30 IDEAS. 10 READY-TO-USE PROMPTS.

The AI Prompt Vault for Developers, Students & Builders

Copy-paste prompts that solve real problems in coding, learning, career growth, and building products — built to work in ChatGPT, Claude, Gemini, and any AI assistant you already use.

haas.dev
Engineering mindset over tutorials.

HOW TO USE THIS VAULT

Get more out of every prompt

This resource has two parts. First, **30 prompt ideas** — a menu of high-value, reusable prompts across coding, learning, career, content, and business. Use this list to see what's possible and to know what to search for when you need something specific.

Second, the **10 strongest ideas** are built out into complete, copy-paste-ready prompts. Each one includes the exact prompt, what to customize, a filled-in example, what output to expect, and a pro tip to get a noticeably better result.

Three habits that make any prompt work better:

- Replace every placeholder in [BRACKETS] — a prompt with unfilled placeholders confuses the model as much as it confuses you.
- Paste in real context (your actual code, resume, or notes) rather than describing it — models work better with source material than summaries.
- Treat the first output as a draft. Follow up with “refine [X]” or “go deeper on [Y]” instead of starting a new chat from scratch.

PART ONE

30 Prompt Ideas at a Glance

The ten marked with ★ are fully built out starting on the next section — complete with the exact prompt text, customization notes, and an example.

#	Title	Category	Target user	Level
1	The Code Review Simulator ★	Coding & Software Dev.	Developers before opening a PR	Intermediate
2	The Error Message Decoder ★	Debugging & Error Fixing	Beginners stuck on cryptic errors	Beginner
3	The Learning Path Architect ★	Learning & Study	Self-taught learners	Beginner
4	The Concept Simplifier (Feynman Method)	Learning & Study	Students on hard topics	Beginner
5	The Research Synthesizer	Research	Students & professionals	Intermediate
6	The Second-Brain Note Converter	Personal Productivity	Knowledge workers	Beginner
7	The Resume Bullet Rewriter ★	Resume & Portfolio	Job seekers	Beginner
8	The Mock Interviewer ★	Interview Preparation	Job seekers	Intermediate
9	The Portfolio Case Study Builder	Resume & Portfolio	Developers	Intermediate
10	The SEO Blog Post Outliner ★	SEO & Blogging	Bloggers & marketers	Beginner
11	The Content Repurposing Engine	Content Creation	Creators	Beginner
12	The Startup Idea Validator ★	Startup Ideas & Validation	Aspiring founders	Intermediate
13	The Business Model Canvas Generator	Business & Entrepreneurship	Founders	Intermediate
14	The PRD Generator ★	Product Building	PMs & indie developers	Intermediate
15	The System Design Interview Coach ★	System Design	Engineers, senior interviews	Advanced
16	The Code Documentation Generator	Documentation	Developers	Beginner
17	The Project Scope & Timeline Planner	Project Planning	Freelancers & PMs	Beginner
18	The Marketing Campaign Brief Builder	Marketing	Small business owners	Intermediate
19	The Data Analysis Interpreter	Data Analysis	Analysts & students	Intermediate
20	The Developer Career Roadmap Planner	Developer Career Growth	Junior developers	Beginner
21	The Technical Debt Auditor	Coding & Software Dev.	Dev teams & tech leads	Advanced
22	The Weekly Planning & Prioritization System ★	Personal Productivity	Anyone with too many tasks	Beginner
23	The Cold Outreach Message Writer	Career & Job Search	Job seekers & freelancers	Beginner
24	The API Design Reviewer	System Design	Backend developers	Advanced
25	The Study Session Quiz Generator	Learning & Study	Students	Beginner
26	The Freelance Proposal Writer	Business & Entrepreneurship	Freelancers	Beginner
27	The Competitive Analysis Report	Research	Founders & marketers	Intermediate

#	Title	Category	Target user	Level
28	The Refactoring Advisor	Coding & Software Dev.	Developers	Intermediate
29	The Onboarding Doc Generator	Documentation	Team leads	Intermediate
30	The Negotiation Prep Assistant	Career & Job Search	Job seekers	Intermediate

Why these made the list: each idea solves a recurring problem (not a one-off task), produces output worth saving, and works across ChatGPT, Claude, and Gemini without modification. Prompts dependent on a tool's temporary feature, a trend, or a single-use task were left out on purpose.

