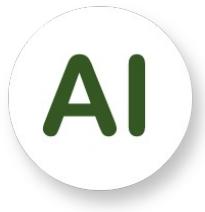


AI VIETNAM
All-in-One Course
(TA Session)

Gradio for Machine Learning Model Demos

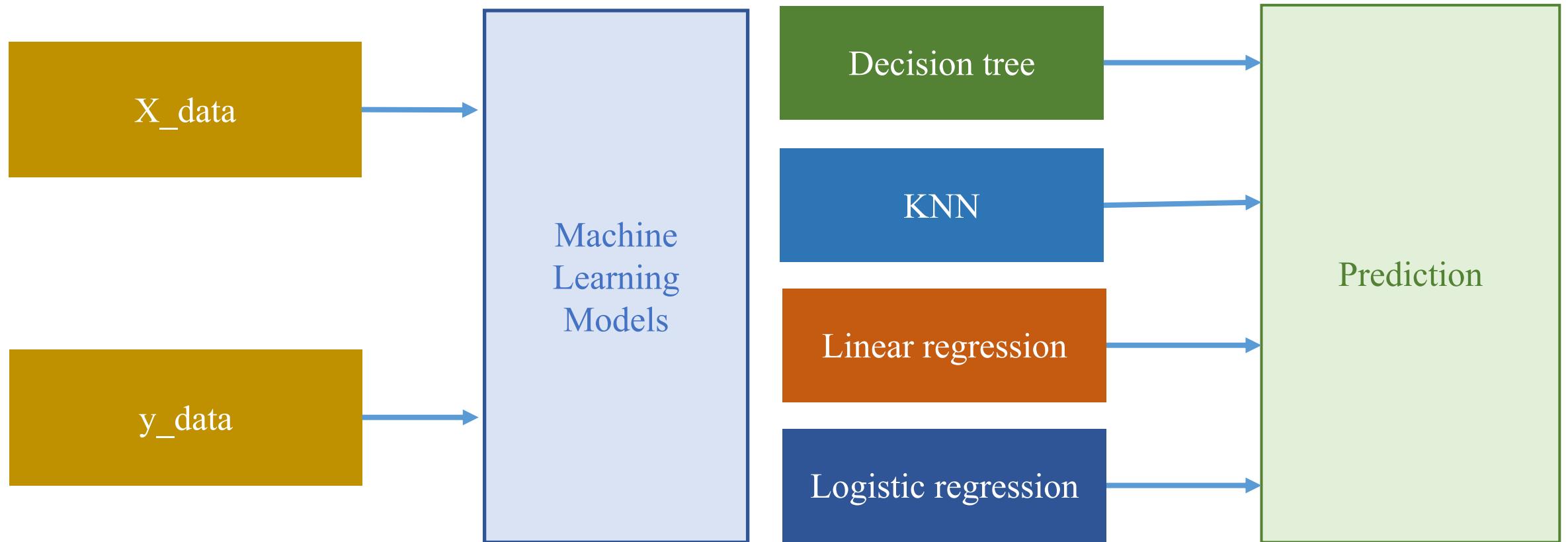


AI VIET NAM
[@aivietnam.edu.vn](http://aivietnam.edu.vn)

Nguyen-Thuan Duong – TA

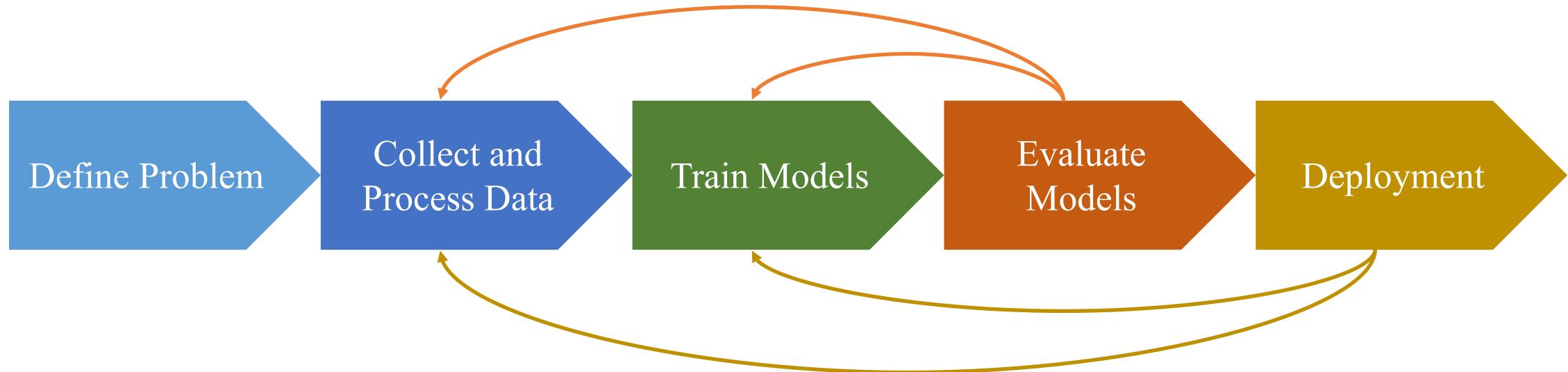
Machine Learning

❖ Machine Learning Models



Machine Learning

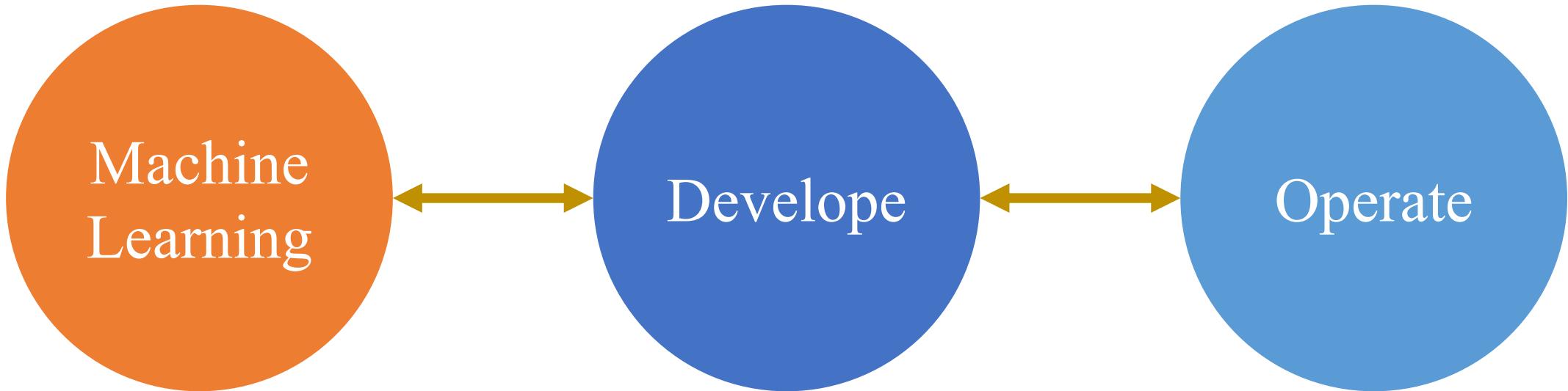
❖ Machine Learning Life Cycle



Machine Learning

❖ Machine Learning Operation (MLOps)

- MLOps is a set of tools and best practices for bringing ML into production.



Outline

- Introduction
- Web Application
- Gradio
- Hands-on
- Conclusion
- Question

Introduction

Introduction

❖ Getting Started

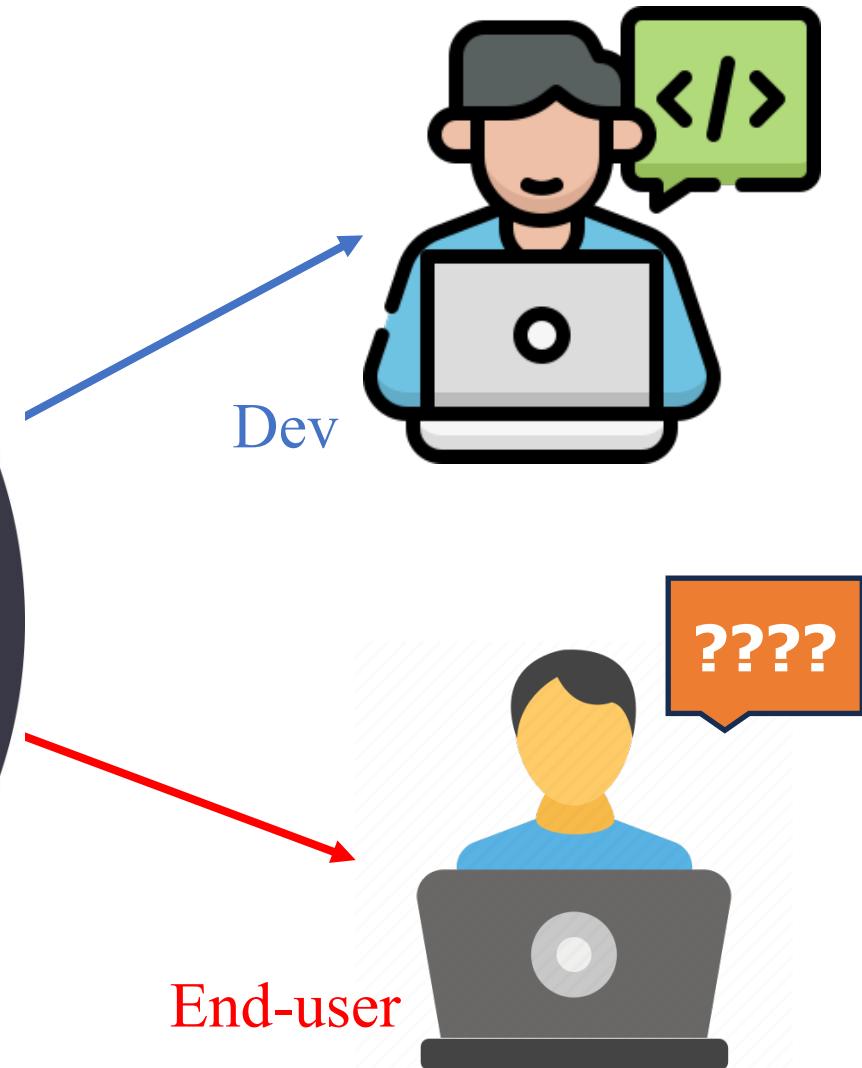
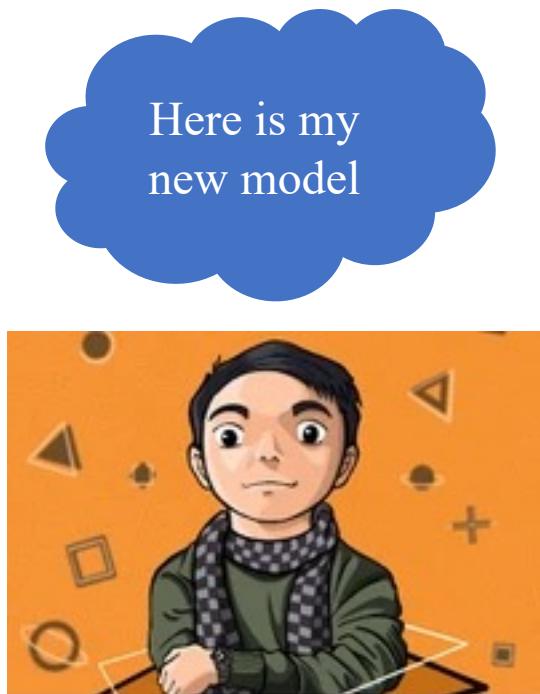
```
● ● ●  
1 from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC  
2 import soundfile as sf  
3 import torch  
4  
5 processor = Wav2Vec2Processor.from_pretrained("nguyenvulebinh/wav2vec2-base-vietnamese-250h")  
6 model = Wav2Vec2ForCTC.from_pretrained("nguyenvulebinh/wav2vec2-base-vietnamese-250h")  
7  
8 # define function to read in sound file  
9 def map_to_array(batch):  
10     speech, _ = sf.read(batch["file"])  
11     batch["speech"] = speech  
12     return batch  
13  
14 # load dummy dataset and read soundfiles  
15 ds = map_to_array({  
16     "file": 'audio-test/t1_0001-00010.wav'  
17 })  
18  
19 input_values = processor(ds["speech"], return_tensors="pt", padding="longest").input_values  
20 logits = model(input_values).logits  
21  
22 # take argmax and decode  
23 predicted_ids = torch.argmax(logits, dim=-1)  
24 transcription = processor.batch_decode(predicted_ids)
```

Here is my
new model



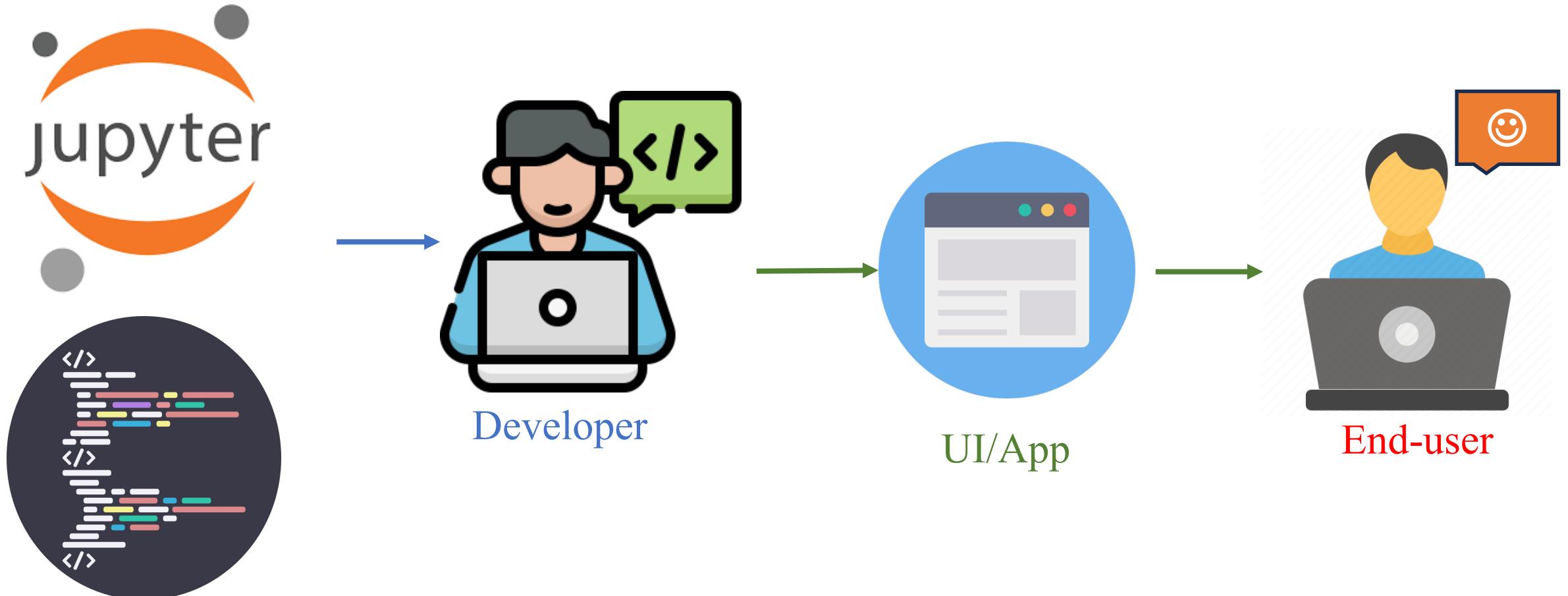
Introduction

❖ Developer vs End-user



Introduction

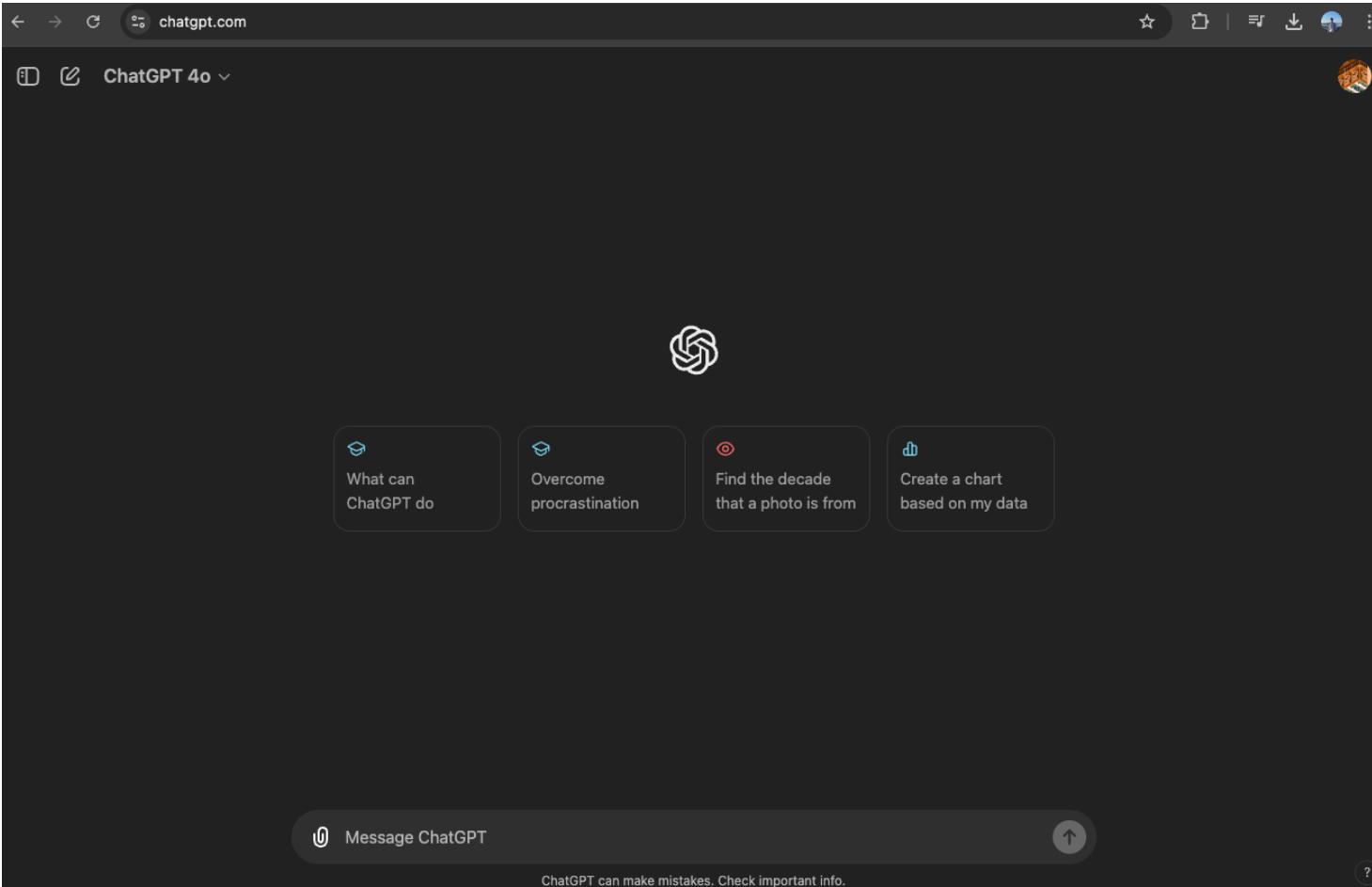
❖ Development



Web application

Web application

❖ ChatGPT web application



Introduction

❖ OCR demo

EasyOCR

Gradio demo for EasyOCR. EasyOCR demo supports 80+ languages. To use it, simply upload your image and choose a language from the dropdown menu, or click one of the examples to load them. Read more at the links below.

