OpenAI API Guide PDF

To use the OpenAI API, create an API key, store it as an environment variable, send a request with a model name and a list of messages, read the generated text from the response and handle errors and rate limits. This guide explains the request structure, key parameters, structured outputs, tool calling, cost control and safe handling of keys and data.

Key takeaways

- Keep API keys secret in environment variables, never in code or the browser.
- A request has a model, input messages and optional parameters such as temperature.
- Ask for structured JSON output when your code must parse the answer.
- Tool calling lets the model request functions that your code runs.
- Control cost by limiting tokens, caching and choosing the right model; check official pricing.

What is the OpenAI API?

The OpenAI API is a web interface that lets developers send prompts and data to hosted language models and receive generated text or other outputs, which they can build into their own applications.

What is the API and when is it useful?

A hosted model API lets your application send text (and in some models images or other data) and receive a generated reply, without you running the model yourself. Typical uses are summarising documents, answering questions over your data, extracting fields from text, classifying messages and writing drafts for human review. Model names, features, limits and prices change often, so always check the official documentation before building.

How do I get started safely?

Create an account and an API key in the provider's dashboard. Store the key in an environment variable or a secrets manager, never in source code, a public repository or front-end JavaScript. Call the API from a server you control. Set usage limits and alerts in the dashboard so that mistakes cannot create a large bill. Rotate a key immediately if it leaks.

What does a request look like?

The exact SDK and endpoint names change, so follow the current reference. The structure is stable: a model name, the input and optional settings.

```
POST https://api.openai.com/v1/<endpoint>
Authorization: Bearer $OPENAI_API_KEY
Content-Type: application/json

{
  "model": "<model-name>",
  "input": [
    {"role": "system", "content": "You are a concise assistant for a training institute."},
    {"role": "user", "content": "Summarise this text in three bullets: ..."}
  ],
  "temperature": 0.2,
  "max_output_tokens": 300
}
```

Messages have roles: a **system** (or developer) message sets behaviour, **user** messages carry the request and **assistant** messages hold earlier replies in a conversation. The API is stateless unless you use a feature that stores state, so you send the history you want the model to see.

Which parameters matter?

Parameter	Effect
model	Which model answers; affects quality, speed and cost
temperature or similar	Lower values give steadier output; higher values give more variety
max output tokens	Caps the length of the reply and the cost
stop sequences	Ends output at a chosen string
response format	Asks for JSON or a schema

Some newer models fix or hide certain settings, so read the model page.

How do I get structured output?

When code must read the answer, ask for JSON that matches a schema, using the structured output feature if the model supports it. Describe each field clearly and still validate the result in your code, because any model output can be wrong or incomplete. For classification, use a small fixed list of allowed labels.

What is tool calling?

You describe functions (name, purpose and parameters) in the request. The model may reply with a request to call a function with arguments. Your code runs the function, such as a database lookup or a calculator, and sends the result back so that the model can write the final answer. The model never runs code by itself; you control what each tool can do. Validate arguments and limit tools to safe, read-only actions unless a human approves.

How do I handle long inputs?

Models have a context limit measured in tokens. Split long documents into chunks, summarise each and then combine, or retrieve only the relevant chunks (retrieval-augmented generation). Count tokens before sending when possible and keep prompts short and clear.

How do I control cost and speed?

Charges depend on input and output tokens and the model. Reduce cost by shortening prompts, limiting output length, caching repeated requests, using smaller models for simple tasks, batching non-urgent jobs where supported and logging token use per feature. Streaming responses show the first words sooner and improves the user experience without changing total cost. Check the current pricing page, which changes.

How do I handle errors and rate limits?

Expect errors such as invalid requests, authentication failures, rate limits and timeouts. Retry only temporary errors with exponential backoff and a limit, log the request ID for support and show users a friendly message. Do not retry blindly in a loop, because it can increase cost and worsen limits.

What data and safety rules apply?

Do not send secrets or personal data unless you have a lawful reason and have checked the provider's data policy and your organisation's rules. Treat user input as untrusted: it can try to override your instructions (prompt injection). Keep system instructions separate from data, limit tool powers, filter outputs where needed and keep a human in the loop for important actions.

How do I test an API-based feature?

Create a set of sample inputs with expected results, run it after every change to a prompt or model, record accuracy, cost and time, and review failures by hand. Pin a model version where possible so behaviour does not change unexpectedly.

Frequently asked questions

How do I keep my OpenAI API key safe?

Store it in an environment variable or secrets manager, call the API from a server and never put it in source code or front-end code.

What does temperature do?

It controls variety in the output. Lower values give steadier answers and higher values give more varied ones, where the model allows the setting.

What is tool calling?

The model can request a function call with arguments, your code runs the function and returns the result so the model can write the final answer.

How can I reduce API cost?

Shorten prompts, cap output length, cache repeated requests, use smaller models for simple tasks and check the current pricing page.

Is the model output always correct?

No. Validate structured output and check important facts, because any model can return wrong or incomplete answers.

Sources and further reading

- OpenAI API documentation
- OpenAI API reference
- OWASP Top Ten

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550