PART TWO

The 10 Complete Prompt Resources

Copy the text inside each dark box exactly, then replace every [PLACEHOLDER] before sending it.

PROMPT 1 of 10 • DEBUGGING & ERROR FIXING

The Error Message Decoder

USE CASE

Turns a confusing error message or stack trace into a plain-English explanation and a concrete fix. Built for beginners and anyone debugging outside their usual stack.

THE PROMPT

```
You are an experienced software engineer acting as a debugging mentor.

Here is an error I'm getting:

[PASTE FULL ERROR MESSAGE OR STACK TRACE]

Context:
- Language/framework: [TECH STACK]
- What I was trying to do: [GOAL]
- Relevant code (if any): [PASTE CODE SNIPPET]

Do the following, in order:
1. Explain in plain English what this error means, as if I have never seen it before.
2. Identify the most likely root cause based on the context I gave you.
3. Give me the specific fix, with corrected code if code was provided.
4. List 2-3 other common causes of this same error, in case my situation is different.
5. Tell me one habit or check that would have caught this earlier.

Keep the explanation clear and avoid unnecessary jargon. If you need more information to
be certain, ask for it before guessing.
```

HOW TO USE IT

Paste the exact error text (don't paraphrase it) and the surrounding code. The more of the real stack trace you include, the more specific the diagnosis.

EXAMPLE INPUT

Error: "TypeError: Cannot read properties of undefined (reading 'map')" — Tech stack: React + JavaScript — Goal: rendering a list of users from an API response — Code: the component that renders the list.

EXPECTED OUTPUT

A plain-English cause (the array is undefined on first render, before the API call resolves), a corrected code snippet with a default value or loading guard, a short list of other common triggers, and one prevention habit (e.g. always initialize state with an empty array).

PRO TIP

> Always include the full, unedited error text. Models can often spot the real cause from formatting and line numbers that a paraphrased summary loses.

PROMPT 2 of 10 • CODING & SOFTWARE DEVELOPMENT

The Code Review Simulator

USE CASE

Gives your code a thorough review before you open a pull request, catching bugs, readability issues, and edge cases a teammate would flag. For developers who want feedback without waiting on a colleague.

THE PROMPT

```
Act as a senior [TECH STACK] engineer doing a thorough code review.

Here is the code:

[PASTE CODE]

Context:
- What this code is supposed to do: [GOAL]
- Experience level of the author: [EXPERIENCE LEVEL]
- Constraints (performance, style guide, must not break X): [CONSTRAINTS]

Review it in this order:
1. Correctness — flag any logic errors, edge cases, or bugs.
2. Readability — point out anything a teammate would struggle to follow, and suggest
clearer alternatives.
3. Structure — note any function, file, or naming issues that would hurt maintainability.
4. Performance — flag anything unnecessarily slow or resource-heavy, only if it matters
at this scale.
5. Security — flag anything unsafe (input validation, injection risk, exposed secrets,
etc.).

For each issue, give: the specific line or section, why it matters, and the corrected
code.
End with a short summary: is this ready to merge, or what must change first?
```

HOW TO USE IT

Paste the real code, not a description of it. Set [EXPERIENCE LEVEL] honestly — it changes how much the model explains versus assumes.

EXAMPLE INPUT

Tech stack: Python/FastAPI — Code: a new endpoint that saves user signups — Goal: validate and store an email and password — Constraints: must follow the team's existing error-handling pattern.

