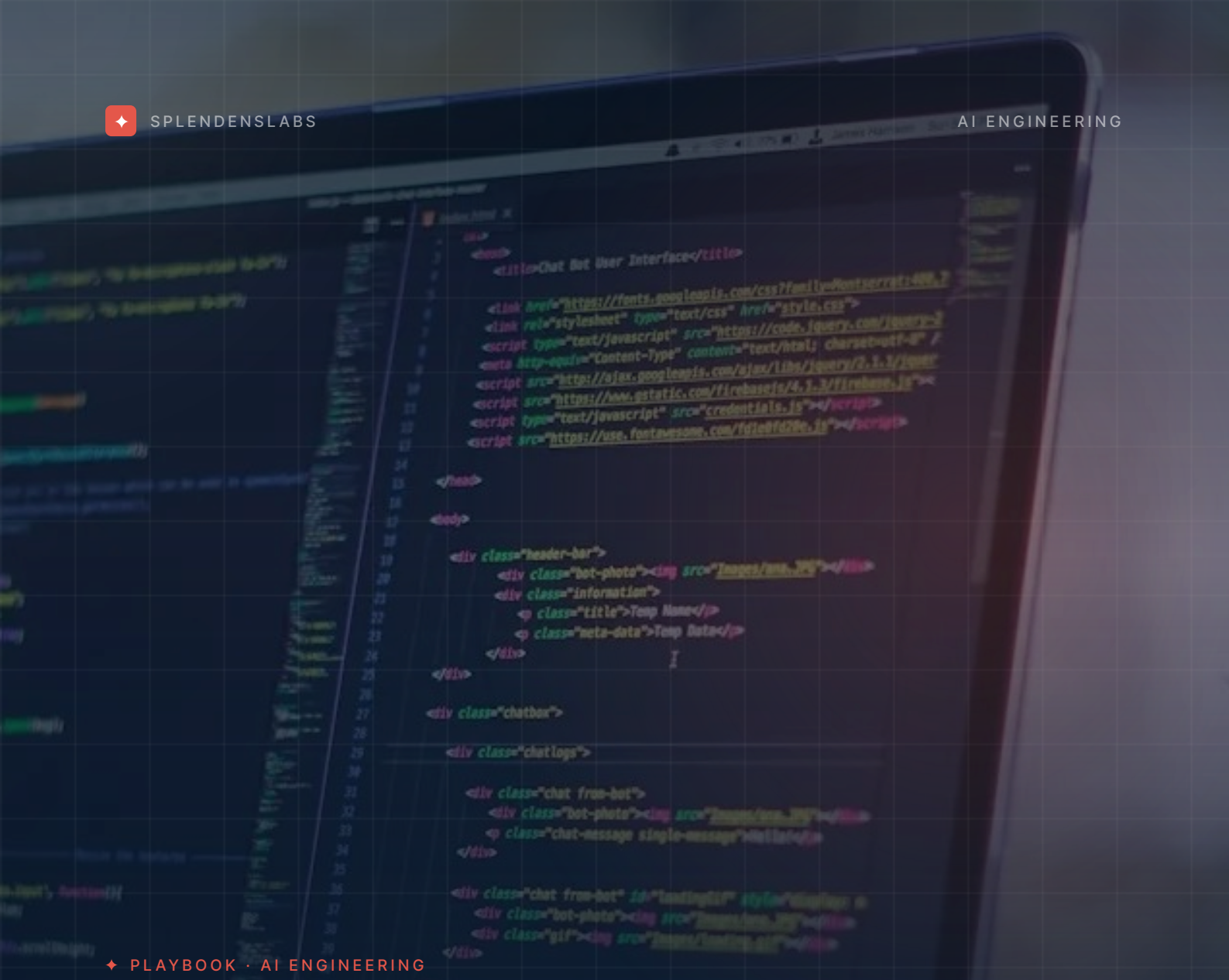


✦ PLAYBOOK · AI ENGINEERING

Prompt Engineering

How to write, structure, version and test prompts like production code: system prompts, examples, structured output, chaining and evals.

FORMAT	LEVEL	READ TIME	UPDATED
Playbook	Beginner	14 min	Oct 2026

OVERVIEW

What this playbook covers

A prompt is the specification for a language-model component. Treated casually, it becomes a pile of patches nobody dares to touch. This playbook covers the habits that make prompts predictable: a clear structure, a deliberate system prompt, well-chosen examples, schema-constrained output, decomposition into smaller steps, and a test suite that tells you whether a change made things better or worse.

