

Mastering Individual Calls to LLMs

Fabio Casati · University of Trento

UI vs API

- *With both, you have 1. defaults and 2. controls.*
- *But – they vary. And you must make choices intentionally*
- *New features come out EVERY DAY*

Capability	Chat UI	API
Tools	Product-managed	Developer-defined
Memory	Product-managed	Developer-managed (if you want)
Structured outputs	Best-effort	Enforced via JSON Schema
...	...	...

Anatomy of an API Call

01

System Message

Sets the persona, rules, and constraints the model follows throughout the conversation. Think of it as the model's "job description."

02

User Message

The actual task or question. Includes the input data, instructions, and any examples (few-shot) you want the model to follow.

03

Parameters

Temperature, top_p, max_tokens, stop sequences, and more. These knobs control how the model generates its response.

Your First API Call

```
from openai import OpenAI

client = OpenAI()

response = client.chat.completions.create(
    model="gpt-4.1-mini",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "What is the capital of France?"},
    ],
    temperature=0.0,
)

print(response.choices[0].message.content)
# >>> The capital of France is Paris.
```

1

System vs. User Messages

Two distinct roles that frame every API call.

System Message

Purpose: Defines the model's persona, tone, constraints, and output format.

Convention: You resend it every call, but keep it the same across turns. The API is stateless.

Tip: Put guardrails and formatting rules here so they can't be overridden by user input.

User Message

Purpose: The actual task, question, or input data for this specific request.

Changes: Every turn. Each new request or follow-up goes here.

Tip: Include few-shot examples, context, and delimiters in the user message for best results.

System = stable rules. User = variable input. Separate concerns cleanly.

System Message Examples

SYSTEM MESSAGE

"You are a formal financial analyst. Be precise and use technical terminology."

USER MESSAGE

"Explain what an API is."

OUTPUT

An Application Programming Interface (API) is a formally defined contract specifying how software components exchange data and invoke operations...

SYSTEM MESSAGE

"You are a friendly teacher explaining concepts to a 10-year-old. Use analogies."

USER MESSAGE

"Explain what an API is."

OUTPUT

Think of an API like a waiter in a restaurant! You tell the waiter what you want, they go to the kitchen, and bring back exactly what you asked for...

Same question, same model, same temperature — completely different tone and depth.

2

Temperature: The Creativity Dial

Low Temperature (0.0 – 0.3)

Behavior: Picks the most likely token **almost** every time. Output is **almost** consistent and repeatable.

Best for: Classification, data extraction.

High Temperature (0.7 – 1.5)

Behavior: Flattens the probability distribution, giving less-likely tokens a real chance. Output varies across runs.

Best for: Creative writing, brainstorming, dialogue, **exploration**, **generating diverse options**.

Rule of thumb: start at 0.0 for precision tasks, 0.7 for creative tasks. Adjust from there.

Temperature in Action

temperature = 0.0

PROMPT

"Write a one-sentence
description of the ocean."

RUN 1

*The ocean is a vast body of
saltwater that covers over 70%
of the Earth's surface.*

RUN 2

*The ocean is a vast body of
saltwater that covers over 70%
of the Earth's surface.*

temperature = 0.7

SAME PROMPT

RUN 1

*The ocean is an immense,
ever-shifting expanse of
saltwater teeming with life.*

RUN 2

*A vast, interconnected body
of water shaping climate and
harboring diverse ecosystems.*

Similar but varied

temperature = 1.5

SAME PROMPT

RUN 1

*Beneath turquoise veils the
ocean hums ancient lullabies
to forgotten shores.*

RUN 2

*Sapphire wilderness sprawling
beyond horizons, cradling
mysteries in salted breath.*

Wildly different

temp=0.0 → deterministic, same every time temp=0.7 → balanced variety temp=1.5 → creative chaos

3

Some useful (?) API Parameters

Six knobs that control how the model generates its response.

top_p (Nucleus)

max_tokens

stop sequences

frequency_penalty

Penalizes tokens proportional to how often they've appeared. Reduces repetition in long-form text.

presence_penalty

Flat penalty for any token that has appeared at all. Encourages the model to explore new topics.