EXPECTED OUTPUT

A numbered list of issues grouped by category (correctness, readability, structure, performance, security), each with the flagged line, the reasoning, and a rewritten version — plus a clear merge-ready verdict at the end.

PRO TIP

> Ask a targeted follow-up like “now just fix issues #2 and #4 and give me the full updated file” instead of re-pasting the whole review request.

PROMPT 3 of 10 • INTERVIEW PREPARATION

The Mock Interviewer

USE CASE

Simulates a realistic technical or behavioral interview, one question at a time, with feedback after each answer. For job seekers who want practice pressure before the real thing.

THE PROMPT

```
Act as an experienced interviewer for a [ROLE/TITLE] position at a [COMPANY TYPE, e.g.
mid-size startup / FAANG-style company].

Interview focus: [TOPIC — e.g. behavioral, system design, JavaScript fundamentals, SQL]
My experience level: [EXPERIENCE LEVEL]

Rules for this interview:
1. Ask me one question at a time. Do not move to the next question until I answer.
2. After each answer, give brief feedback: what was strong, what was missing, and how a
stronger answer would sound — then ask the next question.
3. Mix difficulty realistically for this role — don't make every question maximally hard.
4. At the end of [NUMBER] questions, give me an overall assessment: readiness level, top
2 strengths, top 2 gaps to fix before the real interview.

Start the interview now with your first question.
```

HOW TO USE IT

Set the role, company type, and topic before starting — these shape the question difficulty and style far more than any other input. Answer as if it's a real interview; don't ask the model to answer for you.

EXAMPLE INPUT

Role: Junior Frontend Developer — Company type: Series A startup — Focus: behavioral + React fundamentals — Experience level: 1 year professional, self-taught before that — Number of questions: 6.

EXPECTED OUTPUT

A back-and-forth interview flow: one question, your answer, brief targeted feedback, next question — ending in a short readiness assessment with concrete gaps to close before the real interview.

PRO TIP

> Do the interview in one continuous conversation instead of separate chats — the model's feedback gets sharper as it learns your answering patterns across questions.

PROMPT 4 of 10 • RESUME & PORTFOLIO

The Resume Bullet Rewriter

USE CASE

Converts vague, duty-based resume lines into specific, achievement-driven bullets recruiters actually respond to. For anyone updating a resume before applying.

THE PROMPT

```
Act as a resume writer who specializes in [INDUSTRY, e.g. software engineering] resumes
that pass both ATS systems and human review.

Here are my current resume bullets:

[PASTE CURRENT BULLETS]

Target role: [TARGET ROLE/TITLE]

Rewrite each bullet using this formula: Action verb + what you did + how you did it + the
measurable result.
Rules:
- Do not invent metrics I didn't give you — if I have no number for something, rewrite it
for clarity and impact without fabricating one, and flag it so I can add a real number.
- Keep each bullet to one line where possible.
- Avoid generic verbs like "responsible for" or "helped with."
- Tailor the language toward what a [TARGET ROLE/TITLE] hiring manager scans for.

After the rewrite, list which bullets are strongest, which are weakest, and what specific
missing information (metrics, scope, tools) would make the weak ones stronger.
```

HOW TO USE IT

Paste your actual current bullets, even the rough ones. If you have real numbers (users, %, time saved, revenue), include them in your original text so the model can use them instead of guessing.

EXAMPLE INPUT

Current bullet: "Responsible for maintaining the company website and fixing bugs." — Target role: Frontend Developer.

EXPECTED OUTPUT

Rewritten bullets in the action-plus-result format, an honest flag on any bullet that needs a real metric you'll need to supply, and a ranked note on which bullets are interview-worthy versus which still need work.

PRO TIP

> Do this one role at a time. A single resume rewritten generically for "any developer job" performs worse than one tailored to a specific target title.

