

Web Development with AI

DR ANUPAM SOBTI

PLAKSHA UNIVERSITY

Why this course?

- Barriers to code (of all types) are reducing
- Infinite demand for software automation
- Need to build software quickly with the best tools available and SELL!
- LLMs are a tool to understand the best available methods and generate software.
 - Let's try to learn how to use it best!



Andrew Ng
@AndrewYNg



Despite having worked on AI since I was a teenager, I'm now more excited than ever about what we can do with it, especially in building AI applications. Sparks are flying in our field, and 2025 will be a great year for building!

One aspect of AI that I'm particularly excited about is how easy it is to build software prototypes. AI is lowering the cost of software development and expanding the set of possible applications. While it can help extend or maintain large software systems, it shines particularly in building prototypes and other simple applications quickly.



Naval @naval · 13h

There is infinite demand for computer programs.

13

18

403

14K



Course Philosophy and Approach

Menu card approach

- Know your options. Explore more when required.

LLMs to mimic (expert) human behavior at a workflow level

Use flexible approaches

- With respect to models
- With respect to products, e.g., don't depend too much on Cursor/VS Code.

What are we going to do?

Lect.	Topic	Details
1	WTH is web?	HTML/Styling/Interactivity - JS; Bootstrap, Tailwind, React, Vue, D3.js, Chart.js
2	What's possible in web?	Library usage examples
3	Prompting LLMs	Single shot prompting, Editing with prompts, Maintaining context
4	Expanding web users	Responsive Design, Web Sockets, Hooks
5	Prompting in context	Multi-file editing with prompts, Sharing documentation with LLMs
6	FastHTML/FastAPI	Basics, HTMX, async workflows, DB integration
7	FastHTML - II	Websockets, Authentication
8	Database design basics	User management, DB operations, atomicity, etc. Prompting for DB design.
9	Stitching databases with frontend	Prompts for integrating DBs with web pages
10	Beyond databases	Handling large data, caching
11	Using reasoning models - o1, gemini 2.0	Reason before you respond. Hybrid use of models.
12	RAG	Vector embeddings, Retrieving content, Handling large data, Establish Causation
13	Agentic RAG	Adding helpers to your RAG pipelines
14	Multi agent systems	The benefit of having multi-role agents and orchestrating workflows in between agents
15	Using memory with LLMs	How to orchestrate memory, what to retrieve for context, what kind of context is good context, scratchpadding

Evaluation

The lab presentations would be updates on things you've tried towards your project from the lecture and its effect on your course project.

Please read papers/watch youtube to see if people have tried similar things that you are trying to do. How did they phrase their prompts? What problems did they encounter?

Lab presentations (6) 70%

Demo day 20%

Attendance and class participation 10%

More than 80% = 10 marks (full)
70 to 80 % = 8 marks
60 to 70 % = 6 marks
Less than 60 % = 0 marks

There will be 3 star presentations declared every 2 weeks. If you get more than 3 stars, you get a grade bump.

Recommended read for when prompting isn't enough

https://developer.mozilla.org/en-US/docs/Learn_web_development

What are websites?

Some basic
terminologies

HTML – HyperText Markup Language (latest version 5) [2014 onwards]

- Inbuilt support for multiple modalities like `<audio>`, `<video>`, `<track>`
- More structure for text, e.g., `<header>`, `<footer>`, `<article>`, `<section>`, `<aside>`, `<figure>`, `<figcaption>` etc.
- Enhanced forms
- Scriptable graphics
- LocalStorage/SessionStorage
- Improved scripting and APIs: websockets, web workers (background js)

Programmatic access to a document (html/xml): DOM

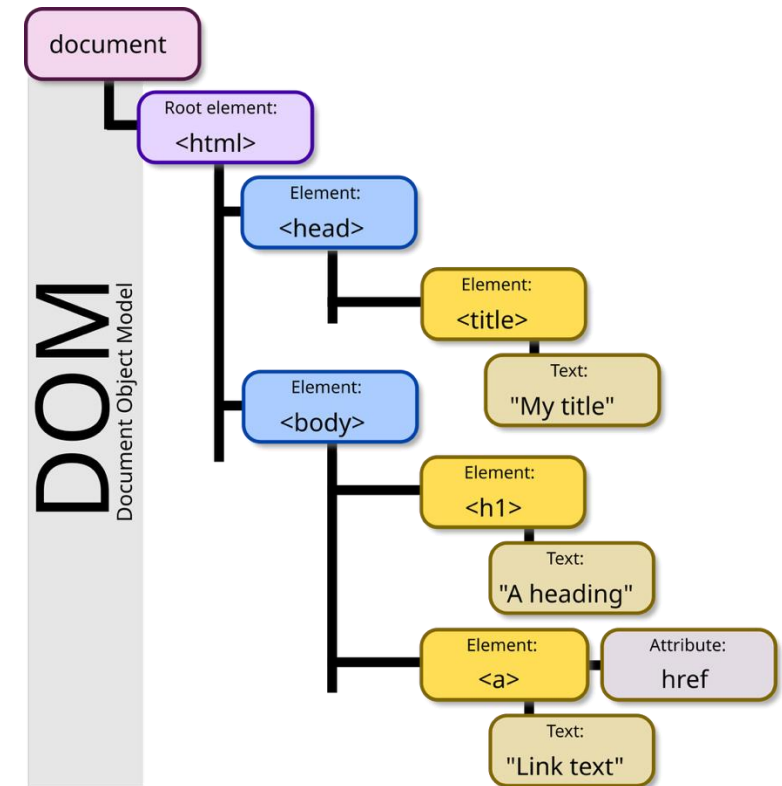
From Wikipedia, the free encyclopedia

The **Document Object Model (DOM)** is a [cross-platform](#) and [language-independent](#) interface that treats an [HTML](#) or [XML](#) document as a [tree structure](#) wherein each [node](#) is an [object](#) representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document.^[2] Nodes can have [event handlers](#) (also known as event listeners) attached to them. Once an event is triggered, the event handlers get executed.^[3]

The principal standardization of the DOM was handled by the [World Wide Web Consortium](#) (W3C), which last developed a recommendation in 2004. [WHATWG](#) took over the development of the standard, publishing it as a [living document](#). The W3C now publishes stable snapshots of the WHATWG standard.

In HTML DOM (Document Object Model), every element is a node:^[4]

- A document is a document node.
- All HTML elements are element nodes.
- All HTML attributes are attribute nodes.
- Text inserted into HTML elements are text nodes.
- Comments are comment nodes.



https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

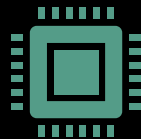
What are scripts?



Things that allow dynamic interactions/content, e.g., mouse hovers, scroll, etc.



- JavaScript (JS), TypeScript (superset compiled to JS), WASM (Web Assembly for running Go/Rust in the browser)



Executable JS code (could be client or server side but mostly client side)

Styling content

Cascading Style Sheets (CSS)

- Style fixed for elements
- or rendered at load time on the server side (SSR – server side rendering)

Famous:

- <https://ui.shadcn.com/>
- <https://tailwindcss.com/>
- <https://m3.material.io/styles/color/system/overview>

<https://playcode.io/css>

Margin

<https://developer.mozilla.org/en-US/docs/Web/CSS/margin>

CSS Demo: margin

RESET

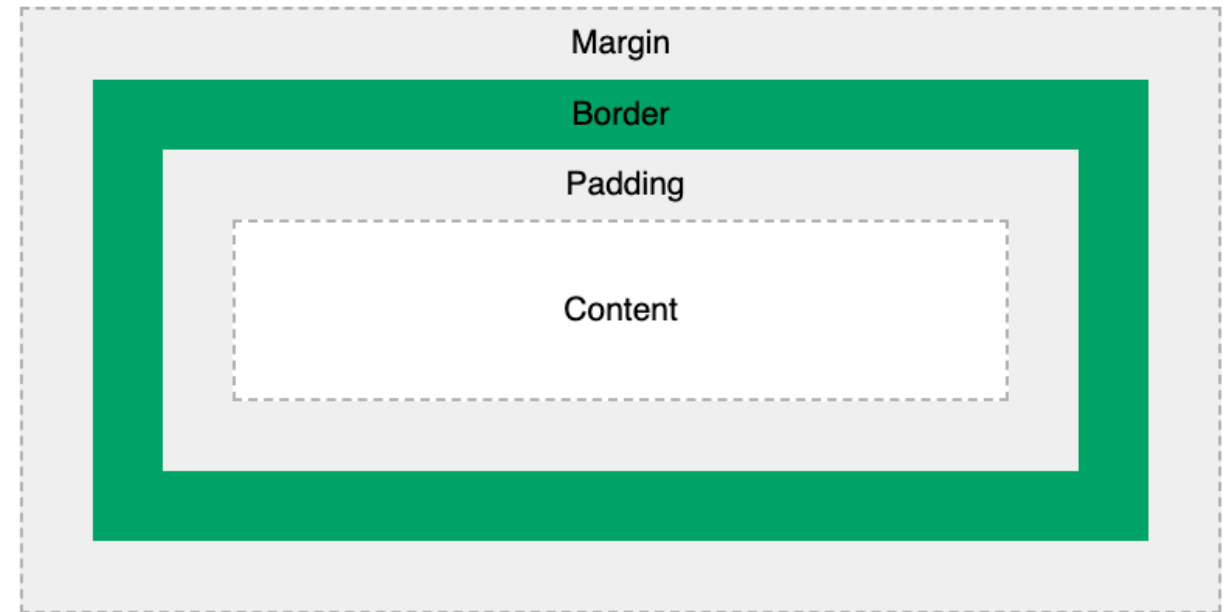
margin: 1em;

margin: 5% 0;

margin: 10px 50px 20px;


margin: 10px 50px 20px 0;

margin: 0;



https://www.w3schools.com/css/css_boxmodel.asp

js

 Copy code

```
const inventory = [
  { name: "Laptop", quantity: 5, price: 800 },
  { name: "Phone", quantity: 10, price: 500 },
  { name: "Tablet", quantity: 8, price: 300 },
];

let totalValue = 0;

for (const item of inventory) {
  const itemValue = item.quantity * item.price;
  console.log(`Item: ${item.name}, Total Value: ${itemValue}`);
  totalValue += itemValue;
}

console.log("Inventory Total Value:", totalValue);
```

JS Basics

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>JavaScript Events Example</title>
</head>
<body>
  <button id="myButton">Click Me</button>
  <p id="message"></p>

  <script>
    // Step 1: Select the button and paragraph elements
    const button = document.getElementById("myButton");
    const message = document.getElementById("message");

    // Step 2: Define a function to handle the event
    function handleClick() {
      message.textContent = "Button was clicked!";
    }

    // Step 3: Attach the event listener to the button
    button.addEventListener("click", handleClick);
  </script>
</body>
</html>
```

Copy code

Events in JS

D0: Let's make something?



A basic website for
this course's (your
project's) information.



Adding interactions.



Code for different
types of events:
onClick, onHover



Responsive designs

D0: Using variables for interactions



TRY TO MAKE SOME
SECTIONS COLLAPSIBLE.



STORE THE STATE IN JS.

D0: Let's make different layouts and styles



Try three different styles and layouts for your website



Check them for responsiveness on three different devices



Add 3 types of interaction through js

Some layout frameworks

Component Libraries

1. **Material-UI (MUI)**
2. **Ant Design**
3. **Chakra UI**
4. **React Bootstrap**
5. **Vuetify**
6. **PrimeReact / PrimeVue**
7. **Blueprint**

CSS Frameworks

1. **Bootstrap**
2. **Tailwind CSS**
3. **Bulma**
4. **Foundation**
5. **Materialize**
6. **Skeleton**
7. **Pure CSS**

Super lightweight: Pico.css

D0: Let's make multipage apps



COMMON CODE FOR
HEADER, FOOTER



USE SCRIPTS TO
INCLUDE ON ALL PAGES

Responsive Design

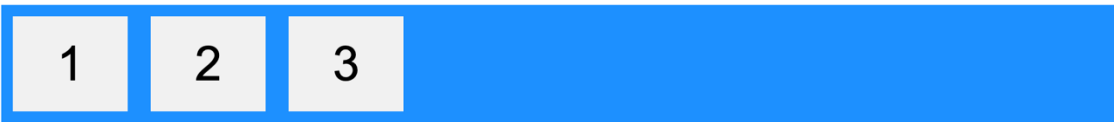
The term *responsive design*, coined by Ethan Marcotte in 2010, described using fluid grids, fluid images, and media queries to create responsive content.

https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design

https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Media_queries

Flexbox model

Understanding FlexBox



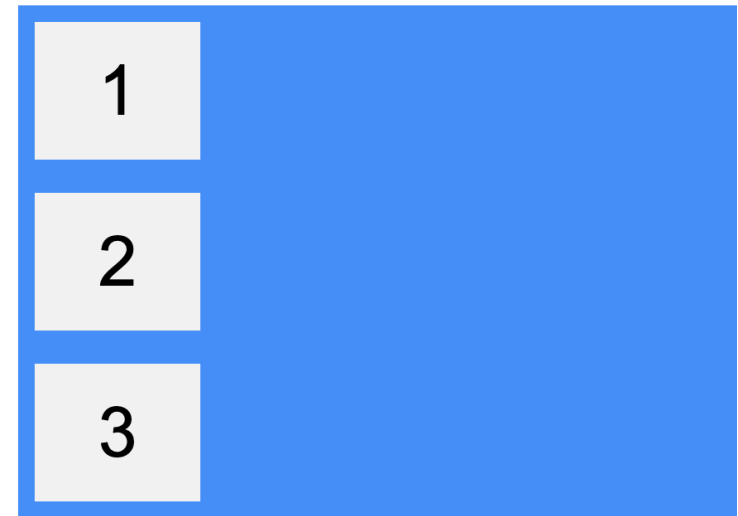
```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

- `flex-direction`
- `flex-wrap`
- `flex-flow`
- `justify-content`
- `align-items`
- `align-content`

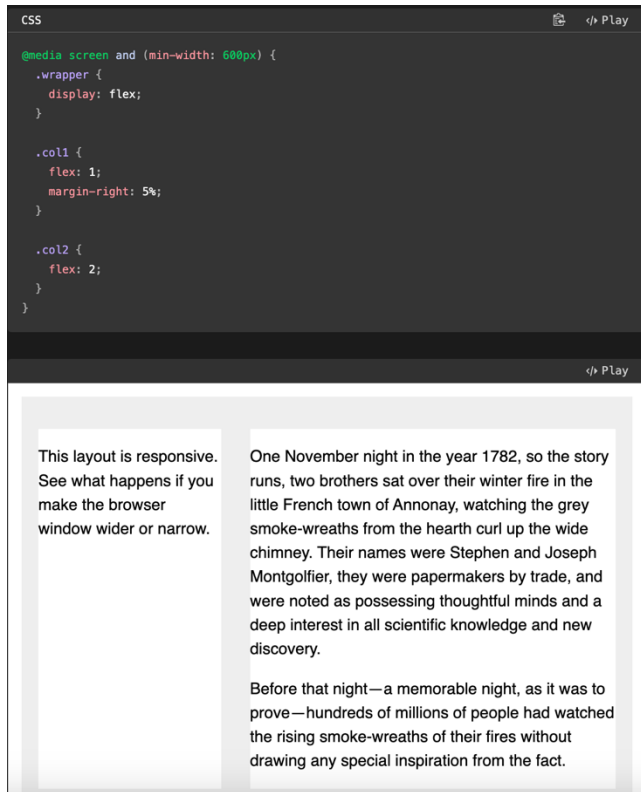
https://www.w3schools.com/css/css3_flexbox.asp

```
.flex-container {
  display: flex;
  flex-direction: column;
}
```

Result:



Flexbox example



Media Queries

CSS



```
@media screen and (min-width: 80rem) {  
  .container {  
    margin: 1em 2em;  
  }  
}
```

Screen-size/Viewport size dependent css

Do you need a media query?

