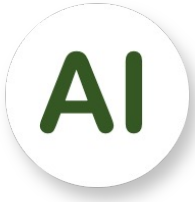
AI VIETNAM
All-in-One Course
(TA Session)

Gradio for Machine Learning Model Demos



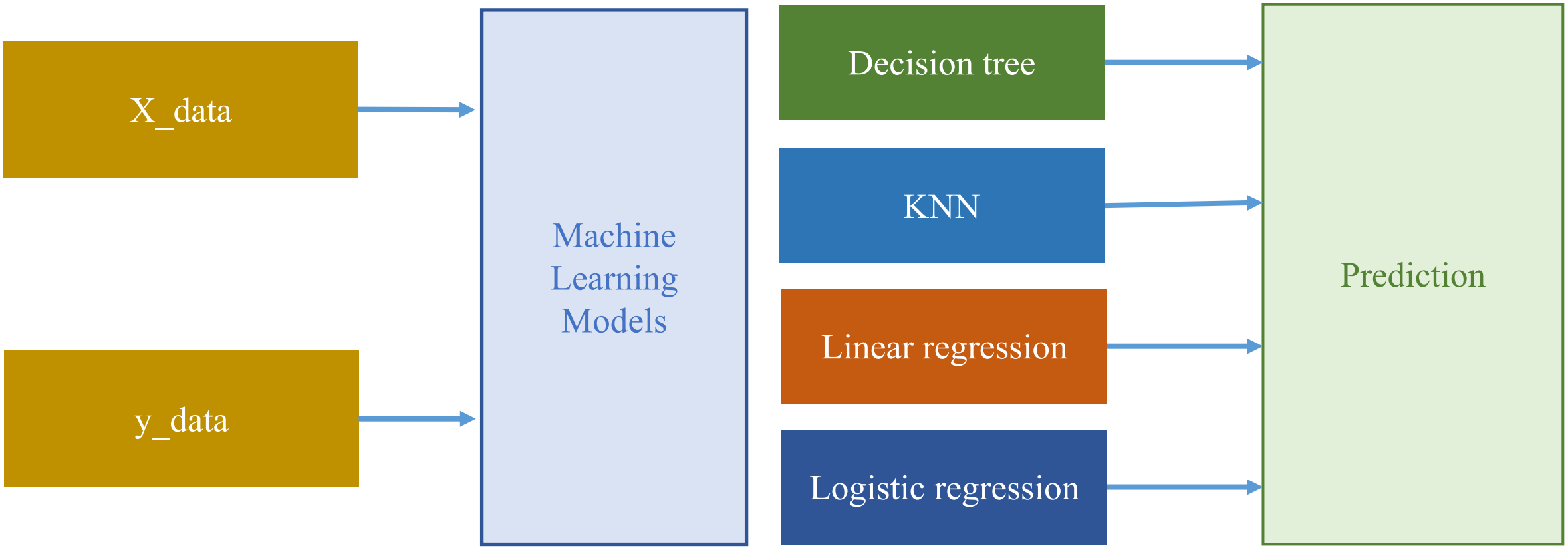
AI VIET NAM
@aivietnam.edu.vn

Nguyen-Thuan Duong – TA

Year 2025 Code&Data Quiz&Feedback

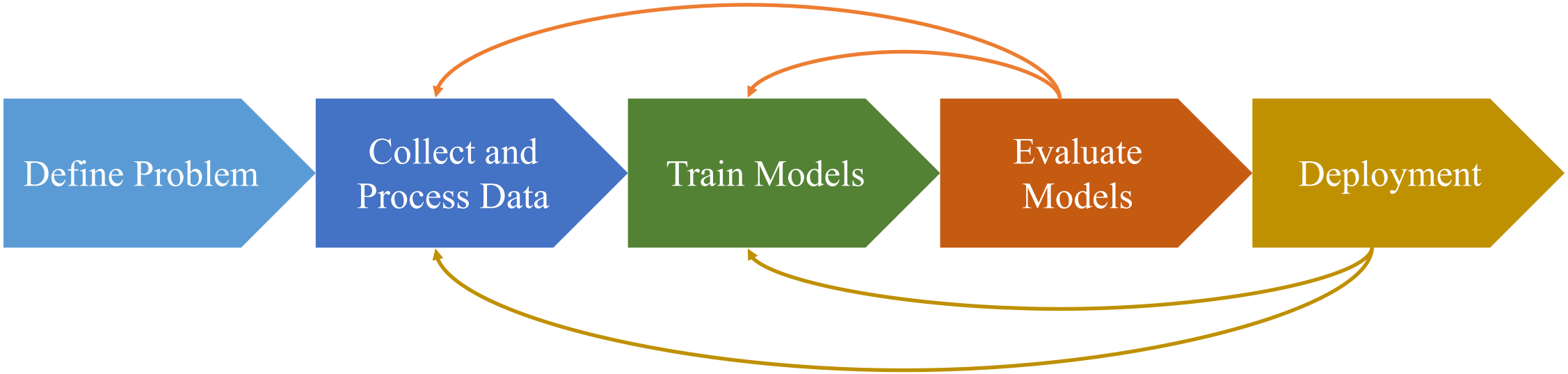
Machine Learning

❖ 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.

Ref: Designing Machine Learning System – Chip Huyen

Outline

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

Introduction

Introduction

❖ Getting Started

```python
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC
import soundfile as sf
import torch

processor = Wav2Vec2Processor.from_pretrained("nguyenvulebinh/wav2vec2-base-vietnamese-250h")
model = Wav2Vec2ForCTC.from_pretrained("nguyenvulebinh/wav2vec2-base-vietnamese-250h")

# define function to read in sound file
def map_to_array(batch):
    speech, _ = sf.read(batch["file"])
    batch["speech"] = speech
    return batch

# load dummy dataset and read soundfiles
ds = map_to_array({
    "file": 'audio-test/t1_0001-00010.wav'
})

input_values = processor(ds["speech"], return_tensors="pt", padding="longest").input_values
logits = model(input_values).logits

# take argmax and decode
predicted_ids = torch.argmax(logits, dim=-1)
transcription = processor.batch_decode(predicted_ids)
```