PROMPT 5 of 10 • LEARNING & STUDY

The Learning Path Architect

USE CASE

Builds a structured, realistic roadmap for learning any skill from your current level to your goal, broken into stages with checkpoints. For self-taught learners who don't know what order to learn things in.

THE PROMPT

```
Act as a curriculum designer and mentor for people learning [SKILL/TOPIC] on their own.

My situation:
- Current level: [EXPERIENCE LEVEL, e.g. complete beginner / know the basics]
- Goal: [GOAL, e.g. get freelance clients / pass a certification / build a specific
project]
- Time available: [HOURS PER WEEK]
- Target timeframe: [TIMEFRAME]

Build me a learning path with these rules:
1. Break it into 4-6 stages, each with a clear theme (not just a list of topics).
2. For each stage: what to learn, why it matters before the next stage, and one project or
exercise to prove I've actually learned it.
3. Flag the 1-2 stages most people get stuck on, and how to push through them.
4. Recommend how to know I'm ready to move to the next stage (a checkpoint, not just
"feeling ready").
5. Do not pad the roadmap with topics that don't serve my stated goal.

End with a realistic estimate of how long this path takes at my available hours per week.
```

HOW TO USE IT

Be specific about your goal, not just the topic — “learn Python to automate my job” produces a very different path than “learn Python for data science.”

EXAMPLE INPUT

Skill: SQL — Current level: knows basic SELECT statements — Goal: land an entry-level data analyst role — Time available: 6 hours/week — Target timeframe: 3 months.

EXPECTED OUTPUT

A staged roadmap (typically 4-6 stages) with what to learn, a proof-of-learning project per stage, the likely sticking points, checkpoint criteria for moving on, and a realistic total time estimate.

PRO TIP

> Come back after finishing each stage and ask the model to evaluate your checkpoint project against the goal before moving on — it catches gaps early instead of at the end.

PROMPT 6 of 10 • STARTUP IDEAS & VALIDATION

The Startup Idea Validator

USE CASE

Stress-tests a business idea against real market questions before you spend time building it. For aspiring founders who want an honest gut-check, not encouragement.

THE PROMPT

```
Act as a skeptical but fair startup advisor who has seen hundreds of early-stage ideas
succeed and fail.

Here is my idea:

[DESCRIBE IDEA IN 2-4 SENTENCES]

Target customer: [WHO IT'S FOR]
Problem it solves: [PROBLEM]

Evaluate it honestly across these dimensions:
1. Problem strength — is this a real, painful, frequent problem, or a mild inconvenience?
2. Market — who else already solves this (directly or indirectly), and why would someone
switch?
3. Willingness to pay — is this the kind of problem people already spend money to solve?
4. Founder fit — based on what I've described, what's the biggest risk in me specifically
executing this?
5. Fastest way to get real evidence — what is the cheapest, fastest test I could run this
week to find out if I'm wrong, before building anything?

Do not soften the feedback to be encouraging. If the idea is weak, say so clearly and
explain exactly why, then suggest what would need to change to make it viable.
```

HOW TO USE IT

Describe the idea the way you'd explain it to a stranger, not the way you'd pitch it — the model can only stress-test what you actually give it.

EXAMPLE INPUT

Idea: A subscription box that sends indie developers a curated tool or resource each month — Target customer: solo/indie developers — Problem: discovering useful tools is time-consuming and scattered across forums.

EXPECTED OUTPUT

A blunt breakdown across problem strength, competition, willingness to pay, founder-fit risk, and one concrete, low-cost validation test you can run this week — not generic encouragement.

PRO TIP

> Follow up with “steelman the strongest version of this idea” after the critique — it's useful to see both the weakest and strongest honest cases before deciding.

PROMPT 7 of 10 • SYSTEM DESIGN

The System Design Interview Coach

USE CASE

Walks through a system design interview question the way a senior interviewer would, pushing on tradeoffs instead of just approving your first answer. For engineers prepping for mid-to-senior interviews.