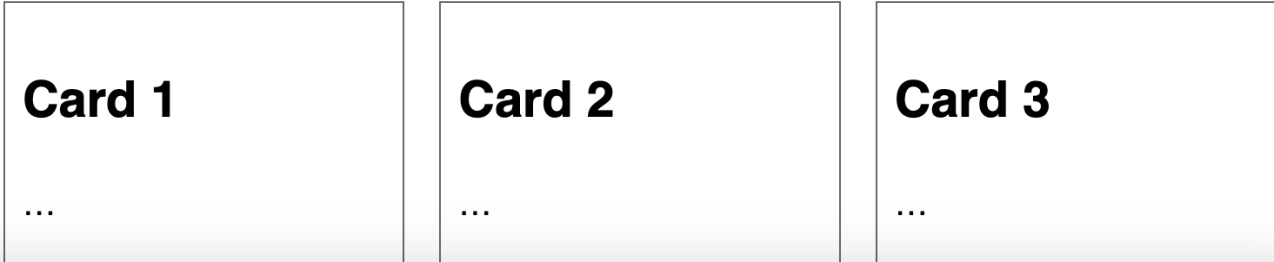
Always prefer flexbox/css grid solutions since these are more general solutions!

```
CSS 📄 </> Play

body {
  font: 1.2em / 1.5 sans-serif;
}
.grid {
  list-style: none;
  margin: 0;
  padding: 0;
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
}

.grid li {
  border: 1px solid #666;
  padding: 10px;
}

</> Play
```



Styling images

https://www.w3schools.com/css/css3_images.asp



Example

Rounded Image:

```
img {  
  border-radius: 8px;  
}
```

Polaroid Images / Cards



Cinque Terre

Guess what this does?

```
// Get the modal  
var modal = document.getElementById('myModal');  
  
// Get the image and insert it inside the modal - use its "alt" text as a caption  
var img = document.getElementById('myImg');  
var modalImg = document.getElementById("img01");  
var captionText = document.getElementById("caption");  
img.onclick = function(){  
  modal.style.display = "block";  
  modalImg.src = this.src;  
  captionText.innerHTML = this.alt;  
}  
  
// Get the <span> element that closes the modal  
var span = document.getElementsByClassName("close")[0];  
  
// When the user clicks on <span> (x), close the modal  
span.onclick = function() {  
  modal.style.display = "none";  
}
```

Frameworks

Common frameworks

React

Vue.js

Next.js

What do these bring to the table?

Large number of interactions in a component style

Good to use when highly functional SPAs (single page applications) are required

Reflections from last class

Guest lecture by Archit

My two cents.

If you're imagining screens which are largely interactive:

- Next js based app

If you're imagining screens which are largely static with interactive components:

- HTMX + Static site generator (Eleventy)

If you're imagining screens with complex visualizations and custom interactions:

- Stick to raw html/css/js

Prompting like a pro

OpenAI

Services

MODEL	DESCRIPTION
GPT-4o	Our versatile, high-intelligence flagship model
GPT-4o-mini	Our fast, affordable small model for focused tasks
o1 and o1-mini	Reasoning models that excel at complex, multi-step tasks
GPT-4o Realtime	GPT-4o models capable of realtime text and audio inputs and outputs
GPT-4o Audio	GPT-4o models capable of audio inputs and outputs via REST API
GPT-4 Turbo and GPT-4	The previous set of high-intelligence models
GPT-3.5 Turbo	A fast model for simple tasks, superseded by GPT-4o-mini
DALL·E	A model that can generate and edit images given a natural language prompt
TTS	A set of models that can convert text into natural sounding spoken audio
Whisper	A model that can convert audio into text

Structured Outputs

Ensure model responses adhere to your supplied JSON schema

```
{
  "name": "math_response",
  "strict": true,
  "schema": {
    "type": "object",
    "properties": {
      "steps": {
        "type": "array",
```

Realtime API

Build low-latency multimodal experiences

User

Can I order two americanos and a chocolate donut?

04:20

Assistants API

Build conversational assistants with tools and File Search

- File Search**
Built-in RAG tool to process and search through files
- Code Interpreter**
Write and run python code, process files and diverse data
- Function Calling**
Use your own custom functions to interact with your application

Async use cases

Batch requests for async, large-scale processing

```
batch_joiYCmvQuzPK
Status Completed
Completion time 24h
Endpoint /v1/chat/completio
Request counts 497 completed, 0 failed
```

Fine-tuning >

Adapt a model to your specific use case with your data

Step 568
Training loss 0.0302
Validation loss 0.02

Distillation

Evaluate and fine-tune models using production logs

o1-preview 90% gpt-4o-mini 86%

Metric	o1-preview 90%	gpt-4o-mini 86%
Grammar	~65%	~45%
Syntax	~65%	~15%
Spelling	~65%	~45%

Prompting

Create a human-like response to a prompt



```
1 from openai import OpenAI
2 client = OpenAI()
3
4 completion = client.chat.completions.create(
5     model="gpt-4o-mini",
6     messages=[
7         {"role": "system", "content": "You are a helpful assistant."},
8         {
9             "role": "user",
10            "content": "Write a haiku about recursion in programming."
11        }
12     ]
13 )
14
15 print(completion.choices[0].message)
```

Generate an image based on a textual prompt

```
1 from openai import OpenAI
2 client = OpenAI()
3
4 response = client.images.generate(
5     prompt="A cute baby sea otter",
6     n=2,
7     size="1024x1024"
8 )
9
10 print(response.data[0].url)
```


Strategies for prompting

Writing Clear Instructions

- Include details in your query to get more relevant answers
- Ask the model to adopt a persona
- Use delimiters to clearly indicate distinct parts of the input
 - `"""` or `<article></article>` or even Santa: Banta:
 - Explain task to the “system”
- Specify the steps required to complete a task
- Provide examples (Few shot prompting)
- Specify the desired length of the output

Write clear instructions

Provide reference text

Split complex tasks into simpler subtasks

Give the model time to "think"

Use external tools

Test changes systematically

Other resources

Providing reference text

- Instruct the model to answer using a reference text
 - Use embeddings when text is too big
- Instruct the model to answer with citations from a reference text

```
SYSTEM    You will be provided with a document delimited by triple quotes and a
          question. Your task is to answer the question using only the provided
          document and to cite the passage(s) of the document used to answer the
          question. If the document does not contain the information needed to
          answer this question then simply write: "Insufficient information." If an
          answer to the question is provided, it must be annotated with a citation. Use
          the following format for to cite relevant passages ({"citation": ...}).

USER      """"<insert document here>""""

          Question: <insert question here>
```

Split complex tasks into simpler subtasks

- Use intent classification to identify the most relevant instructions for a user query
- For dialogue applications that require very long conversations, summarize or filter previous dialogue
 - Think of it as note taking
- Summarize long documents piecewise and construct a full summary recursively

Give the model time to "think"

- Instruct the model to work out its own solution before rushing to a conclusion
- Use inner monologue or a sequence of queries to hide the model's reasoning process
- Ask the model if it missed anything on previous passes

Use tools

- Use embeddings-based search to implement efficient knowledge retrieval
- Use code execution to perform more accurate calculations or call external APIs
 - Use for visualizations and dashboards
- Give the model access to specific functions

Test changes systematically

- Evaluate model outputs with reference to gold-standard answers

More from OpenAI

<https://cookbook.openai.com/>

- [Prompting libraries & tools](#)
- [Prompting guides](#)
- [Video courses](#)
- [Papers on advanced prompting to improve reasoning](#)

Gemini

Services

Model variant	Input(s)	Output	Optimized for
Gemini 2.0 Flash gemini-2.0-flash-exp	Audio, images, videos, and text	Text, images (coming soon), and audio (coming soon)	Next generation features, speed, and multimodal generation for a diverse variety of tasks
Gemini 1.5 Flash gemini-1.5-flash	Audio, images, videos, and text	Text	Fast and versatile performance across a diverse variety of tasks
Gemini 1.5 Flash-8B gemini-1.5-flash-8b	Audio, images, videos, and text	Text	High volume and lower intelligence tasks
Gemini 1.5 Pro gemini-1.5-pro	Audio, images, videos, and text	Text	Complex reasoning tasks requiring more intelligence
Gemini 1.0 Pro gemini-1.0-pro (Deprecated on 2/15/2025)	Text	Text	Natural language tasks, multi-turn text and code chat, and code generation
Text Embedding text-embedding-004	Text	Text embeddings	Measuring the relatedness of text strings
AQA aqa	Text	Text	Providing source-grounded answers to questions



Explore long context

Input millions of tokens to Gemini models and derive understanding from unstructured images, videos, and documents.



Solve tasks with fine-tuning

Modify the behavior of Gemini models to adapt to specific tasks, recognize data, and solve problems. Tune models with your own data to make production deployments more robust and reliable.

```
structured_outputs.py > ...
1 import google.generativeai as genai
2 import typing_extensions as typing
3
4 class Recipe(typing.TypedDict):
5     recipe_name: str
6     ingredients: list[str]
7
8 genai.configure(api_key="GEMINI_API_KEY")
9 model = genai.GenerativeModel("gemini-1.5-pro")
10 result = model.generate_content(
11     "List a few popular python libraries.",
12     generation_config=genai.GenerationConfig(
13         response_mime_type="application/json", response_schema=Recipe
14     ),
15 )
16 print(result)
```

Generate structured outputs

Constrain Gemini to respond with JSON, a structured data format suitable for automated processing.

Basic usage

```
import google.generativeai as genai
import PIL.Image

genai.configure(api_key="GEMINI_API_KEY")

model = genai.GenerativeModel("gemini-1.5-flash")
organ = PIL.Image.open("/path/to/organ.png")
response = model.generate_content(["Tell me about this instrument", organ])
print(response.text)
```



Strategies for prompting

Clear and specific instructions

- Define task clearly, in steps if required
- Specify constraints
- Define format

- Give clear and specific instructions
- Include few-shot examples
- Add contextual information
- Add prefixes
- Let the model complete partial input

Include Few-shot Examples

Experiment with number of examples

Include patterns, not antipatterns

Use consistent formatting in examples

Adding contextual information

- Include information (context) in the prompt that you want the model to use when generating a response.
 - Specially useful when using tools
- Give the model instructions on how to use the contextual information.

Add prefixes

Input, e.g., ARTICLE:

Output, e.g., JSON:

Include examples

Let the model complete partial output

- If you give the model a partial input, the model completes that input based on any available examples or context in the prompt.
- Having the model complete an input may sometimes be easier than describing the task in natural language.
- Adding a partial answer to a prompt can guide the model to follow a desired pattern or format.

Break down prompts into simple components

Break down instructions

- Separate out major “functions” and use different prompts for each function

Chain prompts, e.g., running steps one after the other to ensure intermediate output

Aggregate results, e.g., going through your codebase

Different parameter values

Max output tokens

Temperature

- Higher the temperate, more creative

Top-K

- Sampling during generation

Top-P

- Tokens are selected from the most (see top-K) to least probable until the sum of their probabilities equals the top-P value.
- if tokens A, B, and C have a probability of 0.3, 0.2, and 0.1 and the top-P value is 0.5, then the model will select either A or B as the next token by using temperature and excludes C as a candidate.

Iterating on prompts

Use different paraphrasing

Switch to analogous tasks

Change order of prompt content

```
Version 1:  
[examples]  
[context]  
[input]
```

```
Version 2:  
[input]  
[examples]  
[context]
```

```
Version 3:  
[examples]  
[input]  
[context]
```

Prompting with files

- Prompt design fundamentals

- **Be specific in your instructions:** Craft clear and concise instructions that leave minimal room for misinterpretation.
- **Add a few examples to your prompt:** Use realistic few-shot examples to illustrate what you want to achieve.
- **Break it down step-by-step:** Divide complex tasks into manageable sub-goals, guiding the model through the process.
- **Specify the output format:** In your prompt, ask for the output to be in the format you want, like markdown, JSON, HTML and more.
- **Put your image first for single-image prompts:** While Gemini can handle image and text inputs in any order, for prompts containing a single image, it might perform better if that image is placed before the text prompt.

- Troubleshooting your multimodal prompt

- **If the model is not drawing information from the relevant part of the image:** Drop hints with which aspects of the image you want the prompt to draw information from.
- **If the model output is too generic (not tailored enough to the image):** At the start of the prompt, try asking the model to describe the image(s) before providing the task instruction, or try asking the model to refer to what's in the image.

Claude

Key capabilities

Key capabilities

Claude can assist with many tasks that involve text, code, and images.



Text and code generation

Summarize text, answer questions, extract data, translate text, and explain and generate code.



