APPLIED AI FOR DEVELOPERS · PART 3

Working with LLM APIs

In Part 2, we looked inside the model.

Now we connect it to an application.

Send a request

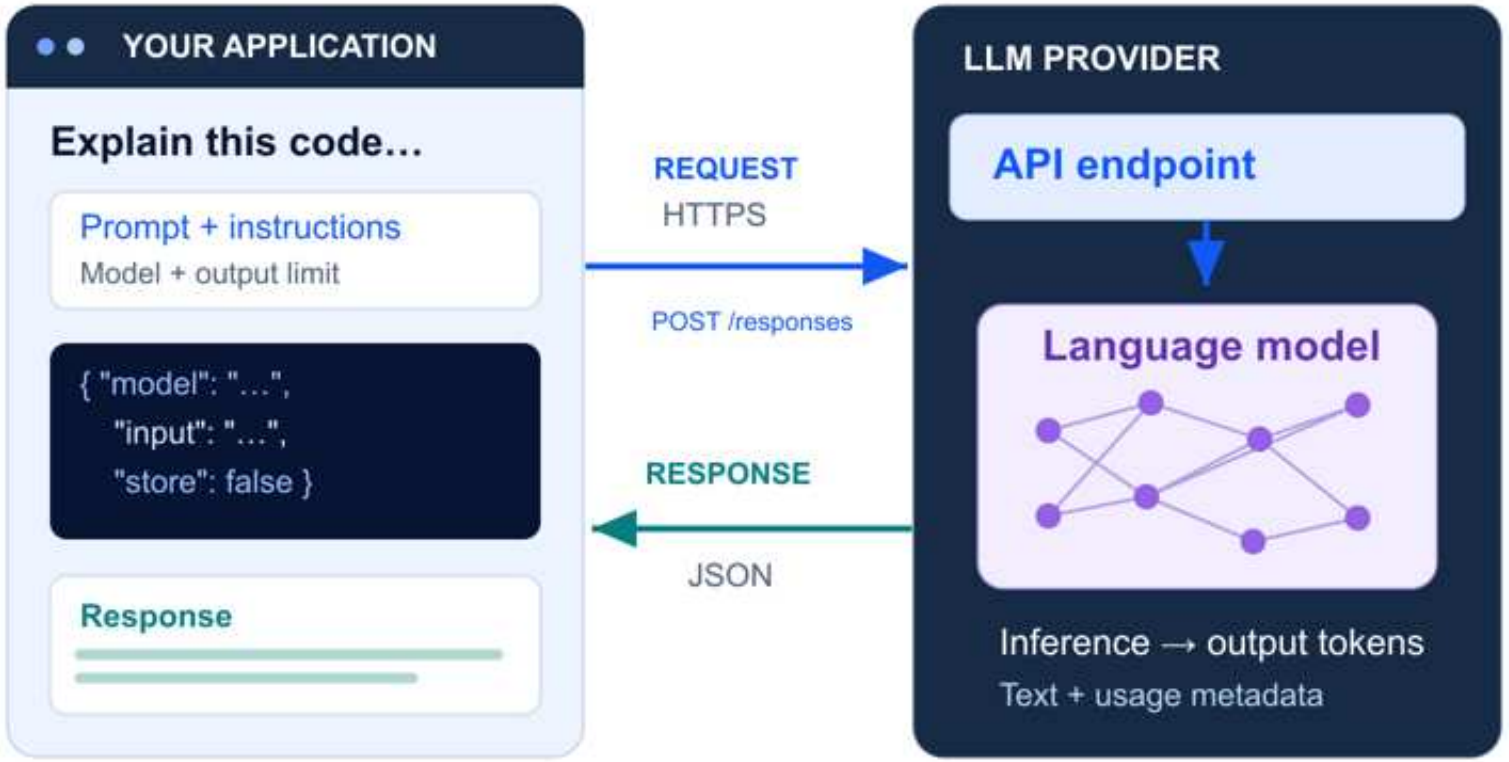
HTTP, authentication and JSON.

Shape the context

Instructions and conversation history.

Build for reliability

Validate, measure and handle failures.



TODAY'S ROUTE Request → response → context → reliable application

OpenAI Docs · Text generation

Recap: From Inside the Model to Your Application

Part 2 opened the black box. Four ideas explain what happens behind an LLM API.

1

Tokens → vectors

bank →



Token IDs map to learned embedding vectors.

2

Layers build context

Causal attention + position

MLP + residual paths

Repeat across decoder blocks.
Each position uses its past.

3

Predict the next token



Vocabulary scores → probabilities.
Illustrative distribution.