Input



language

abq	ady	af	ang	ar	as	ava	az	be
bg	bh	bho	bn	bs	ch_sim	ch_tra	che	
cs	cy	da	dar	de	<input checked="" type="checkbox"/> en	es	et	fa
fr	ga	gom	hi	hr	hu	id	inh	is
it	ja	kbd	kn	ko	ku	la	lbe	lez
lt	lv	mah	mai	mi	mn	mr	ms	mt
ne	new	nl	no	oc	pi	pl	pt	ro
ru	rs_cyrillic	rs_latin		sck	sk	sl	sq	sv
sw	ta	tab	te	th	tjk	tl	tr	ug
uk	ur	uz	<input checked="" type="checkbox"/> vi					

Output



166.5s

Output 2

TEXT	CONFIDENCE
[H	0.3347064751478329
79D	0.24794097406733084
06:31879	0.181738747536989
THAO ĐIÉN	0.5234315327990876
HIỆP BÌNH	0.6633234946059156
CHÁNH	0.5597377871990743
TP_HCM SÊ THI TUYỂN QUỐC TẾ	0.5408944320326364
TƯỞNG QUY HOẠCH	0.9091566919586396
giây	0.9989028573036194
BẢN ĐẢO THANH ĐA	0.9408069748721736
TP HCM: Chủ dô	0.7445287340962483

Introduction

❖ Web application



Introduction

❖ HTML

```
● ● ●  
1  <!DOCTYPE html>  
2  <html lang="en">  
3  <head>  
4      <meta charset="UTF-8">  
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">  
6      <title>Styled Text Input Form</title>  
7  </head>  
8  <body>  
9  
10     <div class="container">  
11         <h1>Styled Form</h1>  
12         <form action="#" method="post">  
13             <label for="input1">Input 1:</label>  
14             <input type="text" id="input1" name="input1" required>  
15             <label for="input2">Input 2:</label>  
16             <input type="text" id="input2" name="input2" required>  
17             <button type="submit">Submit</button>  
18         </form>  
19     </div>  
20  
21 </body>  
22 </html>  
23
```

Styled Form

Input 1: Input 2:

Introduction

❖ HTML + CSS

```
● ● ●  
1  <!DOCTYPE html>  
2  <html lang="en">  
3  <head>  
4      <meta charset="UTF-8">  
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">  
6      <title>Styled Text Input Form</title>  
7      <link rel="stylesheet" href="index.css">  
8  </head>  
9  <body>  
10  
11     <div class="container">  
12         <h1>Styled Form</h1>  
13         <form action="#" method="post">  
14             <label for="input1">Input 1:</label>  
15             <input type="text" id="input1" name="input1" required>  
16             <label for="input2">Input 2:</label>  
17             <input type="text" id="input2" name="input2" required>  
18             <button type="submit">Submit</button>  
19         </form>  
20     </div>  
21  
22 </body>  
23 </html>  
24
```

Introduction

❖ HTML + CSS



```
1 body {  
2     font-family: Arial, sans-serif;  
3     background-color: #f4f4f9;  
4     margin: 0;  
5     padding: 0;  
6     display: flex;  
7     justify-content: center;  
8     align-items: center;  
9     height: 100vh;  
10 }  
11  
12 h1 {  
13     text-align: center;  
14     color: #333;  
15 }
```



```
1 form {  
2     background-color: #fff;  
3     padding: 20px;  
4     border-radius: 8px;  
5     box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
6     width: 100%;  
7     max-width: 400px;  
8 }  
9  
10 label {  
11     font-size: 16px;  
12     color: #555;  
13 }  
14  
15 input[type="text"] {  
16     width: 100%;  
17     padding: 10px;  
18     margin: 8px 0;  
19     border: 1px solid #ccc;  
20     border-radius: 4px;  
21     box-sizing: border-box;  
22 }
```



```
1 button {  
2     background-color: #28a745;  
3     color: white;  
4     padding: 10px 15px;  
5     border: none;  
6     border-radius: 4px;  
7     cursor: pointer;  
8     width: 100%;  
9     font-size: 16px;  
10 }  
11  
12 button:hover {  
13     background-color: #218838;  
14 }  
15  
16 .container {  
17     width: 100%;  
18     max-width: 400px;  
19     margin: 0 auto;  
20 }
```

Introduction

❖ HTML + CSS

Styled Form

Input 1:

Input 2:

Submit

Introduction

❖ HTML + CSS + JS

```
1 <body>
2   <div class="container">
3     <h1>Styled Form</h1>
4     <form id="concatForm" action="javascript:void(0);">
5       <label for="input1">First name:</label>
6       <input type="text" id="input1" name="input1" required>
7       <label for="input2">Last name:</label>
8       <input type="text" id="input2" name="input2" required>
9       <button type="submit">Submit</button>
10    </form>
11    <p id="result"></p>
12  </div>
13  <script src="index.js"></script>
14 </body>
```

```
1 document.getElementById('concatForm').addEventListener('submit', function(event) {
2   event.preventDefault();
3
4   const input1 = document.getElementById('input1').value;
5   const input2 = document.getElementById('input2').value;
6
7   const result = input1 + " " + input2;
8
9   document.getElementById('result').textContent = "Your name is: " + result;
10 });
11
```

Introduction

- ❖ HTML + CSS + JS

Styled Form

First name:

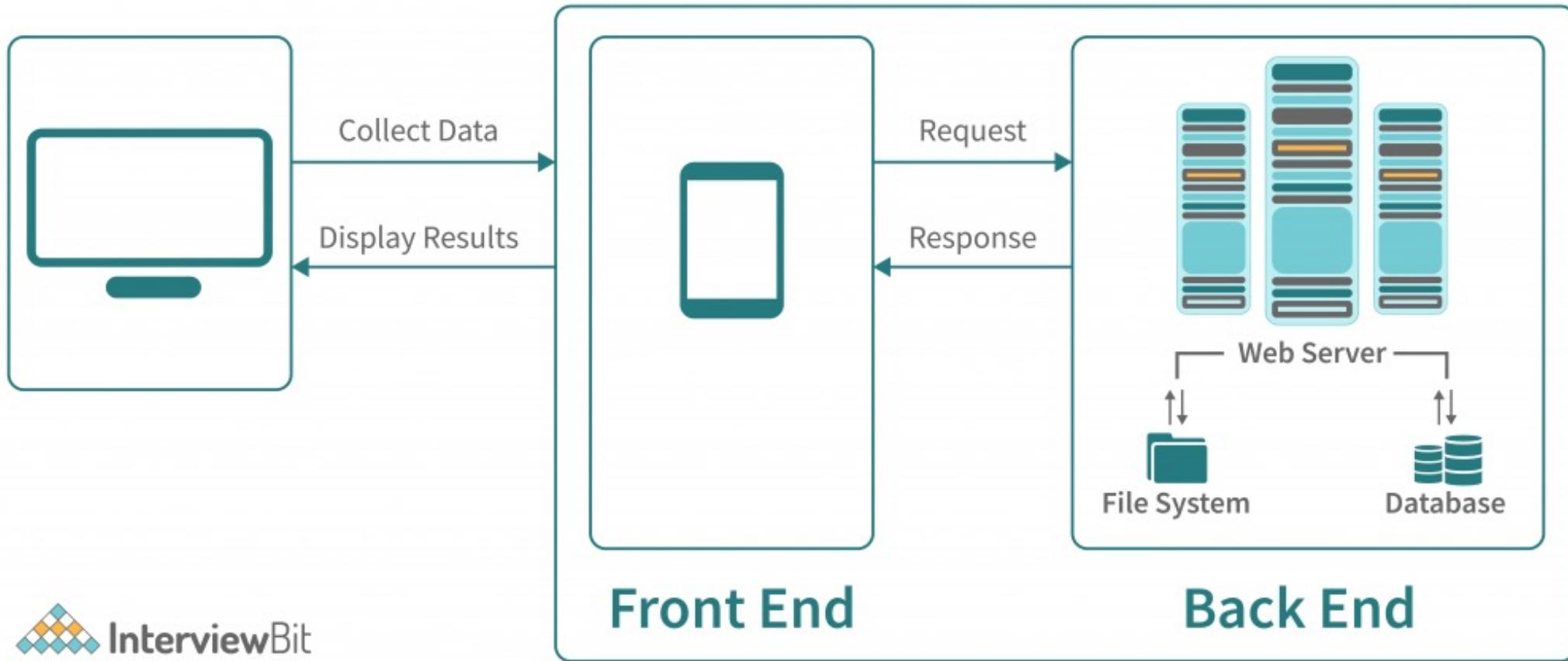
Last name:

Submit

Your name is: Duong Thuan

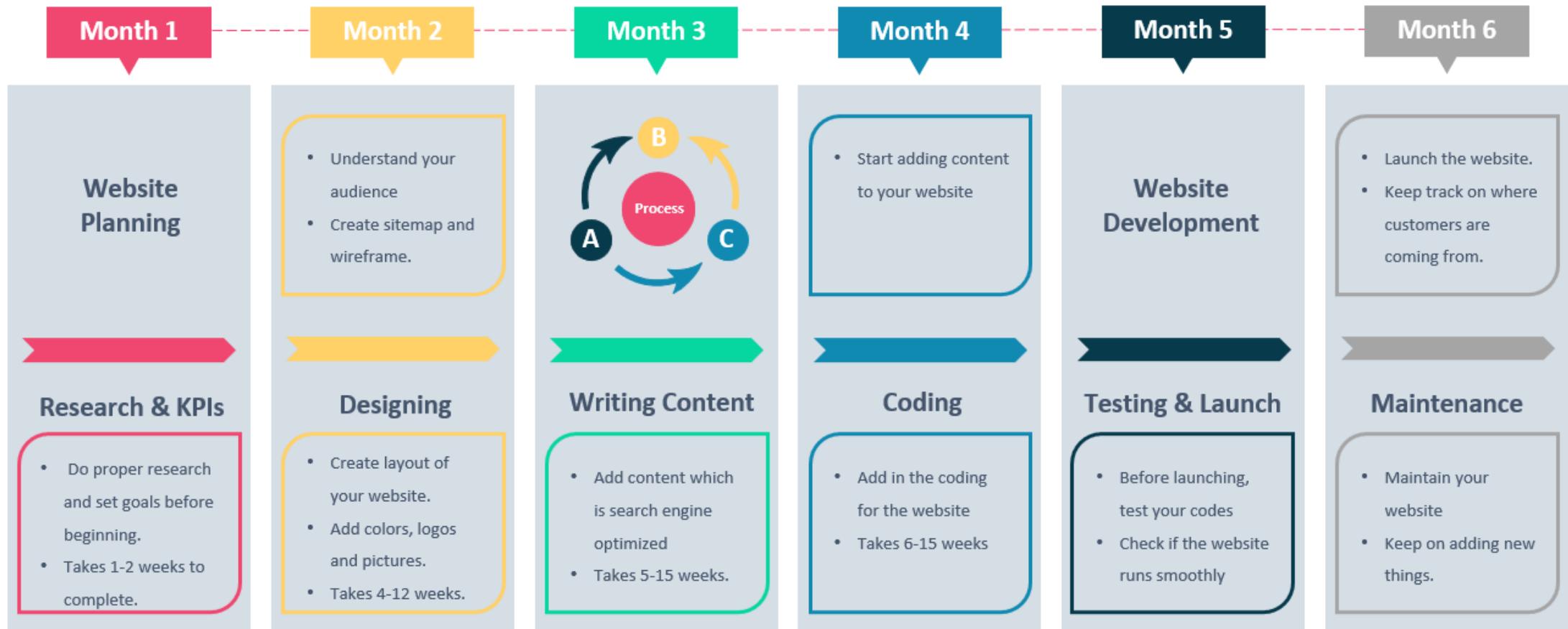
Introduction

❖ Web Development



Introduction

❖ Web Development



Introduction

- ❖ No HTML - No CSS - No JavaScript

Streamlit



gradio

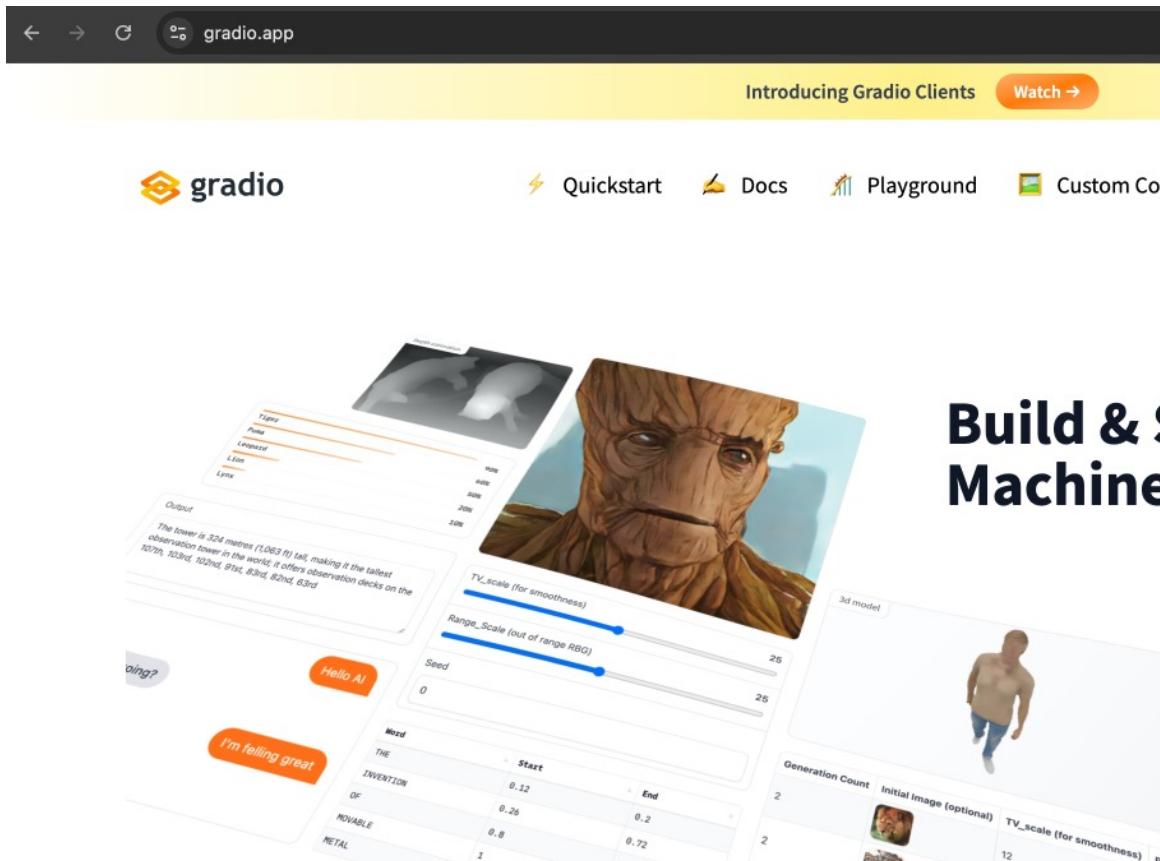
Chainlit



Introduction

Introduction

❖ Getting Started



The screenshot shows the Gradio app homepage with a yellow header bar containing the URL "gradio.app", a "Watch →" button, and a search bar. Below the header, there's a navigation bar with links for "Quickstart", "Docs", "Playground", "Custom Components", and "Community". A large central image displays three different AI-generated outputs: a 3D model of a person walking, a painting of Groot from Guardians of the Galaxy, and a text-based interface for generating images of the Eiffel Tower.

Build & Share Delightful Machine Learning Apps

Gradio is the fastest way to demo your machine learning model with a friendly web interface so that anyone can use it, anywhere!

