a max number of items or a performance target can focus the AI on what’s important​.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "avoid-ambiguity-in-wording)

Avoid ambiguity in wording

If a term could be interpreted in different ways, clarify it. The clearer you are, the less the AI has to guess.

DON’T:

Copy

Add a profile feature

Copy

Support notifications

DO:

The latter gives clear direction on scope and expected outcome.

Copy

Add a user profile page with fields X, Y, Z.

Copy

Send an email notification on form submission.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "mind-your-tone-and-courtesy)

Mind your tone and courtesy

While it doesn’t change functionality, a polite tone can sometimes yield better results​. Phrases like “please” or a respectful ask can add context and make the prompt a bit more descriptive, which can help the AI. For example,

*Please refrain from modifying the homepage, focus only on the dashboard component.*

This reads as polite, and it explicitly tells the AI what not to do. It’s not about the AI’s feelings – it’s about packing in detail. (Plus, it never hurts to be nice​!)

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "use-lovable%E2%80%99s-modes-intentionally)

Use LLMs’s Modes Intentionally

As mentioned, utilize Chat Mode for planning and Default Mode for building. For example, when starting a new feature, you might enter Chat Mode and brainstorm the component breakdown:

I want to add a blog section to my app. Let’s discuss how to structure the data and pages.

The AI might respond with an outline. Once you’re satisfied, you can switch to Default Mode and say:

Create a BlogPost page and a supabase table or schema for blog posts based on the above plan.

Chat Mode is also great for debugging without consuming your prompt credits on failed edits

e.g., in Chat ask “Why might the login function be returning undefined?” to get analysis, then apply the fix in code with an edit prompt.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "use-formatting-to-your-advantage)

Use formatting to your advantage

Structure lists or steps when appropriate. If you want the AI to output a list or follow a sequence, enumerate them in the prompt. By numbering steps, you hint the AI to respond in kind.

Copy

Let's think through the process of setting up a secure authentication system:

1. What are the necessary components?

2. How should they interact?

3. Provide the implementation code.

*First, explain the approach. Second, show the code. Third, give a test example.*

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "leverage-examples-or-references)

Leverage examples or references

If you have a target design or code style, mention it or provide an example. Providing an example (image or code snippet) gives the AI a concrete reference to emulate.

Setting the context

Copy

We are building a project management tool that helps teams track their tasks.

This tool should have features like:

- user authentication

- project creation

- task assignments

- reporting

Now, for the first task, create the UI for project creation.

Another example:

Copy

I need a CRM app with Supabase integration and a secure auth flow. Start by setting up the backend.

Another example:

Copy

We are developing an e-commerce platform focusing on eco-friendly products. Generate a product listing page with filters for category and price.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "using-image-prompts)

Using image prompts

LLMs even allows image uploads with your prompt, so you can show a design and say “match this style”.

There are two main approaches here. The first one is a simple prompting approach.

Simple image upload prompting

You can upload an image and then add an example prompt like this:

Copy

Create and implement a UI that looks as similar as possible to the image attached.

Copy

This screenshot shows a layout issue on mobile. Adjust margins and padding to make it responsive while keeping the same design structure.

Or, you can help AI better understand the content of the image and some additional specifics about it. Excellent results can be achieved by adding specific instructions to the image uploaded. While the image is worth a thousand words, adding a couple of your own to describe desired functionality can go a long way - especially since interactions cannot always be obvious from a static image.

Image prompting with detailed instructions

Copy

I want you to create the app as similar as possible to the one shown in this screenshot.

It's essentially a kanban clone.

It should have the ability to add new cards (tickets) in each column, have the ability to change the order of those tickets within a single column, and even move those cards between columns.

Feel free to use the Pangea home dnd npm package for drag-and-drop functionality.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "feedback-integration)

Feedback integration

Review the AI’s output and provide specific feedback for refinements.

Copy

The login form looks good, but please add validation for the email field to ensure it contains a valid email address.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "emphasizing-accessibility)

Emphasizing Accessibility

Encourage the generation of code that adheres to accessibility standards and modern best practices. This ensures that the output is not only functional but also user-friendly and compliant with accessibility guidelines.

Copy

Generate a React component for a login form that follows accessibility best practices, including appropriate ARIA labels and keyboard navigation support.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "predefined-components-and-libraries)

Predefined Components and Libraries

Specify the use of certain UI libraries or components to maintain consistency and efficiency in your project. This directs the AI to utilize specific tools, ensuring compatibility and a uniform design language across your application.

Copy

Create a responsive navigation bar using the shadcn/ui library with Tailwind CSS for styling.

[​](https://docs.lovable.dev/tips-tricks/prompting-one)

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "multilingual-prompting)

Multilingual Prompting

When working in a multilingual environment, specify the desired language for both code comments and documentation. This ensures that the generated content is accessible to team members who speak different languages, enhancing collaboration.