Introduction

❖ Developer vs End-user

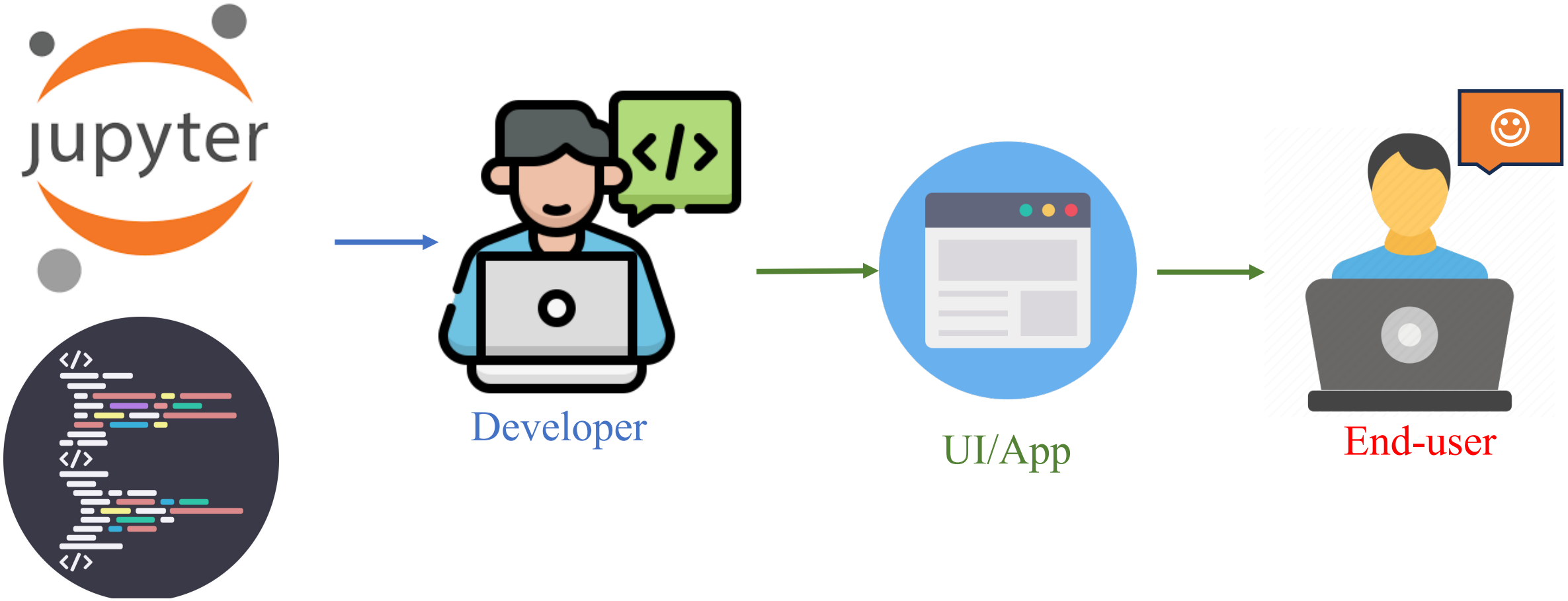
Here is my new model

Dev

End-user

Introduction

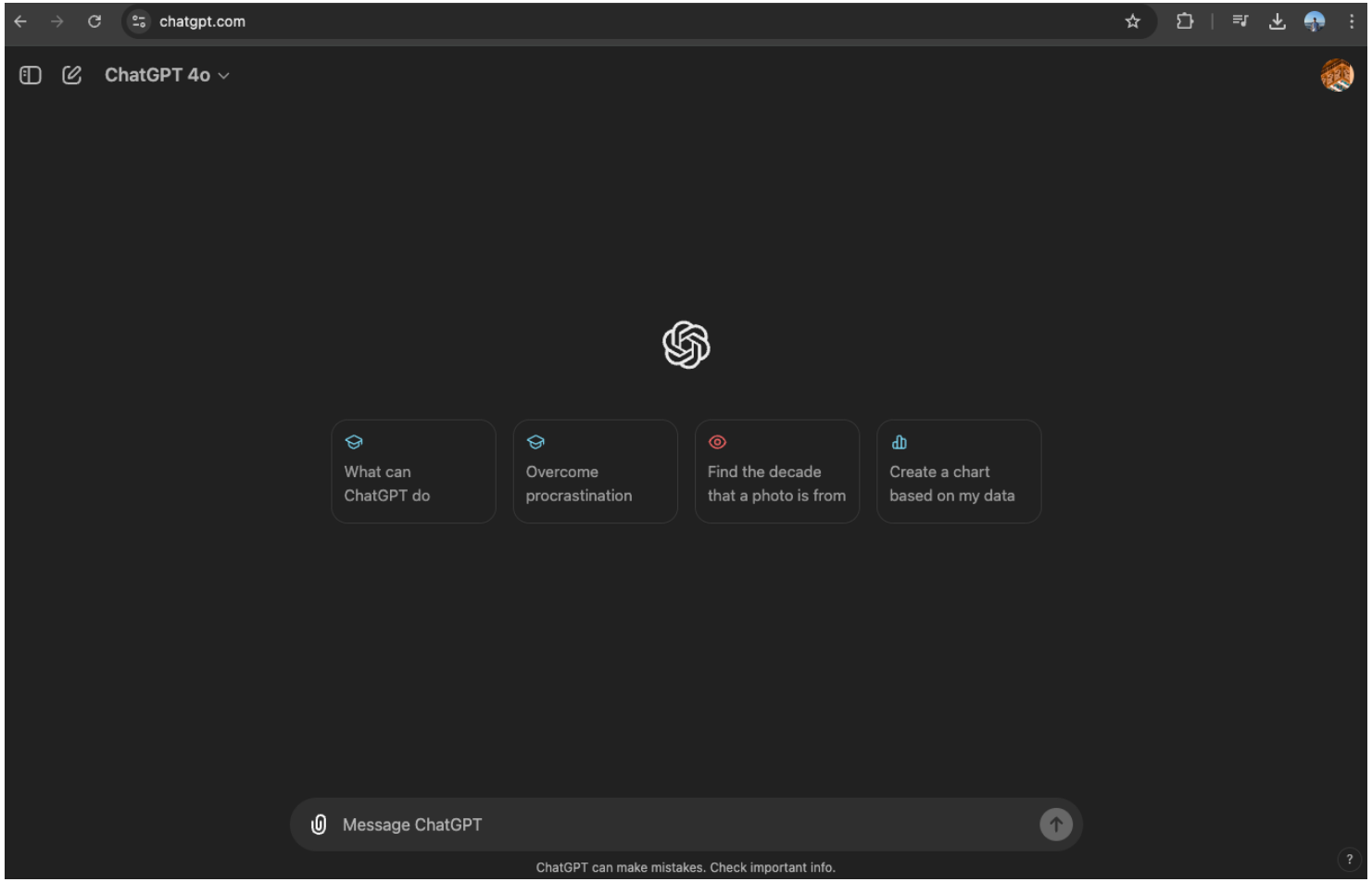
❖ Development



Web application

Web application

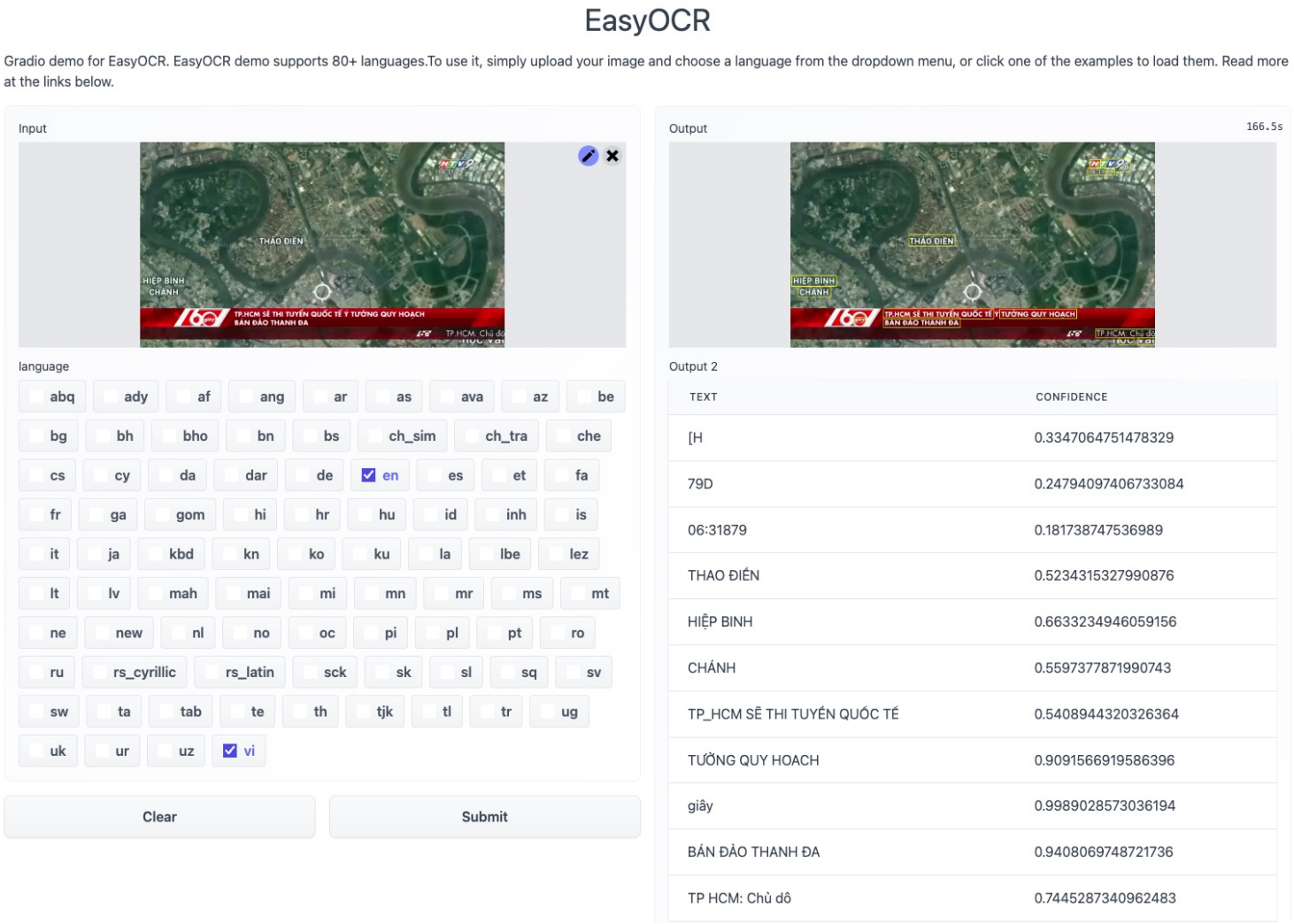
❖ ChatGPT web application



Ref: https://chatgpt.com

Introduction

❖ OCR demo



Ref: https://huggingface.co/spaces/tomofi/EasyOCR

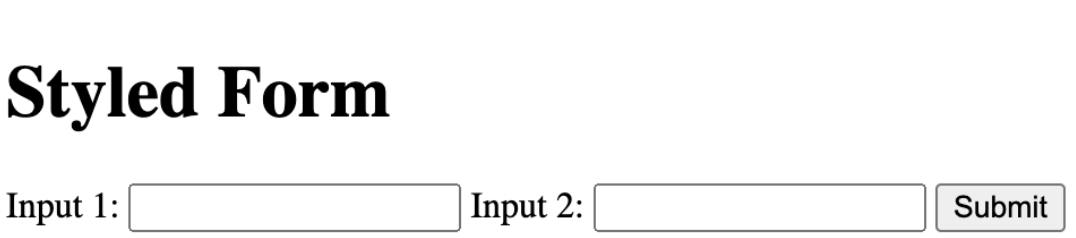
Introduction

❖ Web application

Introduction

❖ HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Styled Text Input Form</title>
</head>
<body>

    <div class="container">
        <h1>Styled Form</h1>
        <form action="#" method="post">
            <label for="input1">Input 1:</label>
            <input type="text" id="input1" name="input1" required>
            <label for="input2">Input 2:</label>
            <input type="text" id="input2" name="input2" required>
            <button type="submit">Submit</button>
        </form>
    </div>

</body>
</html>

```



Introduction

❖ HTML + CSS

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Styled Text Input Form</title>
    <link rel="stylesheet" href="index.css">
</head>
<body>

    <div class="container">
        <h1>Styled Form</h1>
        <form action="#" method="post">
            <label for="input1">Input 1:</label>
            <input type="text" id="input1" name="input1" required>
            <label for="input2">Input 2:</label>
            <input type="text" id="input2" name="input2" required>
            <button type="submit">Submit</button>
        </form>
    </div>

</body>
</html>