[Get Started](#) Star 32485

Introduction

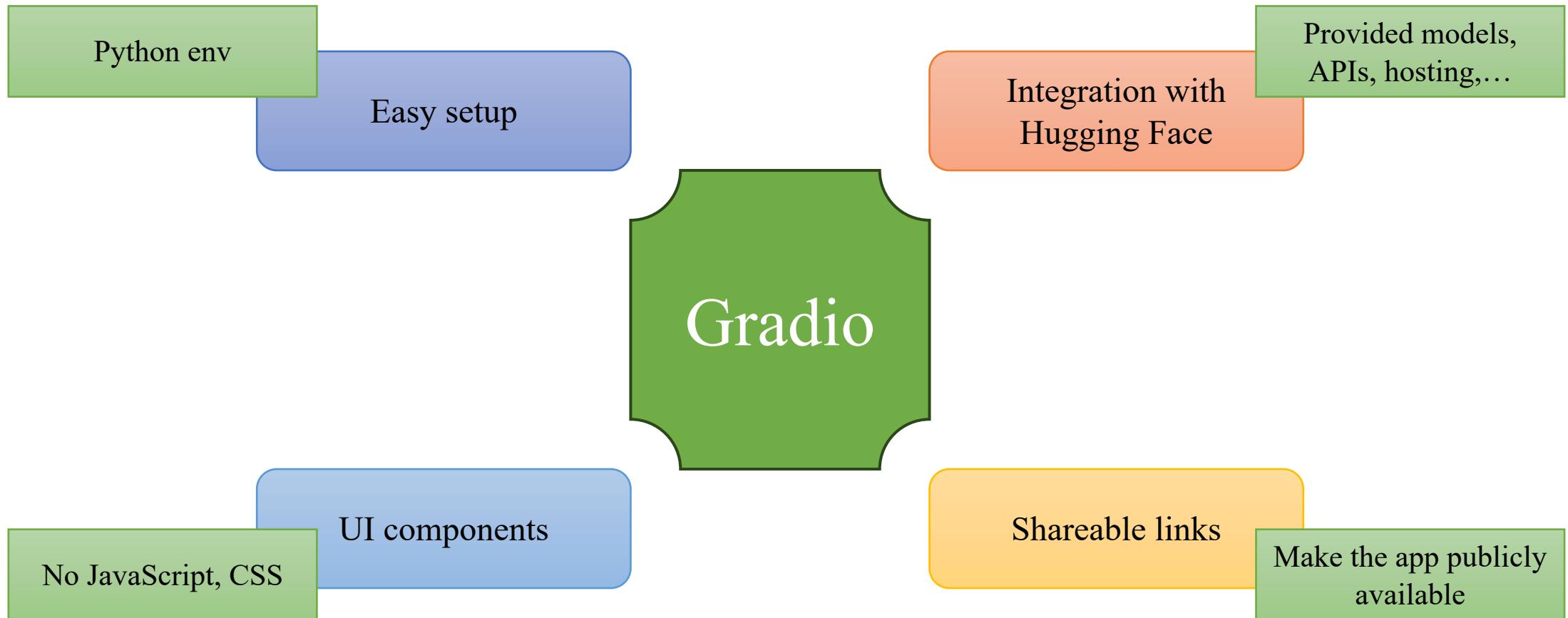
❖ Gradio Overview

```
Users > abidlabs > Desktop > app.py
1   from model import generate_image
2
3
4
```

Lightweight, open-source UI library for ML models

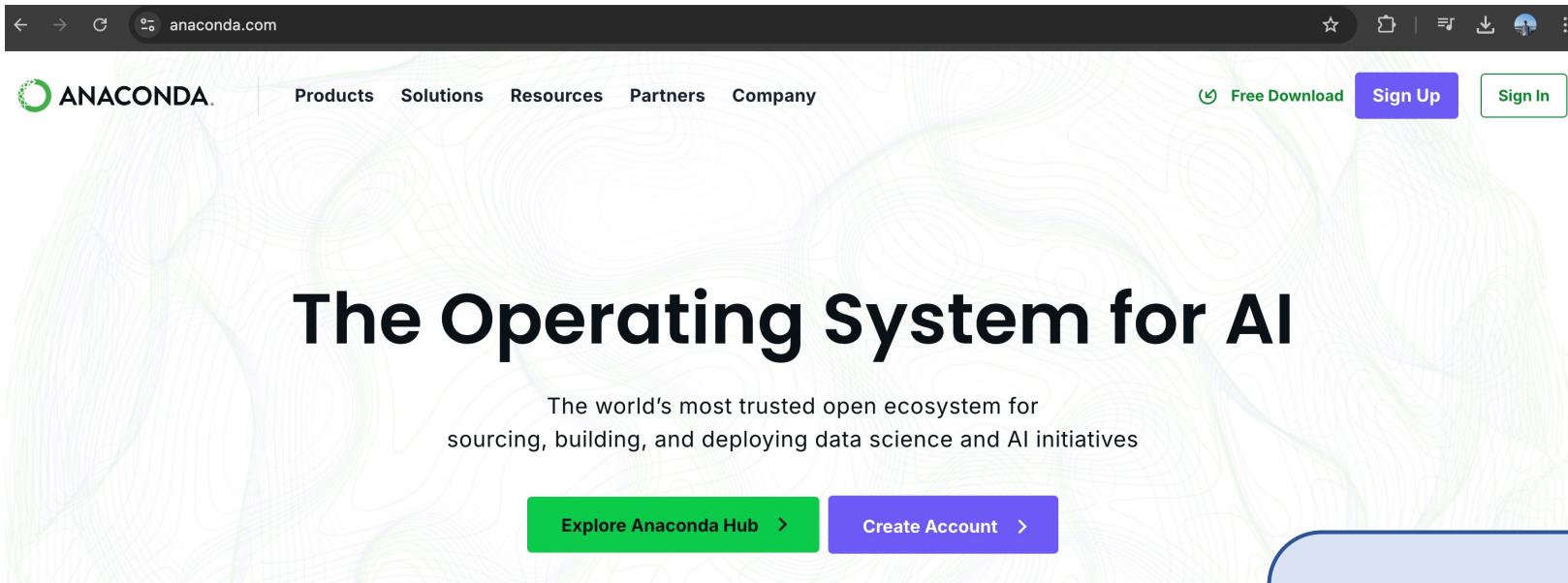
Introduction

❖ Key Features



Introduction

❖ Installation



```
>> conda create -n gradio_env python=3.13.5
>> conda activate gradio _env
>> pip install --upgrade gradio
```

Introduction

❖ Building Your First Demo

The image shows a development environment with two main components: a code editor and a web browser.

Code Editor (Left):

```
1 import gradio as gr
2
3 def greet(name):
4     return "Hello, " + name
5
6 demo = gr.Interface(
7     fn=greet,
8     inputs=["text"],
9     outputs=["text"],
10)
11
12 demo.launch()
```

Terminal (Top Right):

PROBLEMS OUTPUT AZURE DEBUG CONSOLE TERMINAL PORTS

- (gradio_env) thuanduong@MacNaN Code % python 1.first_demo.py
 - * Running on local URL: http://127.0.0.1:7860
 - * To create a public link, set `share=True` in `launch()`.

Browser (Bottom Right):

127.0.0.1:7860

The browser displays a Gradio interface with a single input field labeled "name" containing "Thuan" and a single output field labeled "output" containing "Hello, Thuan". Below the input field is a "Clear" button, and below the output field is a "Flag" button. An orange "Submit" button is positioned between the two fields.