Vision

Process and analyze visual input and generate text and code from images.

claude_quickstart.py

```
import anthropic

client = anthropic.Anthropic()

message = client.messages.create(
    model="claude-3-5-sonnet-20241022",
    max_tokens=1000,
    temperature=0,
    system="You are a world-class poet. Respond only with short poems.",
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "Why is the ocean salty?"
                }
            ]
        }
    ]
)

print(message.content)
```

Before you prompt

Set a good goal

- <https://docs.anthropic.com/en/docs/build-with-claude/define-success>
- Specific
- Measurable
- Achievable
- Relevant

▼ Example task fidelity criteria for sentiment analysis

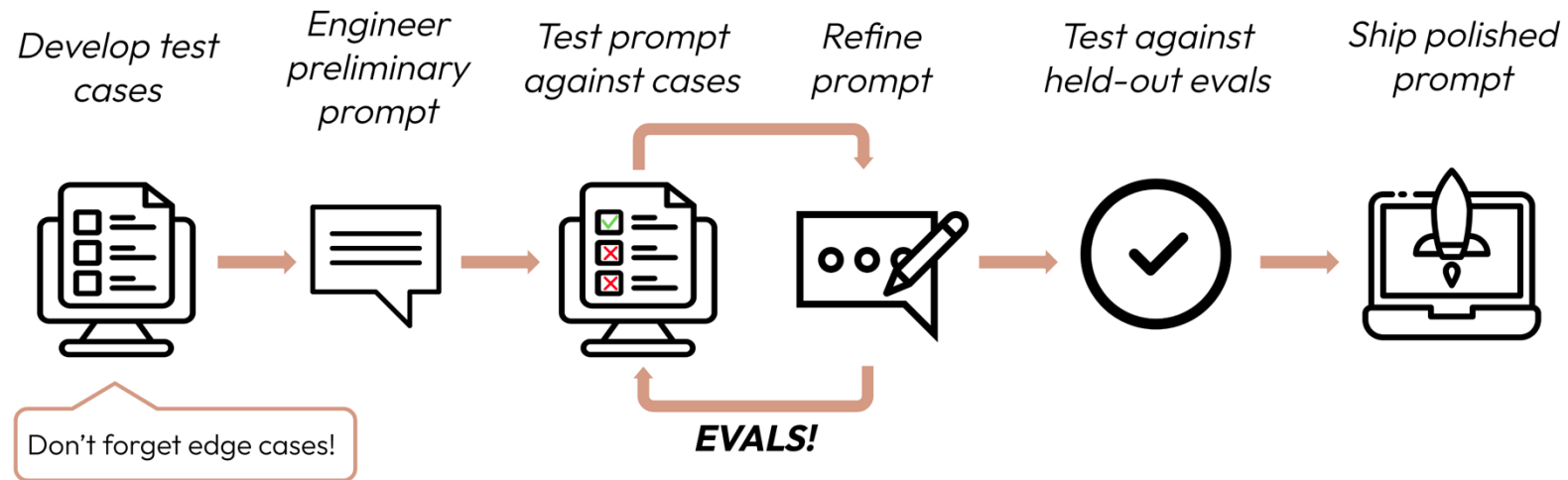
Criteria	
Bad	The model should classify sentiments well
Good	Our sentiment analysis model should achieve an F1 score of at least 0.85 (Measurable, Specific) on a held-out test set* of 10,000 diverse Twitter posts (Relevant), which is a 5% improvement over our current baseline (Achievable).

- Task fidelity
- Consistency
- Relevance and coherence
- Tone and style
- Privacy preservation
- Context utilization
- Latency
- Price

Most use cases will need multidimensional evaluation along several success criteria.

▸ Example multidimensional criteria for sentiment analysis

Create evals



e.g., in limited types of interactions within your website or retrieving data from the DB

Levels of tests

Eval design principles

1. **Be task-specific:** Design evals that mirror your real-world task distribution. Don't forget to factor in edge cases!

▸ Example edge cases

2. **Automate when possible:** Structure questions to allow for automated grading (e.g., multiple-choice, string match, code-graded, LLM-graded).
3. **Prioritize volume over quality:** More questions with slightly lower signal automated grading is better than fewer questions with high-quality human hand-graded evals.

How to prompt Claude?

1. [Prompt generator](#)
2. [Be clear and direct](#)
3. [Use examples \(multishot\)](#)
4. [Let Claude think \(chain of thought\)](#)
5. [Use XML tags](#)
6. [Give Claude a role \(system prompts\)](#)
7. [Prefill Claude's response](#)
8. [Chain complex prompts](#)
9. [Long context tips](#)

Clear and direct prompts

Give contextual information

- Where is this going to be used?
- Who is the audience?

Be specific about what you want Claude to do

Provide instructions as sequential steps

Prompt to think

Basic: Think step-by-step

Intermediate: Outline steps

The notes approach: Use tags <thinking> <answer>.

- Additionally, use summarization on thinking
- Add other elements like absorbing PDFs or notes

System prompts

You are a X. You provide the service Y to the firm Z.

Prefill responses

▼ Example: Structured data extraction without prefilling

Role	Content
User	<p>Extract the name, size, price, and color from this product description as a JSON object:</p> <p><description> The SmartHome Mini is a compact smart home assistant available in black or white for only \$49.99. At just 5 inches wide, it lets you control lights, thermostats, and other connected devices via voice or app—no matter where you place it in your home. This affordable little hub brings convenient hands-free control to your smart devices. </description></p>
Assistant	<p>Here's the extracted information in JSON format:</p> <pre>```json { "name": "SmartHome Mini", "size": "5 inches wide", "price": "\$49.99", "colors": ["black", "white"] } ```</pre> <p>I've extracted the following details from the product description:</p> <ul style="list-style-type: none">- Name: SmartHome Mini- Size: 5 inches wide- Price: \$49.99- Colors: Available in black and white

▼ Example: Structured data extraction with prefilling

Role	Content
User	<p>Extract the name, size, price, and color from this product description as a JSON object:</p> <p><description> The SmartHome Mini is a compact smart home assistant available in black or white for only \$49.99. At just 5 inches wide, it lets you control lights, thermostats, and other connected devices via voice or app—no matter where you place it in your home. This affordable little hub brings convenient hands-free control to your smart devices. </description></p>
Assistant (prefill)	<pre>{</pre>
Assistant (Claude's response)	<pre> "name": "SmartHome Mini", "size": "5 inches wide", "price": "\$49.99", "colors": ["black", "white"] }</pre>

Chain it up

- 1. Identify subtasks:** Break your task into distinct, sequential steps.
- 2. Structure with XML for clear handoffs:**
Use XML tags to pass outputs between prompts.
- 3. Have a single-task goal:** Each subtask should have a single, clear objective.
- 4. Iterate:** Refine subtasks based on Claude's performance.

Example chained workflows:

- **Multi-step analysis:** See the legal and business examples below.
- **Content creation pipelines:** Research → Outline → Draft → Edit → Format.
- **Data processing:** Extract → Transform → Analyze → Visualize.
- **Decision-making:** Gather info → List options → Analyze each → Recommend.
- **Verification loops:** Generate content → Review → Refine → Re-review.

Long prompts

Keep queries at the end (30% improvement in response)

Structure each document and metadata with XML

Add quoting capabilities with explicit indexing

D0: Make screens for your web app

Let's think through

- the kind of things our users require,
- the amount of reactivity required,
- source/destination for data
- Look/feel for the app

D0: Generate each screen using an API

Let's do Ollama APIs this time

Try two models, may be

- a phi3 and
- a deepseek-r1/llama 3.2

Managing data

DATABASES

<https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5>

What are databases?

Which excel sheets would this person maintain?

What information will I require for my screens?

DB -> Table -> Record -> Field

The image displays three overlapping database window screenshots. The top window shows the 'Products' table with columns 'Product ID', 'Product Name', and 'Supplier'. The middle window shows the 'Customers' table with columns 'Company Name' and 'Contact Name'. The bottom window shows the 'Orders' table with columns 'Order ID', 'Customer', and 'Employee'. The 'Orders' table is currently selected and shows a list of records with a status bar at the bottom indicating 'Record: 1 of 830'.

Product ID	Product Name	Supplier
1	Chai	Exotic Liquids
2	Chang	Exotic Liquids

Company Name	Contact Name
Alfreds Futterkiste	Maria Anders
Ana Trujillo Emparedados y helados	Ana Trujillo

Order ID	Customer	Employee
10248	Wilman Kala	Buchanan, Ste
10249	Tradição Hipermercados	Suyama, Mich
10250	Hanari Carnes	Peacock, Marg

Good DB Design

A good database design is, therefore, one that:

- Divides your information into subject-based tables to reduce redundant data.
- Provides Access with the information it requires to join the information in the tables together as needed.
- Helps support and ensure the accuracy and integrity of your information.
- Accommodates your data processing and reporting needs.

Designing your DB: The process

Determine the purpose of your database

Find and organize the information required

Divide the information into tables

Turn information items into columns

Specify primary keys

Set up the table relationships

Refine your design

Apply the normalization rules

The purpose

Number of users

- Is the user a single machine
- Or distributed machines that could work parallelly? DB on server

What will you do with the data?

- Which reports to be generated
- What type of user interactions are required?
- What kind of questions will you need to answer later, e.g., which regions provide the best revenue?

Gathering information

For each type of information, think of

- What field would you keep if you give them a paper form, e.g., a customer
- From your screen, what would you display for different components of the screen

Dividing information into tables

1NF: 1st Normal Form (Atomicity)

Student_ID	Name	Courses
1	Alice	Math, Science
2	Bob	History

Student_ID	Name	Course
1	Alice	Math
1	Alice	Science
2	Bob	History

A table is in **1NF** if:

1. Each column contains **atomic (indivisible) values** (no lists, arrays, or nested values).
2. Each column contains **values of a single type**.
3. Each row has a **unique identifier (Primary Key)**.

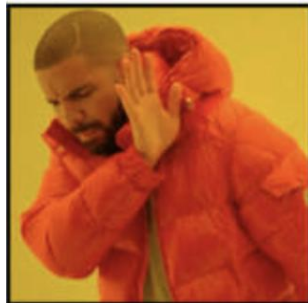


2NF: No partial dependencies

A table is in **2NF** if:

1. It is in **1NF**.
2. **All non-key attributes must depend on the entire primary key**, not just a part of it.

Student_ID	Course_ID	Student_Name	Course_Name
1	101	Alice	Math
2	102	Bob	Science



Student Name depends on
Student_ID and Course_Name
depends on Course ID



Students Table:

Student_ID	Student_Name
1	Alice
2	Bob

Courses Table:

Course_ID	Course_Name
101	Math
102	Science

Student_Courses (Joining Table):

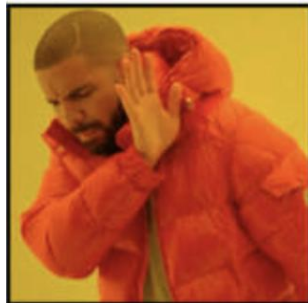
Student_ID	Course_ID
1	101
2	102

3NF: No transitive dependencies

A table is in **3NF** if:

1. It is in **2NF**.
2. **All non-key attributes depend only on the primary key** (no indirect dependencies).

Student_ID	Student_Name	Course_ID	Course_Name	Instructor
1	Alice	101	Math	Dr. Smith
2	Bob	102	Science	Dr. Jones



Here, Course_Name and Instructor depend on Course_ID, not Student_ID, creating a **transitive dependency**.

Courses Table:

Course_ID	Course_Name	Instructor
101	Math	Dr. Smith
102	Science	Dr. Jones

Student_Courses Table:

Student_ID	Course_ID
1	101
2	102

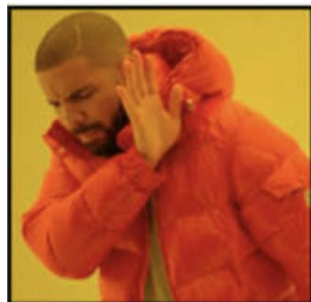


BCNF: Boyce Codd Normal Form

BCNF is a stricter version of **3NF**. A table is in **BCNF** if:

1. It is in **3NF**.
2. **Every determinant is a candidate key.**

Professor	Course	Department
Dr. Smith	Math	Science
Dr. Jones	History	Arts



Here, Professor \rightarrow Course, but Course \rightarrow Department. This means Course should be a primary key.

Courses Table:

Course	Department
Math	Science
History	Arts

Professors Table:

Professor	Course
Dr. Smith	Math
Dr. Jones	History

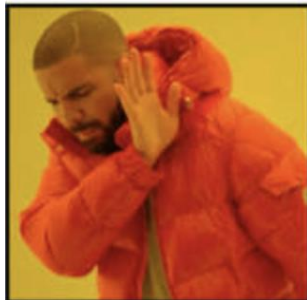


4th NF: No multi-valued dependencies

A table is in **4NF** if:

1. It is in **BCNF**.
2. No column contains two or more independent multi-valued facts.

Course	Instructor	Book
Math	Dr. Smith	Algebra 101
Math	Dr. Smith	Geometry 202
Math	Dr. Brown	Algebra 101



Instructor and
book are
independent

Course_Instructors Table:

Course	Instructor
Math	Dr. Smith
Math	Dr. Brown

Course_Books Table:

Course	Book
Math	Algebra 101
Math	Geometry 202



When to Stop Normalizing?

- 3NF is sufficient for most practical applications.
- BCNF is used in complex enterprise applications.
- 4NF and 5NF are used in advanced use cases like data warehouses.

Normal Form	Rule
1NF	No repeating groups or multi-valued attributes
2NF	No partial dependencies (all attributes depend on the whole primary key)
3NF	No transitive dependencies (only depends on primary key)
BCNF	Every determinant is a candidate key
4NF	No multi-valued dependencies
5NF	No join dependencies

Recap from guest lecture:

Designing your DB

Start with Content

Create a data model

- Break down content model into objects
- Define properties and types
- Map Object relationships

Wanted: A Senior Web Developer

Title

We are looking for a senior level web developer (with around 5–7 years of experience) to help us continue making great websites. This is a remote position. You may be located anywhere in India or the rest of the world (with at least a 4-hour time overlap with the Indian timezone).