Copy

Generate a Python script that calculates the Fibonacci sequence. Provide comments and documentation in French.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "defining-project-structure-and-file-management)

Defining Project Structure and File Management

Clearly outline the project structure, including file names and paths, to ensure organized and maintainable code generation. This provides clarity on where new components should reside within the project, maintaining a coherent file organization.

Copy

Create a new React component named 'UserProfile' and save it as 'components/user-profile.tsx'. Ensure it includes a profile picture, username, and bio section.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "provide-precise-edit-instructions-focus-the-ai)

Provide Precise Edit Instructions (Focus the AI)

By default, when you ask LLMs AI to change something, it might rewrite an entire file or multiple files. To avoid unintended changes, be very specific about *where* and *what* to change. You can use LLMs’s “Select” feature to highlight a component or file, then prompt about that selection only. Or explicitly name the file/component in your prompt. For example:

\_In the \_Header*component, change the signup button’s text to ‘Get Started’ and move it to the left side of the nav bar.*

This way, the AI knows to focus on Header component and just adjust that part. Another trick: tell the AI what not to touch. You might add, “Do not modify any other components or logic unrelated to the header.” This prevents the AI from wandering off and potentially breaking something else. This practice (sometimes called the “Diff & Select” approach) ensures minimal, targeted changes – resulting in faster responses and fewer regression bugs.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "locking-files-workaround)

Locking Files (Workaround)

Currently, LLMs might not have an explicit file-lock feature, but you can simulate it through your prompt wording. If there are critical files that the AI should never alter (maybe a complex component that is working fine), you can repeat an instruction in every prompt like:

*Do not change the authentication.js file.*

By consistently telling the AI to refrain, you reduce the chance of unwanted edits. Similarly, if you only want the AI to work within one part of the project, explicitly constrain it:

\_Focus changes solely on the \_ProfilePage*component; assume all other parts of the app remain as is.*

Being upfront about this in the prompt helps keep the AI within bounds.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "design-and-ui-tweaks)

Design and UI Tweaks

When prompting for UI changes in LLMs, clarity is crucial so you don’t break functionality:

If you want purely visual changes, say so. *“Make the login button blue and 20% larger, but do not alter any of its functionality or onClick logic.”* This ensures the AI doesn’t accidentally rename IDs or change logic while restyling.

For responsiveness (making a design mobile-friendly), guide the AI through a plan. For example: *“Optimize the landing page for mobile: use a mobile-first approach. Start by outlining how each section should rearrange on smaller screens, then implement those CSS changes. Use standard Tailwind breakpoints (sm, md, lg) and avoid custom breakpoints. Ensure nothing in functionality changes, just layout.”* By providing this kind of detailed instruction, you get a thorough adaptation to mobile without breaking the desktop layout.

If you have a design change in mind, describing the desired outcome and any constraints (like “keep the same HTML structure, just update CSS”) will help the AI focus on the right solution. Always test the app after AI design changes to confirm everything still works as expected.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "refactoring-and-optimizing-code)

Refactoring and Optimizing Code

As your project evolves, LLMs’s AI might suggest refactoring to improve performance or maintainability. Prompting for refactoring is an advanced but valuable use-case:

Emphasize no change in behavior: \_“Refactor the *code for clarity and efficiency, but \_ the app’s functionality and outputs must remain identical.”* This tells the AI the refactor should not introduce bugs or feature changes.

You can ask for a refactor plan first: \_“Scan the \_utils/*folder and suggest improvements in code structure or duplication. List changes but do not apply them yet.”* The AI might give you a report of what to improve. Then you can decide which changes to prompt for implementation.

For large-scale refactoring, do it in stages. Prompt one module at a time, test, then move on. This pairs with the step-by-step principle. For example: first refactor the state management logic, later refactor API calls, rather than everything in one go.

After refactoring, it’s wise to prompt a quick post-check: *“Now that the code is refactored, run through a quick checklist: does the UI look the same and do all tests or key flows still pass?”* The AI can self-verify or list things to manually check.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "debugging-with-ai-assistance)

Debugging with AI Assistance

Bugs are inevitable. LLMs has a “Try to Fix” feature for quick fixes, but you can also enlist the AI through prompts:

When an error occurs, copy any error logs or messages into a prompt (in Chat mode ideally) and ask: *“Here’s the error and relevant code snippet – what is causing this and how can we fix it?”* Detailed error context helps the AI pinpoint the issue.

Use the CLEAR principles while debugging: be Explicit about what the code was supposed to do versus what happened. Sometimes just explaining the bug in detail to the AI will lead it to the solution.

If the AI’s first fix doesn’t work, use the Adaptive principle: clarify what changed or provide the new error, and ask it to try again or suggest an alternative approach.

Leverage Chat Mode to discuss the bug: *“The fix didn’t work. The state is still undefined at runtime. What else could be wrong? Let’s think through possible causes.”* You can have a back-and-forth until a plausible solution is found, then apply it in Default Mode.

For UI bugs, you can even share a screenshot (if LLMs supports image input in chat) or describe the visual issue. E.g., “The sidebar is supposed to hide on mobile, but it’s still visible. Here’s the CSS… Why might it be failing?” The AI can reason about CSS or layout issues if given enough info.