Introduction

❖ Input/Output of ML application

Input

Text

Image

Speech

Video

Output

Text

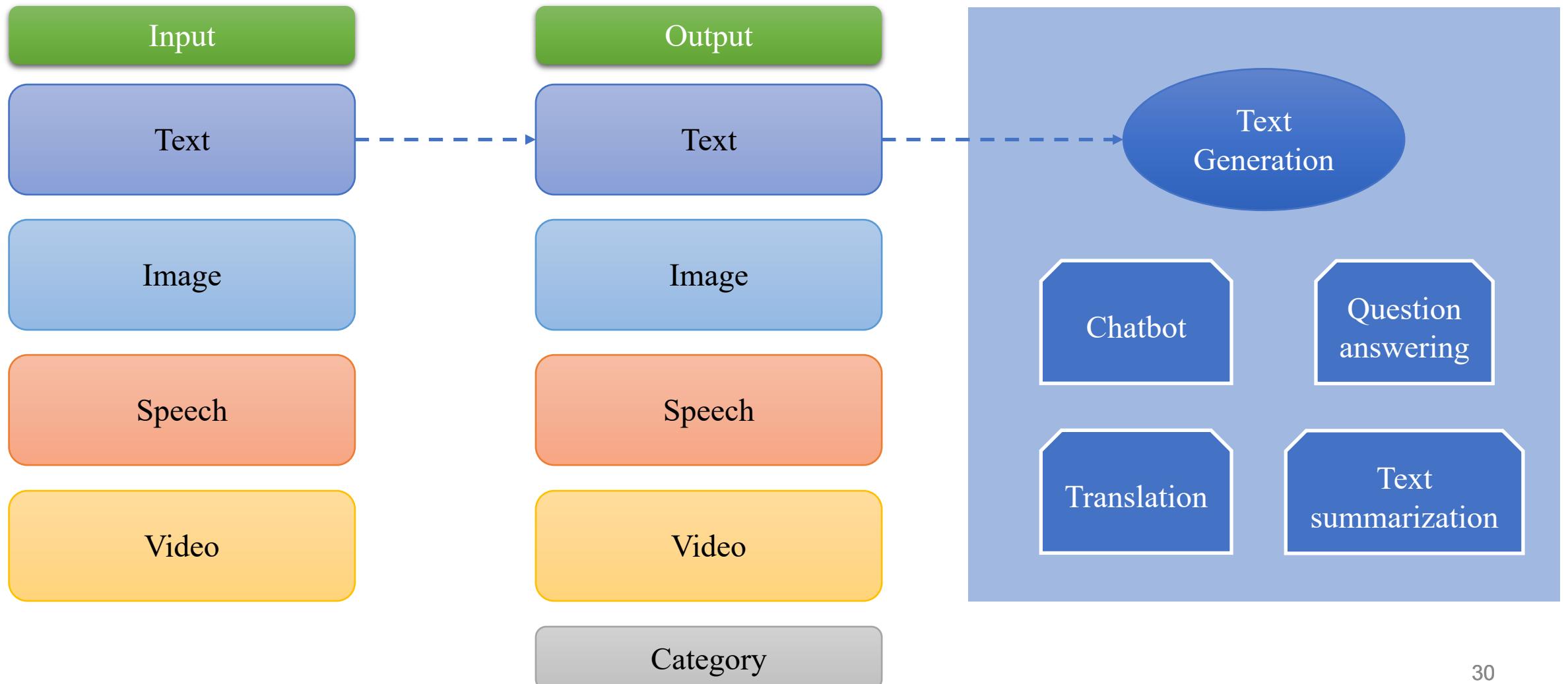
Image

Speech

Video

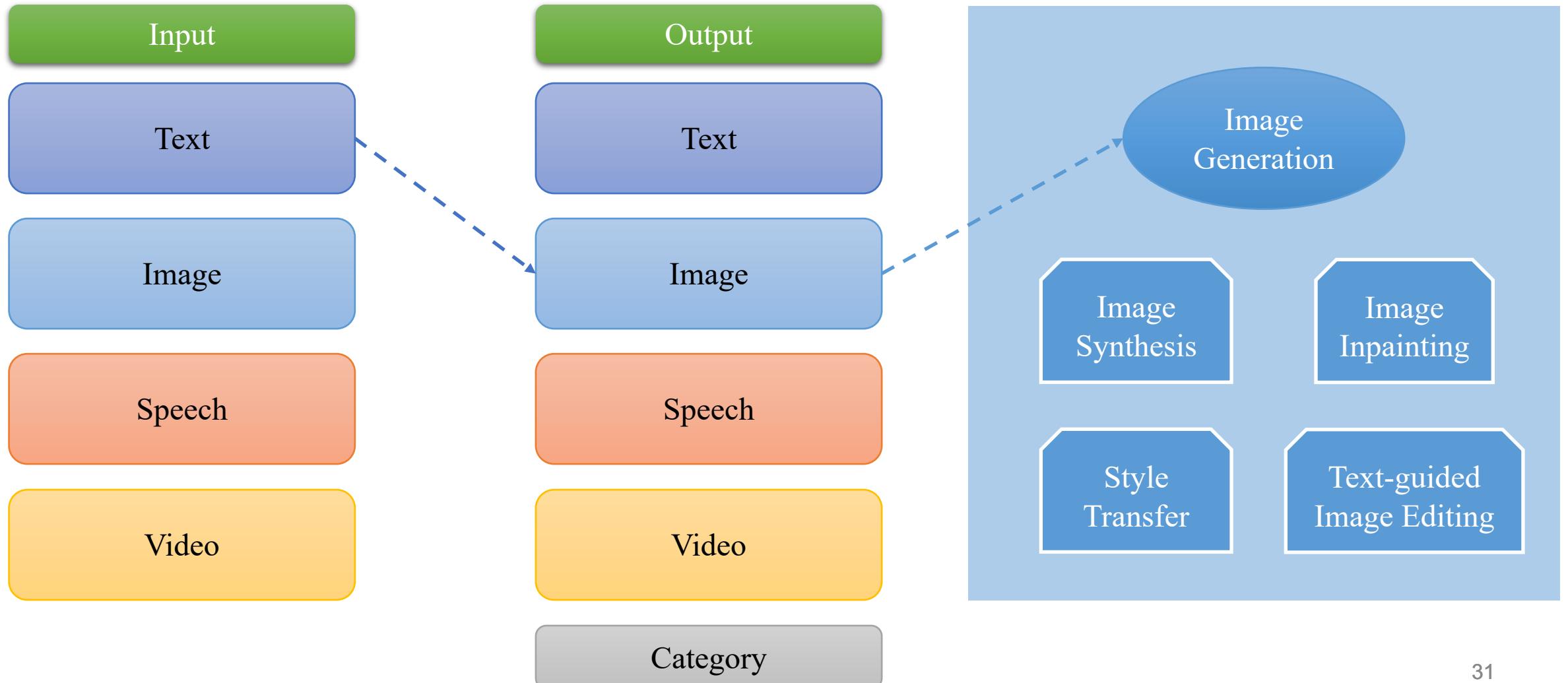
Introduction

❖ Input/Output of ML application



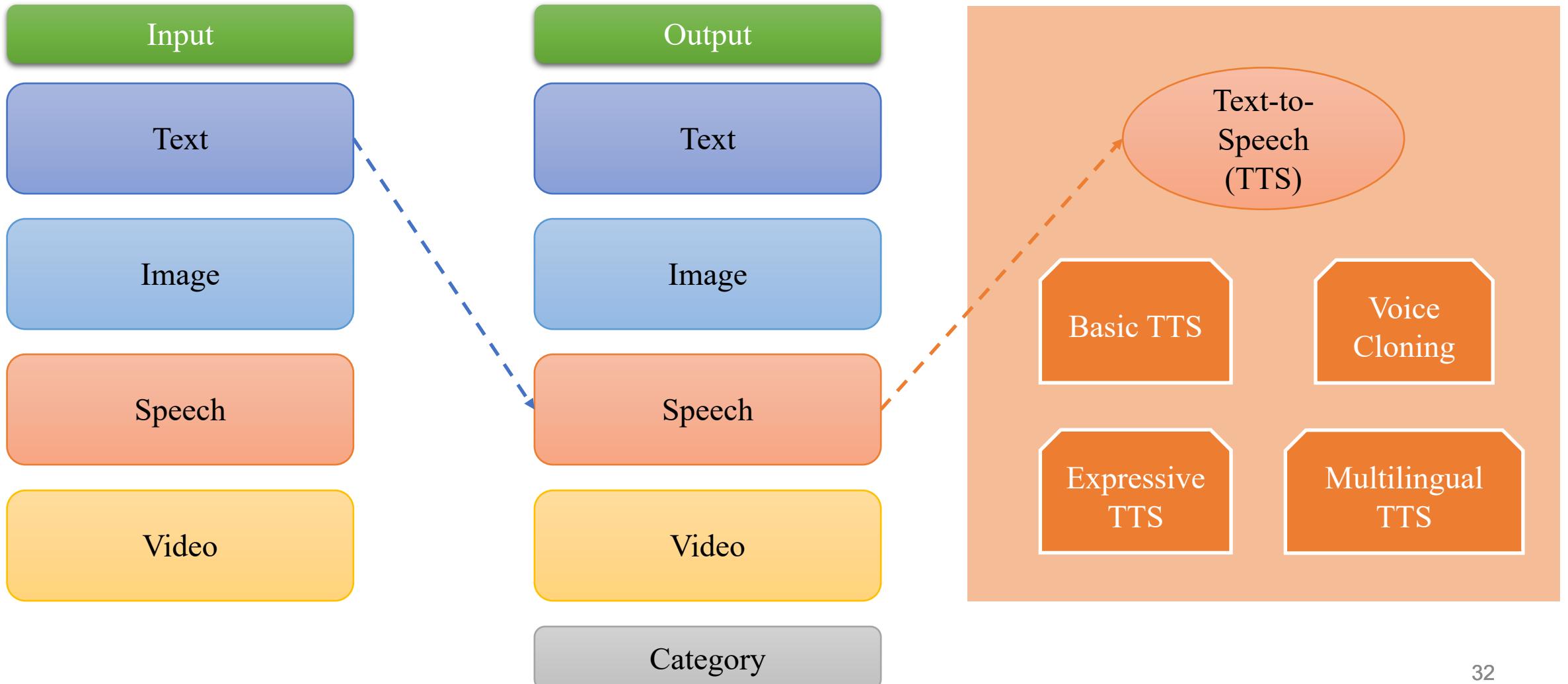
Introduction

❖ Input/Output of ML application



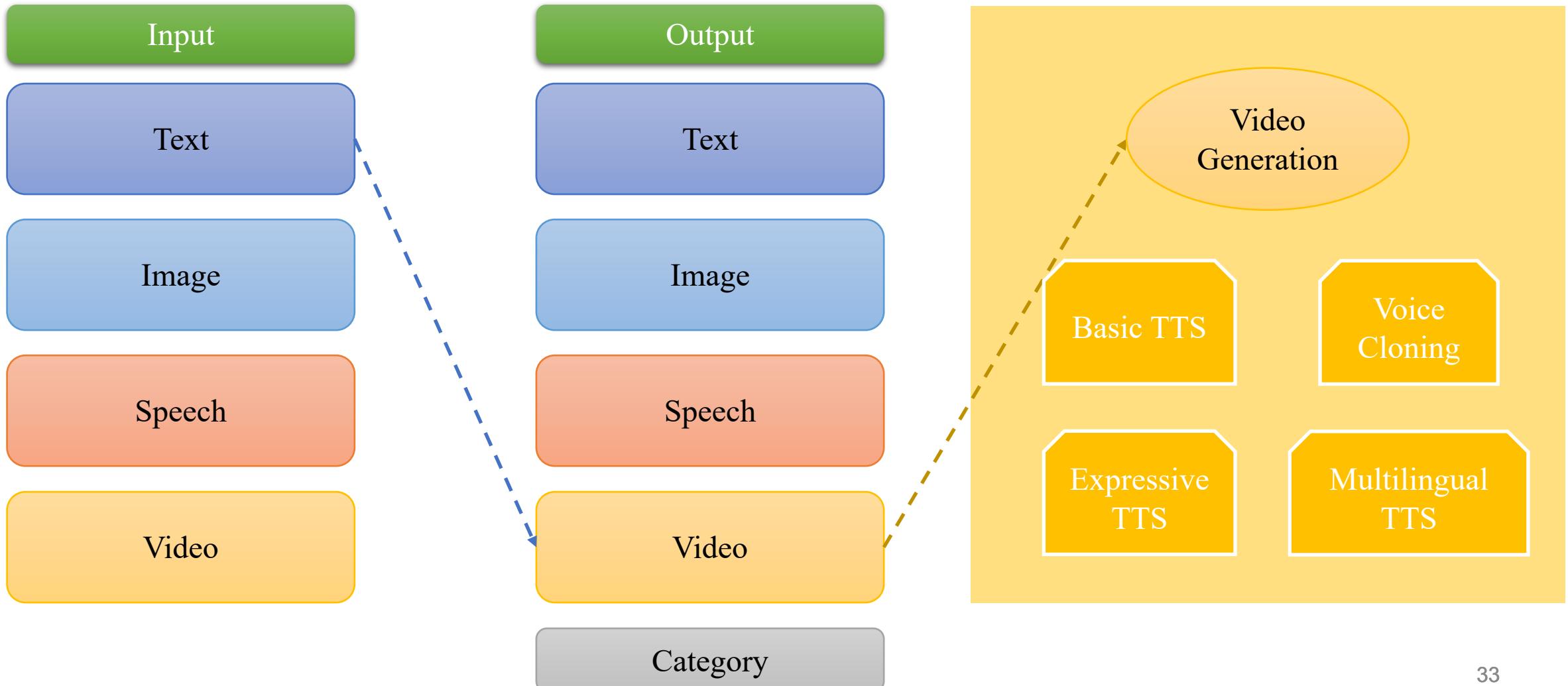
Introduction

❖ Input/Output of ML application



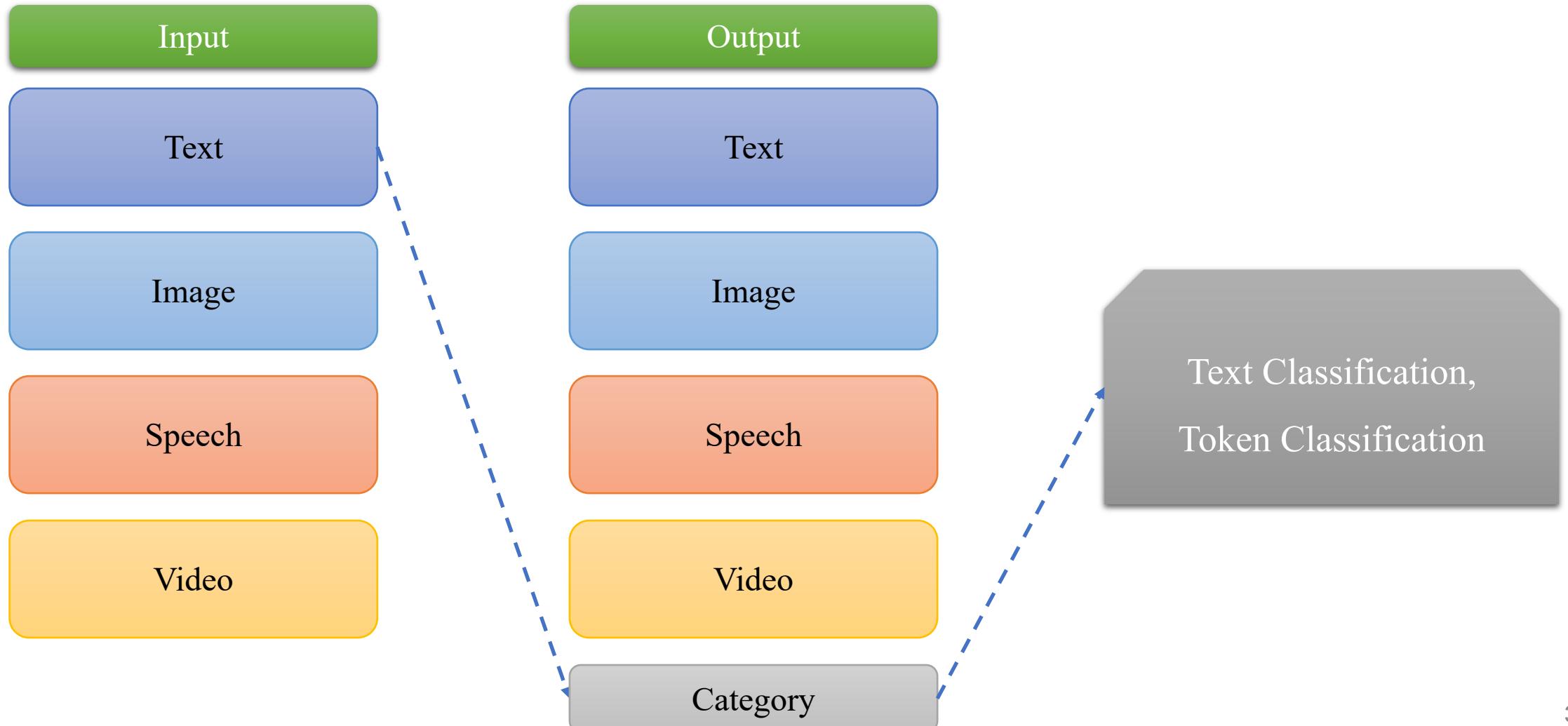
Introduction

❖ Input/Output of ML application



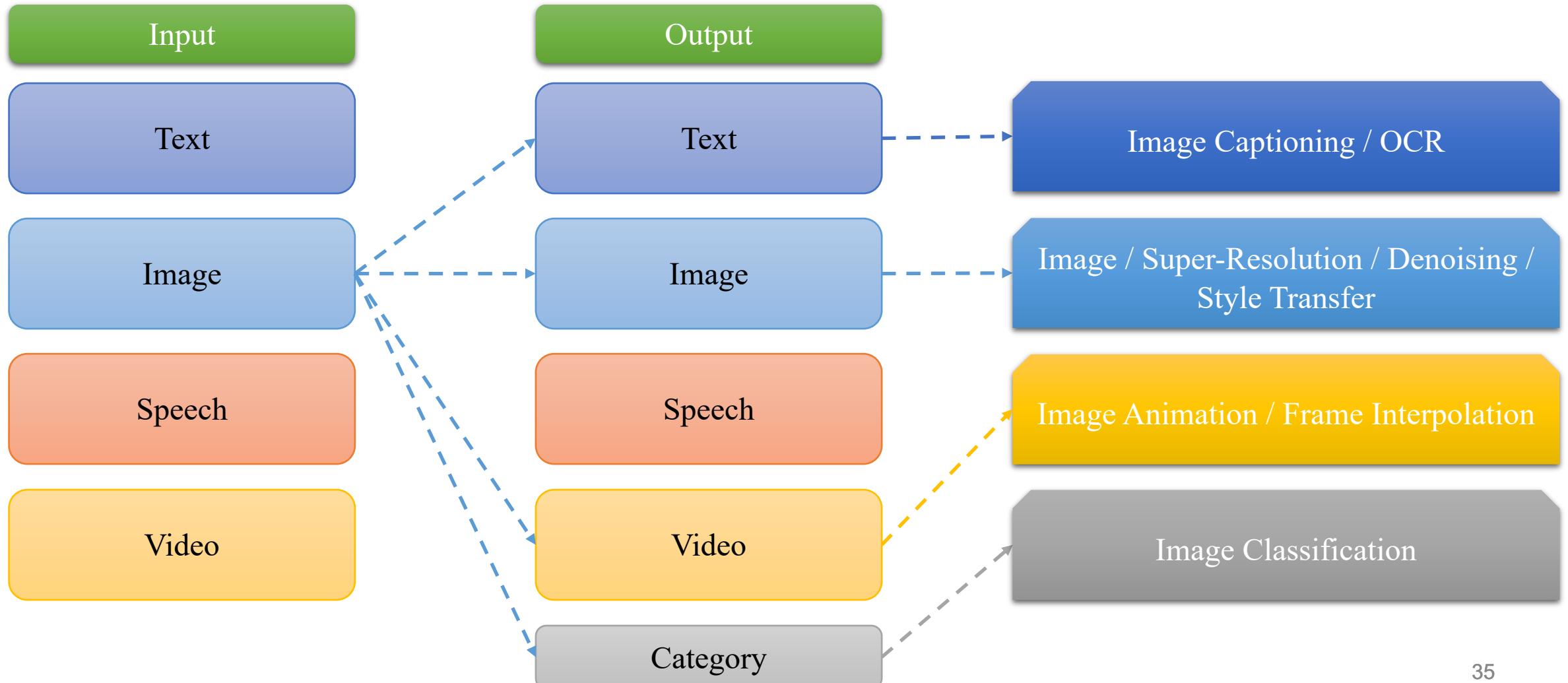
Introduction

❖ Input/Output of ML application



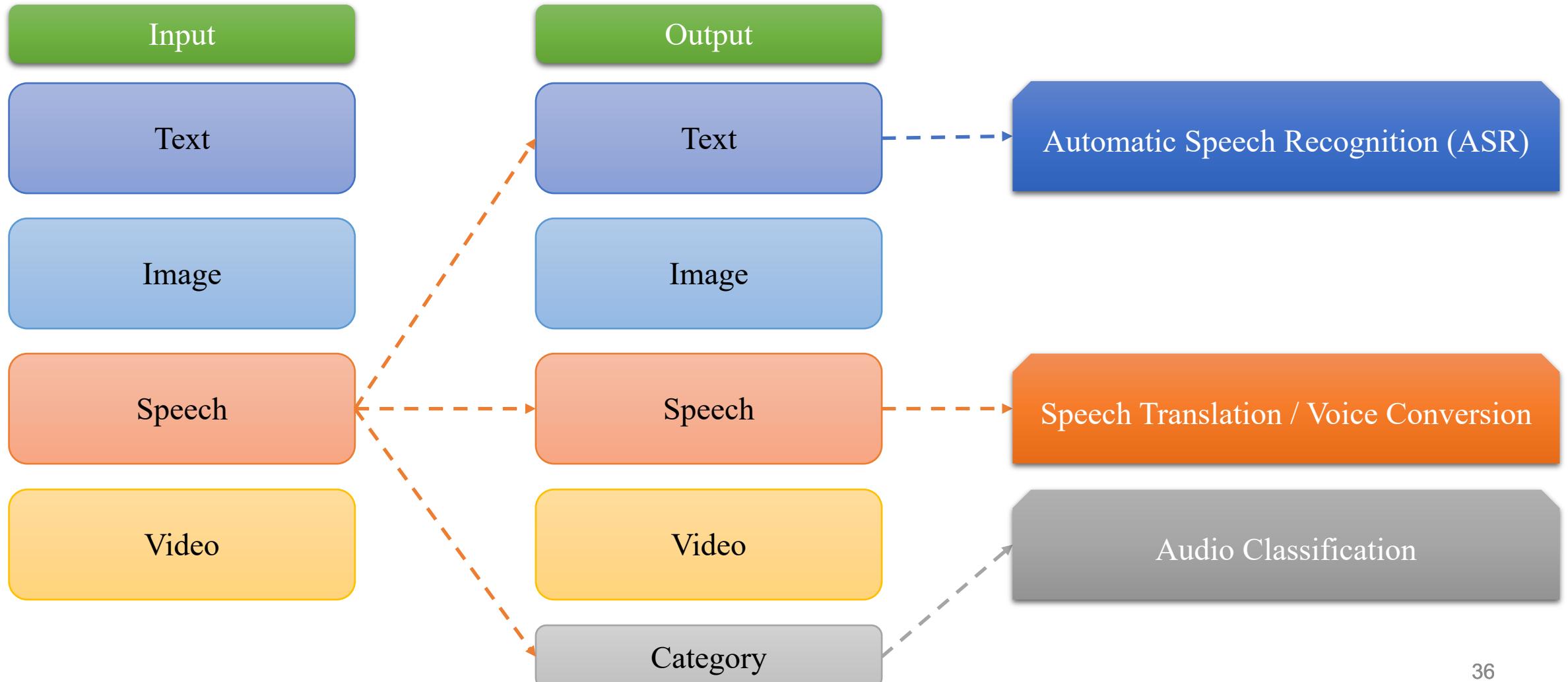
Introduction

❖ Input/Output of ML application



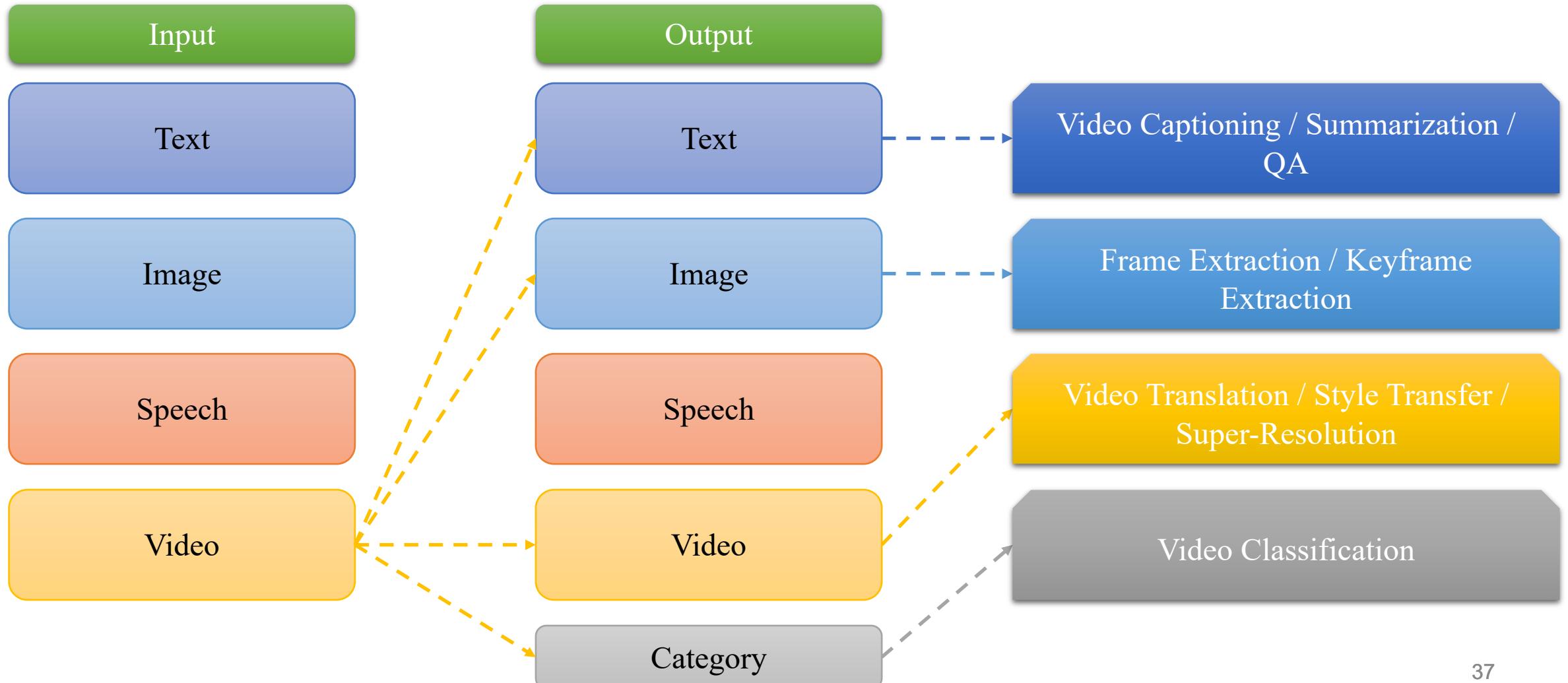
Introduction

❖ Input/Output of ML application



Introduction

❖ Input/Output of ML application

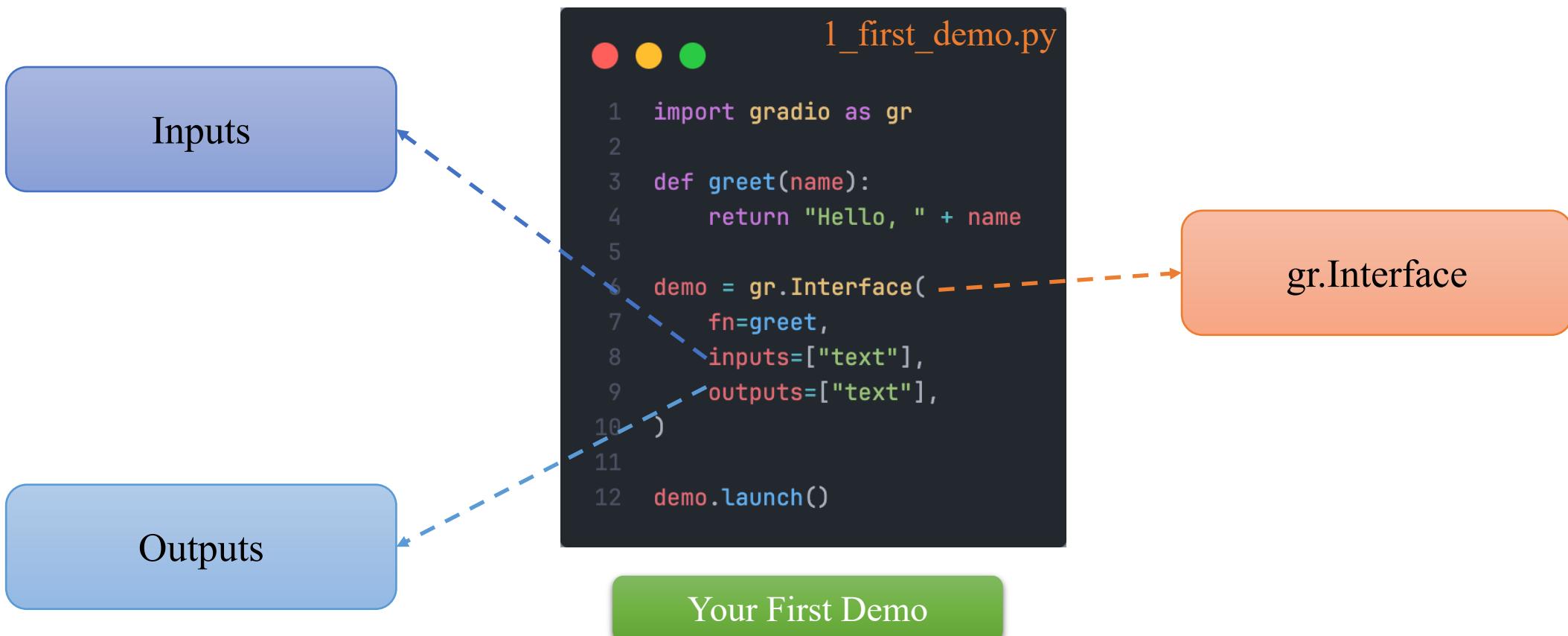


Gradio Basics

Gradio Basics

❖ Gradio Interface

Interface is Gradio's primary high-level class; it lets you build a web-based GUI or demo for a machine-learning model (or any Python function) with just a few lines of code.



Gradio Basics

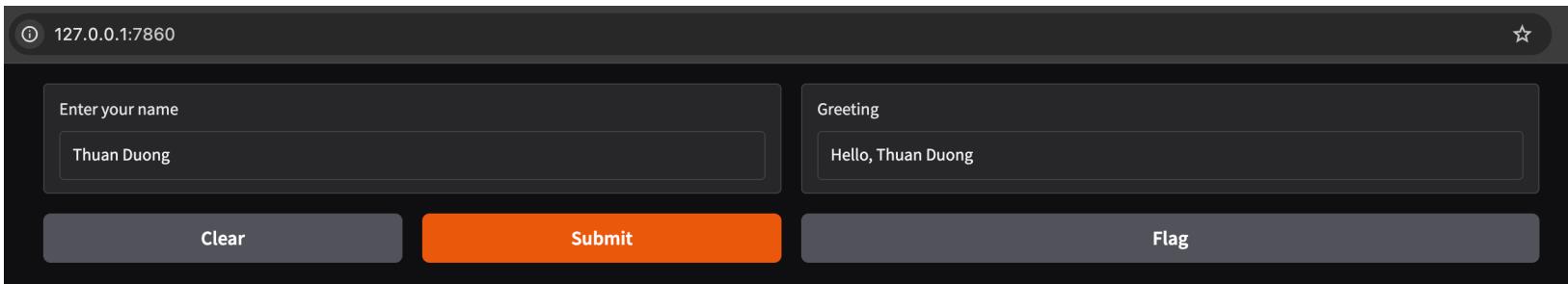
❖ Inputs: Text

Textbox

Generates a textarea that lets users input a string or view the resulting string output.

2_text_inputs.py

```
● ● ●  
1 import gradio as gr  
2  
3 def greet(name):  
4     return "Hello, " + name  
5  
6 demo = gr.Interface(  
7     fn=greet,  
8     inputs=gr.Textbox(label="Enter your name",  
9                         placeholder="Type your name here..."),  
10    outputs=gr.Textbox(label="Greeting"),  
11 )  
12  
13 demo.launch()
```



Gradio Basics

❖ Inputs: Text

HighlightedText

Shows text where individual spans are highlighted based on their category or numerical value.

POS

NER

HighlightedText Demo

Tiny sentiment demo: highlights a few positive/negative words.

Input text

```
I love this course, it is amazing!
```

Clear Submit

Token highlights

```
I love POSITIVE this course , it is amazing POSITIVE !
```

8_highlight_text_input.py

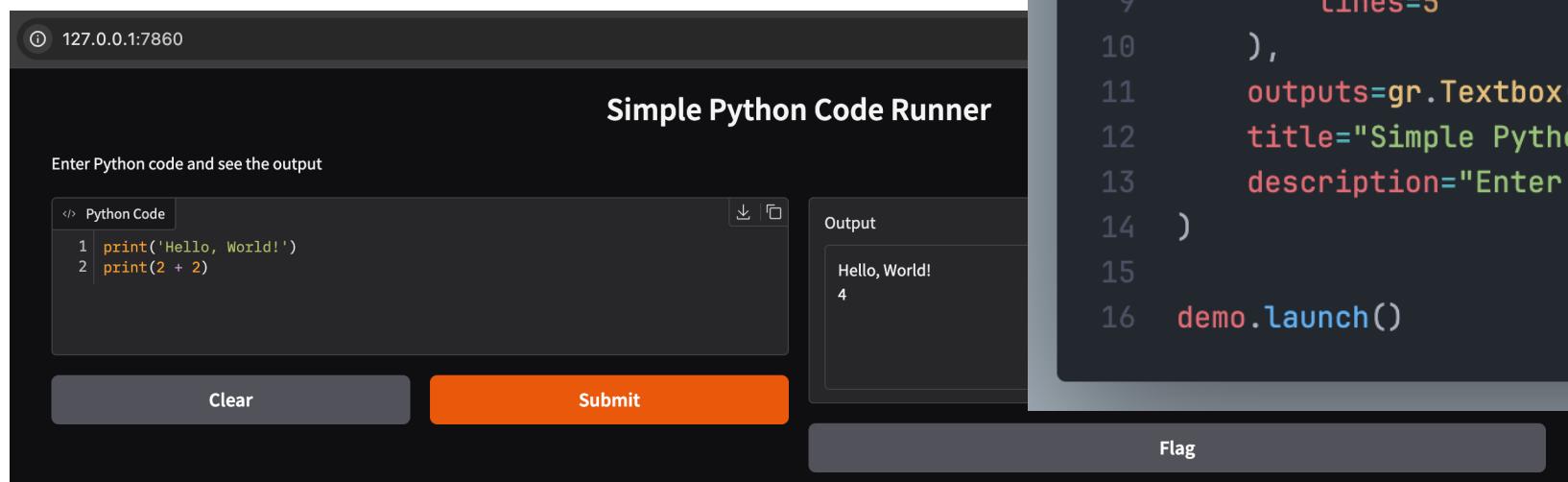
```
● ● ●
1  demo = gr.Interface(
2      fn=highlight_sentiment,
3      inputs=gr.Textbox(
4          label="Input text",
5          placeholder="I love this amazing course.",
6          lines=4,
7      ),
8      outputs=gr.HighlightedText(
9          label="Token highlights",
10         color_map={
11             "positive": "#10b981",
12             "negative": "#ef4444",
13         },
14     ),
15     title="HighlightedText Demo",