```

Introduction

❖ HTML + CSS

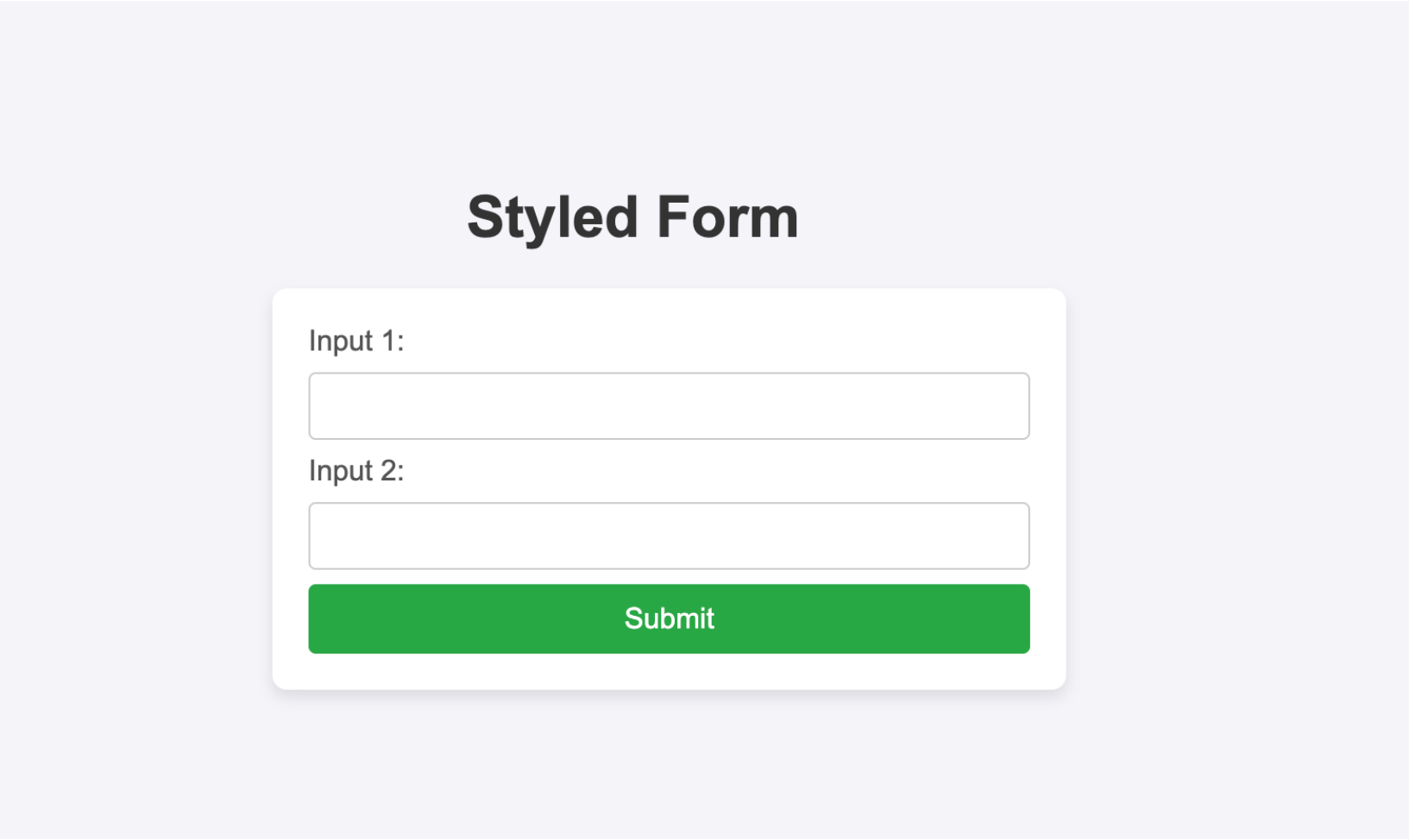
```css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

h1 {
    text-align: center;
    color: #333;
}
```

```css
form {
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 400px;
}

label {
    font-size: 16px;
    color: #555;
}

input[type="text"] {
    width: 100%;
    padding: 10px;
    margin: 8px 0;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}
```

```css
button {
    background-color: #28a745;
    color: white;
    padding: 10px 15px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
    font-size: 16px;
}

button:hover {
    background-color: #218838;
}

.container {
    width: 100%;
    max-width: 400px;
    margin: 0 auto;
}
```

Introduction

❖ HTML + CSS



Introduction

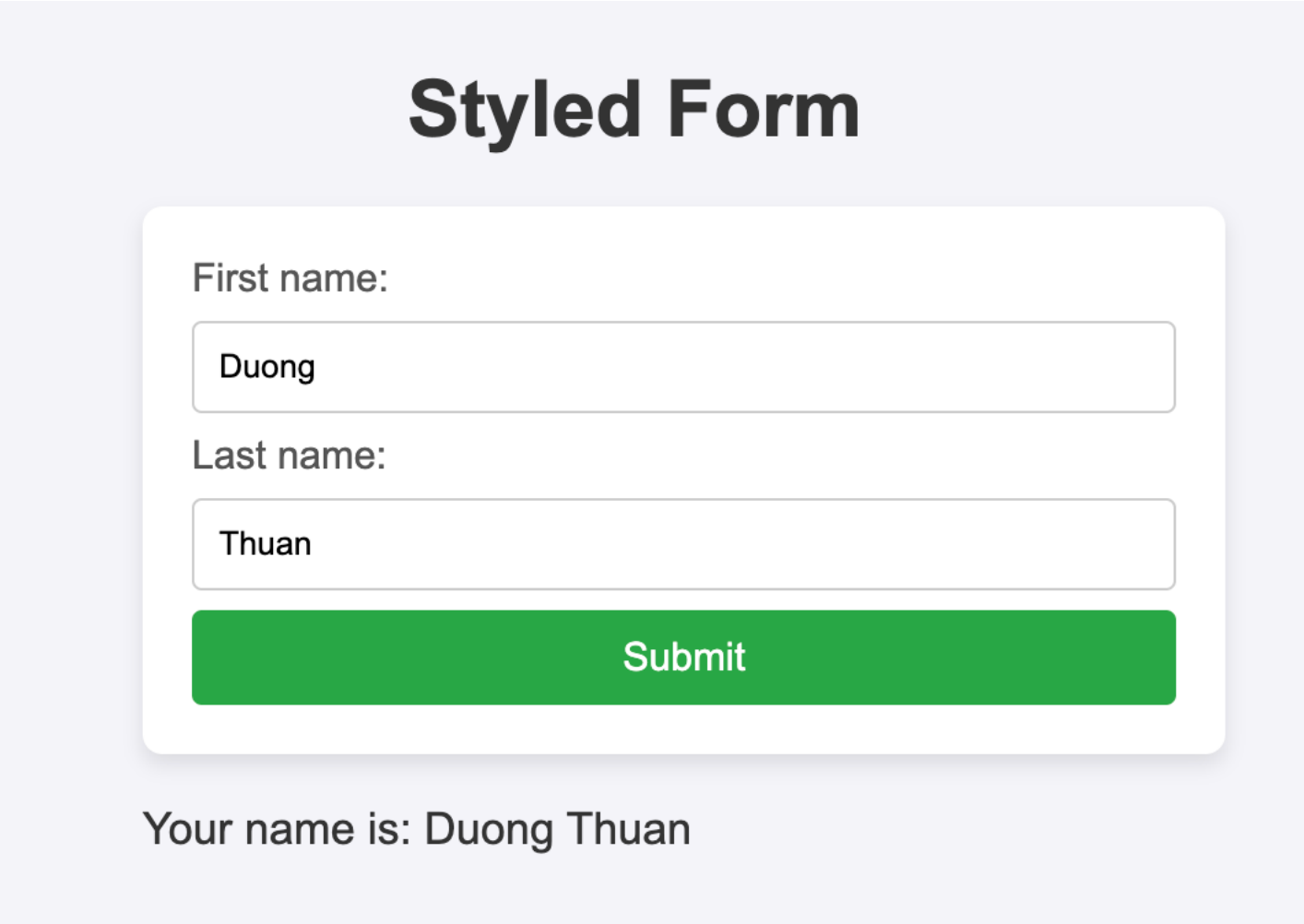
❖ HTML + CSS + JS

```html
<body>
    <div class="container">
        <h1>Styled Form</h1>
        <form id="concatForm" action="javascript:void(0);">
            <label for="input1">First name:</label>
            <input type="text" id="input1" name="input1" required>
            <label for="input2">Last name:</label>
            <input type="text" id="input2" name="input2" required>
            <button type="submit">Submit</button>
        </form>
        <p id="result"></p>
    </div>
    <script src="index.js"></script>
</body>
```

```javascript
document.getElementById('concatForm').addEventListener('submit', function(event) {
    event.preventDefault();

    const input1 = document.getElementById('input1').value;
    const input2 = document.getElementById('input2').value;

    const result = input1 + " " + input2;

    document.getElementById('result').textContent = "Your name is: " + result;
});