4

Append and repeat

The future of AI is

bright

Choose a token; predict again.
Inference keeps weights fixed.

Context is finite.
Your app selects the history and evidence to include.

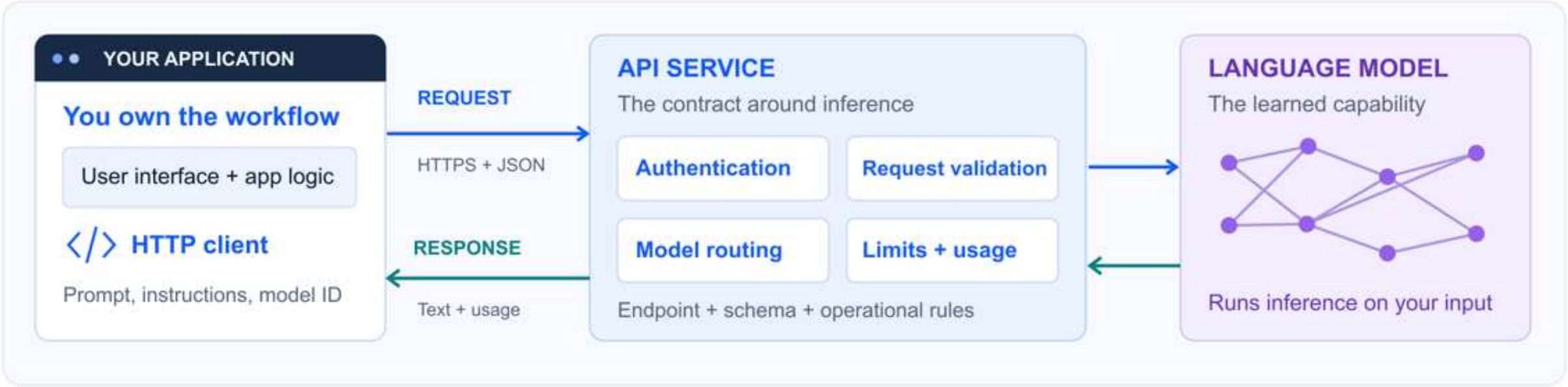
Fluent does not mean verified.
Ground important claims and validate the result.

PART 3 **Your application** → **LLM API** → **this mechanism** → **useful, checked output**

Recap of Part 2 · LLM Fundamentals II: Inside the Model

What Is an LLM API?

A programmable interface that lets your application use a language model over the network.



API = interface + service
How you access and operate the capability.

Model = learned parameters + inference
What generates the output from your context.

A chat product adds a ready-made user interface. An API lets you build your own.

OpenAI Docs · Text generation

The Anatomy of an API Request

One HTTPS request combines a destination, credentials and a structured JSON body.

1 **Destination**
POST starts a generation request.
Use the provider's HTTPS endpoint.

2 **Authentication**
The header carries your API key.
Supply it at runtime; never hard-code it.

3 **Model**
Choose the capability to call.
The model must be accessible to your key.

4 **Input**
Send the user's prompt or input items.
Add instructions and context when needed.

5 **Controls**
Bound output; choose storage behavior.
Supported options depend on the model.

HTTP REQUEST · OPENAI RESPONSES API

```
1 POST https://api.openai.com/v1/responses
2 Authorization: Bearer RUNTIME_KEY
  Content-Type: application/json
  {
3   "model": "gpt-5.4-mini",
4   "input": "Explain an API in 3 sentences.",
5   "max_output_tokens": 400,
    "store": false
  }
```

NEXT: SEND IT **Enter a temporary key and prompt on slide 5 — then inspect the response.**

OpenAI Docs · Text generation

APPLIED AI FOR DEVELOPERS · PART 3

Your First LLM API Call

LIVE DEMO · WEB EDITION

01 · CONNECT TO OPENAI

API Key

Enter a temporary API key

Model

gpt-5.4-mini

02 · WRITE YOUR REQUEST

Prompt

Explain how an LLM works in 3 sentences.

Send →

Clear

Check browser CORS

03 · OpenAI Response

The model's response appears here.

Input tokens: — · Output tokens: — · Time: —

Static PDF preview. Run the interactive demo in the web presentation.

No API key is included. Enter a temporary key in the web edition and revoke it afterwards.

Understanding the API Response

The response is a structured result — generated text is only one part of it.

RESPONSE BODY · SIMPLIFIED JSON

```
{
"status": "completed",                                    1
"output": [{
"type": "message",
"content": [{
"type": "output_text",
"text": "An API lets apps communicate."                   2
}]
}],
"usage": {
"input_tokens": 24,
"output_tokens": 38                                       3
}
}
```

Illustrative JSON and token counts; not a measured demo response.