16     description="Tiny sentiment demo: " \
17                 "highlights a few positive/negative words.",
18     examples=[
19         "I love this course, it is amazing!",
20         "The movie was good but the ending was terrible.",
21         "What a wonderful day, I'm so happy.",
22         "The service was bad and the food was awful.",
23     ],
24 )
25 demo.launch()
```

Gradio Basics

❖ Inputs: Text

Code

Provides a code-editor component that can serve as an output view for displaying code or as an input area for entering and editing code.



```
● ● ● 3_code_input.py

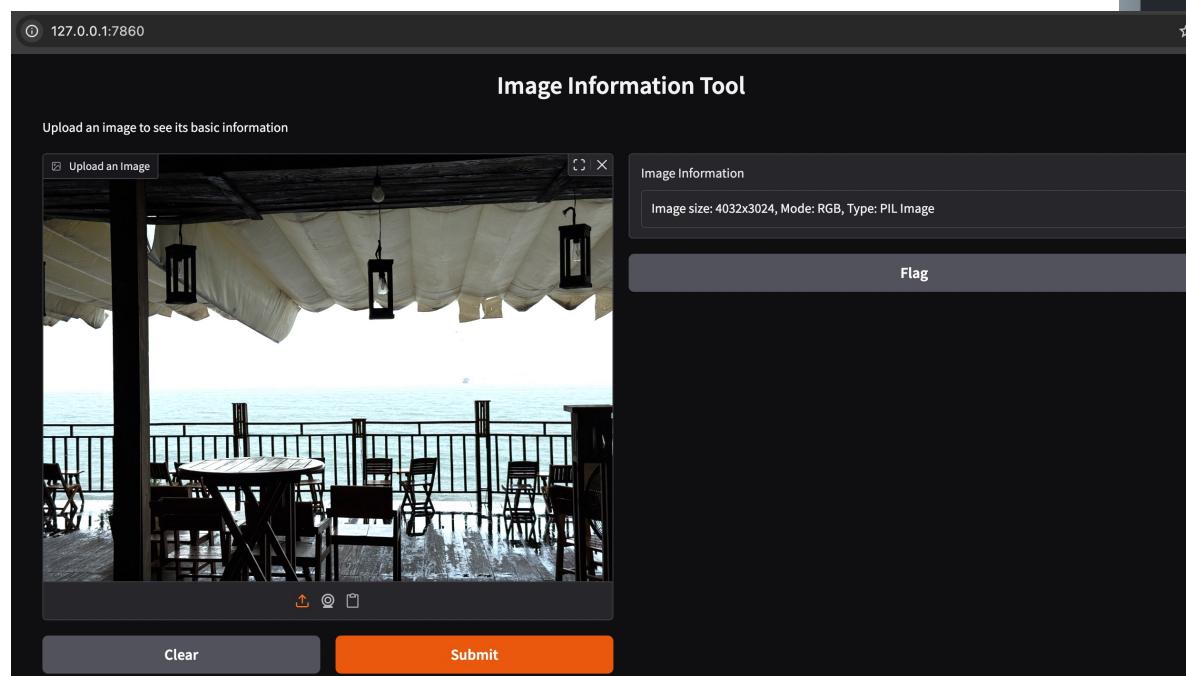
1 import gradio as gr
2
3 demo = gr.Interface(
4     fn=execute_python,
5     inputs=gr.Code(
6         label="Python Code",
7         language="python",
8         value="print('Hello, World!')\nprint(2 + 2)",
9         lines=5
10    ),
11    outputs=gr.Textbox(label="Output", lines=5),
12    title="Simple Python Code Runner",
13    description="Enter Python code and see the output"
14 )
15
16 demo.launch()
```

Gradio Basics

❖ Inputs: Image

Image

.Creates an image component that functions either as an input for uploading pictures or as an output for displaying them.



5_image_input_1.py

```
1 import gradio as gr
2 import numpy as np
3 from PIL import Image
4
5 def get_image_info(image):
6     """Get basic information about an image"""
7     if image is None:
8         return "No image provided"
9
10    # Handle different image types
11    if isinstance(image, np.ndarray):
12        height, width = image.shape[:2]
13        channels = image.shape[2] if len(image.shape) == 3 else 1
14        return f"Image size: {width}x{height}, Channels: {channels}, Type: NumPy array"
15    elif isinstance(image, Image.Image):
16        width, height = image.size
17        return f"Image size: {width}x{height}, Mode: {image.mode}, Type: PIL Image"
18    else:
19        return f"Image type: {type(image)}"
20
21
22    demo = gr.Interface(
23        fn=get_image_info,
24        inputs=gr.Image(label="Upload an Image", type="pil"),
25        outputs=gr.Textbox(label="Image Information"),
26        title="Image Information Tool",
27        description="Upload an image to see its basic information"
28    )
29    demo.launch()
```

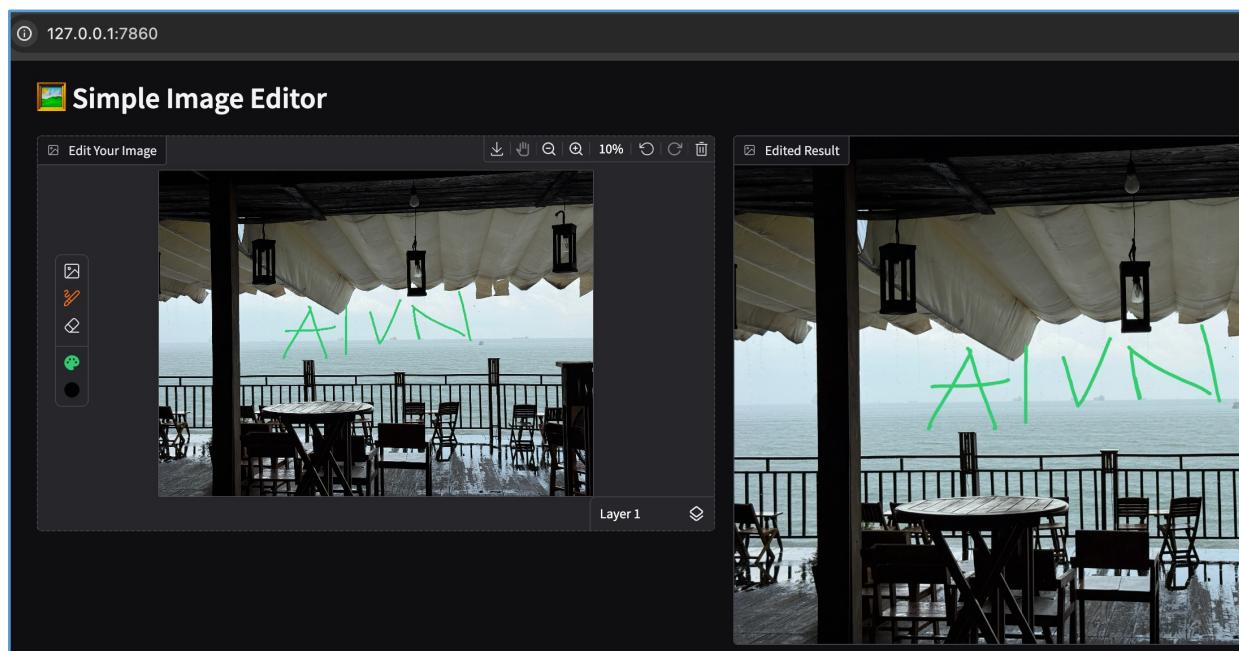
Gradio Basics

❖ Inputs: Image

ImageEditor

Provides an image component that serves two purposes:

- Input mode – lets users upload photos and perform basic edits.
- Output mode – simply displays the resulting image.