Description

• This is a full-time position

Type

• We work remotely

Location

• Experience required: 5-7 years

Experience

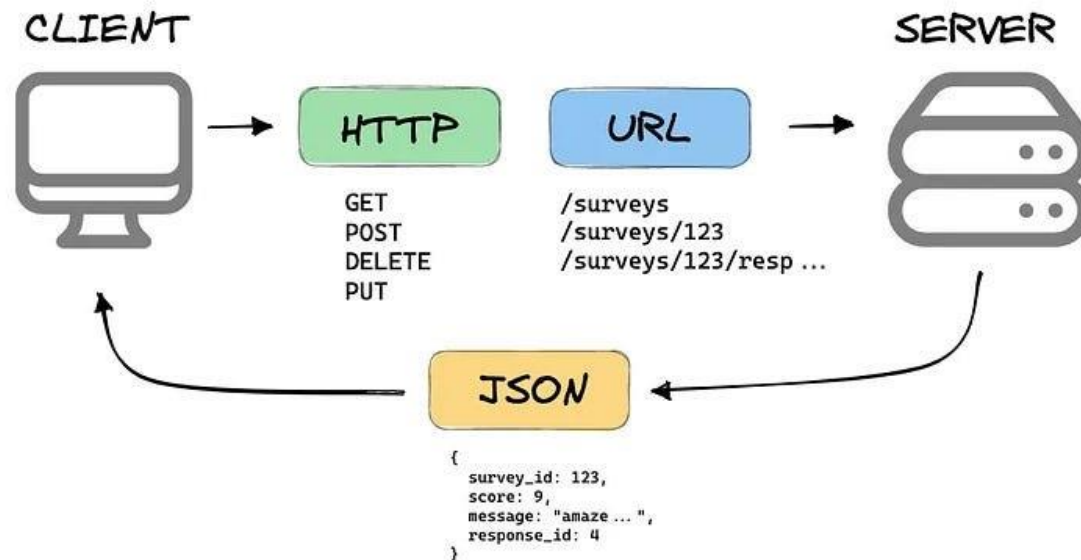
• Typical compensation is Rs. 15L per annum

Compensation

Requests and API

Making requests

WHAT IS A REST API?



Source: <https://medium.com/@shikha.ritu17/rest-api-architecture-6f1c3c99f0d3>



Let's see a quick demo

Todo List API Demo

Add New Todo POST

Add Todo

Update Todo PUT

Update Todo

Delete Todo DELETE

Delete Todo



FastAPI is a modern, fast (high-performance), web framework for building APIs with Python based on standard Python type hints.

The key features are:

- Fast: Very high performance, on par with NodeJS and Go (thanks to Starlette and Pydantic). [One of the fastest Python frameworks available](#).
- Fast to code: Increase the speed to develop features by about 200% to 300%. *
- Fewer bugs: Reduce about 40% of human (developer) induced errors. *
- Intuitive: Great editor support. Completion everywhere. Less time debugging.
- Easy: Designed to be easy to use and learn. Less time reading docs.
- Short: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- Robust: Get production-ready code. With automatic interactive documentation.
- Standards-based: Based on (and fully compatible with) the open standards for APIs: [OpenAPI](#) (previously known as Swagger) and [JSON Schema](#).

Typing hints for FastAPI

With **FastAPI** you declare parameters with type hints and you get:

- **Editor support.**
- **Type checks.**

...and **FastAPI** uses the same declarations to:

- **Define requirements:** from request path parameters, query parameters, headers, bodies, dependencies, etc.
- **Convert data:** from the request to the required type.
- **Validate data:** coming from each request:
 - Generating **automatic errors** returned to the client when the data is invalid.
- **Document** the API using OpenAPI:
 - which is then used by the automatic interactive documentation user interfaces.

The basic app



Async calls

If you are using third party libraries that tell you to call them with `await`, like:

```
results = await some_library()
```

Then, declare your *path operation functions* with `async def` like:

```
@app.get('/')
async def read_results():
    results = await some_library()
    return results
```



Common reasons to wait:

- the data from the client to be sent through the network
- the data sent by your program to be received by the client through the network
- the contents of a file in the disk to be read by the system and given to your program
- the contents your program gave to the system to be written to disk
- a remote API operation
- a database operation to finish
- a database query to return the results etc.

Concurrency vs Parallelism

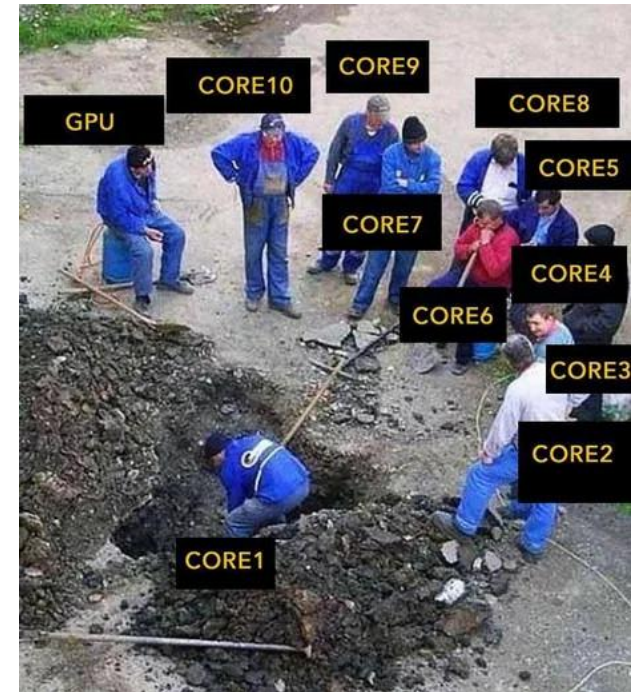
Concurrency

Tell someone to finish something while you do your work, e.g., ordering food

Parallelism

Get a number of agents supporting you in your work, e.g., cleaning your house

FastAPI is both, feat Starlette.



Parallelism is not always useful, nor is concurrency

Let's go through

- Basic FastAPI app
- Doc/Redoc

What is a schema?

- The definition of format and purpose of a certain path/variable
 - API schema
 - Data schema

Servers

One of:

- POST
- GET
- PUT
- DELETE

...and the more exotic ones:

- OPTIONS
- HEAD
- PATCH
- TRACE

Request



Response

- HTML
- Text
- JSON



Refer main.py in
fastapi-demo for
examples



Collecting data for APIs

```
javascript Copy Edit

document.getElementById("myForm").addEventListener("submit", async function(event) {
    event.preventDefault(); // Prevent default form submission

    const formData = new FormData(this); // Create FormData object from form

    try {
        const response = await fetch("https://example.com/api/submit", {
            method: "POST",
            body: formData, // Send form data
        });

        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }

        const result = await response.json();
        console.log("Success:", result);
        alert("Form submitted successfully!");
    } catch (error) {
        console.error("Error:", error);
        alert("Failed to submit form.");
    }
});
```

Form

```
function createTodo() {
    const text = document.getElementById('newTodo').value;
    if (!text) return;

    fetch('http://localhost:3000/api/todos', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ text })
    })
    .then(response => {
        if (!response.ok) throw new Error('Network response was not ok');
        return response.json();
    })
    .then(() => {
        loadTodos();
        document.getElementById('newTodo').value = '';
    })
    .catch(error => {
        console.error('Error:', error);
        alert('Failed to create todo');
    });
}
```

From individual fields (DOM Tree)

Collecting files

```
168
169 @app.post("/uploadfile/")
170 async def create_upload_file(file: UploadFile):
171     return {"filename": file.filename}
172
173
174 @app.post("/uploadfiles/")
175 async def create_upload_files(files: List[UploadFile]):
176     return {"filenames": [file.filename for file in files]}
177
178 @app.get("/upload")
179 async def main():
180     content = ""
181     <body>
182     <form action="/files/" enctype="multipart/form-data" method="post">
183     <input name="files" type="file" multiple>
184     <input type="submit">
185     </form>
186     <form action="/uploadfiles/" enctype="multipart/form-data" method="post">
187     <input name="files" type="file" multiple>
188     <input type="submit">
189     </form>
190     </body>
191     ""
192     return HTMLResponse(content=content)
```

You, 45 seconds ago • Uncommitted cha

Choose Files

No file chosen

Submit

Choose Files

No file chosen

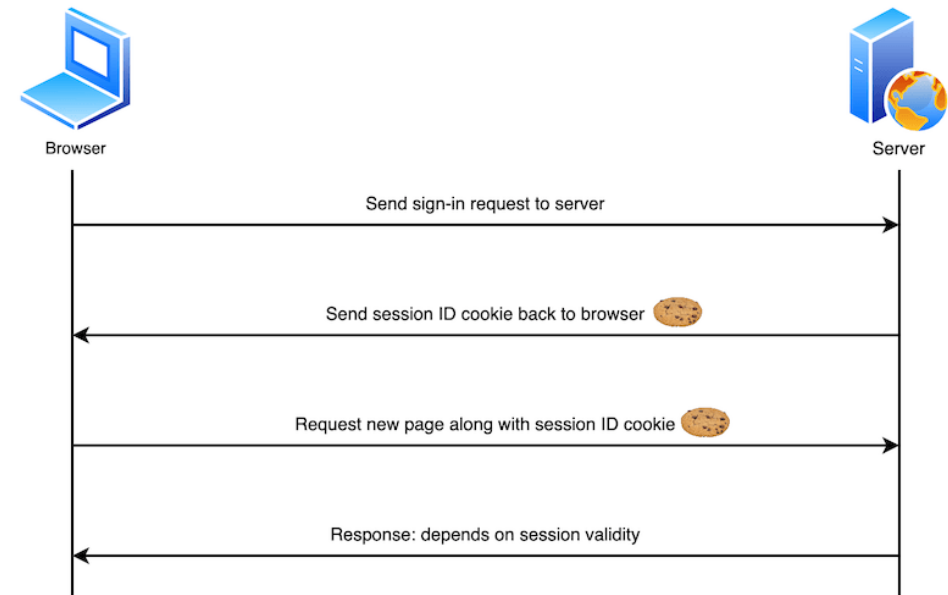
Submit

Cookies

Small amount of data sent by server to your browser

Remember state information

- Session management
- Personalization
- Tracking



Storage

sessionStorage

- Maintains a storage area for active sessions (including reload/restore)
- Resets if you close the browser

localStorage

- Persists on browser re-open

Response

FastAPI limits response to the defined types

- Gives server error if incomplete or ill-formed data is received from the server

JSON Response

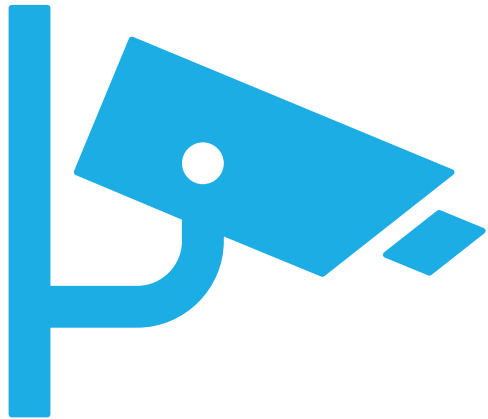
Redirect Response

File Response



- **100 - 199** are for "Information". You rarely use them directly. Responses with these status codes cannot have a body.
- **200 - 299** are for "Successful" responses. These are the ones you would use the most.
 - **200** is the default status code, which means everything was "OK".
 - Another example would be **201**, "Created". It is commonly used after creating a new record in the database.
 - A special case is **204**, "No Content". This response is used when there is no content to return to the client, and so the response must not have a body.
- **300 - 399** are for "Redirection". Responses with these status codes may or may not have a body, except for **304**, "Not Modified", which must not have one.
- **400 - 499** are for "Client error" responses. These are the second type you would probably use the most.
 - An example is **404**, for a "Not Found" response.
 - For generic errors from the client, you can just use **400**.
- **500 - 599** are for server errors. You almost never use them directly. When something goes wrong at some part in your application code, or server, it will automatically return one of these status codes.

Status Codes

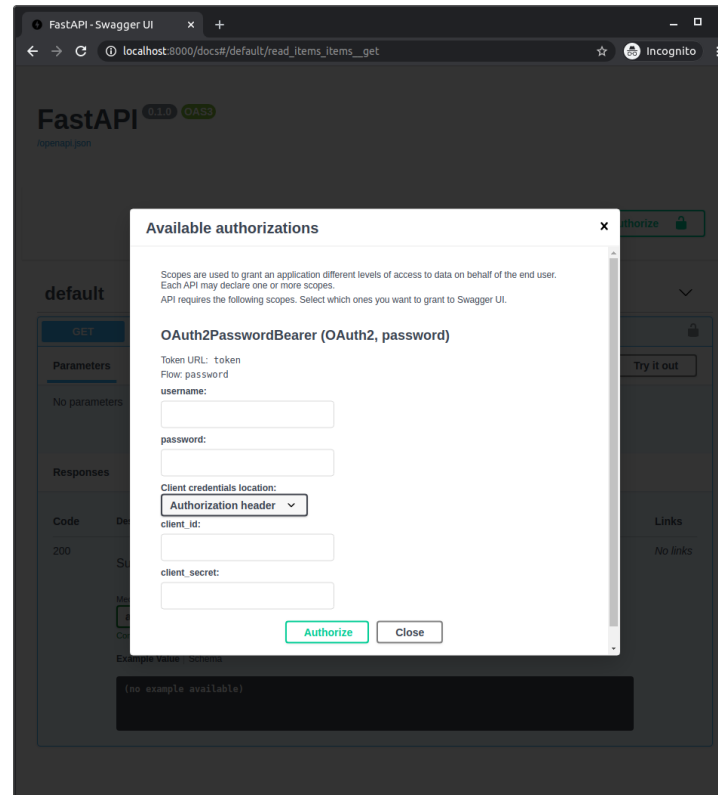
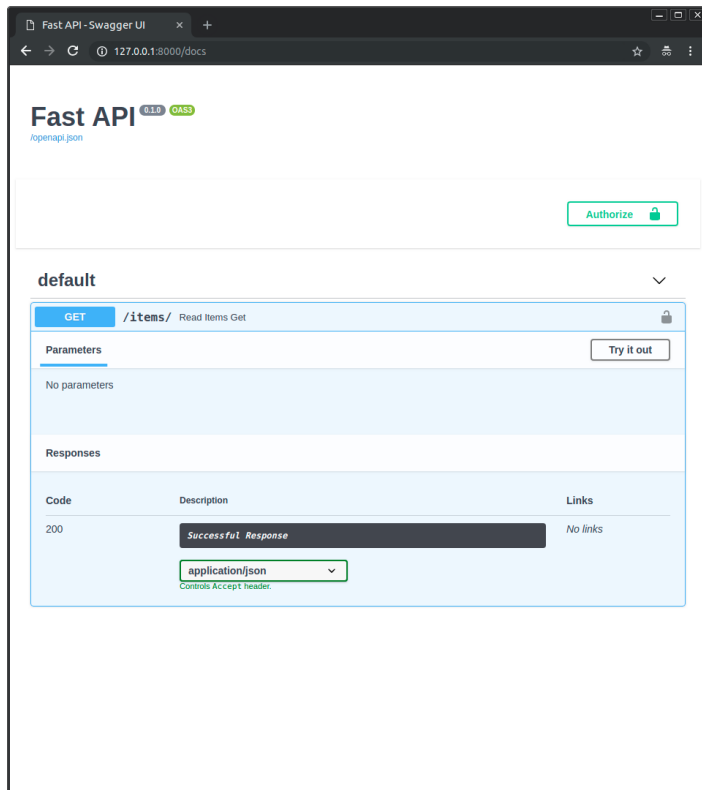