Default values work for most cases. Tune one parameter at a time and measure the impact.

Prompting Cookbook

Reusable Patterns for Modern LLMs

reliableai.github.io

Fabio Casati · University of Trento

The Key Meta-Insight

01

Calibrate Specificity

Find the Goldilocks zone between brittle over-specification and vague under-specification. Anthropic calls this "the right altitude."

02

Models Follow Literally

Claude 4.x does exactly what you ask — no more, no less. If you want thoroughness, ask for it. If you want brevity, specify it.

03

Cut Redundancy, Not Structure

Remove hedging and over-explanation. Keep output format, constraints, audience, and verification criteria.

Prompting Cookbook — Methods

Prompt Foundations

Prompt Skeleton

Direct Instruction & Role Prompting

Scope & Structure

Reasoning Techniques

Few-Shot Prompting

Chain of Thought

Long-Context Placement

Structured Output

Schema Extraction

Reasoning Depth

Quality & Production

Quality Patterns

Reflection Loops

Prompt Chaining

Error Handling & Retries

Guardrail Models

And now, the Real Key Insights

01 Ask the model – iterate – “fine tune” the prompt

02 Don't trust people with with “key meta-insight slides”

Just try stuff

A Possible Prompt Skeleton

ROLE

Expert stance (optional)

OBJECTIVE

One-sentence goal

CONTEXT

Facts, input data

CONSTRAINTS

Audience, length, tone

OUTPUT FORMAT

Exact structure

VERIFY

Quality gate

Claude prefers XML tags (<context>, <instructions>). OpenAI works with both XML and LABEL: sections.

Direct Instruction & Role Prompting

Direct Instruction

The baseline. Clear, literal instructions for simple tasks.

```
Summarize for a VP:  
- exactly 5 bullets  
- include 1 risk + 1 recommendation  
Text: [paste]
```

If output drifts, add Recipe 5 (output clamp).

Role Prompting

Shapes tone, vocabulary, depth. Does NOT improve factual correctness.

```
Act as a senior SRE.  
Return:  
1) Findings (major)  
2) Missing evidence  
3) Follow-ups (top 5)
```

Gemini takes personas very seriously — test for conflicts.

Scope & Structure



Scope Discipline

Set explicit boundaries. "Fix this function" without scope may trigger a full refactor.

Implement EXACTLY this and nothing else.
If ambiguous, pick simplest valid interpretation.



Structured Formatting

Delimiters prevent the model from confusing instructions with data.

Separate instructions from data.
Put long data first, instructions last.

Format	Example	Best for
XML tags	<code><context>...</context></code>	Claude, OpenAI
LABEL: sections	<code>ROLE: / CONSTRAINTS:</code>	Quick UIs, Gemini 3
Markdown	<code>## Instructions</code>	General readability
Triple quotes	<code>"""content"""</code>	Data vs instructions

Few-Shot Prompting

Show the model what you want by providing examples in the prompt itself.

The Technique

Zero-shot: No examples. Relies entirely on the model's training.

One-shot: One input–output pair shows the pattern.

Few-shot (3–5): Multiple examples lock in format, tone, and edge cases.

Best Practices

Diverse examples: Cover typical cases AND edge cases.

Consistent format: Use identical delimiters (Input: / Output:) in every example.

Order matters: Place the most similar example last, closest to the actual task.

Few-shot is the single most reliable way to control output format without fine-tuning.

Chain of Thought (CoT)

Force the model to show its work before giving a final answer.

Why It Works

LLMs generate tokens sequentially. When you force intermediate reasoning steps, each subsequent token is conditioned on better context.

Key insight: The model "thinks" by writing. No writing = no thinking. CoT gives the model scratch space.

CoT Variants

Zero-shot CoT: Add "Let's think step by step" to trigger reasoning.

Structured CoT: Prescribe steps: "First analyze X, then evaluate Y, finally conclude Z."

Few-shot CoT: Provide examples that include reasoning traces, not just answers.

CoT is most impactful for math, logic, multi-step reasoning, and complex decision-making tasks.