```

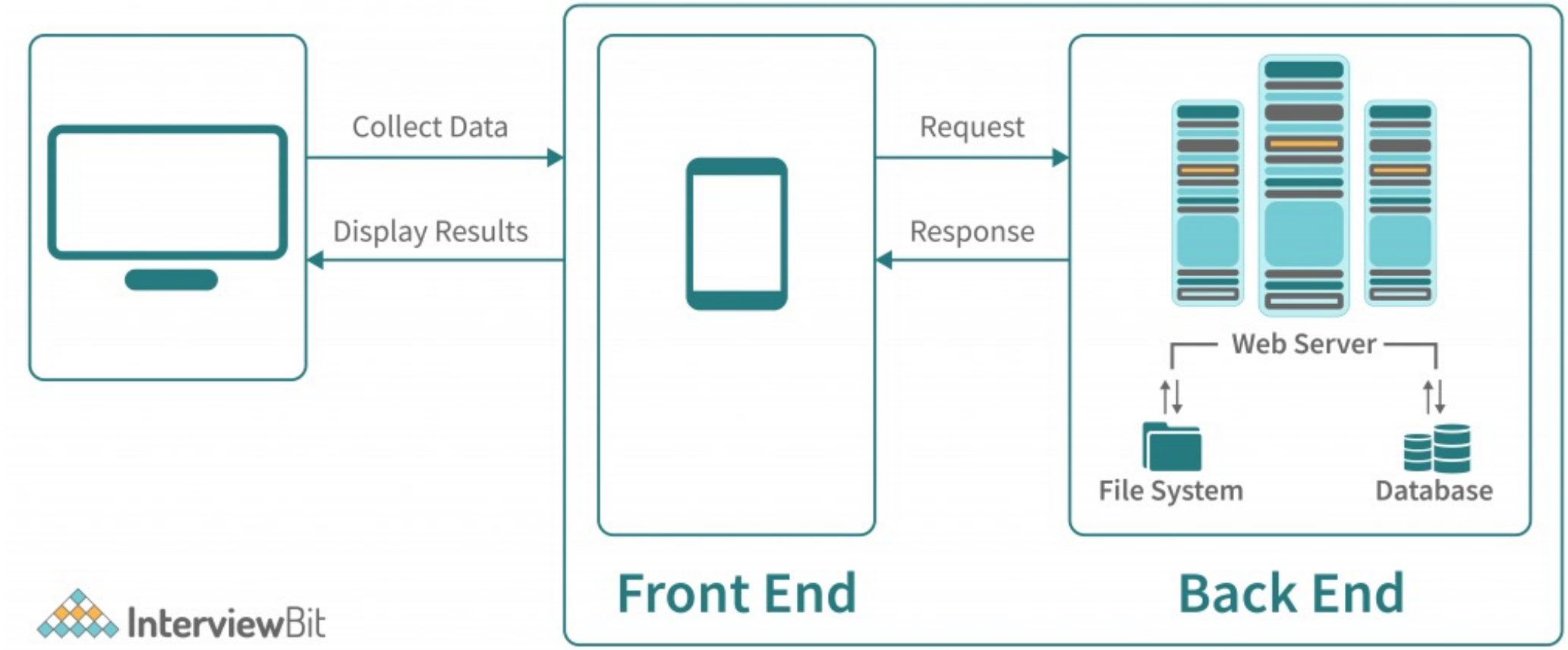
Introduction

- **HTML + CSS + JS**



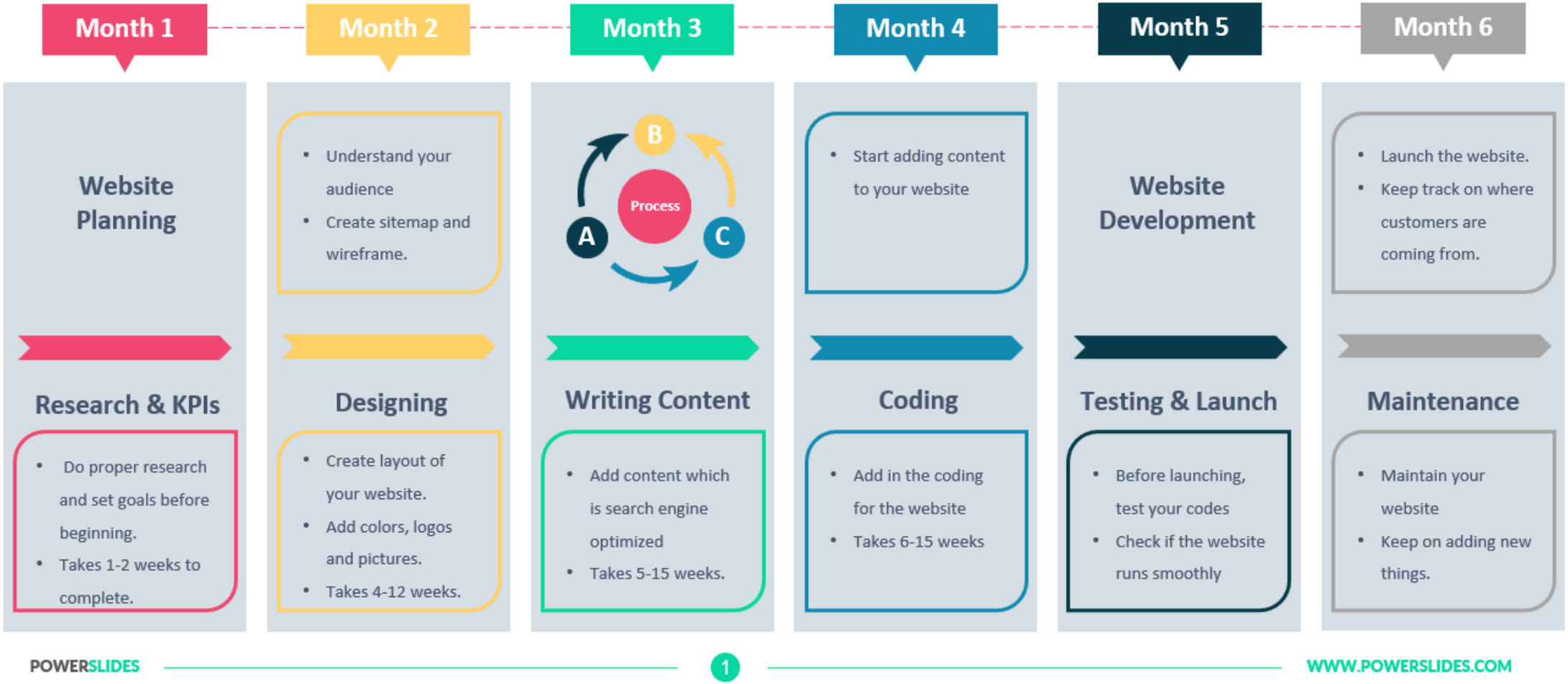
Introduction

❖ Web Development



Introduction

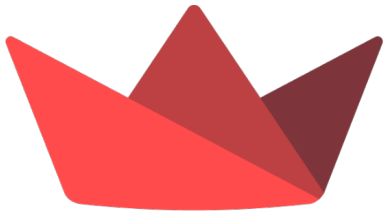
❖ Web Development



Introduction

❖ No HTML - No CSS - No JavaScript

gradio



Streamlit

Chainlit

Introduction

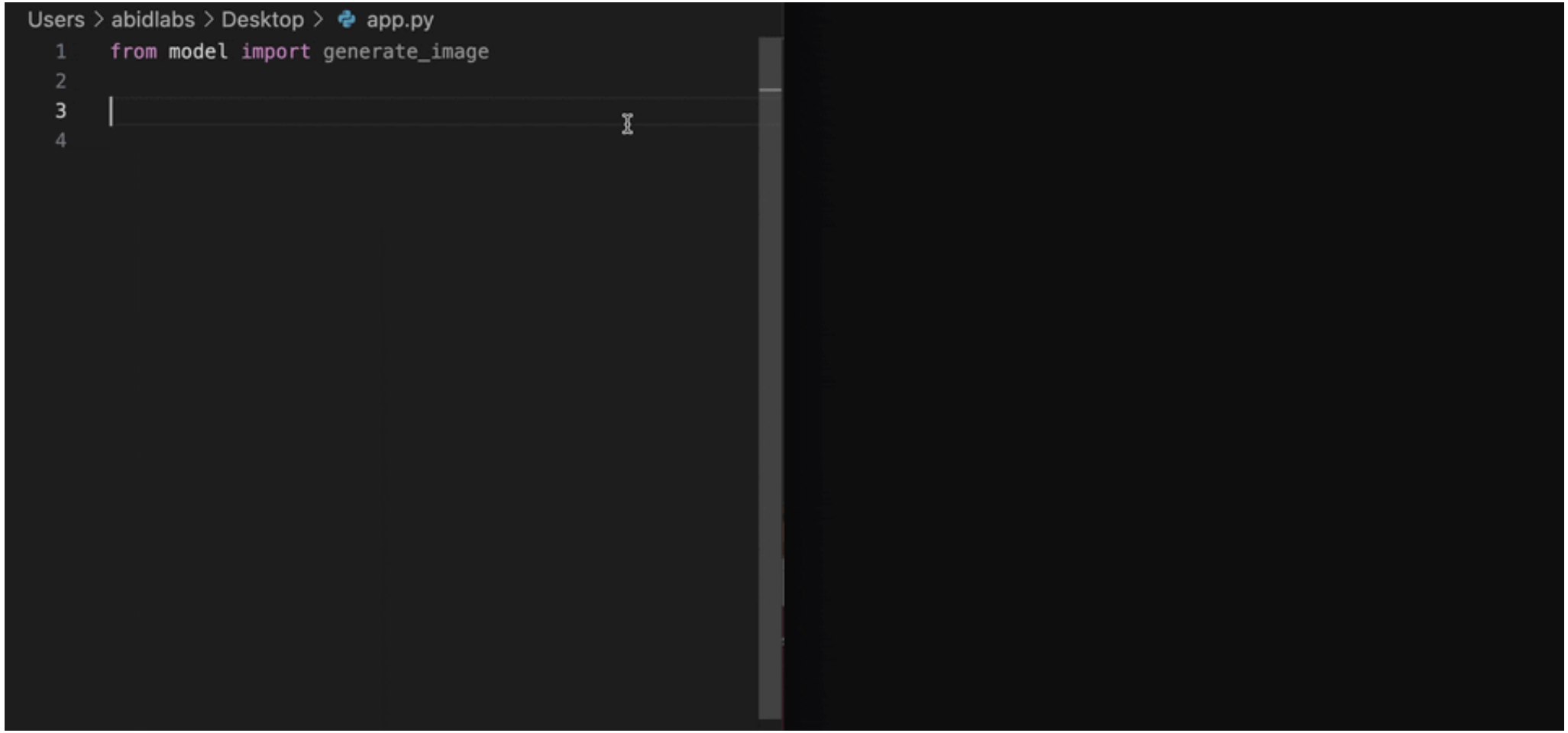
Introduction

❖ Getting Started

Ref: https://www.gradio.app

Introduction

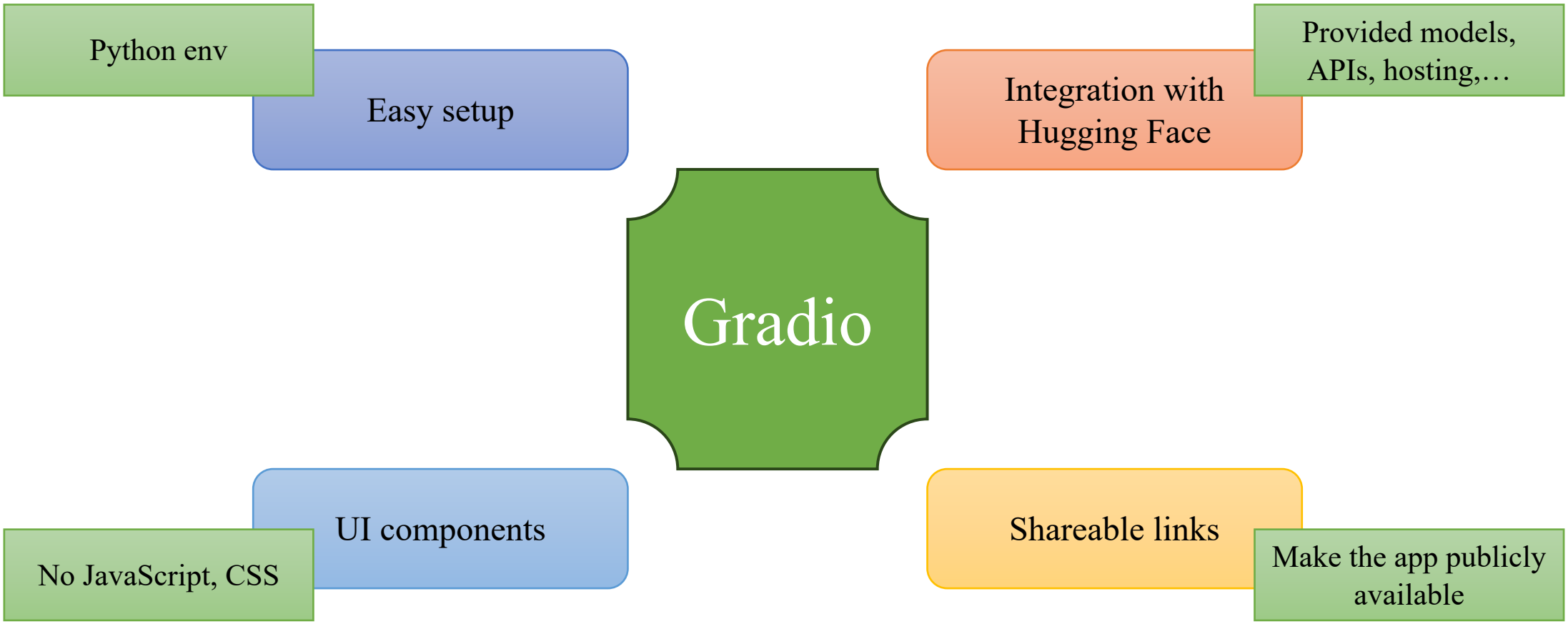
❖ Gradio Overview



Lightweight, open-source UI library for ML models