<https://www.gradio.app/docs/gradio/imageeditor>



6_image_input_2.py

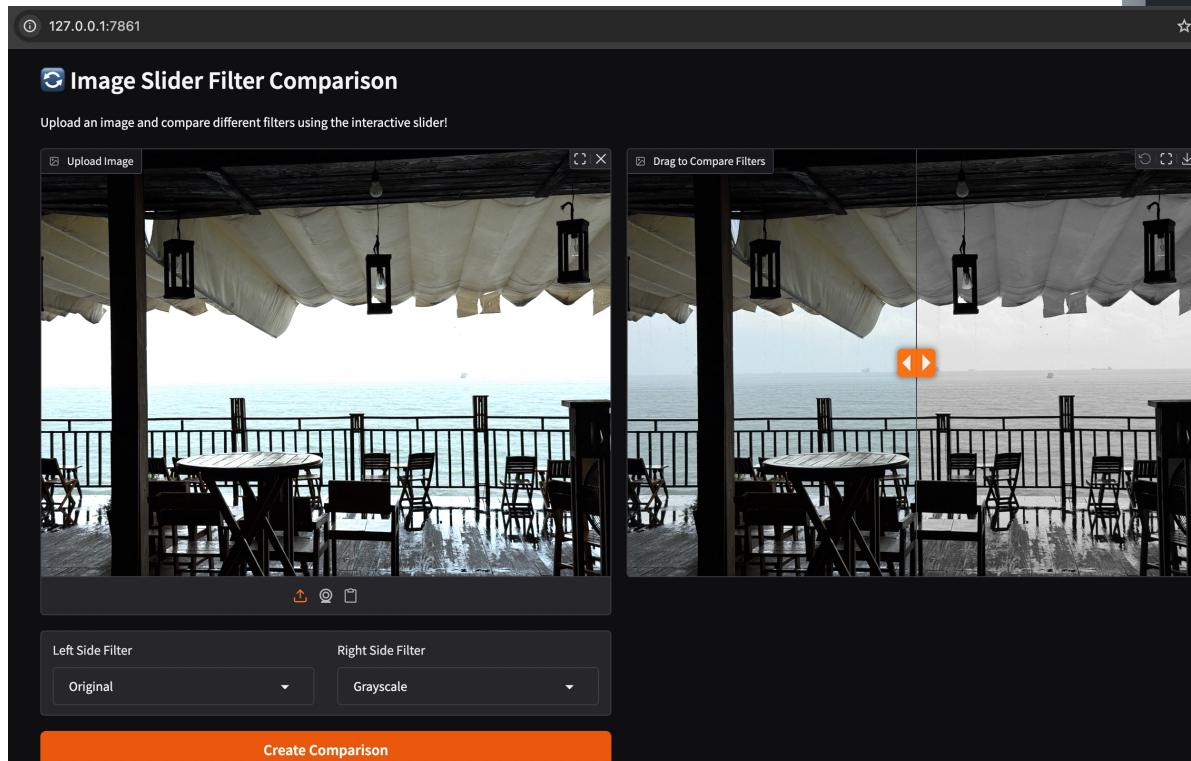
```
1 import gradio as gr
2
3 def process_image(image_data):
4     if image_data is None:
5         return None
6     return image_data.get('composite', None)
7
8 with gr.Blocks(title="Simple Image Editor") as demo:
9     gr.Markdown("# 🖼 Simple Image Editor")
10
11     with gr.Row():
12         image_editor = gr.ImageEditor(
13             label="Edit Your Image",
14             type="pil",
15             sources=["upload", "webcam"], # Allow upload and webcam
16             transforms=["crop"], # Enable cropping
17             brush=gr.Brush(default_size=10), # Drawing brush
18             eraser=gr.Eraser(default_size=10) # Eraser tool
19         )
20         output_image = gr.Image(label="Edited Result", type="pil")
21
22         image_editor.change(
23             fn=process_image,
24             inputs=[image_editor],
25             outputs=[output_image]
26         )
27
28 demo.launch()
```

Gradio Basics

❖ Inputs: Image

ImageSlider

Creates an image component that can function as an input for uploading pictures or as an output for displaying them.



<https://www.gradio.app/docs/gradio/imageslider>

7_image_input_3.py

```
1 choices = ["Original", "Grayscale", "Blur", "Sharpen",
2             "Brightness", "Contrast", "Sepia", "Vintage"]
3
4 with gr.Blocks(title="Image Slider Demo") as demo:
5     gr.Markdown("# 📸 Image Slider Filter Comparison")
6
7     with gr.Row():
8         with gr.Column():
9             input_image = gr.Image(label="Upload Image", type="pil")
10            with gr.Row():
11                filter1_dropdown = gr.Dropdown(
12                    choices=choices,
13                    value="Original",
14                    label="Left Side Filter"
15                )
16                filter2_dropdown = gr.Dropdown(
17                    choices=choices,
18                    value="Grayscale",
19                    label="Right Side Filter"
20                )
21
22            compare_btn = gr.Button("Create Comparison", variant="primary")
23
24            with gr.Column():
25                comparison_slider = gr.ImageSlider(
26                    label="Drag to Compare Filters",
27                    elem_id="comparison",
28                )
```

Gradio Basics

❖ Output: Label

Displays a classification label and, when available, the confidence scores for the top categories.

<https://www.gradio.app/docs/gradio/label>

8_label_output.py

```
● ● ●
1 demo = gr.Interface(
2     fn=classify_sentiment,
3     inputs=gr.Textbox(
4         label="Input text",
5         placeholder="e.g. I love this course, it is amazing!",
6         lines=4,
7     ),
8     outputs=gr.Label(label="Predicted sentiment"),
9     title="Label Output Demo",
10    description="gr.Label - simple sentiment classifier demo. " \
11                  "Type text to see predicted label and confidences.",
12    examples=[
13        "I love this amazing course but the pacing is bad.",
14        "The movie was good but the ending was terrible.",
15        "What a wonderful day, I'm so happy.",
16        "The service was bad and the food was awful.",
17    ],
18 )
19 demo.launch()
```

Label Output Demo

gr.Label — simple sentiment classifier demo. Type text to see predicted label and confidences.

Input text

What a wonderful day, I'm so happy.

Predicted sentiment

positive

Category	Confidence (%)
positive	51%
neutral	42%
negative	7%

Clear **Submit**

Gradio Basics

❖ Output: JSON

```
9_json_output.py
1 demo = gr.Interface(
2     fn=analyze,
3     inputs=gr.Textbox(
4         label="Input text",
5         placeholder="e.g. Gradio is great for prototyping ML apps!",
6         lines=6,
7     ),
8     outputs=gr.JSON(label="Analysis JSON"),
9     title="JSON Output Demo",
10    description="Type some text and get a structured JSON summary.",
11    examples=[
12        "Hello world. Hello Gradio. This is a simple JSON output demo.",
13        "One two two three three three.",
14        "Gradio makes it easy to build demos. JSON shows structured data.",
15    ],
16)
17
18 demo.launch()
```

Used to render arbitrary JSON data in a nicely formatted, readable way.

127.0.0.1:7860

JSON Output Demo

Type some text and get a structured JSON summary.

Input text

```
One two two three three three.
```

Clear Submit

Analysis JSON

```
1 {
2     "summary": {
3         "num_tokens": 6,
4         "num_unique": 3,
5         "num_sentences": 1
6     },
7     "top_5_tokens": [
8         {
9             "token": "three",
10            "count": 3
11        },
12        {
13            "token": "two",
14            "count": 2
15        },
16        {
17            "token": "one",
18            "count": 1
19        }
20    ],
21    "unique_tokens_sample": [
22        "one"
23    ]
24}
```

Flag

Gradio Basics

❖ Multi-Input / Multi-Output



10_multi_inputs_outputs.py

```
1  demo = gr.Interface(
2      fn=multi_analyze,
3      inputs=[
4          gr.Textbox(label="Text", placeholder="Type a sentence...", lines=4),
5          gr.Slider(label="Threshold", minimum=0.0, maximum=1.0, step=0.05, value=0.4),
6          gr.Checkbox(label="Verbose JSON", value=True),
7      ],
8      outputs=[
9          gr.Label(label="Sentiment"),
10         gr.JSON(label="JSON Summary"),
11         gr.Textbox(label="Text Summary", lines=3),
12     ],
13     title="Multi-Input / Multi-Output Demo",
14     description="Enter text, set a threshold, toggle verbosity. " \
15                 "Returns a label, JSON summary, and a short text summary.",
16     examples=[
17         ["I love this amazing course but some parts are bad.", 0.4, True],
18         ["The service was awful and the food was horrible.", 0.5, True],
19         ["Just a regular day.", 0.3, False],
20     ],
21 )
22 demo.launch()
```

Gradio Basics

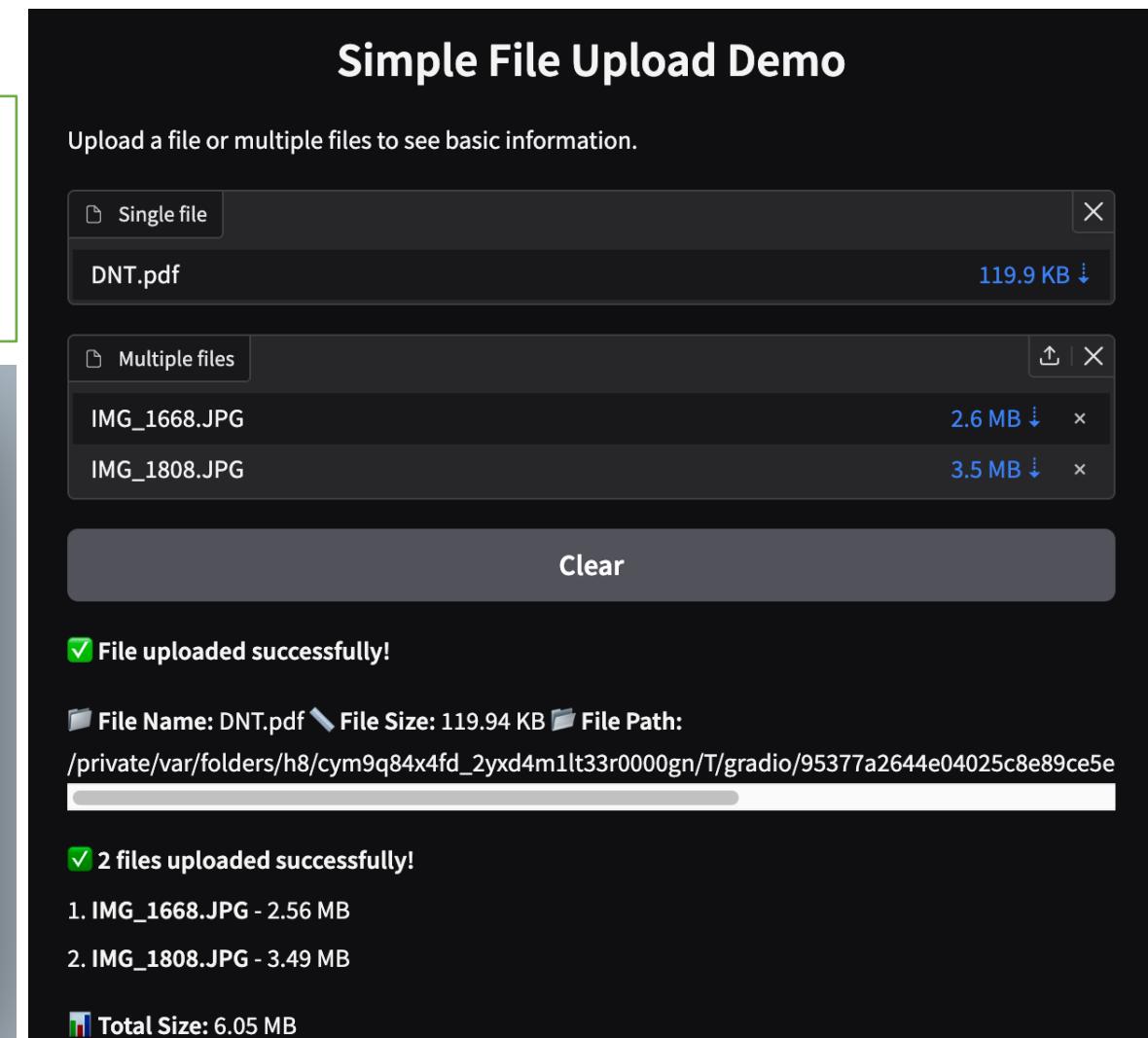
❖ Inputs: File

Simple & Multi Files

Creates a file component that can:

- **Input mode** – upload one or multiple generic files.
- **Output mode** – display generic files or provide URLs for downloading them.

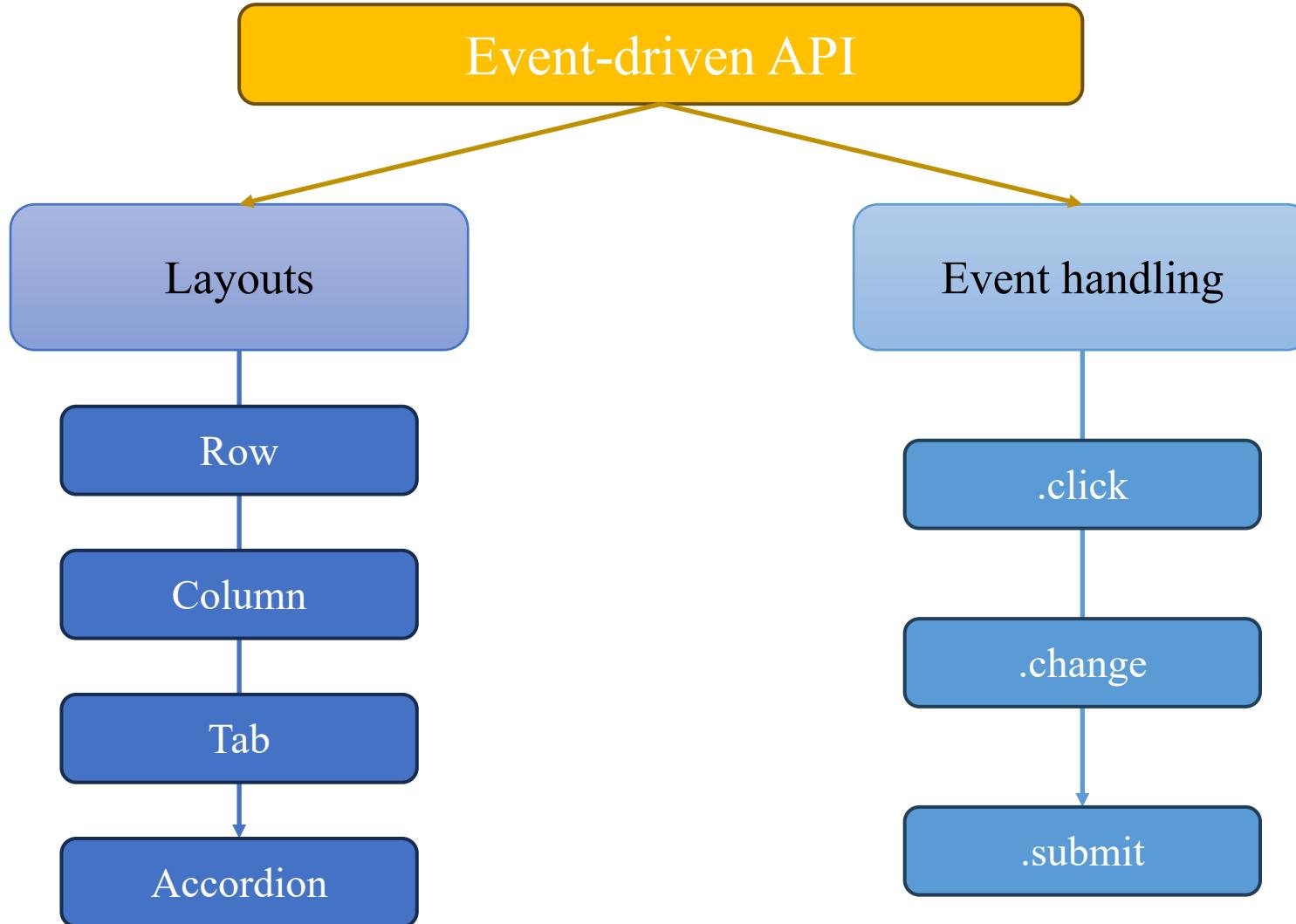
```
7_file_input.py
1 demo = gr.Interface(
2     fn=process_pair,
3     inputs=[
4         gr.File(label="Single file", file_count="single"),
5         gr.File(label="Multiple files", file_count="multiple"),
6     ],
7     outputs=[
8         gr.Markdown(label="Single file info"),
9         gr.Markdown(label="Multiple files info"),
10    ],
11    title="Simple File Upload Demo",
12    description="Upload a file or multiple files to see basic information.",
13    live=True,
14 )
15
16 demo.launch()
```



Gradio Blocks

Blocks

❖ Getting Started



Blocks

❖ Gradio Blocks

Blocks is Gradio's low-level API, enabling you to build more customized web applications and demos than what **Interface** offers—while still staying entirely within Python.

Feature	gr.Interface	gr.Interface
Ease of use	Very easy, minimal code	More verbose
Customization	Limited	Very flexible
Layout control	Layout control	Layout control
Best for	Simple model demos	Complex apps (chatbots, dashboards)
Events	Only function binding	Multiple event handlers

Blocks

❖ Layout

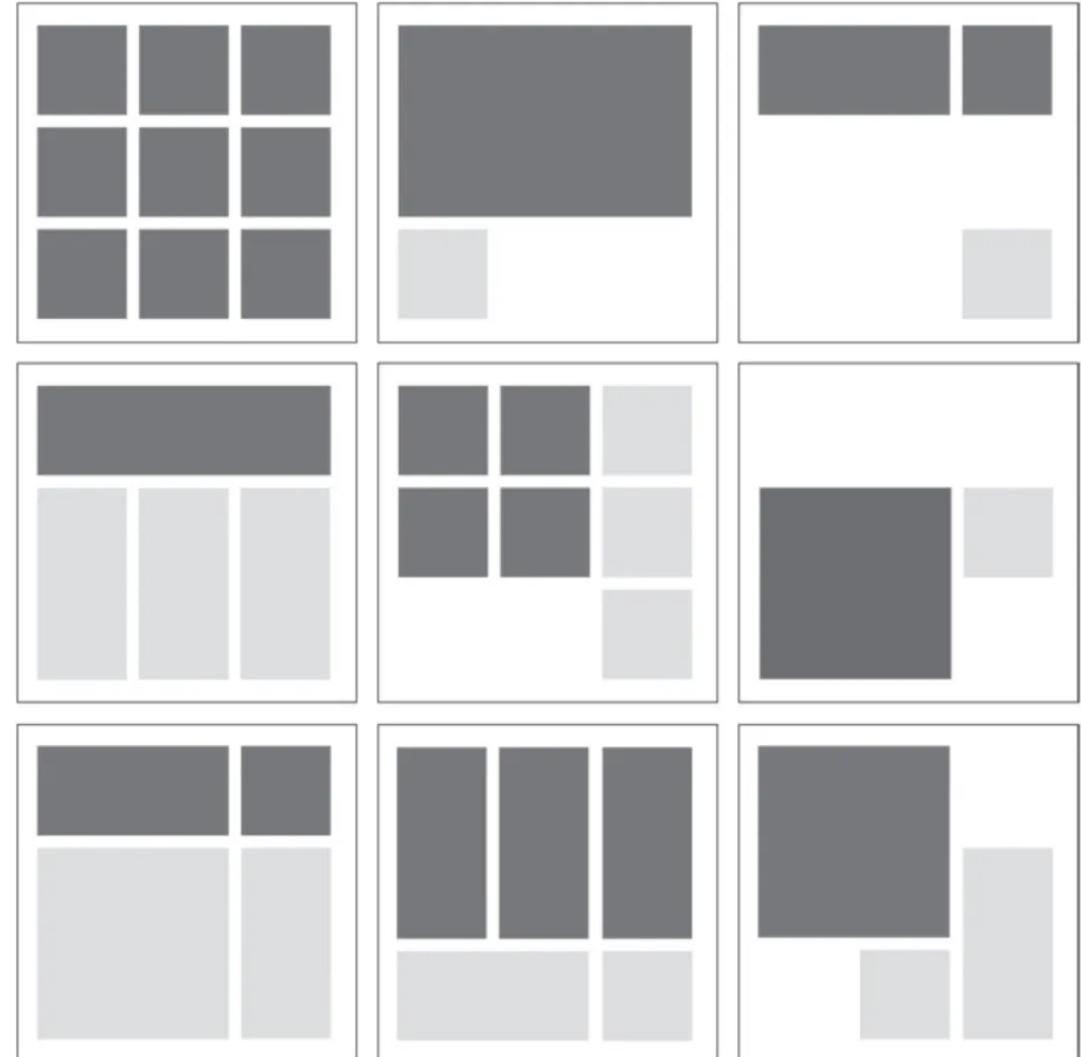
In UI/UX design:

A layout is the arrangement of components (buttons, text, images, input fields, etc.) on a screen or web page to ensure usability and aesthetics.



