Prompt Engineering Field Guide

Tools Library

September 17, 2025

1 Introduction

Prompt engineering is the disciplined process of steering large language models (LLMs) toward dependable, factual, and controllable responses. Well-crafted prompts close the gap between business intent and model capability, accelerating development of writing assistants, analytical copilots, and automated support flows.

In day-to-day practice prompt engineering serves as:

- **A specification bridge**: mapping product requirements onto precise model instructions.
- **An experience console**: constraining tone, structure, and reasoning paths for consistent outputs.
- **An optimization lever**: combining experimentation logs and regression suites to refine prompts over time.

2 Getting Started

A productive workflow begins with dependable tooling, followed by lightweight experiments to understand how the model reacts.

2.1 Environment Setup

Table **??** summarizes a baseline setup that keeps experiments reproducible and shareable.

Automate the setup via scripts (Makefile, PowerShell, Python) so newcomers can bootstrap with one command.

2.2 First Experiments

With tooling ready, run small-scale tasks to observe model behavior. The following loop is a reliable starting point:

1. **Define the persona**: "You are a senior technical writer" or similar context.
2. **State the goal**: Clarify the data source, task boundaries, and expected output format.
3. **Show exemplars**: Provide one or two ideal answers to convey tone and structure.

Table 1: Recommended setup components

Component	Notes
Models and APIs	Request access to OpenAI, Azure OpenAI, Anthropic, or local open-source models; record API keys and rate limits.
Development tooling	Use editors such as VS Code or Cursor; install Python/JavaScript SDKs and CLI utilities for quick calls.
Version control	Track prompts and scripts with Git; branch per experiment for clean diffs.
Experiment logs	Capture prompts, responses, metrics, and reflections in Markdown or notebooks.
Team knowledge base	Centralize best practices and reusable assets in a shared repo or wiki.

4. **Log findings**: Score accuracy, completeness, tone, and hallucination risk after each run.

Example interaction:

```
System: You are a senior legal advisor who explains contract terms plainly.
User: Summarize the core obligations and carve-outs from the following clause in <=12
Expected format: 1) Key obligations 2) Carve-outs 3) Risk warnings.
```

Vary tone, exemplars, and structure to see which levers meaningfully change the response.

3 Guidelines

The sections below adapt the Prompt Engineering Guide into actionable patterns with concrete illustrations.

3.1 Prompt Principles

Effective prompts exhibit clarity, structure, and data alignment.

1. **Make the goal explicit**: Name the task, audience, and quality criteria.
2. **Structure complex work**: Break instructions into ordered steps, headings, or tables.
3. **Anchor on facts**: Embed critical figures or references to limit speculation.
4. **Set hard constraints**: e.g., "Return valid JSON only" or "Avoid subjective judgment."

Sample prompt:

```
Task: Analyze the incident report and return JSON only.
Requirements:
1. Extract root causes with supporting quotes.
2. Respond using {"impact":[], "root_causes":[], "actions":[]}.
3. If information is missing, insert null and explain why.
```

3.2 Iterative Refinement

High-quality prompts emerge from tight experimentation cycles.

1. **Assemble a replay set**: Collect representative inputs for your mission-critical flows.
2. **Change one lever at a time**: Adjust tone, structure, or exemplars individually to isolate effects.
3. **Quantify success**: Define measurable criteria such as correct fields or fact accuracy.
4. **Automate regression**: Batch-call prompts via scripts and archive metrics per revision.

Iteration log example:

```
v0: Lacked citations -> add explicit "quote the source" requirement.
v1: Quotes too long -> constrain to <=30 words.
v2: Stable across replay set -> promote to production.
```

3.3 Summarizing

Summaries need well-defined scope and perspective.

Design checklist:

- Specify target length, audience, and focus points.
- Provide a reference summary to demonstrate voice and formatting.
- Request a list of missing questions or uncertainties for follow-up research.

Prompt example:

```
Summarize the following release notes in <=150 words for product managers:
- Emphasize user-facing impact
- Present as bullet points
- Flag hypotheses that remain unvalidated
Source notes: [...]
```

3.4 Inferring

Inference tasks revolve around labeling, classification, or judgement calls.

Best practices:

- Provide label definitions and mutual exclusivity rules.
- Require supporting evidence or quotations to reduce speculation.

- Offer an “uncertain” option for ambiguous cases.

Prompt example:

```
Determine the sentiment label for the review using {"positive","neutral","negative","
Return JSON with:
- label
- evidence (quoted text)
- confidence (0-1 float)
Review: [...]
```

3.5 Transforming

Transformation keeps semantics intact while changing representation.

Recommendations:

- Include explicit input and output examples, especially JSON keys or table headers.
- Define error handling, e.g., output ”missing” for absent fields.
- Remind the model not to add commentary beyond the requested format.

Prompt example:

```
Convert the CSV rows below into JSON Lines.
Input sample:
name,email,role
Zhang Yan,zhang@example.com,Account Manager
Output: one JSON object per line with keys {"name","email","role"}.
```

3.6 Expanding

Expansion tasks require balancing creativity with narrative coherence.

Guidance:

- State the expansion goal: add detail, extend the storyline, or elaborate on arguments.
- Fix tone, reading level, and length limits.
- Ask the model to highlight how new content connects to the source.

Prompt example:

```
Extend the following paragraph to ~300 words for first-year university students in an
- Preserve the original thesis
- Begin each new paragraph with a topic sentence
Source text: [...]
```

3.7 Chatbot

Designing a chatbot prompt involves persona, memory, and guardrails.

Checklist:

- **Persona**: Define background, tone, and knowledge boundaries.
- **Dialogue state**: Describe how to summarize history, when to call tools, and how to ask clarifying questions.
- **Refusal policy**: Supply patterns for declining sensitive or out-of-scope requests.
- **Memory management**: Specify when to condense or drop older turns.

Prompt fragment:

```
System: You are Lucy, a digital banker who only answers personal savings questions.
- If the user asks about loans or investments, decline politely and route to humans.
- After 3 turns without key details, summarize current info and ask for what is missi
User: I'm curious about credit card perks...
```

3.8 Wrap-up

A resilient prompt engineering workflow can be summarized as "Set the goal -> Structure instructions -> Iterate intentionally -> Regression test."

To institutionalize the practice:

- Maintain a shared prompt template and example library.
- Track quality metrics over time to catch regressions across model versions.
- Encourage team reviews so everyone converges on consistent bar-raising standards.

Resources

- Prompt Engineering Guide
- OpenAI Prompt Engineering Guide
- Prompting Guide