Always test after a fix. If it works, consider using Reverse Meta Prompting to have the AI summarize what the root cause was and how to avoid it in the future, enriching your knowledge base.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "when-and-when-not-to-involve-the-ai)

When (and When Not) to Involve the AI

A master prompter knows that sometimes, you don’t need to prompt at all. If a change is extremely small or you already know how to do it quickly (e.g., changing a text label, adjusting one padding value), it can be faster to just do it manually in the code editor. Over-relying on the AI for trivial tasks can slow you down and use up your prompt quota. Use the AI where it adds value – complex logic, boilerplate generation, multi-step operations, or things you’re unsure about. For simpler issues, you might:

Use your own knowledge or a quick search (or even ask ChatGPT outside of LLMs) to figure it out, especially if it avoids burning a prompt on something the AI might misunderstand.

Utilize developer tools: open the browser DevTools console to inspect an element or debug a JavaScript error in real time. Once you identify the fix, you can either implement it directly or confirm via a prompt.

If you notice a button is the wrong color, it might be faster to just fix the CSS class yourself than to describe the issue to the AI and risk it changing more than intended. On the other hand, if you need to implement a new feature from scratch, that’s a perfect job for the AI – you describe the what and why, and it figures out the how in code.

Remember, LLMs’s AI is like an assistant developer. You manage it by giving clear tasks and oversight. It can drastically speed up development, but you remain the lead who reviews and directs the work.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "applying-these-strategies-in-different-tools)

Applying These Strategies in Different Tools

The prompting principles above apply not just in LLMs’s chat, but anywhere you interact with AI or automation tools:

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "in-lovable%E2%80%99s-builder)

In LLMs’s Builder

You’ll primarily use these prompts in the LLMs chat interface to build and refine your app.

Start with a broad project prompt, then iterate feature by feature.

Use Chat-Only mode when you need to discuss or debug without changing code.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "with-make-com-or-n8n-workflow-automation)

W[ith make.com or n8n (workflow automation)](http://make.com/)

You might not prompt these platforms in natural language the same way, but designing an automation still benefits from clear AI instructions.

For instance, you can have LLMs generate integration logic:

[*When a form is submitted, send the data to a Make.com webhook for Slack notification.*](http://make.com/)

In fact, LLMs can help set up automation by integrating with webhooks. If your app needs to hand off tasks (like sending emails, updating a CRM), you can prompt LLMs to use Make or n8n.

[*After a user signs up in the app, trigger a Make.com workflow that creates a record in Salesforce.*](http://make.com/)

LLMs will write the code to call that webhook or API. Keeping the prompt structured ensures the AI knows exactly how to connect LLMs with those external services.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "edge-cases-and-external-integrations)

Edge cases and external integrations

LLMs integrates with many services (Stripe, GitHub, Supabase, etc.). When prompting for these, treat the integration details as part of your Context/Constraints. For example,

*Connect the form to Stripe (test mode) for payments. On success, redirect to /thank-you.*

Be clear about what external services should do. The same goes for using n8n (self-hosted automation) – you might write,

*Send a POST request to the n8n webhook URL after form submission, and wait for its response to show a confirmation message.*

Clarity here is key so the AI produces the correct calls.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "summary)

Summary

Strong prompting is about clarity, structure, and context. Whether you’re telling LLMs to build a feature, or orchestrating a [Make.com](http://make.com/) scenario, the goal is to paint a picture of what you want.

Start with structured prompts if you’re unsure, and evolve to more conversational style as you gain confidence.

Use meta techniques to improve and learn from each interaction.

With practice, you’ll guide the AI like an extension of your dev team – and it will feel natural to get exactly the output you need.

[​](https://docs.lovable.dev/tips-tricks/prompting-one" \l "conclusion)

Conclusion

By now, you should have a solid grasp of how to craft prompts that are clear, effective, and tailored to LLMs’s AI. From the foundational CLEAR principles to advanced strategies like few-shot examples and meta prompting, these techniques empower you to get exactly what you need from the AI – no more, no less. You’ve learned to structure your requests, provide context, avoid pitfalls like hallucinations, and leverage LLMs-specific features (Knowledge Base, Chat mode, etc.) to streamline your workflow.

Master-level prompting is a game changer: it turns AI from a gimmick into a reliable teammate. With practice, you’ll find that you can build apps faster, debug with less frustration, and even explore creative solutions by simply *asking the right questions* and giving the right guidance. The key is to stay smart, concise, direct, and adaptive in your instructions – much like a seasoned engineer communicating with their team.

Finally, always keep learning from each interaction (that Reflective habit). Every prompt/response is feedback for you to refine your technique further. As you continue to build in LLMs, you’ll develop an intuition for what the AI needs to hear to produce great results. Combine that with your own ingenuity, and there’s little you can’t achieve.

Focus on your big ideas – let LLMs’s AI handle the execution details once you clearly tell it what to do.