In web development:

Layout means how elements are positioned and sized (using CSS Grid, Flexbox, floats, etc.).



Blocks

❖ Layout: Row

Arrange components side by side.

<https://www.gradio.app/docs/gradio/row>

ⓘ 127.0.0.1:7860

Simple Blocks + Row demo

Name

Type your name

Times

1

Greet

Output



12_blocks_row.py

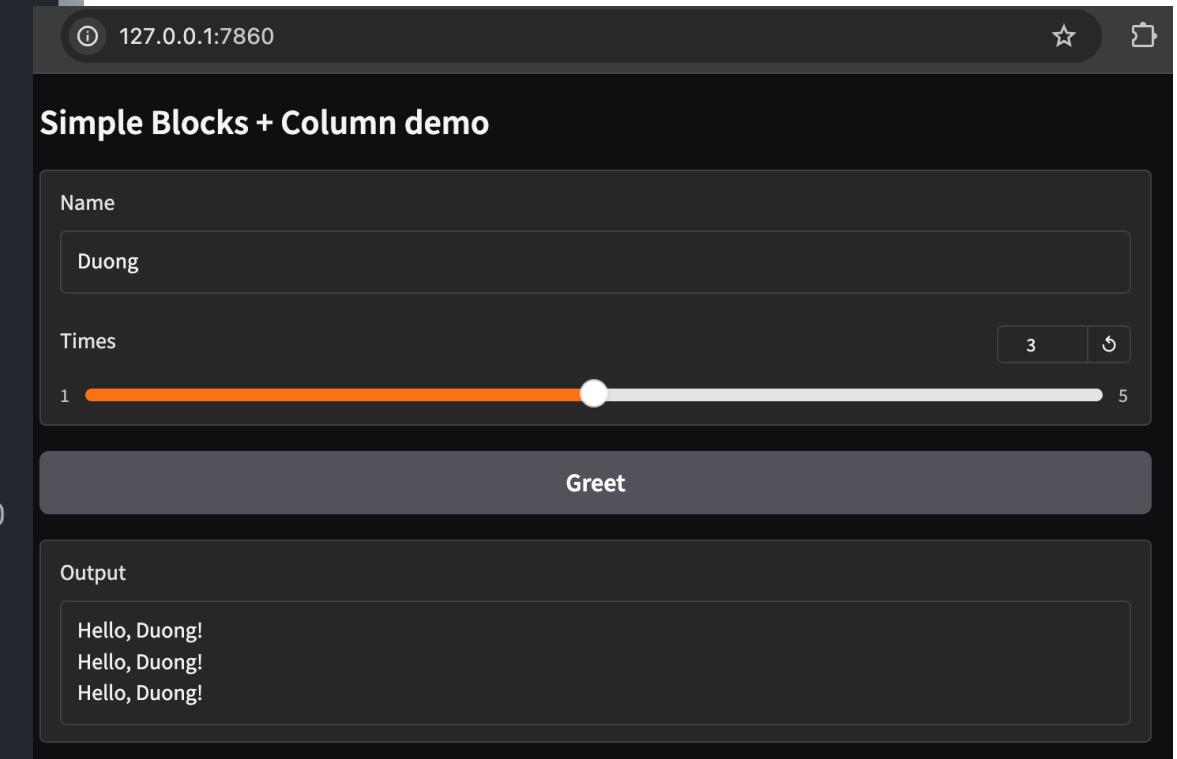
```
1 import gradio as gr
2
3 def greet(name: str, times: float) -> str:
4     n = int(times or 1)
5     name = name or "there"
6     return "\n".join([f"Hello, {name}!" for _ in range(max(n, 1))])
7
8
9 with gr.Blocks(title="Blocks + Row: simplest") as demo:
10     gr.Markdown("## Simple Blocks + Row demo")
11
12     with gr.Row():
13         name = gr.Textbox(label="Name", placeholder="Type your name")
14         times = gr.Number(value=1, label="Times")
15
16     with gr.Row():
17         btn = gr.Button("Greet")
18         out = gr.Textbox(label="Output")
19
20     demo.launch()
```

Blocks

❖ Layout: Column

Stack components vertically.

```
● ● ● 13_blocks_column.py
1 import gradio as gr
2
3 def greet(name: str, times: float) -> str:
4     n = int(times or 1)
5     name = name or "there"
6     return "\n".join([f"Hello, {name}!" for _ in range(max(n, 1))])
7
8
9 with gr.Blocks(title="Blocks + Column: simplest") as demo:
10    gr.Markdown("## Simple Blocks + Column demo")
11
12    with gr.Column():
13        name = gr.Textbox(label="Name", placeholder="Type your name")
14        times = gr.Slider(1, 5, value=1, step=1, label="Times")
15        btn = gr.Button("Greet")
16        out = gr.Textbox(label="Output", interactive=False)
17
18 demo.launch()
19
```



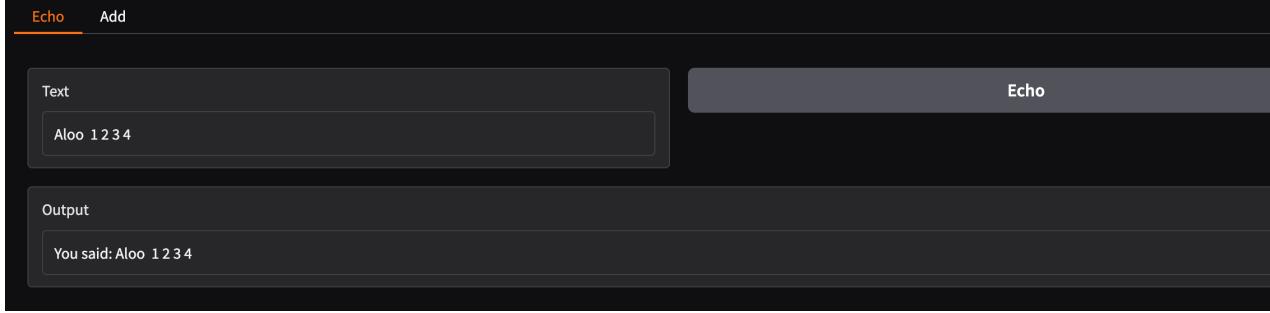
<https://www.gradio.app/docs/gradio/column>

Blocks

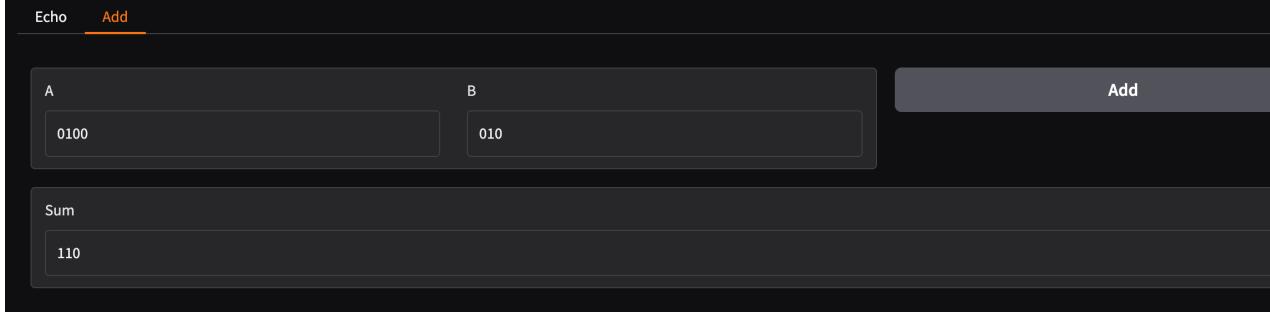
❖ Layout: Tab

Organize components in different tabs

Simple Blocks + Tabs demo



Simple Blocks + Tabs demo



<https://www.gradio.app/docs/gradio/tab>

14_blocks_tabs.py

```
● ● ●
1 import gradio as gr
2
3 def echo(text: str) → str:
4     text = text or ""
5     return f"You said: {text}"
6
7
8 def add(a: float, b: float) → float:
9     return (a or 0) + (b or 0)
10
11
12 with gr.Blocks(title="Blocks + Tabs: simplest") as demo:
13     gr.Markdown("## Simple Blocks + Tabs demo")
14
15     with gr.Tabs():
16         with gr.TabItem("Echo"):
17             with gr.Row():
18                 inp = gr.Textbox(label="Text", placeholder="Type something")
19                 btn1 = gr.Button("Echo")
20                 out1 = gr.Textbox(label="Output", interactive=False)
21                 btn1.click(echo, inputs=inp, outputs=out1)
22
23         with gr.TabItem("Add"):
24             with gr.Row():
25                 a = gr.Number(value=0, label="A")
26                 b = gr.Number(value=0, label="B")
27                 btn2 = gr.Button("Add")
28                 out2 = gr.Number(label="Sum", interactive=False)
29                 btn2.click(add, inputs=[a, b], outputs=out2)
30
31     demo.launch()
32
```

Blocks

❖ Layout: Tab

A toggle element that reveals or conceals the content placed inside it.



```
1  with gr.Blocks(title="Accordion Demo") as demo:
2      gr.Markdown("# Gradio Blocks - Accordion Demo")
3      gr.Markdown("This simple app shows how to group UI "
4                  "in expandable sections using `gr.Accordion`.")
5
6      with gr.Accordion("Greet Me", open=True):
7          name = gr.Textbox(label="Your name",
8                              value="Alice",
9                              placeholder="Type your name...")
10         mood = gr.Radio(["Great", "Good", "OK", "Tired"],
11                          label="How are you?",
12                          value="Great")
13         greet_btn = gr.Button("Greet")
14         greet_out = gr.Textbox(label="Message", interactive=False)
15         greet_btn.click(greet, [name, mood], greet_out)
16
17     with gr.Accordion("Quick Utilities", open=False):
18         with gr.Row():
19             num = gr.Number(label="Number to square", value=10)
20             square_btn = gr.Button("Square it")
21             result = gr.Number(label="Result", interactive=False)
22             square_btn.click(square, num, result)
23
24 demo.launch()
```

15_blocks_accordion.py

Gradio Blocks – Accordion Demo

This simple app shows how to group UI in expandable sections using `gr.Accordion`.

The screenshot displays a user interface for a "Gradio Blocks – Accordion Demo". The interface is contained within a dark-themed window. At the top, there is a header bar with the title "Greet Me" and a small downward arrow icon. Below the header, the "Greet Me" section is expanded, showing a text input field labeled "Your name" containing "Alice", and a radio button group labeled "How are you?" where the "Great" option is selected. A "Greet" button is located below these controls. Below the "Greet Me" section, the "Quick Utilities" section is collapsed, indicated by a downward arrow icon. This section contains a text input field labeled "Number to square" with the value "10", a "Square it" button, and a text output field labeled "Result". The overall design is clean and modern, utilizing a dark color palette and rounded corners for its components.

❖ Event handling

```
● ● ●  
1 def greet(name: str, times: float) -> str:  
2     n = int(times or 1)  
3     name = name or "there"  
4     return "\n".join([f"Hello, {name}!" for _ in range(max(n, 1))])  
5  
6 def live_preview(name: str) -> str:  
7     return f"Typing: {name or ''}"  
8  
9 def on_times_change(times: float) -> str:  
10    return f"Times set to {int(times or 1)}"  
11  
12 with gr.Blocks(title="Event Handling: simplest") as demo:  
13     gr.Markdown("## Event Handling demo")  
14     with gr.Row():  
15         with gr.Column():  
16             name = gr.Textbox(label="Name", placeholder="Type your name and press Enter")  
17             times = gr.Slider(1, 5, value=1, step=1, label="Times")  
18             greet_btn = gr.Button("Greet")  
19         with gr.Column():  
20             live_md = gr.Markdown("Start typing to see live preview")  
21             times_md = gr.Markdown("Times set to 1")  
22             out = gr.Textbox(label="Output", interactive=False)  
23  
24     # Events  
25     name.input(live_preview, inputs=name, outputs=live_md)  
26     name.submit(greet, inputs=[name, times], outputs=out)  
27     greet_btn.click(greet, inputs=[name, times], outputs=out)  
28     times.change(on_times_change, inputs=times, outputs=times_md)  
29  
30 demo.launch()
```

Event Handling demo

Name

Times

1 5

Greet

Typing: Thuan

Times set to 2

Output

16_event_handling.py

Event handling

is the mechanism that detects user actions—such as clicks, keystrokes, form submissions, or selections—and triggers a specific function (the handler) in response.

Blocks

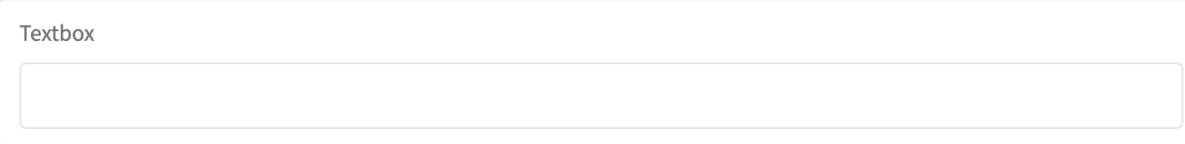
❖ Event Listeners

New to Gradio? Start here: [Getting Started](#)

← State See the [Release History](#) Timer →

Textbox

```
gradio.Textbox(...)
```



The right side of the page features a sidebar with the following sections:

- Textbox
- Description
- Behavior
- Initialization
- Shortcuts
- Demos
- Event Listeners
- Guides

A large orange arrow points from the "Event Listeners" section in the sidebar back towards the "change" button at the bottom of the page.

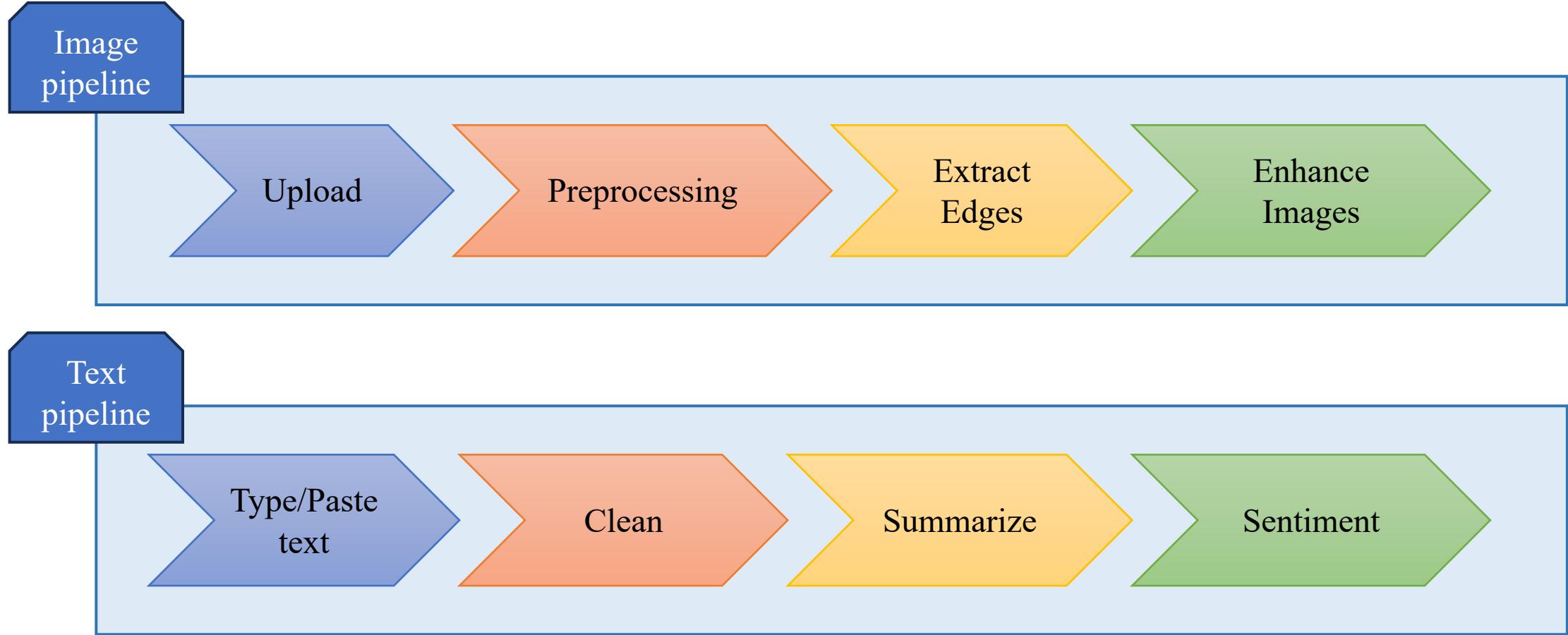
change input select submit blur stop copy

<https://www.gradio.app/docs/gradio/textbox#event-listeners>

Advanced Usage

Advanced Usage

❖ Complex Apps: Multi-step workflows



Advanced Usage

❖ Complex Apps: Multi-step workflows



17_complex_app.py

```
1  with gr.TabItem("Image Pipeline"):
2      with gr.Row():
3          with gr.Column(scale=1):
4              image_in = gr.Image(label="Upload Image", type="pil")
5              strength = gr.Slider(0.1, 3.0, value=1.0, step=0.1, label="Edge Strength")
6              # Removed Demo Delay slider
7              with gr.Row():
8                  btn_pre = gr.Button("Step 1: Preprocess")
9                  btn_edge = gr.Button("Step 2: Detect Edges")
10                 btn_enh = gr.Button("Step 3: Enhance")
11             with gr.Row():
12                 btn_run_all = gr.Button("Run All", variant="primary")
13                 btn_reset_img = gr.Button("Reset")
14
15                 # Internal states to pass between steps
16                 st_pre = gr.State()
17                 st_edge = gr.State()
18
19                 with gr.Column(scale=1):
20                     out_pre = gr.Image(label="Preprocessed", interactive=False)
21                     out_edge = gr.Image(label="Edges", interactive=False)
22                     out_enh = gr.Image(label="Enhanced", interactive=False)
```

Each step behind requires the previous step to be completed.