Security

Authentication – OAuth2

```
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
```

```
@app.get("/items/")
async def read_items(token: Annotated[str, Depends(oauth2_scheme)]):
    return {"token": token}
```



OAuth2

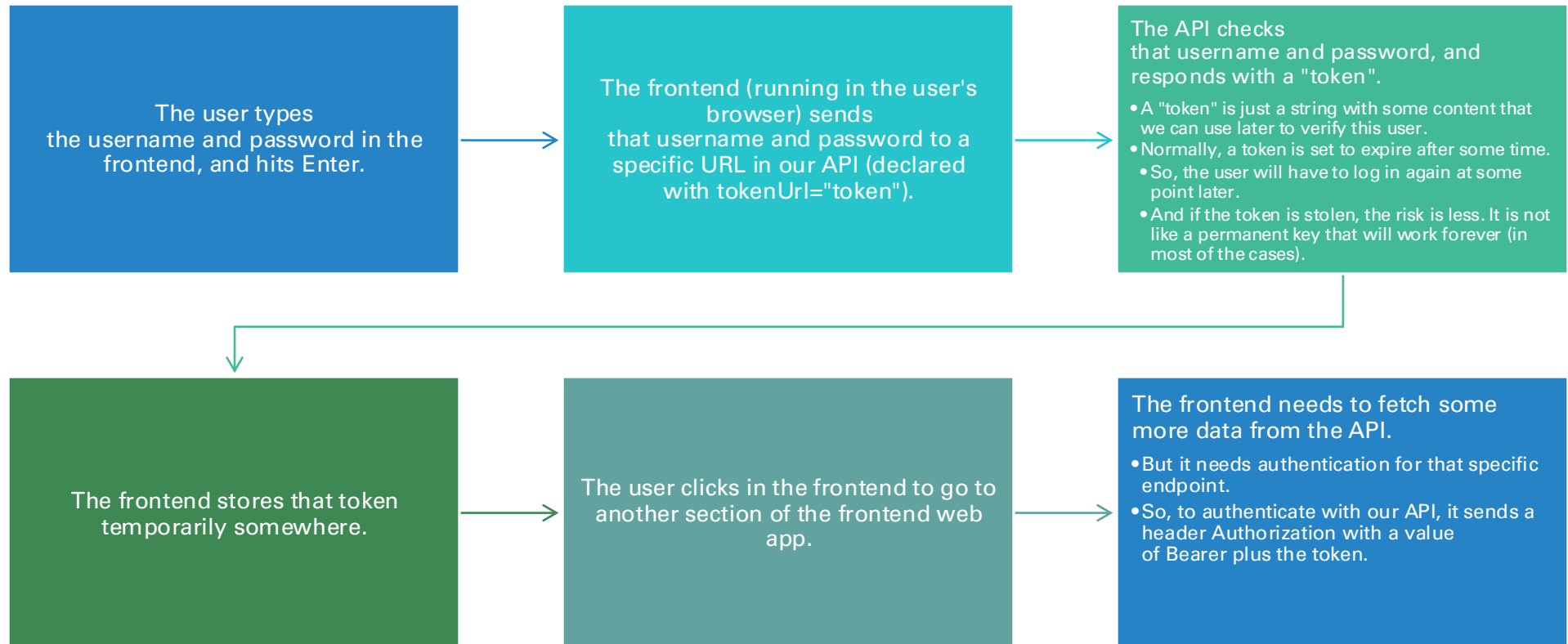
Designed to isolate authentication from backend/API

- Same FastAPI application can also handle the API and authentication (like here)

Defines different types of flows

- Password flow
- Refresh token flow
- Device authorization flow, etc.

Authentication Flow



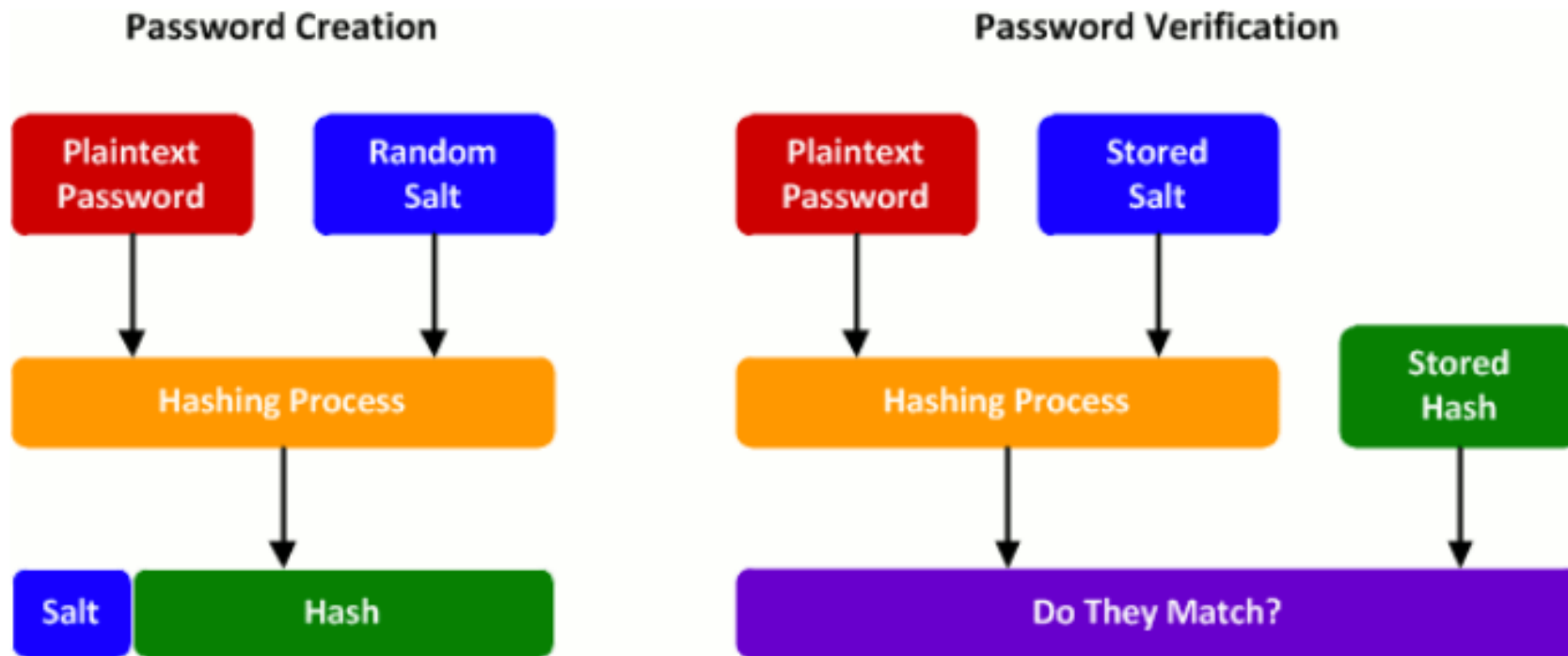
What does it do?

It will go and look in the request for that Authorization header, check if the value is Bearer plus some token, and will return the token as a str.

If it doesn't see an Authorization header, or the value doesn't have a Bearer token, it will respond with a 401 status code error (UNAUTHORIZED) directly.

You don't even have to check if the token exists to return an error. You can be sure that if your function is executed, it will have a str in that token.

Passwords (login routine)



The complete auth with JWT tokens and password hashing (bcrypt)

<https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/#update-the-dependencies>

TODO: Screens + User Journey

Please complete and document if not already done

TODO: Design an API interface

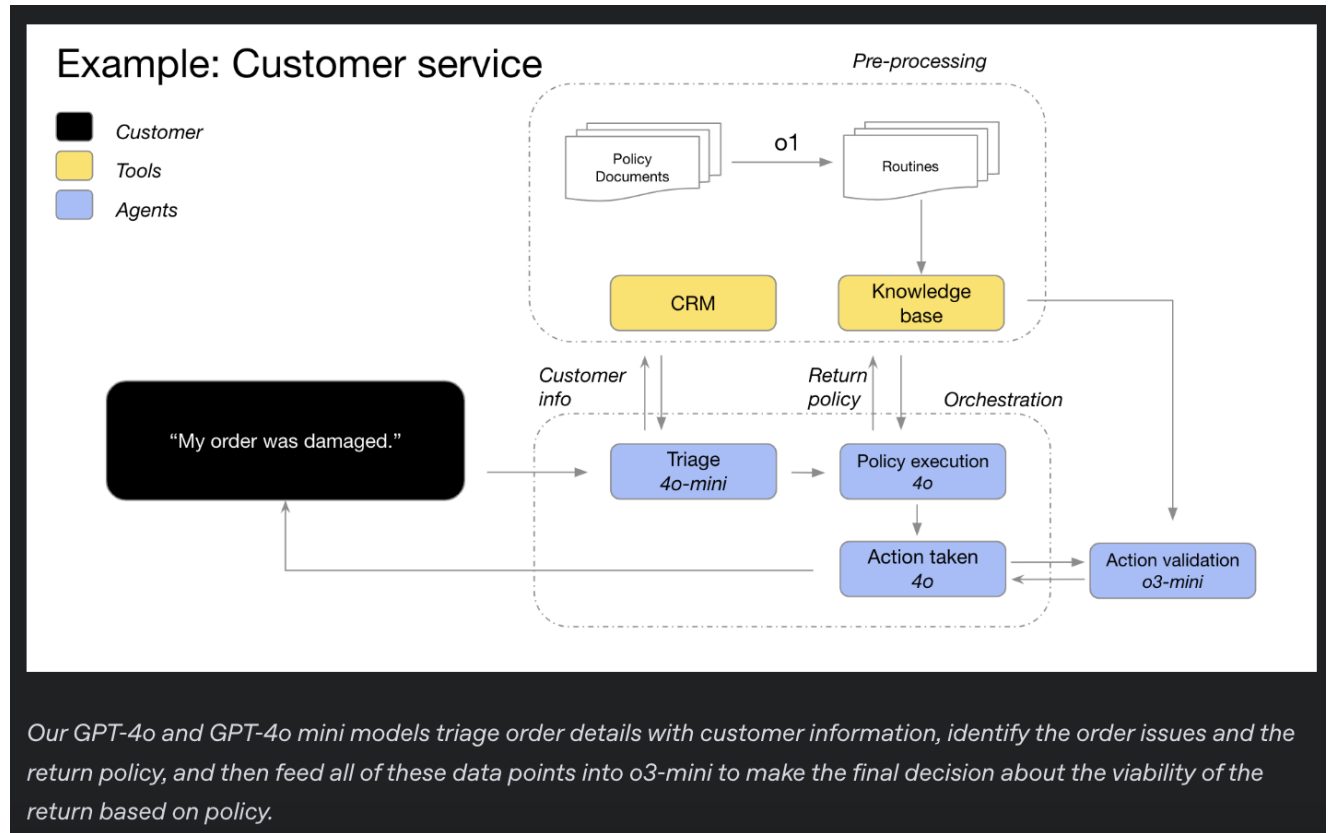
Based on the functionality you imagine that you need, design an API interface with dummy calls but decent return response

Build your web pages and verify your user journey

Using reasoning models

<https://platform.openai.com/docs/guides/reasoning-best-practices>

The orchestrator



Recommended use cases

Navigating ambiguity

Finding needles in a haystack

Finding relationships and nuances
across a large dataset

“o1’s reasoning capabilities enable our multi-agent platform Matrix to produce exhaustive, well-formatted, and detailed responses when processing complex documents. For example, o1 enabled Matrix to easily identify baskets available under the restricted payments capacity in a credit agreement, with a basic prompt. No former models are as performant. o1 yielded stronger results on 52% of complex prompts on dense Credit Agreements compared to other models.”

—[Hebbia](#), AI knowledge platform company for legal and finance

“To analyze a company’s acquisition, o1 reviewed dozens of company documents—like contracts and leases—to find any tricky conditions that might affect the deal. The model was tasked with flagging key terms and in doing so, identified a crucial “change of control” provision in the footnotes: if the company was sold, it would have to pay off a \$75 million loan immediately. o1’s extreme attention to detail enables our AI agents to support finance professionals by identifying mission-critical information.”

—[Endex](#), AI financial intelligence platform

“Tax research requires synthesizing multiple documents to produce a final, cogent answer. We swapped GPT-4o for o1 and found that o1 was much better at reasoning over the interplay between documents to reach logical conclusions that were not evident in any one single document. As a result, we saw a 4x improvement in end-to-end performance by switching to o1—incredible.”

—[Blue J](#), AI platform for tax research

Recommended use cases contd

Multi-step agentic reasoning

Visual reasoning

Code

“We use o1 as the planner in our agent infrastructure, letting it orchestrate other models in the workflow to complete a multi-step task. We find o1 is really good at selecting data types and breaking down big questions into smaller chunks, enabling other models to focus on execution.”

—[Argon AI](#), AI knowledge platform for the pharmaceutical industry

“We automate risk and compliance reviews for millions of products online, including luxury jewelry dupes, endangered species, and controlled substances. GPT-4o reached 50% accuracy on our hardest image classification tasks. o1 achieved an impressive 88% accuracy without any modifications to our pipeline.”