KEY TAKEAWAYS

- Write prompts as specifications: role, task, context, constraints and output format, in that order.
- A handful of varied, realistic examples teaches format and judgment better than paragraphs of rules.
- Ask for structured output against a JSON schema and validate it in code — never parse prose with regex.
- Split complex tasks into a chain of small prompts, each with one job you can test on its own.
- Version prompts in source control and gate every change on an eval set, exactly like code.

WHO IT'S FOR

Engineers adding LLM features to products

Product managers writing prompt specs

QA engineers asked to test AI features

Tech leads reviewing prompt changes

HOW TO USE IT

Read sections 01–03 before writing your first production prompt. Use sections 05–06 to set up versioning and evals, and the anti-pattern checklist as a code-review aid.

CONTENTS

01 Anatomy of a production prompt
02 System prompts and examples
03 Structured output and JSON schemas
04 Decomposition and chaining
05 Versioning prompts like code
06 Testing prompts with evals
07 Anti-patterns to catch in review

01 / STRUCTURE

Anatomy of a production prompt

Good prompts read like a brief to a capable new colleague: who they are, what to do, what they need to know, what they must not do, and what to hand back.

LAYER 01 **Role**	Who the model is acting as	Audience it serves	Tone and register
LAYER 02 **Task**	One clear instruction	Definition of done	Why the task matters
LAYER 03 **Context**	Retrieved documents	User profile & state	Delimited with tags
LAYER 04 **Constraints**	What to refuse	What to do when unsure	Length & style limits
LAYER 05 **Output**	Format or JSON schema	Field definitions	Example of a valid response

01

Delimit your inputs

Wrap documents, user text and data in clearly named tags such as `<document>` so the model never confuses instructions with content.

02

Say what to do, not only what to avoid

“Answer in two sentences” works better than “don’t be verbose”. Positive instructions are easier to follow and to test.

03

Give the reason

A short explanation of why a rule exists helps the model generalize it to cases you did not anticipate.

04

Plan for uncertainty

Tell the model exactly what to output when information is missing, so it does not fill the gap by guessing.

✦ 02 / SYSTEM PROMPTS

System prompts and examples

The system prompt sets stable behavior for every turn; examples show the model what good looks like. Together they do more than any amount of clever wording in the user message.

BELONGS IN THE SYSTEM PROMPT	BELONGS IN THE USER TURN
Role, audience and tone	The specific request
Standing rules and refusals	Retrieved context for this request
Output format and schema	Per-request parameters
Tool usage policy	Data the user supplied

Choosing few-shot examples

01 Use real inputs

Pull examples from production logs or realistic samples, not tidy cases written for the prompt.

02 Cover the edges

Include at least one ambiguous input and one where the right answer is to decline or ask for clarification.

03 Vary the surface

Mix lengths, phrasings and topics so the model learns the pattern rather than copying one example.

04 Keep them consistent

Every example must follow the exact output format you ask for — the model will copy inconsistencies.

05 Keep them out of the eval set

Examples used in the prompt cannot also be used to measure it, or your scores will flatter you.

✦ FIELD NOTE

Examples outrank instructions

When an example contradicts a written rule, models tend to follow the example. If output drifts from your rules, check the examples before rewriting the instructions.

✦ 03 / OUTPUT

Structured output and JSON schemas

If code consumes the output, define it as a schema. Most major model APIs now support structured outputs or tool definitions that constrain the response to a JSON schema; use them, then validate anyway.

01 Define schema in code → 02 Pass schema to the API → 03 Model returns JSON → 04 Validate against schema → 05 Retry or fall back on failure

Schema design rules

RULE	WHY	EXAMPLE
Use enums for categories	Prevents near-duplicate labels	`"priority": "low" | "medium" | "high"`
Allow an explicit unknown	Stops the model inventing values	`"due_date": null` when not stated
Describe every field	Descriptions act as instructions	“ISO 8601 date the customer requested”
Put reasoning before the answer	Lets the model think before committing	`reasoning` field ahead of `decision`
Keep nesting shallow	Deep trees raise error rates	Flatten to two levels where possible

Validate with the same schema library your application uses (for example Pydantic or Zod). On a validation failure, retry once with the error message included, then fall back to a safe default or a human queue — never pass unvalidated output downstream.