THE PROMPT

```
Act as a senior engineer conducting a system design interview.

Design prompt: Design [SYSTEM, e.g. a URL shortener / a rate limiter / a notification
system]
My target level: [EXPERIENCE LEVEL, e.g. mid-level / senior]
Constraints given: [SCALE/CONSTRAINTS, e.g. 10M daily users, must be low-latency]

Run this as an interactive interview:
1. Ask me to state my assumptions and clarifying questions first — don't give me the
requirements upfront.
2. Let me propose a high-level design before you respond.
3. Push back with realistic follow-up questions on the weakest parts of my design
(scaling bottlenecks, failure modes, data consistency, tradeoffs I glossed over) — one at
a time.
4. If I get stuck, give me a hint before the answer, not the answer immediately.
5. At the end, summarize: what a strong answer at my target level would have included
that I missed, and rate my readiness.

Start by asking me the first clarifying question I should be asking you.
```

HOW TO USE IT

Set your real target level — a senior-level bar and a mid-level bar expect very different depth on scaling and tradeoffs. Try to actually design it before reading ahead.

EXAMPLE INPUT

System: a rate limiter for a public API — Target level: mid-level backend engineer — Constraints: 5,000 requests/second peak, must work across multiple servers.

EXPECTED OUTPUT

An interactive back-and-forth: clarifying questions, your proposed design challenged with realistic follow-ups (one at a time), hints when you're stuck, and a final readiness summary naming exactly what was missing.

PRO TIP

> Resist the urge to ask for the “ideal answer” upfront. The struggle of being pushed on your own design is what mimics the real interview and builds the skill.

PROMPT 8 of 10 • SEO & BLOGGING

The SEO Blog Post Outliner

USE CASE

Turns a target keyword into a search-optimized blog post outline that's structured to actually rank, not just to contain the keyword. For bloggers and marketers planning content.

THE PROMPT

```
Act as an SEO content strategist who writes for [NICHE/INDUSTRY].

Target keyword: [KEYWORD]
Target reader: [WHO'S SEARCHING FOR THIS, e.g. beginner developers / small business
owners]
Search intent: [INTENT — e.g. how-to, comparison, buying decision]

Build a blog post outline with:
1. 3 title options that include the keyword naturally and would make someone click from a
search results page.
2. A meta description (under 160 characters) that matches the search intent.
3. A section-by-section outline (H2s and H3s) that fully answers what someone searching
this keyword actually wants — not just what's easy to write about.
4. For each section, one sentence on what specific point it needs to make.
5. 3-5 related questions or subtopics worth covering to capture "People Also Ask"-style
search traffic.
6. A suggested internal linking angle — what related topic on the same site this post
should link to.

Do not pad the outline with generic sections like "Introduction" and "Conclusion" without
specifying what they need to actually say.
```

HOW TO USE IT

Give the real target keyword you're trying to rank for, not a topic area — the outline changes significantly based on search intent (someone comparing options writes/reads differently than someone learning basics).

EXAMPLE INPUT

Keyword: "how to debug javascript errors" — Target reader: beginner developers — Intent: how-to / educational.

EXPECTED OUTPUT

Three click-worthy titles, a compliant meta description, a full H2/H3 structure with a one-line brief per section, a short list of related questions to cover, and one concrete internal linking suggestion.

PRO TIP

> After the outline, ask the model to write just the introduction section using your actual voice/examples — outlines from AI are strong, but drafting section-by-section keeps the writing from sounding generic.

PROMPT 9 of 10 • PRODUCT BUILDING

The Product Requirements Doc (PRD) Generator

USE CASE

Turns a rough feature idea into a clear, dev-ready product requirements document. For PMs and indie developers who need to go from idea to build-ready spec quickly.