1 Check the outcome

Read status before trusting the result.

Handle incomplete output, refusals and errors.

2 Extract the generated text

`output[] → message → content[] → output_text`

Collect matching text parts; render them as text.

Do not assume output[0] is always a message.

3 Read actual token usage

Input: 24 tokens Output: 38 tokens

Elapsed time is measured separately by the browser.

RAW REST ≠ SDK HELPER The SDK's output_text helper collects text; raw fetch returns typed output items.

OpenAI Docs · Text generation

Instructions and Conversation Context

Tell the model how to behave — and supply the information it needs for this request.

01 · INSTRUCTIONS

Define application behavior.

Task, audience, constraints and format.

Keep guidance explicit and consistent.

02 · INPUT + CONTEXT

Supply the question and evidence.

Current user message

Relevant earlier user / assistant turns

Documents or tool results, when needed

Your app decides what to include.

RESPONSES API · REQUEST BODY EXCERPT

```
{
"instructions": "Explain APIs clearly. Be concise.",

"input": [
{ "role": "user",
"content": "What is an API?" },
{ "role": "assistant",
"content": "An API lets applications communicate." },
{ "role": "user",
"content": "Now give me a GET request example." }
]
}
```

KEEP THE BOUNDARY

Retrieved text is untrusted data. Keep secrets out of prompts.
Instructions guide behavior; validate important outputs in your application.

OpenAI Docs · Text generation

Controlling Model Behavior

Use the controls your model supports — then measure whether the result meets your needs.

1

Choose the capability

MODEL
Task-appropriate model

Match model strengths to your task.

Compare quality, latency and cost.

Evaluate on representative examples.

2

Bound the output

max_output_tokens



Set an explicit upper limit.

Request a schema when supported.

Validate the returned structure.

A token limit may truncate the answer.

3

Temperature

API parameter: temperature

Low · e.g. 0.2

Favors likely tokens; less variation.

High · e.g. 0.8

More varied token choices.

Only where supported by the model
and its selected configuration.

EXPECT VARIATION **Lower temperature does not promise identical answers.**

API BODY EXCERPT

```
{ "temperature": 0.2 }
```

Not a truth or quality control.
Check support; omit if unsupported. Slide 5 does not set it yet.

OpenAI Docs · Text generation

Multi-Turn Conversations

For the follow-up, your app sends the earlier exchange together with the new question.

THE CONVERSATION

USER · FIRST QUESTION
What is an API?

ASSISTANT · FIRST ANSWER
An API lets applications communicate.

USER · FOLLOW-UP
Now show me a GET request example.

The next call receives all three messages.

No automatic memory is assumed.

SECOND REQUEST · BODY EXCERPT

```
{
"instructions": "Explain APIs clearly. Be concise.",
"input": [
{
"role": "user",
"content": "What is an API?"
},
{
"role": "assistant",
"content": "An API lets applications communicate."
},
{
"role": "user",
"content": "Now show me a GET request example."
}
]
}
```

ANOTHER OPTION

Use supported provider state, e.g. previous_response_id; resend needed instructions.
Check retention and context growth. Slide 5 still sends independent requests.

OpenAI Docs · Conversation state

Streaming Responses

Show text as it arrives, instead of waiting for the entire response to finish.



01 · REQUEST

{ "stream": true }

Enable streaming in the JSON body.

02 · READ + RENDER

Parse SSE events; append text deltas.

Network chunks ≠ complete events.

03 · FINISH SAFELY

Handle completion and errors.

Support cancellation; render as text.

OUR LIVE DEMO Slide 5 uses a non-streaming request. Streaming requires additional event-handling code.

OpenAI Docs · Streaming responses

APPLIED AI FOR DEVELOPERS · PART 3

Tokens, Latency and Cost

Measure the whole request: how much it uses, how long it takes and what it costs.

01 · TOKENS

Count input and output

INPUT

Instructions | History | Prompt

OUTPUT

Generated tokens

Read usage metadata from the API.

Tokens are not the same as words.

02 · LATENCY

Follow the elapsed time



Network → input processing → model work → output transfer

Streaming: first text ≠ completion.

Demo time: browser wall-clock duration.

03 · COST

Different models, different prices

Same token usage ≠ same cost.

Each model has its own input / output rates.

Output usually costs more per token.
But long input can cost more in total.
Cached input has separate rates.

Cost = token count × applicable rate

Sum categories; use current model pricing.

TOTAL COST MATTERS Input cost + output cost; divide token counts by 1M when using per-million rates.