—[SafetyKit](#), AI-powered risk and compliance platform

“o3-mini consistently produces high-quality, conclusive code, and very frequently arrives at the correct solution when the problem is well-defined, even for very challenging coding tasks. While other models may only be useful for small-scale, quick code iterations, o3-mini excels at planning and executing complex software design systems.”

—[Windsurf](#), collaborative agentic AI-powered IDE, built by Codeium

Recommended use cases contd

Evaluation and benchmarking for other model responses

- LLM as a judge

Do's and Don'ts with reasoning models

DOs

Developer messages are the new system messages

- Platform > Developer > User > Tool
- Keep prompts simple and direct

Better zero-shot success

Provide all constraints

DON'Ts

Avoid asking to 'think step-by-step' or 'explain your reasoning'

Be vague about end goal

Level of reasoning

Type of data:

- Patient ID: A randomly generated patient id
- Date of Birth: Date of birth of the patient
- Gender: M/F
- Medical History: Past diagnoses
- Current Medications: Medication the patient is taking
- Allergies: Identified allergies
- Lab Results (Glucose mg/dL)
- Diagnoses: Current diagnosis
- Treatment Plan: Current treatment plan
- Is Valid: Whether or not the current row of data is valid (True/False)
- Issue: If the row of data is not valid, what the issue is

Validations to be performed:

- Prescribing medications that the patient is allergic to
- Current medications do not match medical history
- Treatment plan does not match diagnosis

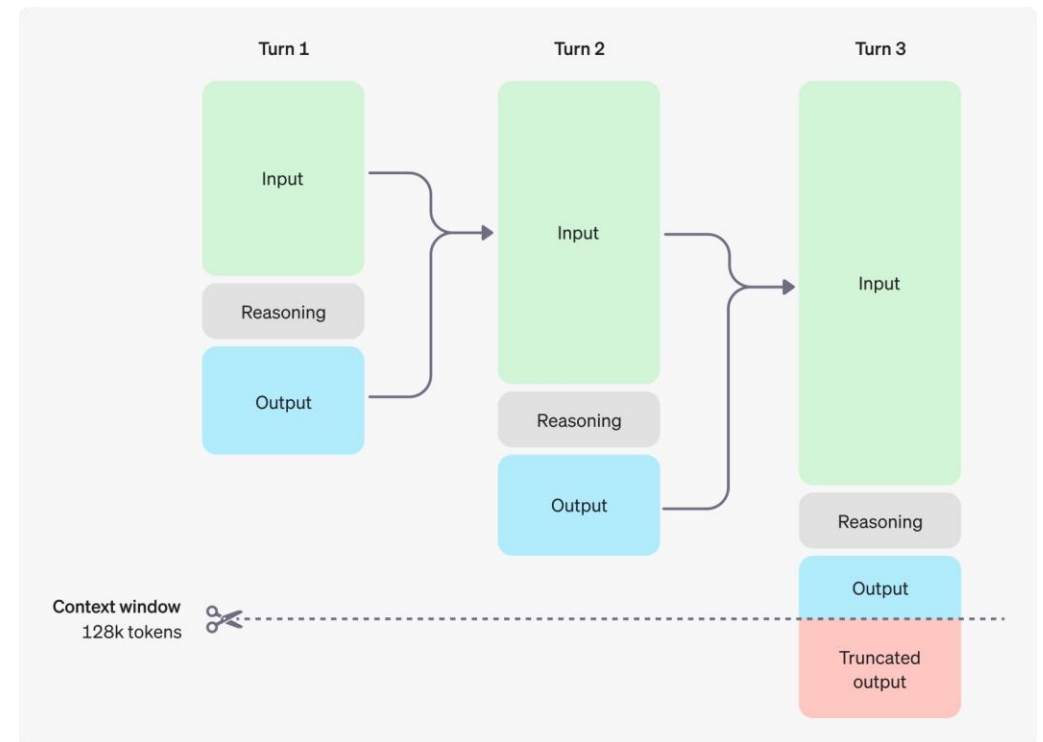
https://cookbook.openai.com/examples/o1/using_reasoning_for_data_validation

How to?

Using a reasoning model in Chat Completions

python ↕

```
1 from openai import OpenAI
2
3 client = OpenAI()
4
5 prompt = """
6 Write a bash script that takes a matrix represented as a string with
7 format '[1,2],[3,4],[5,6]' and prints the transpose in the same format.
8 """
9
10 response = client.chat.completions.create(
11     model="o3-mini",
12     reasoning_effort="medium",
13     messages=[
14         {
15             "role": "user",
16             "content": prompt
17         }
18     ]
19 )
20
21 print(response.choices[0].message.content)
```



Note: your reasoning is part of tokens

D0: Which of your endpoints can use reasoning?

List these API endpoints separately

Try out models

- O1
- O3-mini
- Gemini Flash 2.0 Thinking
- Deepseek-r1

D0: Within your prompts for web dev, which prompts took too much effort?

Try reasoning models to generate prompts

Try reasoning models to do the task

In-built tools

OpenAI

- Web search
- File search
- Computer use!

Claude

- Use function calling

Google

- [Google search](#)

```
from google import genai
from google.genai.types import Tool, GenerateContentConfig, GoogleSearch

client = genai.Client()
model_id = "gemini-2.0-flash"

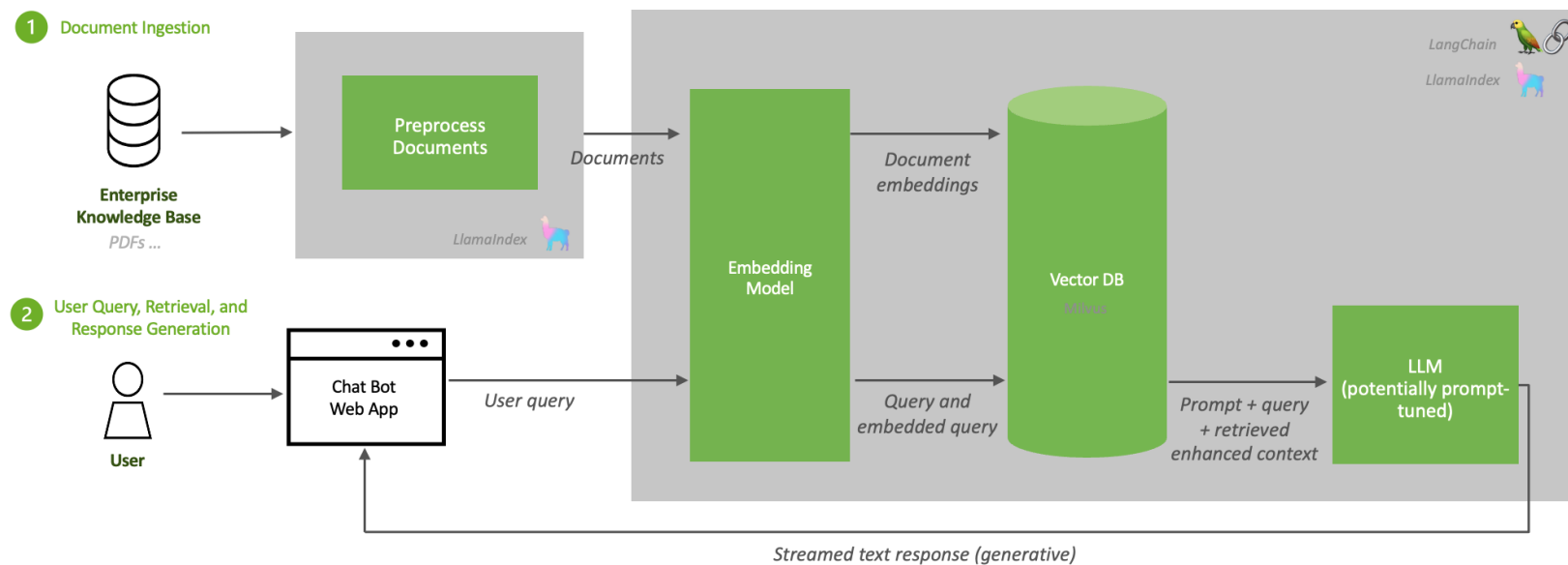
google_search_tool = Tool(
    google_search = GoogleSearch()
)

response = client.models.generate_content(
    model=model_id,
    contents="When is the next total solar eclipse in the United States?",
    config=GenerateContentConfig(
        tools=[google_search_tool],
        response_modalities=["TEXT"],
    )
)
```

Introduction to RAG

What's RAG?

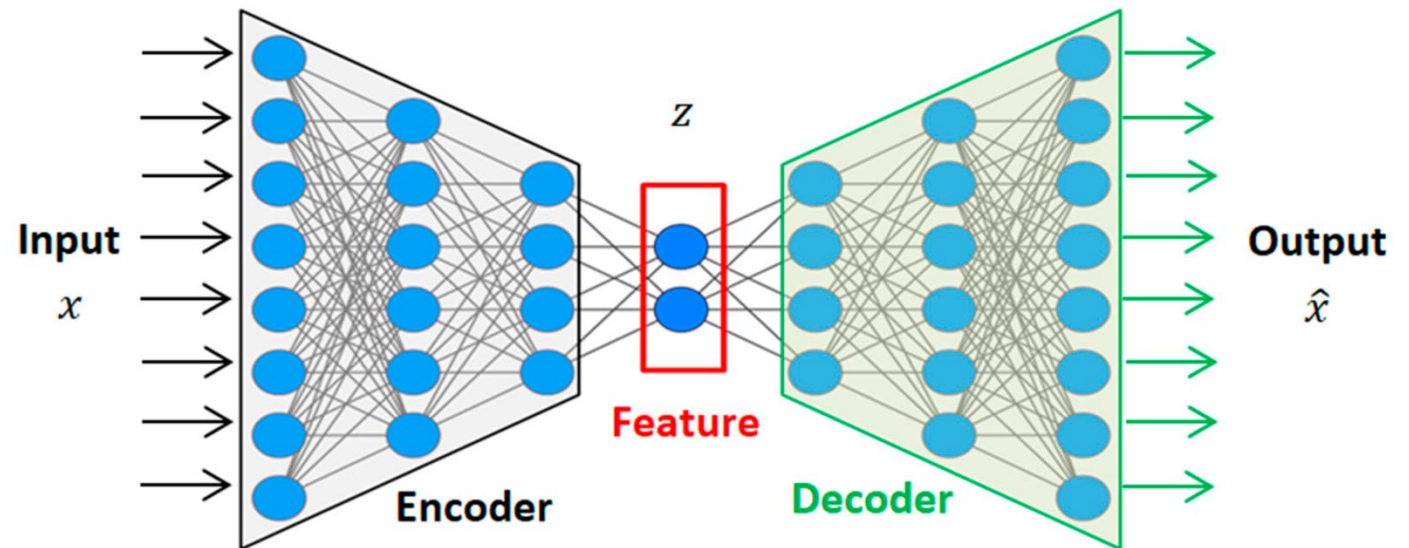
Retrieval Augmented Generation (RAG) Sequence Diagram



Source: <https://developer.nvidia.com/blog/rag-101-demystifying-retrieval-augmented-generation-pipelines/>

Vector embeddings

- Compressed latent space, typically built to capture 'meaning'
- Training pipelines typically bring similar inputs together. Hence, utility in
 - Search
 - Clustering
 - Recommendation
 - Anomaly, etc.



Do we need a DB?

Functions such as search, cluster, rank, retrieve are efficiently implemented using vector databases

Some of them are also optimized for use in cloud, being a common use case

Weaviate

Pinecone

ChromaDB

Redis Stack

ElasticSearch

Different types of search

Sparse vs Dense Search

- **Bag of Words**

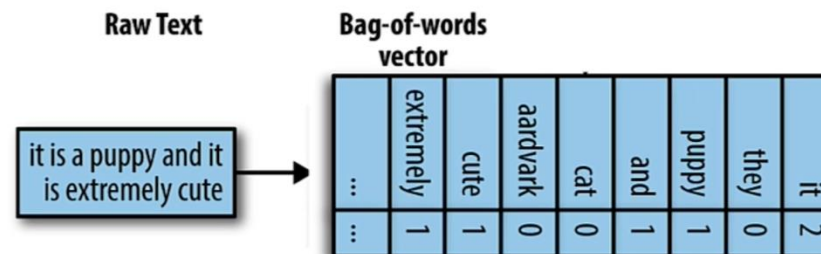
- The easiest way to do keyword matching is using Bag of Words – to count how many times a word occurs in the query and the data vector and then return objects with the highest matching word frequencies.

- **This is known as Sparse Search**

- because the text is embedded into vectors by counting how many times every unique word in your vocabulary occurs in the query and stored sentences.

- **Mostly Zeroes (Sparse Embedding)**

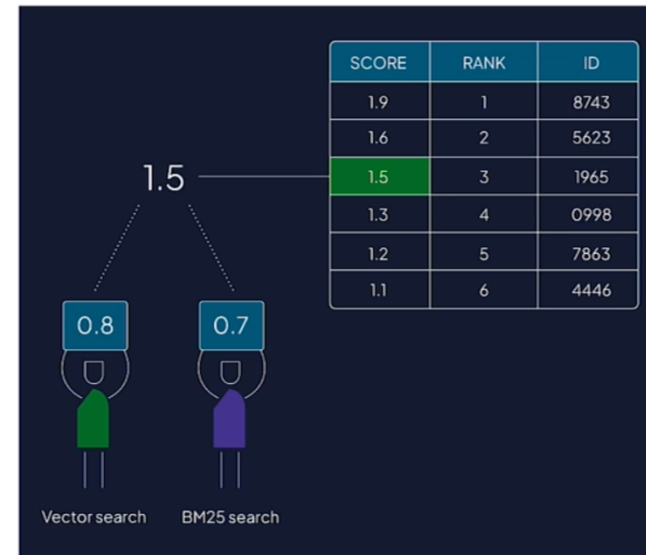
- Since the likelihood of any given sentence containing every word in your vocabulary is quite low the embeddings is mostly zeroes and thus is known as a sparse embedding.



Hybrid search

Hybrid Search

- **What is Hybrid Search?**
 - Hybrid search is the process of performing both vector/dense search and keyword/sparse search and then combining the results
- **Combination based on a Scoring System**
 - This combination can be done based on a scoring system that measures how well each objects matches the query using both dense and sparse searches.