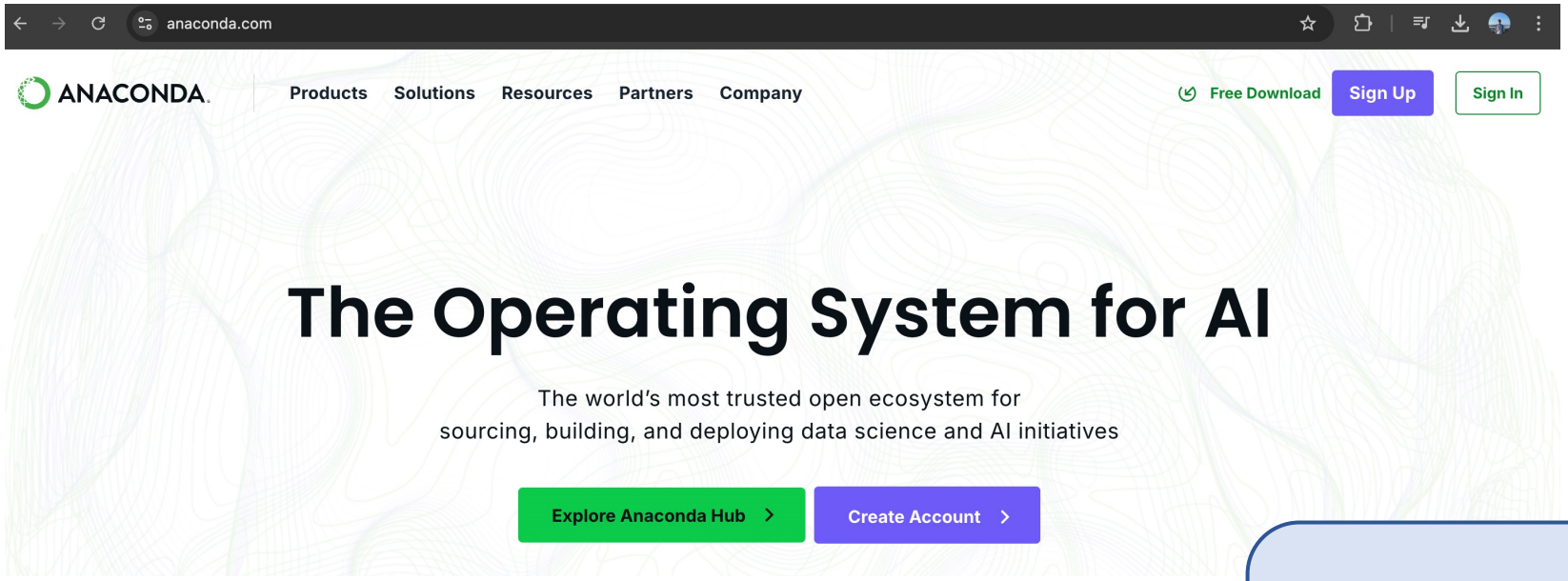
Introduction

❖ Key Features



Introduction

❖ Installation



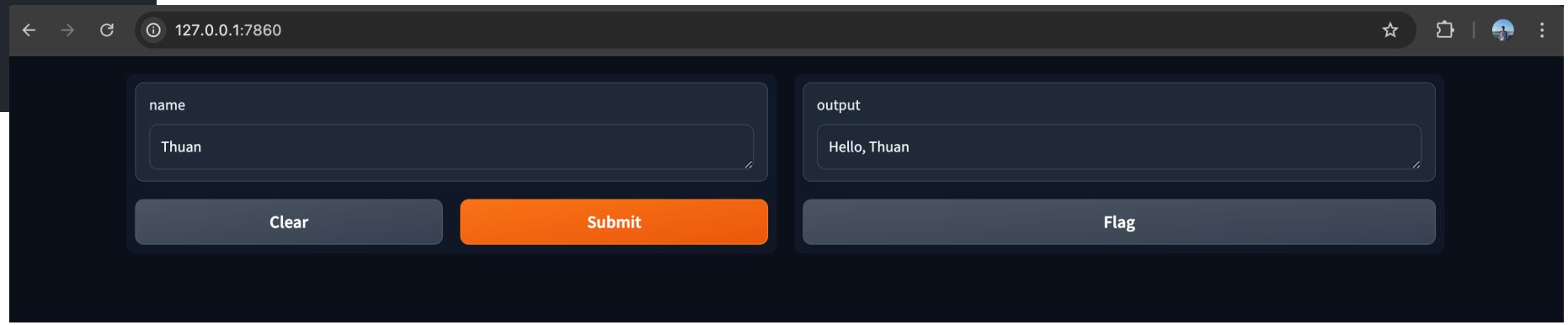

```
>> conda create –n gradio_env python=3.13.5
>> conda activate gradio _env
>> pip install --upgrade gradio
```

Introduction

❖ Building Your First Demo

```python
import gradio as gr

def greet(name):
    return "Hello, " + name

demo = gr.Interface(
    fn=greet,
    inputs=["text"],
    outputs=["text"],
)

demo.launch()
```

```
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()`.
```