COMPARE PER TASK **Quality + latency + total cost**

Token counts can vary by model. Cached input and reasoning affect accounting; demo timing is not a billing quote.

OpenAI Docs · Current model pricing

Handling Errors and Limits

Classify the failure first. A retry only helps when the cause is temporary.

SIGNAL	WHAT TO CHECK	APPLICATION ACTION
400 / 404	Request, parameters, endpoint or model access	Fix the request or configuration.
401 / 403	Key validity, project permissions and access	Fix credentials / access; avoid blind retries.
429	Rate limit or exhausted quota? Read the error.	Rate limit: backoff. Quota: account action.
5xx	Temporary provider-side failure	Use bounded retries with backoff + jitter.
TIMEOUT	The request may still have completed	Cancel locally; check outcome before repeating.

BROWSER FETCH FAILED?

Network, CORS or browser policy may block the call.

JavaScript cannot always distinguish these causes.

RETRY POLICY · ILLUSTRATIVE

Wait → longer wait → stop after a bounded limit

Add jitter; respect provider retry guidance and deadlines.

SAFE FAILURE **Show useful messages. Never log API keys. Keep request IDs for diagnosis.**

OpenAI Docs · Error codes

Choosing the Right Model

Choose for your task — using measured results, rather than the model name alone.

01 · DEFINE THE REQUIREMENTS

What must the model deliver?

- ☐ Quality and acceptable failure rate
- ☐ Response-time target
- ☐ Budget per completed task
- ☐ Output format and tool support
- ☐ Privacy and deployment constraints

02 · TEST CANDIDATES ON THE SAME TASKS

Include normal inputs, hard cases and expected failures.

Candidate	Quality	Latency	Cost / task
Model A	Measure	Measure	Measure
Model B	Measure	Measure	Measure
Model C	Measure	Measure	Measure

Template only — no invented scores or universal model ranking.

03 · CHOOSE

Meet the quality threshold first; then optimize total cost and response time.

CHECK BEFORE DEPLOYING **Model access + parameter support + limits** → re-evaluate when things change

OpenAI Docs · Model selection

Different API Providers

The request travels over HTTP — but the service contract and operating model can differ.

01 · DIRECT HOSTED API



Call the provider directly.

Examples: OpenAI, Anthropic, Google

Provider manages model serving.

Use its credentials, schema and billing.

02 · CLOUD PLATFORM



Use a cloud access layer.

Microsoft Azure · Amazon Bedrock

Cloud identity and deployment options.

Model availability and endpoints vary.

03 · SELF-HOSTED



Operate the serving stack.

Your infrastructure + a model server

Own hardware, scaling and monitoring.

Check model license and capabilities.

CHECK BEFORE SWITCHING

- **Authentication + schemas**
- **Streaming + tools + output formats**
- **Retention + usage accounting**

KEY IDEA **A similar HTTP interface does not guarantee identical behavior. Test your integration.**

Building a Simple LLM Application

Example: a code explainer that turns a submitted snippet into a clear explanation.

01 · USER INTERFACE

```
const total = items.reduce(
(sum, x) => sum + x, 0);
```

Collect the code snippet.

Show loading, answer or error.

Render generated text safely.

02 · YOUR BACKEND

Validate input + add instructions

Choose model + bound output

Call the provider with server-held key

Check the response before returning it

Own the workflow and operational controls.

03 · LLM PROVIDER

Instructions + snippet
→ inference → explanation

Generate the explanation.

Return output + usage metadata.

Your app validates important results.

BEFORE PRODUCTION

- **Access + input limits**
- **Timeouts + budgets**
- **Logs without secrets**
- **Quality evaluations**

TEACHING DEMO Slide 5 uses a temporary browser key. A production app keeps credentials on the server.

YOUR EXERCISE **Build the code explainer; test useful answers, invalid input and provider failures.**

APPLIED AI FOR DEVELOPERS · PART 3

Key Takeaways & What's Next

One request is easy. A reliable application needs engineering.

01 Understand the contract

Endpoint, authentication and JSON.

Read typed output; handle completion.

02 Design the context

Give clear instructions and relevant history.

Treat retrieved content as untrusted data.

03 Measure the system

Compare quality, latency and cost per task.

Model prices and supported controls differ.

04 Build for failure

Validate output and protect credentials.

Handle limits, errors and uncertain outcomes.

NEXT · PART 4

Prompting & Structured Output

System prompts · few-shot prompting · JSON outputs · schemas · reliable prompt design

TAKE IT WITH YOU

Start with one useful task. Define success. Test the failures as well as the happy path.