Common use cases

Who uses Weaviate?

- **Software Engineers**
 - Weaviate is an ML-first database engine
 - Out-of-the-box modules for AI-powered searches, automatic classification
 - Full CRUD support
 - Cloud-native, distributed system that runs well on Kubernetes
 - Scales with your workloads
- **Data Engineers**
 - Weaviate is a fast, flexible vector database
 - Use your own ML model or third party models
 - Run locally or with an inference service
- **Data Scientists**
 - Seamless handover of Machine Learning models to engineers and ML
 - Deploy and maintain your ML models in production reliably and efficiently
 - Easily package custom trained models

Who's providing embeddings?

Ollama

- Nomic
- Snowflake

OpenAI

- Text-embedding-3-small
- Text-embedding-3-large

Gemini

- gemini-embedding-exp-03-07
- text-embedding-004
- embedding-001

Claude

Available Models

Voyage recommends using the following text embedding models:

Model	Context Length	Embedding Dimension	Description
<code>voyage-3-large</code>	32,000	1024 (default), 256, 512, 2048	The best general-purpose and multilingual retrieval quality.
<code>voyage-3</code>	32,000	1024	Optimized for general-purpose and multilingual retrieval quality. See blog post for details.
<code>voyage-3-lite</code>	32,000	512	Optimized for latency and cost. See blog post for details.
<code>voyage-code-3</code>	32,000	1024 (default), 256, 512, 2048	Optimized for code retrieval. See blog post for details.
<code>voyage-finance-2</code>	32,000	1024	Optimized for finance retrieval and RAG. See blog post for details.
<code>voyage-law-2</code>	16,000	1024	Optimized for legal and long-context retrieval and RAG. Also improved performance across all domains. See blog post for details.

Sample use cases

Constructing context from many different sources, often requiring embedding based search/match

- "Suggest trending tech and non-tech jobs I can apply for with my degree."
- "Compare short-term courses for data science vs. digital marketing with fees and reviews."

Why RAG?: Retrieves real-time course and job data from websites and combines it with personalized advice.

• "Plan a Goa trip under ₹10,000 for 4 friends with hotel, food, and places to visit."

• "What are offbeat travel places near Delhi for a 2-day trip?"

Why RAG?: Combines latest travel blogs, real-time hotel/transport prices, and customizes itinerary.

- "Suggest 5 trending research topics in computer vision for final-year B.Tech project."
- "What are the latest AI techniques being used for water management in Indian cities?"

Why RAG?: Retrieves recent papers, conference proceedings, and Google Scholar trends to recommend topics.

Implement with weaviate

Code

```
collection_name = "GitBookChunk"

chunks = client.collections.get(collection_name)
response = chunks.query.near_text(query="history of git", limit=3)
```

Result

```
{
  "data": {
    "Get": {
      "GitBookChunk": [
        {
          "chapter_title": "01-introduction",
          "chunk": "=== A Short History of Git\n\nAs with many great things in life, Git began in 2005.",
          "chunk_index": 0
        },
        {
          "chapter_title": "01-introduction",
          "chunk": "== Nearly Every Operation Is Local\n\nMost operations in Git need only local files and resources, making them fast and efficient.",
          "chunk_index": 2
        },
        {
          "chapter_title": "02-git-basics",
          "chunk": "==\n\nYou can specify more than one instance of both the `--author` and `--committer` options.",
          "chunk_index": 2
        }
      ]
    }
  }
}
```

```
collection_name = "GitBookChunk"

chunks = client.collections.get(collection_name)
response = chunks.generate.near_text(
    query="history of git",
    limit=3,
    grouped_task="Summarize the key information here in bullet points"
)

print(response.generated)
```

RAG
on
result

The G in RAG

- Git began as a replacement for the proprietary DVCS called BitKeeper, which was used by both
- The relationship between the Linux development community and BitKeeper broke down in 2005,
- Git was designed with goals such as speed, simple design, strong support for non-linear development
- Most operations in Git only require local files and resources, making them fast and efficient
- Git allows browsing project history instantly and can calculate differences between files
- Git allows offline work and does not require a network connection for most operations.
- This book was written using Git version 2, but most commands should work in older versions

Apply LLM to each result

```
collection_name = "WineReview"

reviews = client.collections.get(collection_name)
response = reviews.generate_near_text(
    query="fruity white wine",
    limit=3,
    single_prompt="""
        Translate this review into French, using emojis:
        ===== Country of origin: {country}, Title: {title}, Review body: {review_body}
        =====
    """)
```

==== Generated text ====



Origine : États-Unis

Titre : Schmitz 24 Brix 2012 Sauvignon Blanc (Sierra Foothills)

Corps de la critique : Pas du tout un Sauvignon Blanc typique, il sent l'abricot et le chèvrefeuille et a le goût de la marmelade. Il est sec, mais a le goût d'un vin de dessert tardif. Attendez-vous à une petite aventure gustative ici.

===== Original review =====

Country: US,

Title: Schmitz 24 Brix 2012 Sauvignon Blanc (Sierra Foothills)

Review body Not at all a typical Sauvignon Blanc, this smells like apricot and honeysuckle and tastes like marmalade. It is dry, yet tastes like a late-harvest dessert wine. Expect a little taste adventure here.

Consolidated example

Refer

<https://github.com/rjalexa/booksimil/blob/main/booksimil/mybooks.ipynb>

Fresh off the oven

OpenAI now supports file searches without explicit chunking strategies, etc. using Responses API

https://cookbook.openai.com/examples/file_search_responses

New tools in town

- Web Search
- File Search
- Computer Use

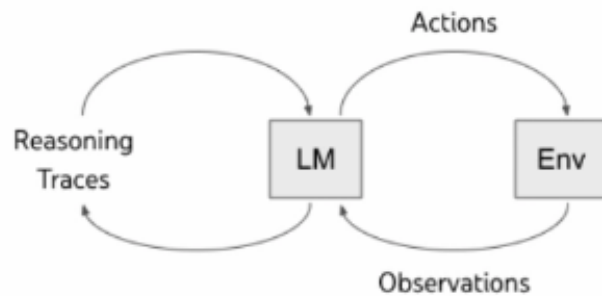
Agents

Agent prototypes

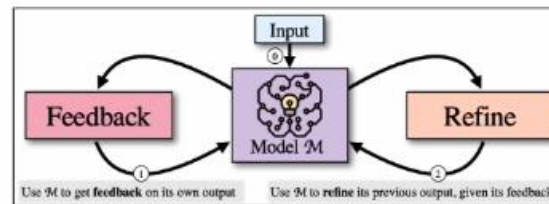
Narrow vertical expertise

- Outlining and planning
- Someone good at searching sources and writing a first draft
- A master editor

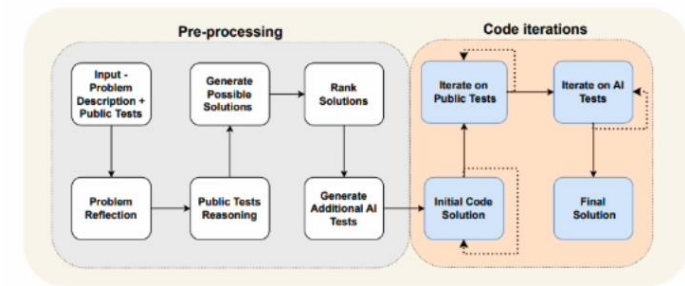
Agentic Workflows



ReAct (Reason + Act)



SELF-REFINE:
Iterative Refinement with Self-Feedback



Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering

Why?

Planning before execution

Tool use

Reflecting on intermediate results

Multiple perspectives

Common memory to track progress

Let's build one

Refer to notebook in web-dev-with-ai repo

- dlai-build-an-agent.ipynb
- DO: Modify this for gemini and run!
- Questions to ponder:
 - How will you use this for multiple conversations in parallel? (Streaming)
 - How will you do very long tasks, e.g., build this game (Persistence)
 - How to implement this across different runs and possibly shutdowns?

Use tools (and build tools)

<https://python.langchain.com/docs/integrations/tools/>

Remember, tool's output is for an LLM, not a human!

```
search = DuckDuckGoSearchResults(output_format="list")  
  
search.invoke("Obama")
```

```
[{'snippet': 'Obama was headed to neighboring Michigan later Tuesday, among the several stops th',  
  'title': 'Obama and Walz host rally in Wisconsin as early voting kicks off | AP News',  
  'link': 'https://apnews.com/article/wisconsin-voting-trump-harris-obama-walz-aefff20ab17a54172',  
  'stop': True},  
 {'snippet': 'Obama has directed plenty of criticisms at Trump over the years, so some might per',  
  'title': 'Why Obama slamming Trump on his response to Covid matters - MSNBC',  
  'link': 'https://www.msnbc.com/rachel-maddow-show/maddowblog/obama-slamming-trump-response-cov',  
  'stop': True},  
 {'snippet': 'Learn about the life and achievements of Barack Obama, the 44th president of the U',  
  'title': 'Barack Obama | Biography, Parents, Education, Presidency, Books ...',  
  'link': 'https://www.britannica.com/biography/Barack-Obama'},  
 {'snippet': 'He maintains a close friendship with Mr. Obama. He first weighed in on presidential',  
  'title': 'Bruce Springsteen to Appear With Harris and Obama at Atlanta and ...',  
  'link': 'https://www.nytimes.com/2024/10/22/us/politics/springsteen-harris-atlanta-obama-phil'}]
```

Building a tool

Building a tool is mostly just implementing function calls

- Retrieve something from the web api
 - Typically returns html
- Parse with beautifulsoup
- Return in natural language/list/json

Letting tools run code?

<https://github.com/OpenInterpreter/open-interpreter>



Agents



Tech Job
Researcher



Personal Profiler
for Engineers



Resume Strategist
for Engineers



Engineering
Interview Preparer

- Search the internet
- Read websites
- Read resume
- Perform RAG (Retrieval Augmented Generation) on resume

Need a crew? (crew.ai)

Source: <https://www.deeplearning.ai/short-courses/multi-ai-agent-systems-with-crewai/>

Making a crew with crew.ai

Step 1: Hire the people

Planner

Writer

Editor

```
planner = Agent(  
    role="Content Planner",  
    goal="Plan engaging and factually accurate content on {topic}",  
    backstory="You're working on planning a blog article "  
        "about the topic: {topic}."  
        "You collect information that helps the "  
        "audience learn something "  
        "and make informed decisions. "  
        "Your work is the basis for "  
        "the Content Writer to write an article on this topic.",  
    allow_delegation=False,  
    verbose=True  
)
```

```
writer = Agent(  
    role="Content Writer",  
    goal="Write insightful and factually accurate "  
        "opinion piece about the topic: {topic}",  
    backstory="You're working on a writing "  
        "a new opinion piece about the topic: {topic}. "  
        "You base your writing on the work of "  
        "the Content Planner, who provides an outline "  
        "and relevant context about the topic. "  
        "You follow the main objectives and "  
        "direction of the outline, "  
        "as provide by the Content Planner. "  
        "You also provide objective and impartial insights "  
        "and back them up with information "  
        "provide by the Content Planner. "  
        "You acknowledge in your opinion piece "  
        "when your statements are opinions "  
        "as opposed to objective statements.",  
    allow_delegation=False,  
    verbose=True  
)
```

```
editor = Agent(  
    role="Editor",  
    goal="Edit a given blog post to align with "  
        "the writing style of the organization. ",  
    backstory="You are an editor who receives a blog post "  
        "from the Content Writer. "  
        "Your goal is to review the blog post "  
        "to ensure that it follows journalistic best practices,"  
        "provides balanced viewpoints "  
        "when providing opinions or assertions, "  
        "and also avoids major controversial topics "  
        "or opinions when possible.",  
    allow_delegation=False,  
    verbose=True  
)
```

Step 2: Give them tasks

Task: Plan

```
plan = Task(
    description=(
        "1. Prioritize the latest trends, key players, "
        "and noteworthy news on {topic}.\n"
        "2. Identify the target audience, considering "
        "their interests and pain points.\n"
        "3. Develop a detailed content outline including "
        "an introduction, key points, and a call to action.\n"
        "4. Include SEO keywords and relevant data or sources."
    ),
    expected_output="A comprehensive content plan document "
        "with an outline, audience analysis, "
        "SEO keywords, and resources.",
    agent=planner,
)
```

Task: Write

```
write = Task(
    description=(
        "1. Use the content plan to craft a compelling "
        "blog post on {topic}.\n"
        "2. Incorporate SEO keywords naturally.\n"
        "3. Sections/Subtitles are properly named "
        "in an engaging manner.\n"
        "4. Ensure the post is structured with an "
        "engaging introduction, insightful body, "
        "and a summarizing conclusion.\n"
        "5. Proofread for grammatical errors and "
        "alignment with the brand's voice.\n"
    ),
    expected_output="A well-written blog post "
        "in markdown format, ready for publication, "
        "each section should have 2 or 3 paragraphs.",
    agent=writer,
)
```

Task: Edit

```
edit = Task(
    description=("Proofread the given blog post for "
        "grammatical errors and "
        "alignment with the brand's voice."),
    expected_output="A well-written blog post in markdown format, "
        "ready for publication, "
        "each section should have 2 or 3 paragraphs.",
    agent=editor
)
```

Step 3: See the magic

```
crew = Crew(  
    agents=[planner, writer, editor],  
    tasks=[plan, write, edit],  
    verbose=2  
)
```

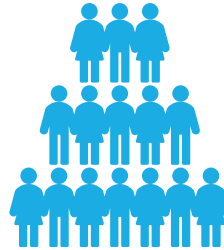
```
result = crew.kickoff(inputs={"topic": "Artificial Intelligence"})
```

Works with Ollama, Gemini, OpenAI, Mistral, HuggingFace

More features



Memory



Self-delegation

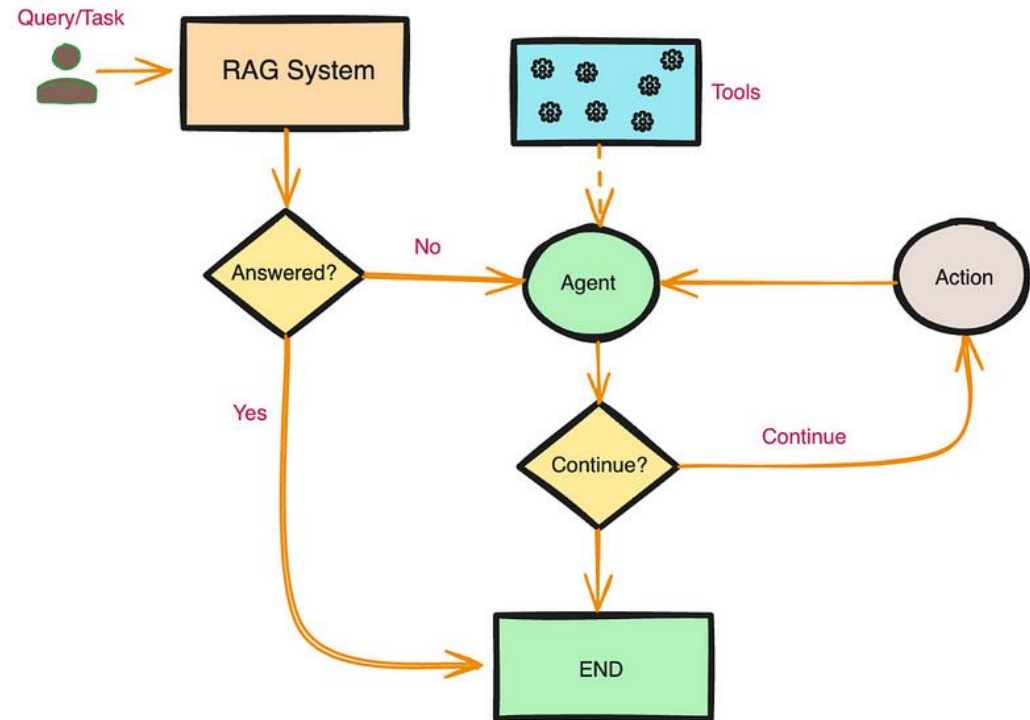
(allow agents to communicate with each other requesting help with something)



Events

Tools: <https://docs.crewai.com/concepts/tools>

Agentic RAG



Source: <https://levelup.gitconnected.com/agentic-rag-using-crewai-langchain-bf935d26bc21>

Agents that live and learn (humans?)