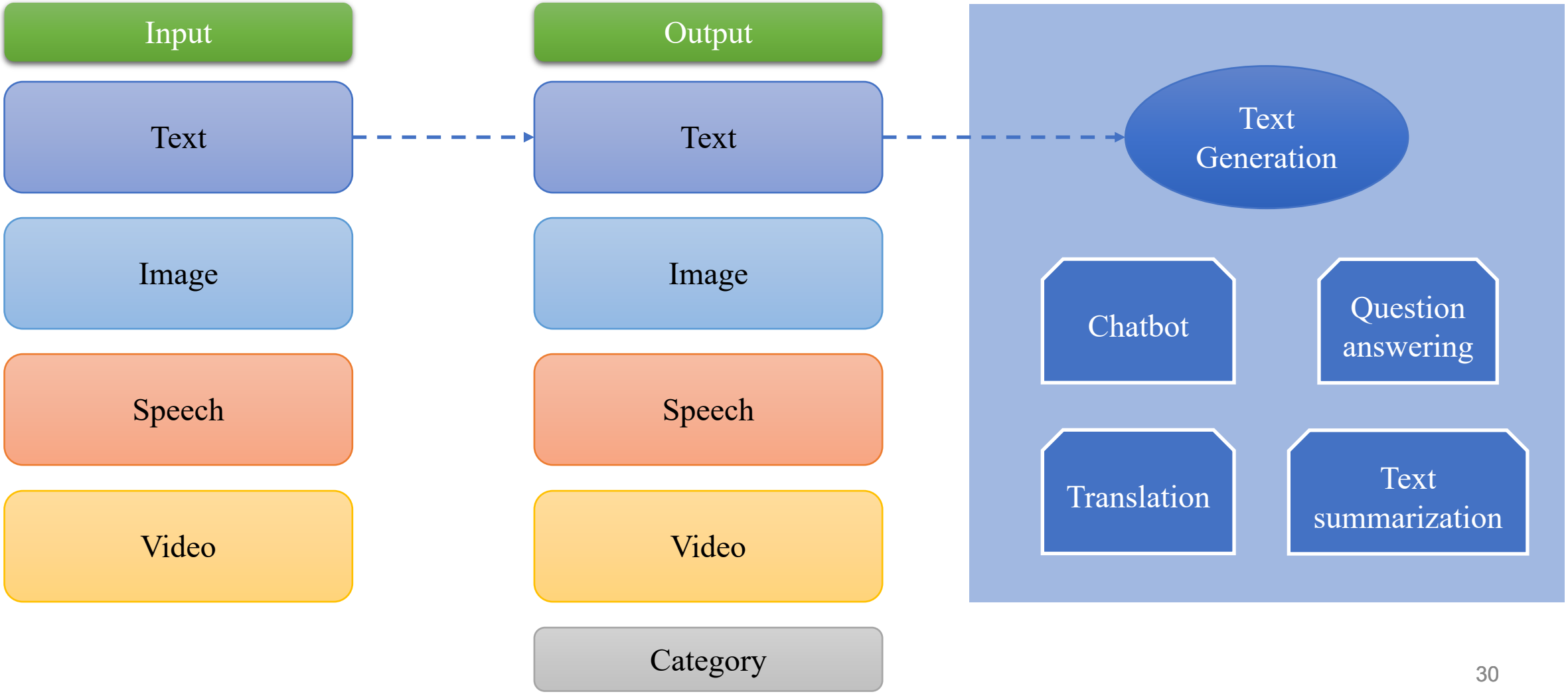
Introduction

❖ Input/Output of ML application

Input	Output
Text	Text
Image	Image
Speech	Speech
Video	Video

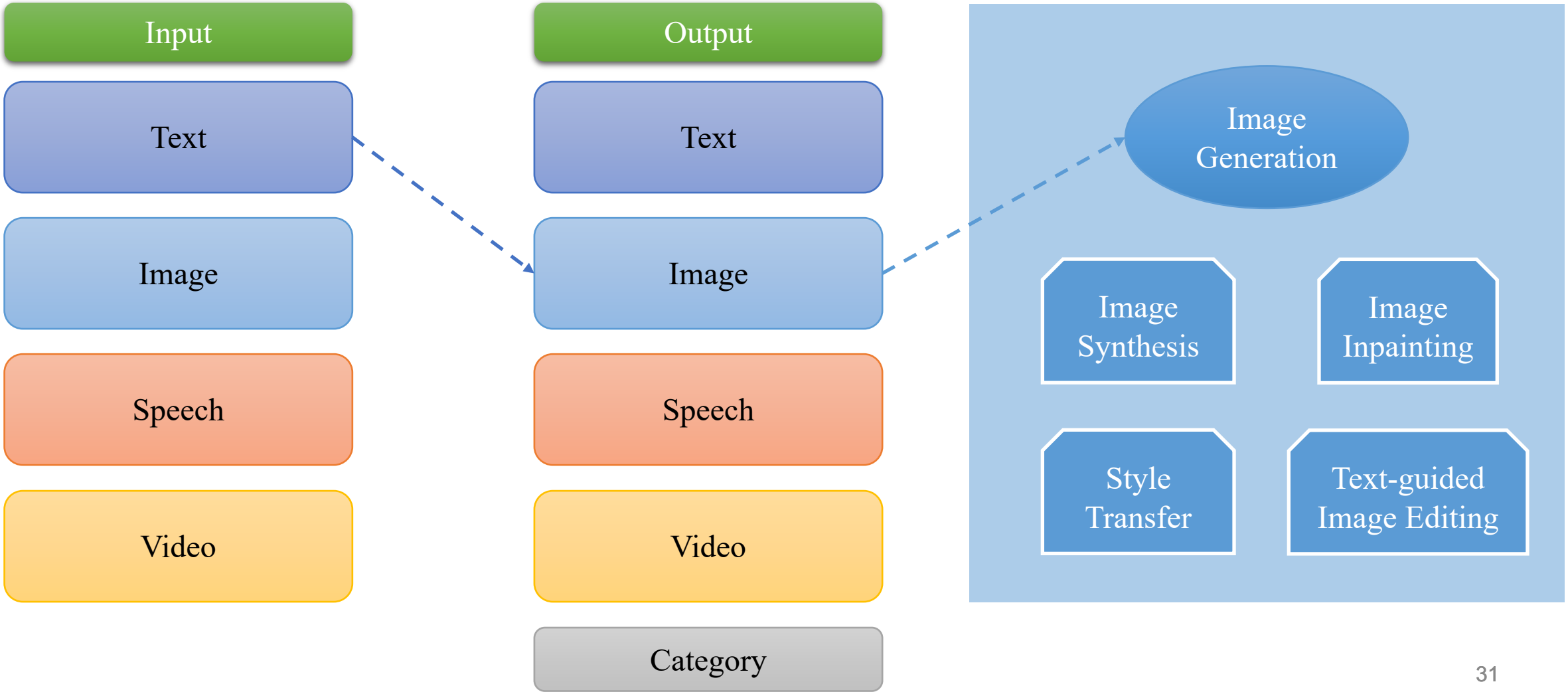
Introduction

❖ Input/Output of ML application



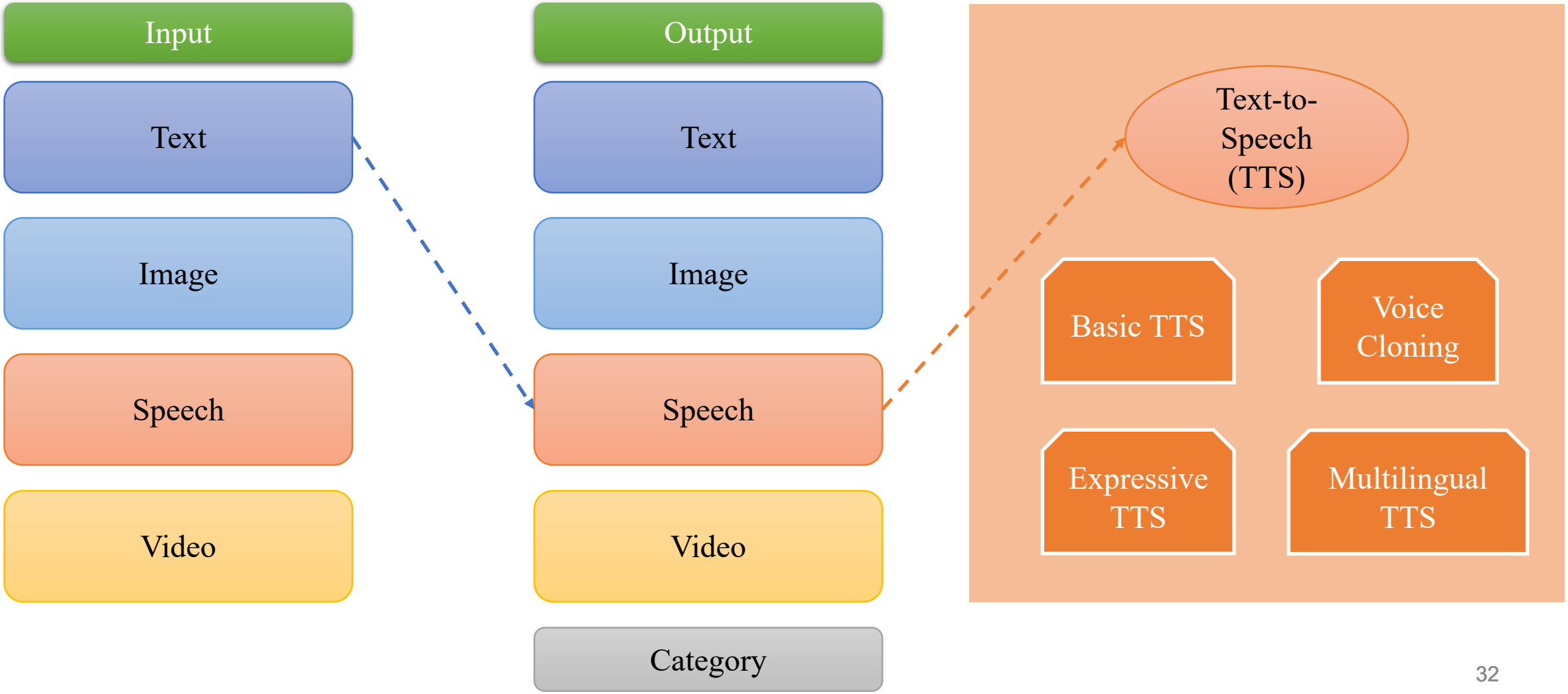
Introduction

❖ Input/Output of ML application



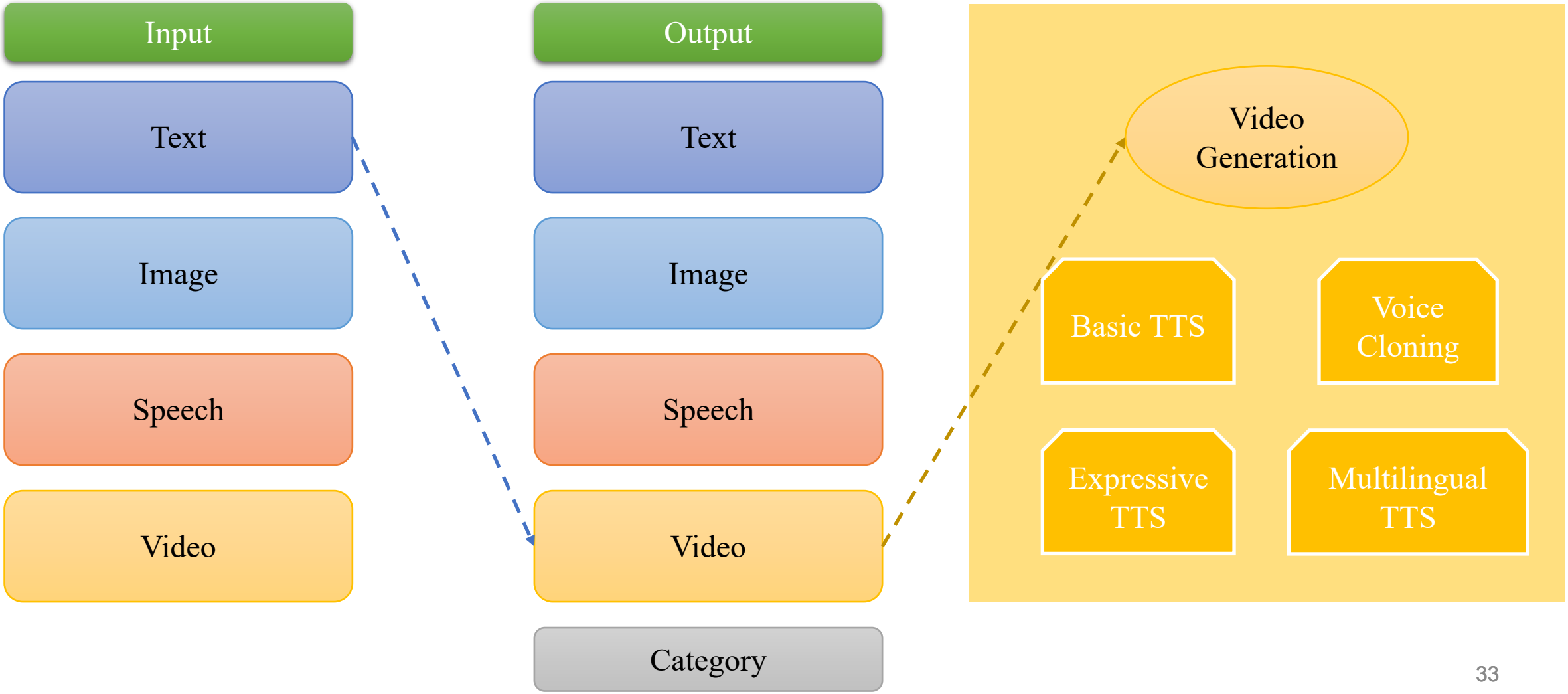
Introduction

❖ Input/Output of ML application



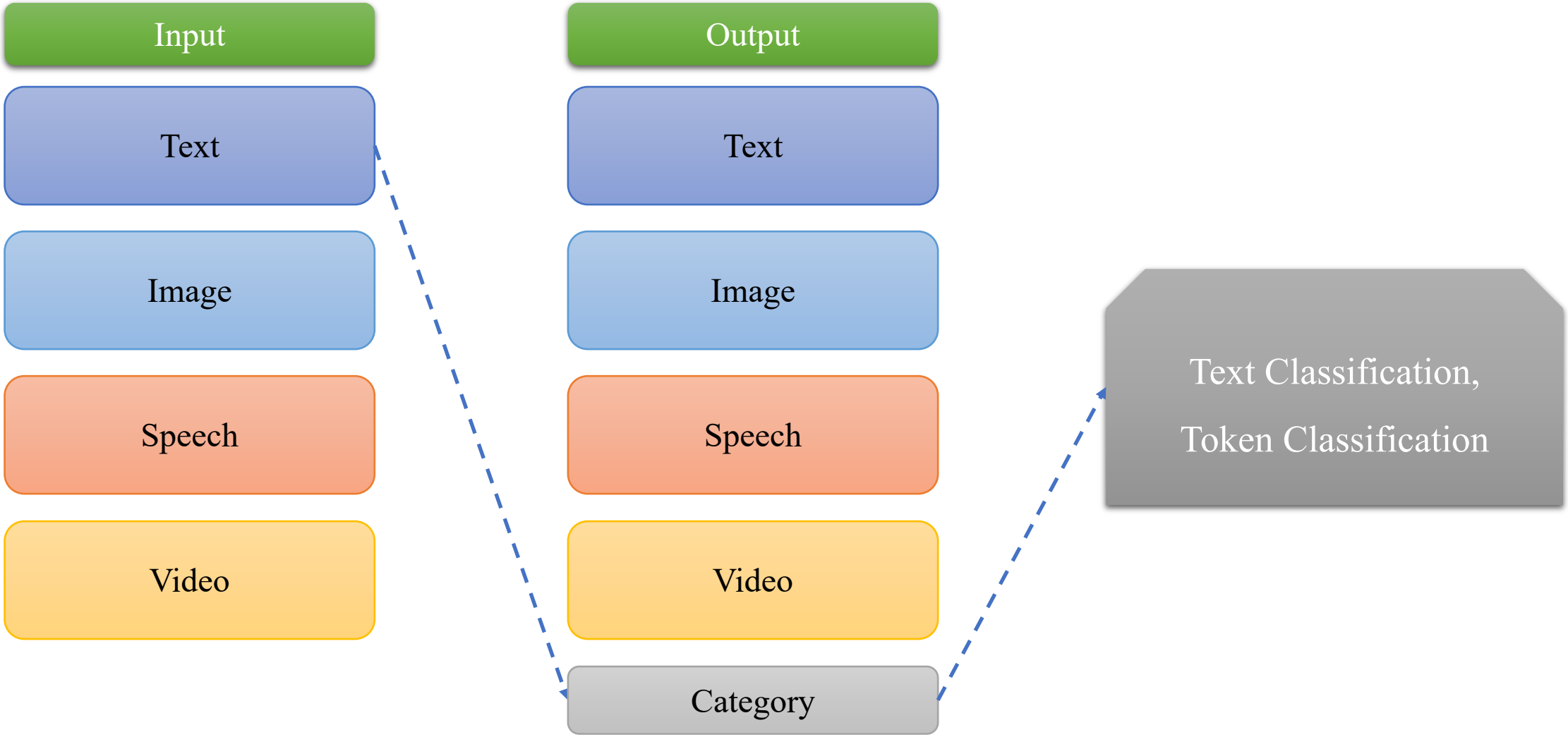
Introduction