❖ Complex Apps

```
1 # Wiring events for image pipeline
2 def _preprocess_and_store(img, progress=gr.Progress(track_tqdm=True)):
3     p = preprocess_image(img, progress=progress)
4     return p, p
5
6 btn_pre.click(_preprocess_and_store, inputs=[image_in], outputs=[out_pre, st_pre])
7
8 def _edge_and_store(img_pre, k, progress=gr.Progress(track_tqdm=True)):
9     if img_pre is None:
10         gr.Warning("Run Step 1 first.")
11         return None, None
12     e = detect_edges(img_pre, strength=k, progress=progress)
13     return e, e
14
15 btn_edge.click(_edge_and_store, inputs=[st_pre, strength], outputs=[out_edge, st_edge])
16
17 def _enhance(img_edge, progress=gr.Progress(track_tqdm=True)):
18     if img_edge is None:
19         gr.Warning("Run Step 2 first.")
20         return None
21     return enhance_image(img_edge, progress=progress)
22
23 btn_enh.click(_enhance, inputs=[st_edge], outputs=out_enh)
24
25 def _run_all(img, k, progress=gr.Progress(track_tqdm=True)):
26     p, e, h = run_all_image(img, k, progress=progress)
27     # Also store states for continuity
28     return p, e, h, p, e
29
30 btn_run_all.click(_run_all, inputs=[image_in, strength], outputs=[out_pre, out_edge, out_enh, st_pre, st_edge])
31
32 def _reset_img():
33     return None, None, None, None, None
34
35 btn_reset_img.click(_reset_img, outputs=[image_in, out_pre, out_edge, out_enh, st_pre])
36
```

Advanced Usage

❖ Customization: Theming

<https://www.gradio.app/guides/theming-guide>

The screenshot illustrates the process of customizing a Gradio application's theme. On the left, the 'Event Handling demo' interface is shown with a dark theme. It features a 'Name' input field, a 'Times' slider set to 1, and a 'Greet' button. On the right, the code for this interface is displayed in a Python file named `18_theming.py`:

```
1 with gr.Blocks(
2     title="Event Handling: simplest",
3     theme=gr.themes.Soft()
4 ) as demo:
```

Two yellow arrows point from the UI elements on the left to the corresponding code blocks on the right, indicating how each part of the UI is defined in the theme file. The bottom half of the image shows the same 'Event Handling demo' interface, but with a different color scheme (purple and blue), demonstrating the result of applying the `gr.themes.Soft()` theme.

Advanced Usage

❖ Customization: JS/HTML integration

The screenshot shows a web browser window with the URL huggingface.co/spaces/VLAI-AIVN/AIO2025M03_DEMO_KNN. The page title is "KNN Demo" under "AIO2025: Module 03.". The interface includes a sidebar with "Data & Configuration" and "Results & Visualization" sections, and a main area with "Interactive KNN Visualization" and "Prediction Result" sections.

AI VIET NAM
@aivietnam.edu.vn

KNN Demo

AIO2025: Module 03.

How to Use: Select data → Configure target → Set parameters → Enter new point → Run prediction!

Data & Configuration

Start with sample datasets or upload your own CSV/Excel files.

Upload Your Data

Drop File Here

- or -

Click to Upload

Results & Visualization

Interactive KNN Visualization

Prediction Result

Advanced Usage

❖ Customization: JS/HTML integration

```
17 force_light_theme_js = """
18 () => {
19   const params = new URLSearchParams(window.location.search);
20   if (!params.has('__theme')) {
21     params.set('__theme', 'light');
22     window.location.search = params.toString();
23   }
24 }
25 """
```

```
def create_footer():
    logo_base64_vlai = image_to_base64("static/vlai_logo.png")
    footer_html = """
<style>
    .sticky-footer{position:fixed;bottom:0px;left:0;width:100%;background:#F
        padding:10px;box-shadow:0 -2px 10px rgba(0,0,0,0.1);z-ind
    .content-wrap{padding-bottom:60px;}
</style>"""+ f"""
<div class="sticky-footer">
    <div style="text-align:center;font-size:18px; color: #888">
        Created by
        <a href="https://vlai.work" target="_blank" style="color:#465C88;text-
         from <a href="https://aivietnam.edu.vn/" target="_blank" style="c
    </div>
</div>
"""
```

```
56 custom_css = """
57
58 .gradio-container {
59   min-height: 100vh !important;
60   width: 100vw !important;
61   margin: 0 !important;
62   padding: 0px !important;
63   background: linear-gradient(135deg, #F5EFE6 0%, #E8DFCA 50%, #AEBDCA 100%);
64   background-size: 600% 600%;
65   animation: gradientBG 7s ease infinite;
66 }
67
68 @keyframes gradientBG {
69   0% {background-position: 0% 50%;}
70   50% {background-position: 100% 50%;}
71   100% {background-position: 0% 50%;}
72 }
73
74 /* Minimize spacing and padding */
75 .content-wrap {
76   padding: 2px !important;
77   margin: 0 !important;
78 }
```

EXAMPLES

Deployment & Sharing

Deployment & Sharing

❖ Public share links

The screenshot shows a Jupyter Notebook interface with two code cells and a terminal output.

Code Cell 1:

```
17
18     (method) def launch(
19         inline: bool | None = None,
20         inbrowser: bool = False,
21         share: bool | None = None, ← Set True
22         debug: bool = False,
23         max_threads: int = 40,
24         #
25         auth: ((str, str) → bool) | tuple[str, str] | list[tuple[str, str]] | None = None,
26         n
27         n
28         prevent_thread_lock: bool = False,
29         g
30         show_error: bool = False,
31         t
32         server_name: str | None = None,
33         server_port: int | None = None,
```

Code Cell 2:

```
31
32     demo.launch(share=True)
```

Terminal Output:

```
PROBLEMS OUTPUT AZURE DEBUG CONSOLE TERMINAL PORTS
(gradio_env) thuanduong@MacNaN Code % python 16_event_handling.py
* Running on local URL: http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.
```

A red arrow points from the 'share' parameter in the first code cell to the 'share=True' argument in the second code cell. A red dashed line connects the two cells.

Output Box:

```
PROBLEMS OUTPUT AZURE DEBUG CONSOLE TERMINAL PORTS
(gradio_env) thuanduong@MacNaN Code % python 16_event_handling.py
* Running on local URL: http://127.0.0.1:7860
* Running on public URL: https://d392123aa6e9817c6a.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU up
/huggingface.co/spaces)
```

Deployment & Sharing

❖ Hugging Face Spaces (free hosting)

<https://huggingface.co/spaces>

The screenshot shows the Hugging Face Spaces interface. At the top, there are two browser tabs: one for 'huggingface.co/new-space' and another for 'huggingface.co/spaces/thuanan/Multi-step-workflows/tree/main'. The main content area displays a repository named 'Multi-step-workflows' owned by 'thuanan'. The repository has one commit from 'thuanan' labeled 'initial commit' with hash '38b10b7' and status 'VERIFIED'. Two files are listed: '.gitattributes' (1.52 KB) and 'README.md' (239 Bytes). A sidebar on the left shows fields for 'Owner' (thuanan), 'Short description' (Short Description), and 'License' (License). A red arrow points from the 'Gradio' template card in this sidebar to the 'Space hardware' section at the bottom. The bottom section also includes options for 'CPU basic · 2 vCPU · 16 GB · FREE'.

Spaces are Git repositories. You can build Spaces

Owner: thuanan

Short description: Short Description

License: License

Select the Space SDK: You can choose between Gradio, Docker, or Static to host your Space.

Space hardware: Free

CPU basic · 2 vCPU · 16 GB · FREE

Gradio (NEW)
3 templates

Docker
17 templates

Static
6 templates

Space hardware: Free

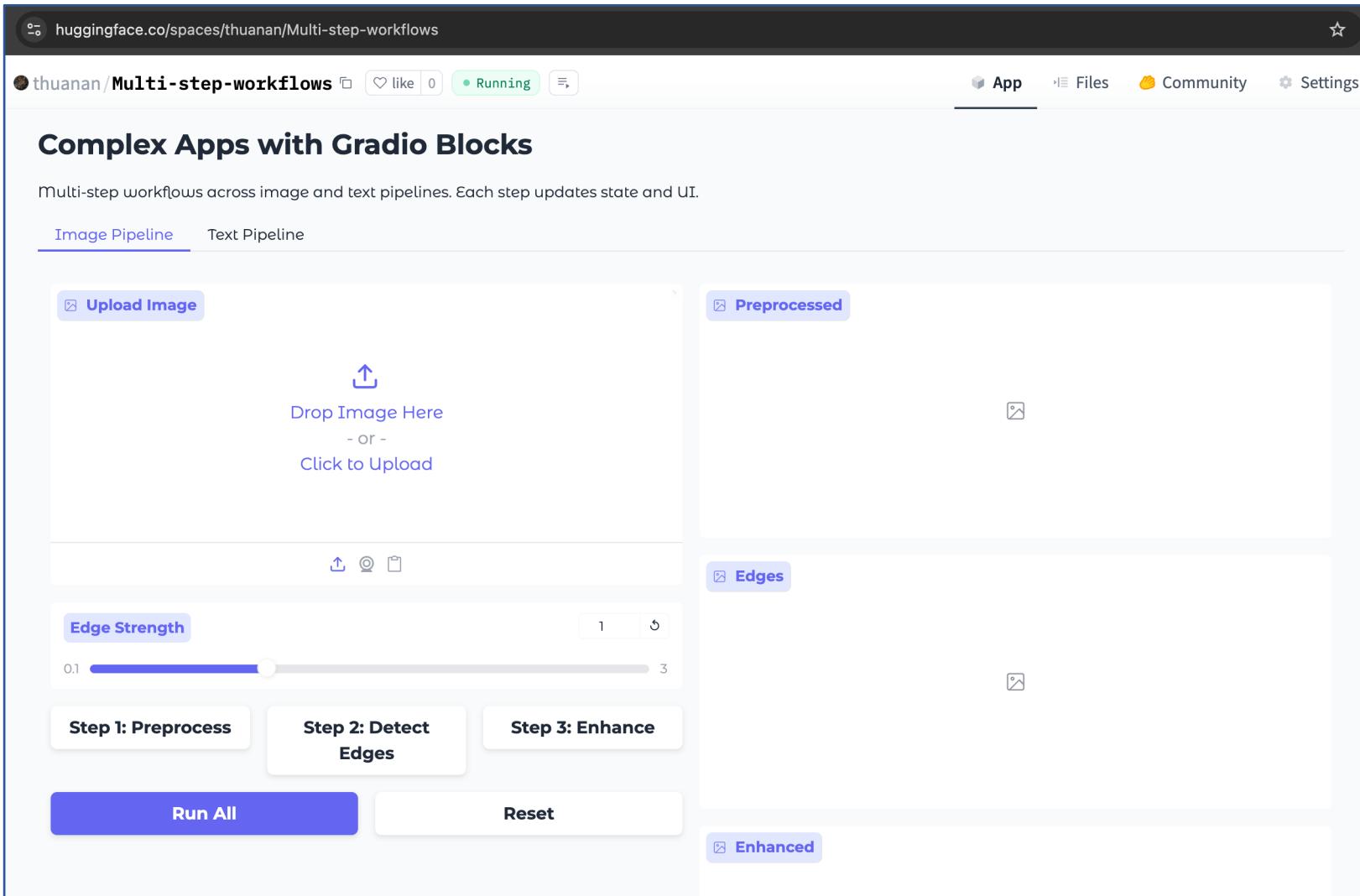
CPU basic · 2 vCPU · 16 GB · FREE

https://huggingface.co/spaces

Deployment & Sharing

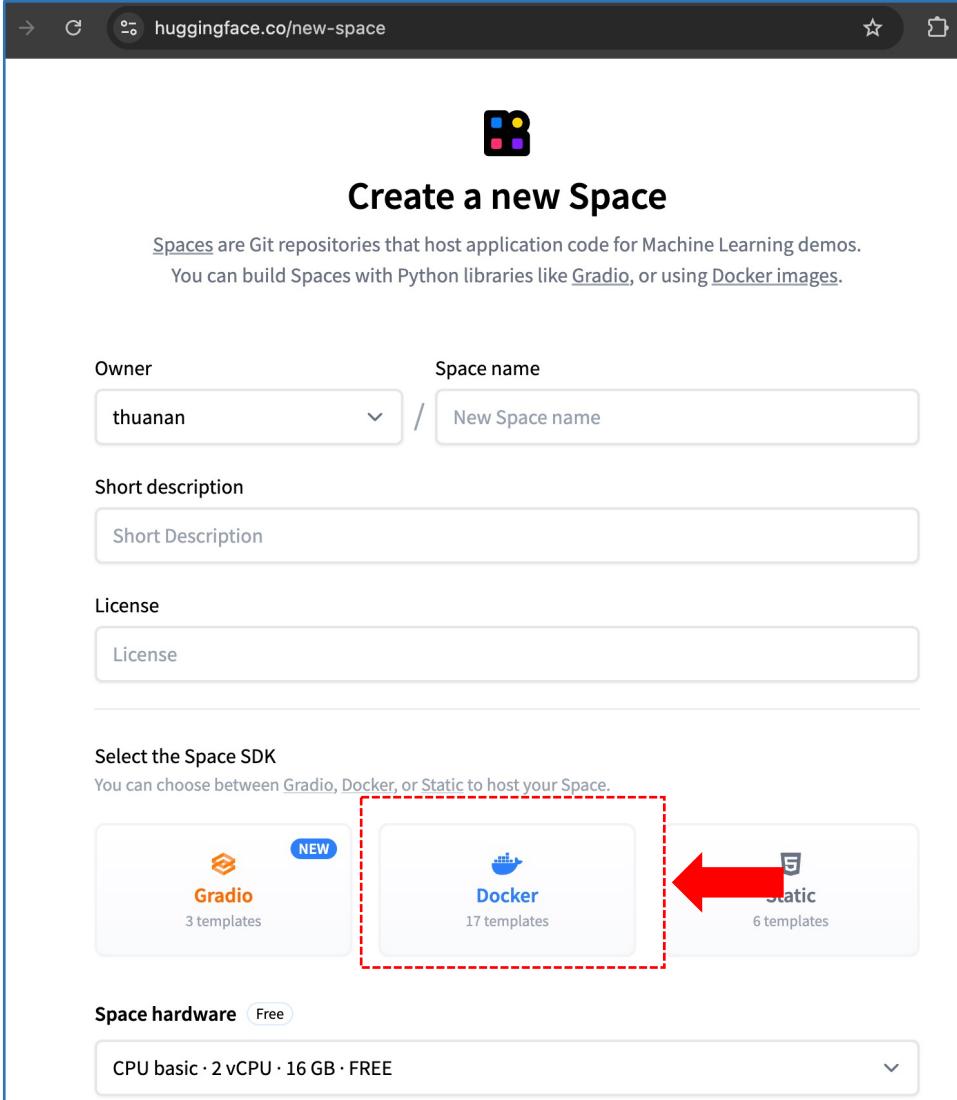
❖ Hugging Face Spaces (free hosting)

<https://huggingface.co/spaces/thuanan/Multi-step-workflows>



Deployment & Sharing

❖ Dockerize a Gradio app



```
1 FROM python:3.11-slim
2
3 ARG PORT=7860
4 ENV PORT=${PORT} \
5     GRADIO_SERVER_NAME=0.0.0.0 \
6     GRADIO_SERVER_PORT=${PORT}
7
8 WORKDIR /app
9
10 COPY requirements.txt /app/requirements.txt
11 RUN pip install --upgrade pip && \
12     pip install -r requirements.txt
13
14 COPY app.py /app/app.py
15
16 RUN useradd -m appuser && chown -R appuser /app
17 USER appuser
18
19 EXPOSE ${PORT}
20
21 # Run the Gradio app
22 CMD ["python", "app.py"]
23
24
```

Question