Long-Context Placement

Where you place instructions relative to data significantly affects quality.

Poor Placement (maybe)

Answer these questions:

1. Retention period?
2. Backup procedures?

Document:

[15,000 words of policy...]

Good Placement (maybe)

```
<document>
[15,000 words of policy...]
</document>
```

Based on the document above:

1. Retention period?
 2. Backup procedures?
- Cite relevant section numbers.

The rule: Long data first → instructions last → query at the very end

Schema Extraction

The biggest practical difference between UI and API.

UI: Best-Effort

Extract into JSON with
exactly these keys:
party_name, jurisdiction,
effective_date,
termination_summary

If unknown, set null.
No extra keys.

API: Enforced

OpenAI: text.format with
strict: true → guaranteed shape

Anthropic: Prefilling + tool-use
schemas → enforced structure

If you're building a pipeline that parses model output, always use API-level enforcement.

Reasoning Depth

Trade off latency/cost vs. correctness. Not all tasks need deep reasoning.

	OpenAI	Anthropic	Google
Knob	reasoning.effort (none/low/med/high)	thinking.budget_tokens (extended thinking)	Model selection (Flash vs Pro)
Best for	Granular effort tuning	Complex multi-step; interleaved thinking	Binary model choice

Quality Patterns



Answer → Review → Re-answer

A single-call self-critique: Draft your answer, critique it for errors and gaps, then revise. All three steps in one API call — no orchestration needed.

Self-check criteria:

- Especially powerful for
- factual tasks, math, and
- any output that can be
- objectively verified.



Evaluation-First

Define success before proposing solutions. The prompting equivalent of test-driven development.

Before proposing, do this:

- 1) Define success metrics
- 2) Define baseline + thresholds
- 3) Propose 3 interventions
- 4) Predict impact + validation plan

Reflection Loops

Multi-turn self-improvement: ask the LLM to evaluate and iterate on its own output.

The Loop

Call 1: Generate initial answer.

Call 2: "Here is your previous answer. Rate it 1–10 on [criteria]. What would you improve?"

Call 3: "Apply those improvements and produce the final version."

Repeat calls 2–3 until quality plateaus or budget is reached.

When to Use

High-stakes output: Legal drafts, medical summaries, customer-facing content.

Latency-tolerant: Batch processing, async workflows where extra calls are acceptable.

vs. Answer-Review-Reanswer: That pattern is single-call; reflection loops use multiple calls for deeper iteration.

Set a max iteration count (2–3 rounds). Diminishing returns kick in fast.

Prompt Chaining

The bridge between "prompting" and "building an AI system."

Call 1

EXTRACT

Read 5 paper abstracts.
For each: research question,
methodology, key finding.
Return as JSON array.

effort: low



Call 2

ANALYZE

Group by methodology.
Identify agreements,
contradictions, and gaps.
Structured comparison.

effort: medium



Call 3

GENERATE

Write a 500-word literature
review. Academic tone.
Highlight the key
unaddressed gap.

effort: high

Why chain? Verify intermediate results · Different models/reasoning per step ·
Clear debugging surface · Prevent corner-cutting

Error Handling & Retries

Production LLM calls fail. Plan for it.

Common Failures

Rate limits (429): Back off exponentially. Use jitter to avoid thundering herds.

Timeouts: Set aggressive timeouts (30–60s). Retry with the same prompt; non-determinism means a retry may succeed.

Malformed output: JSON didn't parse? Retry once. If it fails again, fall back to a simpler prompt or model.

Robustness Patterns

Retry with backoff: 1s → 2s → 4s with random jitter. Max 3 retries for most use cases.

Model fallback: If GPT-4o fails, fall back to GPT-4o-mini. Graceful degradation beats total failure.

Validation layer: Always validate output before using it. Schema checks, length checks, safety checks.

Start simple.

Escalate on failure.

Measure everything.

Patterns persist across model updates; knobs don't.

Sources: Anthropic Prompting Best Practices · OpenAI GPT-5.x Guides · Google Gemini 3 Prompting Guide

Fabio Casati · University of Trento