The Platform for Building Stateful Agents

Build agents with infinite context and human-like memory,
that can learn from data and improve with experience

Install Letta

Request Cloud Access

Prompting Review

The types of prompting



Step-back



Chain-of-
thought



Tree-of-
thought



Self-
consistency



Agentic
(ReAct)

Model Context Protocol (MCPs)

Connecting with other tools/resources

Server:

Exposes app specific data, e.g.,
Whatsapp would expose chats

Client:

AI application that uses this data,
e.g., Claude Desktop

<https://www.anthropic.com/news/model-context-protocol>

Example interaction

1. Claude Receives the User Message

```
json
{
  "type": "user_message",
  "content": "Send a message to Raj on WhatsApp: 'See you at 7 PM!'"
}
```

2. Claude Emits an MCP Tool Call

```
json
{
  "type": "tool_use",
  "tool_name": "whatsapp.send_message",
  "parameters": {
    "contact_name": "Raj",
    "message": "See you at 7 PM!"
  }
}
```

3. WhatsApp Tool Executes and Responds

The WhatsApp plugin, listening for `whatsapp.send_message`, performs the action and replies with a result:

```
json
{
  "type": "tool_result",
  "tool_name": "whatsapp.send_message",
  "status": "success",
  "data": {
    "contact_name": "Raj",
    "delivery_status": "sent"
  }
}
```

4. Claude Responds to the User

```
json
{
  "type": "model_response",
  "content": "✅ Message sent to Raj on WhatsApp: 'See you at 7 PM!'"
}
```

Integrating with FastAPI



FastAPI-MCP

A zero-configuration tool for automatically exposing FastAPI endpoints as Model Context Protocol (MCP) tools.

[pypi package](#) [0.2.0](#) [python](#) [3.10](#) | [3.11](#) | [3.12](#) [FastAPI](#) [MCP](#) [Dev](#) [CI](#) [passing](#) [codecov](#) [81%](#)

```
1 from fastapi import FastAPI
2 # (I'll use the 3K to generate
3 app = FastAPI(
4     title="Greeting API",
5     description="A simple API that returns greetings",
6     version="1.0.0"
7 )
8
9 @app.get("/")
10 async def root():
11     return {"message": "Hello, welcome to FastAPI!"}
12
```

Features

- **Direct integration** - Mount an MCP server directly to your FastAPI app
- **Zero configuration** required - just point it at your FastAPI app and it works
- **Automatic discovery** of all FastAPI endpoints and conversion to MCP tools
- **Preserving schemas** of your request models and response models
- **Preserve documentation** of all your endpoints, just as it is in Swagger
- **Flexible deployment** - Mount your MCP server to the same app, or deploy separately

Deployment

Types of servers

LAMP – Most commonly used

Linux Apache MySQL PHP/Perl/Python

W - Windows

M - Mac

X – Cross platform (PHP, Perl)

XAMPP

vs

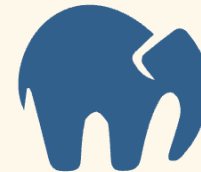
WAMP

vs

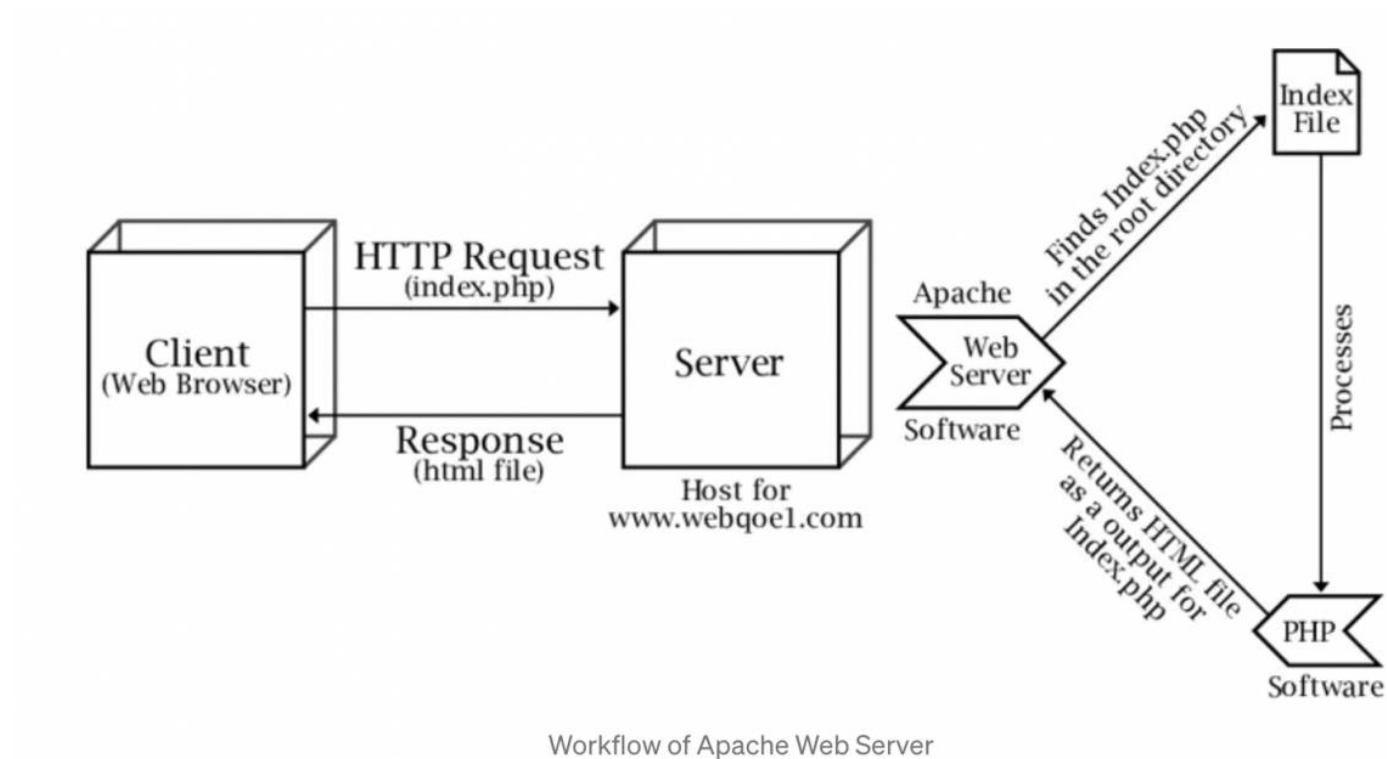
MAMP

vs

LAMP



What does Apache do?



Types of websites

Static – HTML, JS, CSS → Github.io

Backend APIs – FastAPI, Flask → Railway or Render

Full Stack apps - React/Next + Express → Railway/Render

- Requires two servers for next + express but can be combined for react + express

Environments requirements

Next/react js -> node

- When installing on a server, requires npm install

Fastapi/flask -> python

- When installing on a server, requires pip install -r requirements.txt

Therefore, suffer from cold starts (5 – 20s) on serverless deployments

Or require paid plans to keep the instance on

But what about my DB?

SQLite -> Works only on persistent disk, doesn't work on serverless containers since they only offer ephemeral disks

You can use sqlite if you choose to deploy on aws ec2 instances

To deploy,

- use a managed DB, typically postgres using supabase or railway postgres

What about files?

Typically stored on aws s3 buckets due to cheaper and scalable object storage

URLs are stored in database

Other options: MinIO, Wasabi (much cheaper storage, also available for self hosting)

Securing your website

Certificates

- HTTPS and SSL certificates to keep communication with website encrypted

CORS

- Only allow API requests from sources that you recognize (unless the APIs are meant for public use)

Rate limits

- Restrict logins through captcha
- Restrict API call frequency per user/IP

```
from fastapi import Depends
from slowapi.decorators import limit

@app.get("/items")
@limit("5/minute")
async def read_items():
    return {"message": "This endpoint is rate limited to 5 requests per minute"}

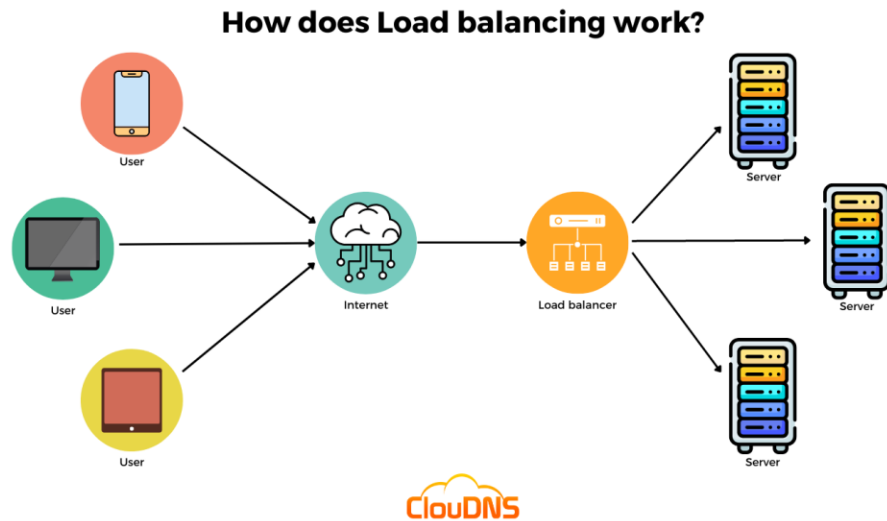
@app.post("/submit")
@limit("2/minute")
async def submit_item():
    return {"message": "This endpoint is rate limited to 2 requests per minute"}
```

Example of limiting rates using fastapi's slowapi

Scaling your website

Load balancing

Caching (Distributed caching)



Use Case	What is Stored (5 Words)	Common System / Library	Expected Benefit
Web App Data Caching	Frequently accessed app-level data	Redis, Memcached, Spring Cache	Faster page loads, less DB load
API Rate Limiting	Request count per user/IP	Redis + Lua scripts	Prevents abuse, ensures fair access
Session Management	Temporary user session state	Redis, Express-session, Django-Redis	Enables stateless servers, fast access
Job Queue / Task Queue	Pending background processing tasks	Redis (with Celery, Bull)	Async processing, load off main app
Pub/Sub Messaging	Real-time messages in channels	Redis Pub/Sub, Socket.io	Low-latency, real-time notifications/messages
ML Feature Caching	Precomputed feature vectors or scores	Redis, Feast (feature store)	Reduces recomputation, speeds up inference
Database Query Caching	Results of expensive DB queries	Redis, Memcached, Hibernate Cache	Reduced latency, less DB pressure
Content Delivery / CDN	Static files like images/CSS/JS	Cloudflare, Akamai, Varnish	Global fast access to assets
Distributed Locking	Lock tokens for shared resources	Redis + RedLock algorithm	Prevents race conditions, ensures consistency
User Profile Caching	Cached serialized user profile	Redis, Flask-Caching	Quicker page loads, lower DB reads

Source: <https://www.cloudns.net/blog/load-balancing/>

Hooks and Sockets

Real time communication

- Set up a socket

Event based communication

- Set up a hook

HTMX

Library that allows you to access modern browser features (like animations, editing, etc.) directly using html instead of using js

See examples here:

<https://htmx.org/examples/>

To understand htmx, first let's take a look at an anchor tag:

```
<a href="/blog">Blog</a>
```

This anchor tag tells a browser:

“When a user clicks on this link, issue an HTTP GET request to ‘/blog’ and load the response content into the browser window”.

With that in mind, consider the following bit of HTML:

```
<button hx-post="/clicked"
        hx-trigger="click"
        hx-target="#parent-div"
        hx-swap="outerHTML">
  Click Me!
</button>
```

This tells htmx:

“When a user clicks on this button, issue an HTTP POST request to ‘/clicked’ and use the content from the response to replace the element with the id `parent-div` in the DOM”

htmx extends and generalizes the core idea of HTML as a hypertext, opening up many more possibilities directly within the language:

- Now any element, not just anchors and forms, can issue an HTTP request
- Now any event, not just clicks or form submissions, can trigger requests
- Now any [HTTP verb](#), not just `GET` and `POST`, can be used
- Now any element, not just the entire window, can be the target for update by the request

Unofficial course feedback:
WebDevAI