THE PROMPT

```
Act as a senior product manager writing a PRD for an engineering team to build from.

Feature idea: [DESCRIBE THE FEATURE IN A FEW SENTENCES]
Product context: [WHAT THE PRODUCT IS / WHO USES IT]
Constraints: [CONSTRAINTS — e.g. must ship in 2 weeks, mobile-first, no new dependencies]

Write a PRD with these sections:
1. Problem statement — what user problem this solves and why it matters now.
2. Goals and non-goals — explicitly state what this feature will NOT do, to prevent scope
creep.
3. User stories — written as "As a [user], I want to [action], so that [benefit]."
4. Requirements — split into Must-have, Should-have, and Nice-to-have.
5. Edge cases — the situations most likely to be missed (empty states, errors,
permissions, etc.).
6. Success metrics — how we'll know this feature actually worked after launch.
7. Open questions — anything that needs a decision before engineering can start.

Keep it concrete and specific enough that an engineer could start estimating work from
it, not vague enough to mean anything.
```

HOW TO USE IT

Describe the feature the way you'd explain it in a Slack message to a teammate — rough is fine. The model needs your real constraints (timeline, platform, team size) more than a polished pitch.

EXAMPLE INPUT

Feature: let users save items to a "favorites" list — Product: a recipe discovery app — Constraints: 1 backend + 1 frontend developer, must ship in 2 weeks.

EXPECTED OUTPUT

A structured PRD with a clear problem statement, explicit non-goals, user stories, must/should/nice-to-have requirements, a list of edge cases, measurable success metrics, and open questions to resolve before build.

PRO TIP

> Send the finished PRD back with “what would an engineer push back on in this doc?” — it surfaces ambiguity before your team's actual planning meeting does.

PROMPT 10 of 10 • PERSONAL PRODUCTIVITY

The Weekly Planning & Prioritization System

USE CASE

Turns a messy brain-dump of tasks into a prioritized, realistic weekly plan. For anyone with more on their plate than hours in the week.

THE PROMPT

```
Act as a productivity coach who prioritizes based on impact, not just urgency.

Here is everything on my plate this week:

[LIST/PASTE ALL TASKS, MESSY IS FINE]

My biggest goal this week: [GOAL]
Hours I realistically have available: [HOURS]
Hard deadlines (if any): [DEADLINES]

Organize this into a plan using these rules:
1. Sort tasks into: Must do this week, Should do if time allows, Can wait / doesn't need
to happen this week.
2. For "Must do," identify which 1-3 tasks would move me closest to my biggest goal, even
if other tasks feel more urgent.
3. Flag anything on my list that looks like it's not actually necessary, or could be
delegated, shortened, or skipped — and say why.
4. Give me a rough day-by-day distribution based on my available hours, not just one big
list.
5. Point out if I've listed more than I can realistically finish, and tell me honestly
what to cut.

Be direct if my list is unrealistic for the time I have — don't just fit everything in to
be agreeable.
```

HOW TO USE IT

Dump the real, messy list — don't pre-sort it yourself. Part of the value is the model catching what you'd naturally over-prioritize because it feels urgent.

EXAMPLE INPUT

Tasks: finish client report, respond to 12 emails, gym 3x, plan sister's birthday, fix a bug in a side project, read one chapter of a course — Goal: ship the client report — Hours available: 25 — Deadline: client report due Friday.

EXPECTED OUTPUT

A sorted plan (must/should/can-wait), the 1-3 tasks most tied to your stated goal, honest flags on anything that can be cut or delegated, and a rough day-by-day distribution that fits your real available hours.

PRO TIP

> Run this at the start of every week with the same phrasing — consistency makes it easy to spot patterns in what keeps landing in “can wait” and whether it should be removed from your life entirely.

KEEP BUILDING

More prompt packs at haas.dev

This vault is part of an ongoing library of practical resources for developers, students, and builders using AI to move faster — without losing the underlying skill. New prompt packs, curriculum PDFs, and guides are added regularly.

— haas.dev
Engineering mindset over tutorials.