❖ Input/Output of ML application



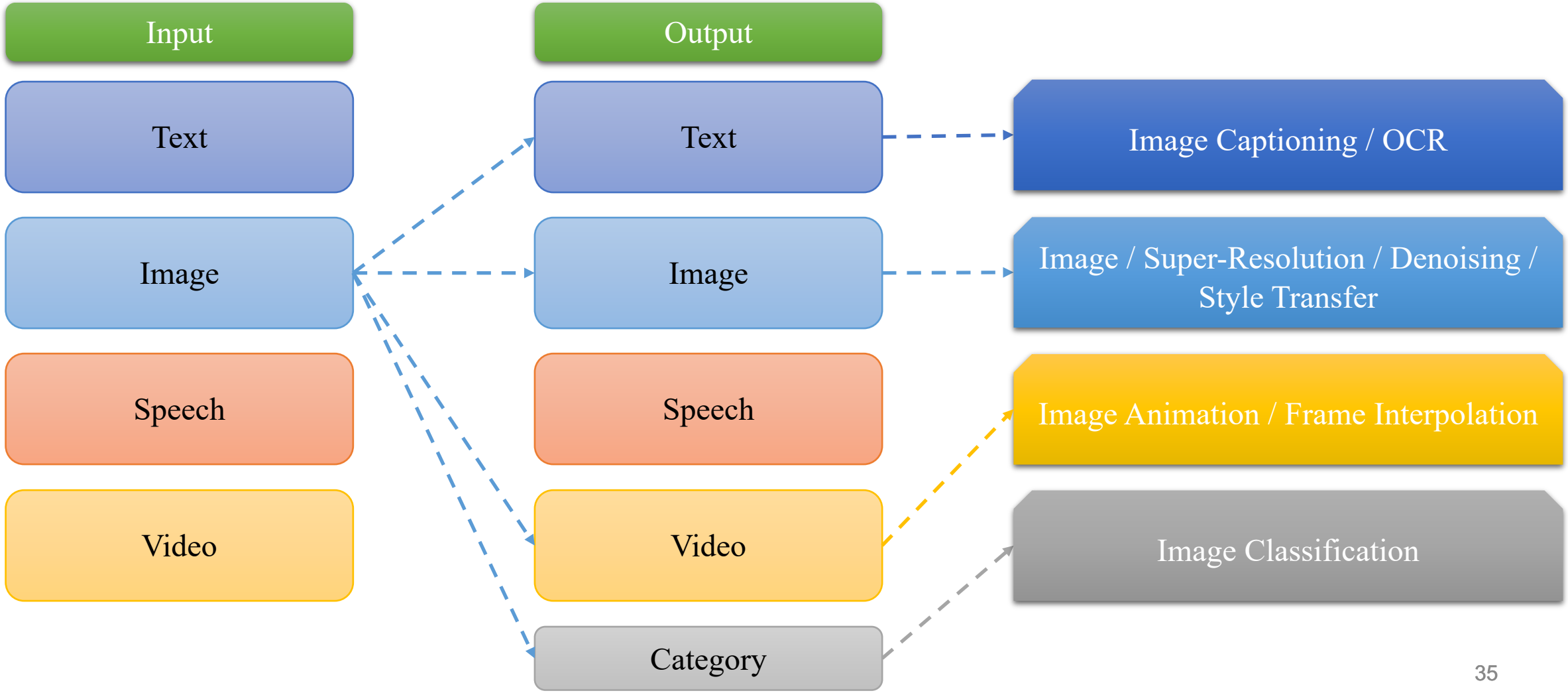
Introduction

❖ Input/Output of ML application



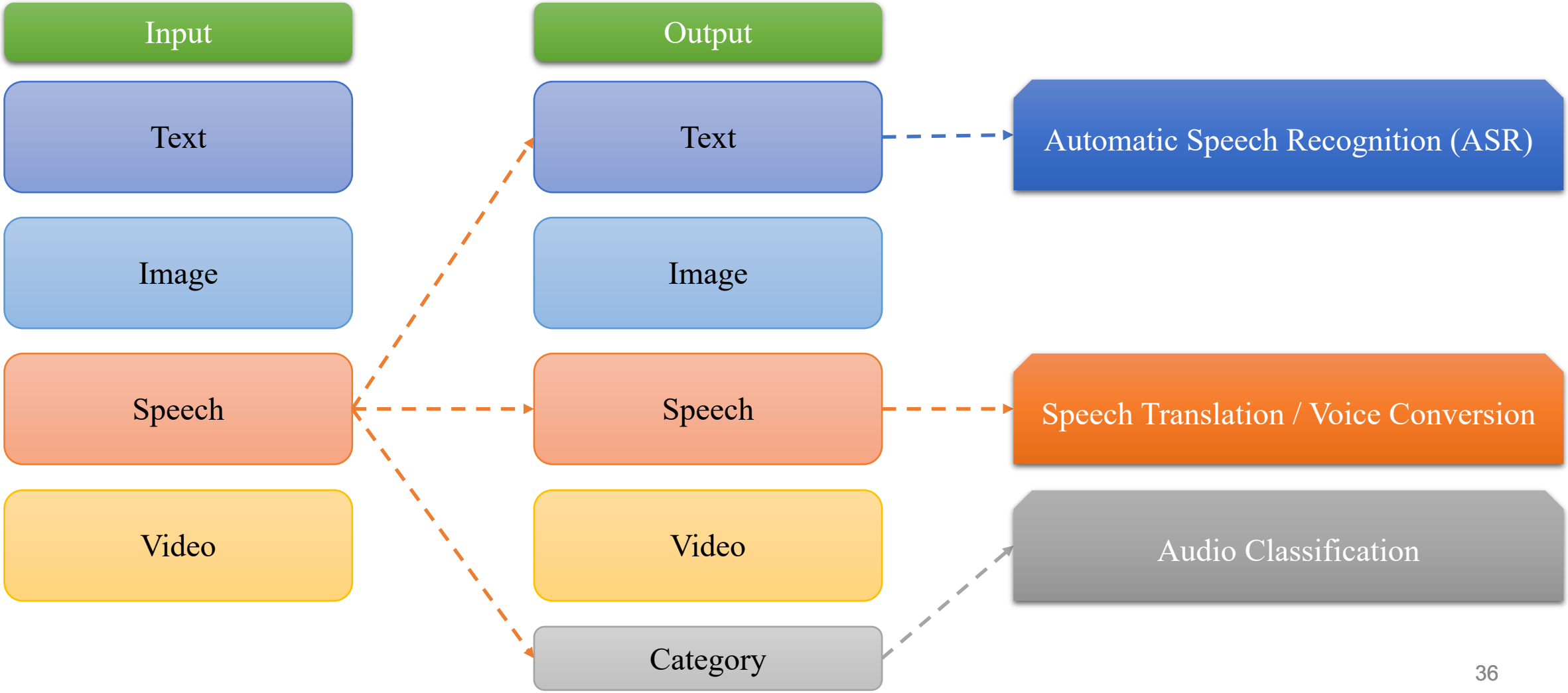
Introduction

❖ Input/Output of ML application



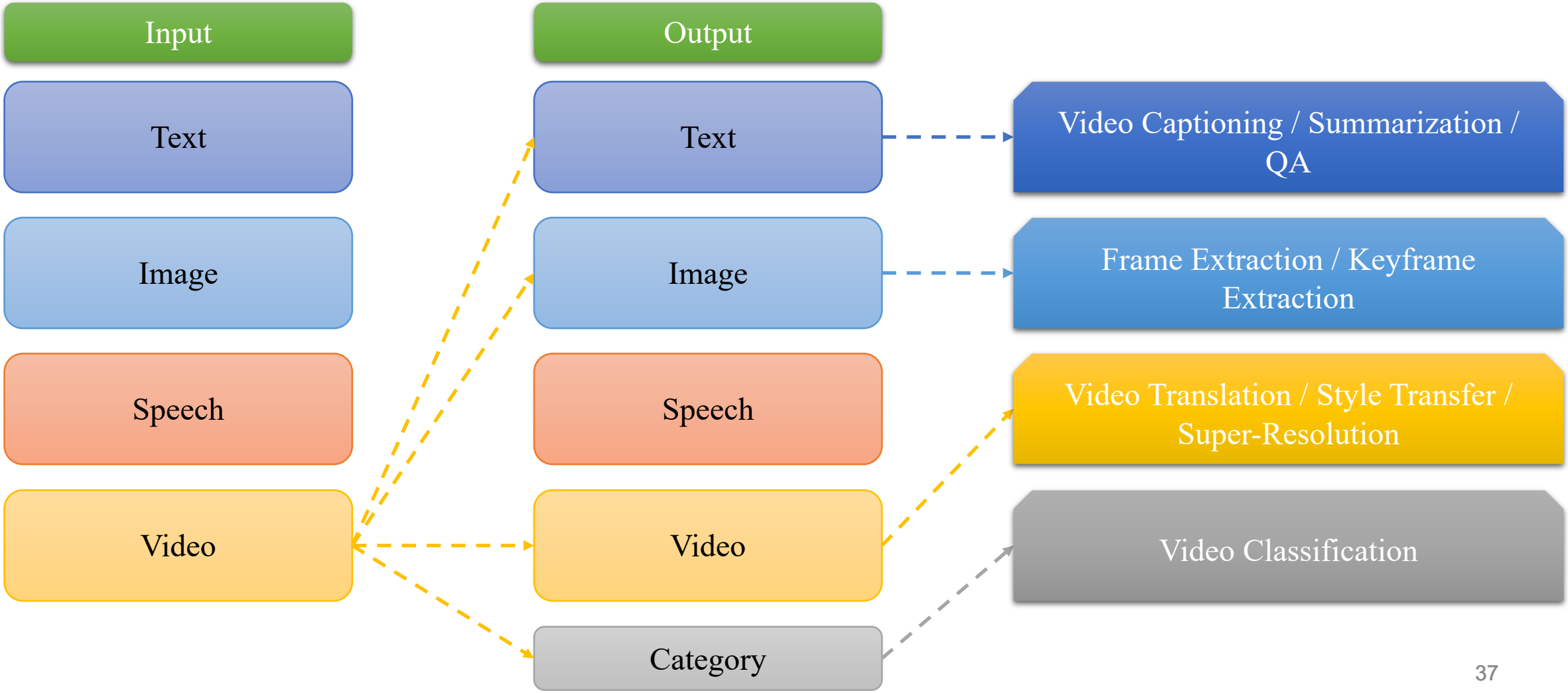
Introduction

❖ Input/Output of ML application



Introduction

❖ Input/Output of ML application

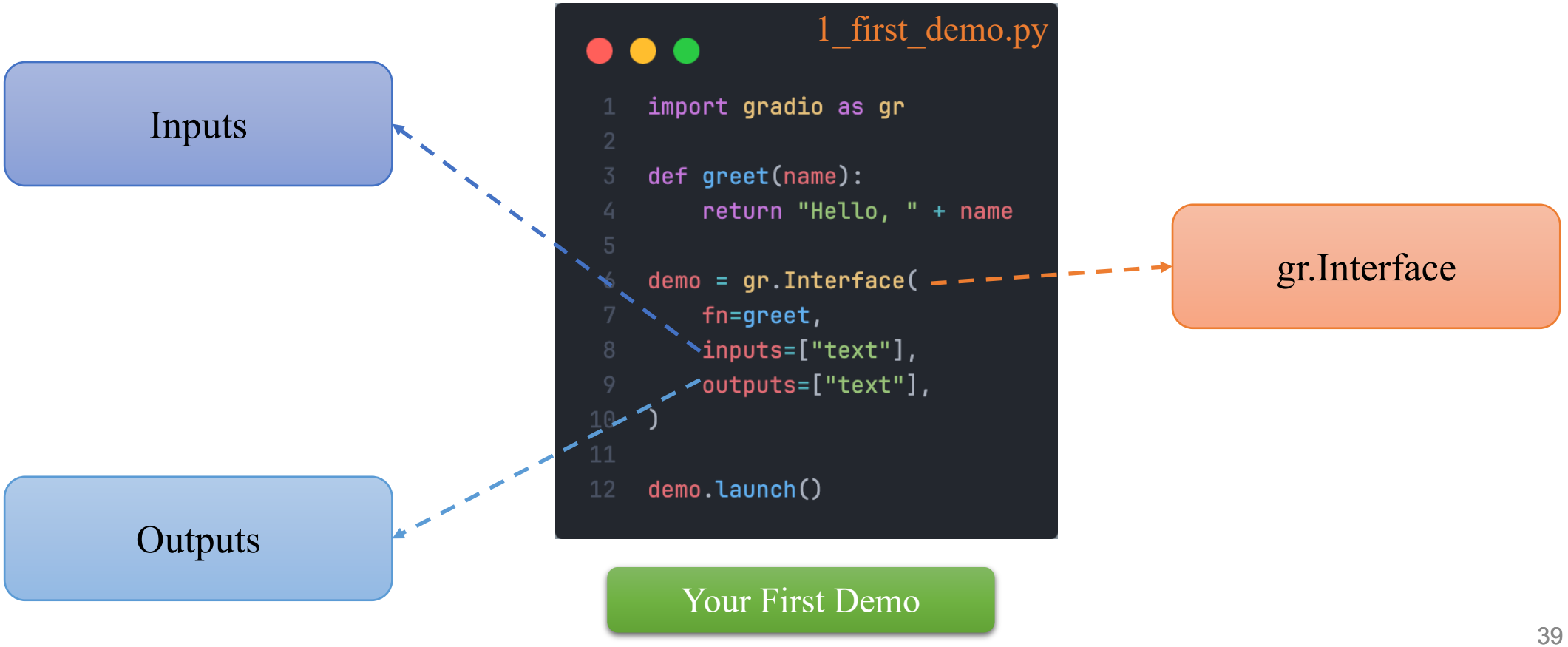


Gradio Basics

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