04 / DECOMPOSITION

Decomposition and chaining

One prompt that classifies, retrieves, reasons, drafts and formats is hard to debug and impossible to test in parts. Break it into a chain where each step has one job.

Example: support-ticket reply chain



PATTERN

Sequential chain

Each step's output is the next step's input. Simple to trace; latency adds up.

PATTERN

Router

A cheap classifier sends each request to a specialized prompt or model.

PATTERN

Parallel fan-out

Independent sub-tasks run at once and a final step merges results.

PATTERN

Generate then critique

A second prompt reviews the draft against explicit criteria and requests fixes.

PATTERN

Extract then reason

Pull the relevant facts first, then reason only over the extracted facts.

PATTERN

Deterministic steps in code

Anything a function can do reliably — math, lookups, dates — stays out of the prompt.

TRADE-OFF

Chains cost latency and tokens

Every extra call adds time and spend. Start with the fewest steps that you can test independently, and merge steps back together when evals show no loss.

05 / VERSIONING

Versioning prompts like code

A prompt edited live in a dashboard is an untracked production change. Prompts deserve the same lifecycle as any other source file.

01 **Store prompts in the repository**

Keep templates as files next to the code that calls them, with variables named explicitly.

02 **Pin the model version**

Record the exact model identifier and sampling parameters with each prompt version — a model upgrade is a prompt change.

03 **Review changes in pull requests**

Show the diff and the eval results side by side so reviewers judge behavior, not wording.

04 **Log the version on every call**

Each request trace records prompt version and model, so any output can be traced to its source.

05 **Roll out gradually**

Ship new versions behind a flag or to a slice of traffic, and keep the previous version one switch away.

> If you cannot say which prompt produced an answer, you cannot fix it when it is wrong.
>
> SPLENDENSLABS ENGINEERING PRINCIPLE

06 / TESTING

Testing prompts with evals

“It looked fine on the five examples I tried” is not a test. An eval set turns prompt work into measurable engineering.

CHECK TYPE	HOW IT IS SCORED	GOOD FOR
Schema validity	Parse and validate in code	Every structured-output prompt

CHECK TYPE	HOW IT IS SCORED	GOOD FOR
Exact / set match	Compare to expected label	Classification and extraction
Assertion rules	Regex, keyword or length checks	Format, banned phrases, required fields
LLM-as-judge	Rubric-scored by a second model	Tone, helpfulness, faithfulness
Human review	Sampled outputs graded by experts	Calibrating the judge; high-stakes tasks

Eval set checklist

- [ ] Built from real or realistic inputs, including messy and adversarial ones
- [ ] Each case has an expected output or a clear pass/fail rubric
- [ ] Covers cases where the correct behavior is to refuse or ask a question
- [ ] Runs automatically on every prompt, model or parameter change
- [ ] LLM judges are spot-checked against human grades before you trust them
- [ ] New production failures are added as cases so they never regress

✦ 07 / ANTI-PATTERNS

Anti-patterns to catch in review

These show up in almost every prompt codebase that grew without discipline. Use the list as a review checklist.

WORDING

- [ ] Instructions shouted in capitals instead of explained
- [ ] Contradictory rules added over time and never reconciled
- [ ] Vague asks such as “be accurate” with no definition

STRUCTURE

- [ ] User data mixed into instructions without delimiters

- [ ] One giant prompt doing five jobs
- [ ] Business logic the code could do left to the model

OUTPUT

- [ ] Prose parsed with regex instead of a schema
- [ ] No defined behavior for missing information
- [ ] Unvalidated output passed to downstream systems

PROCESS

- [ ] Prompts edited in production without version control
- [ ] Changes judged on a few hand-picked examples
- [ ] Model upgraded without rerunning evals

✦ GET IN TOUCH

Let's Talk

We help product teams turn fragile prompts into tested, versioned LLM components with eval suites that gate every release.

- ✦ Prompt and output-schema review
- ✦ Eval set design and CI integration
- ✦ Prompt chain and routing architecture

BOOK AN AI STRATEGY SESSION →

EMAIL
hello@splendenslabs.com

WEB
splendenslabs.com

OFFICE
Sona Incubation Foundation, Salem