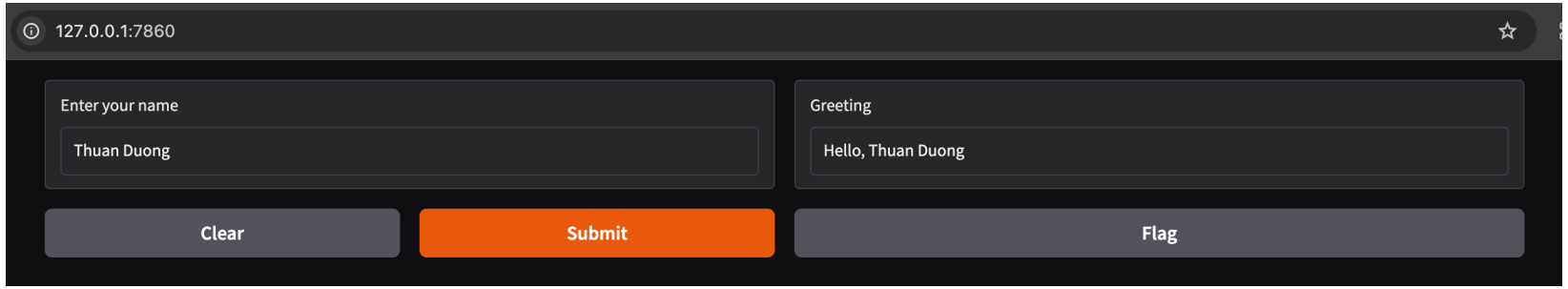
❖ Inputs: Text

Textbox

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

2_text_inputs.py

```python
import gradio as gr

def greet(name):
    return "Hello, " + name

demo = gr.Interface(
    fn=greet,
    inputs=gr.Textbox(label="Enter your name",
                      placeholder="Type your name here..."),
    outputs=gr.Textbox(label="Greeting"),
)

demo.launch()
```



https://www.gradio.app/docs/gradio/textbox

Gradio Basics

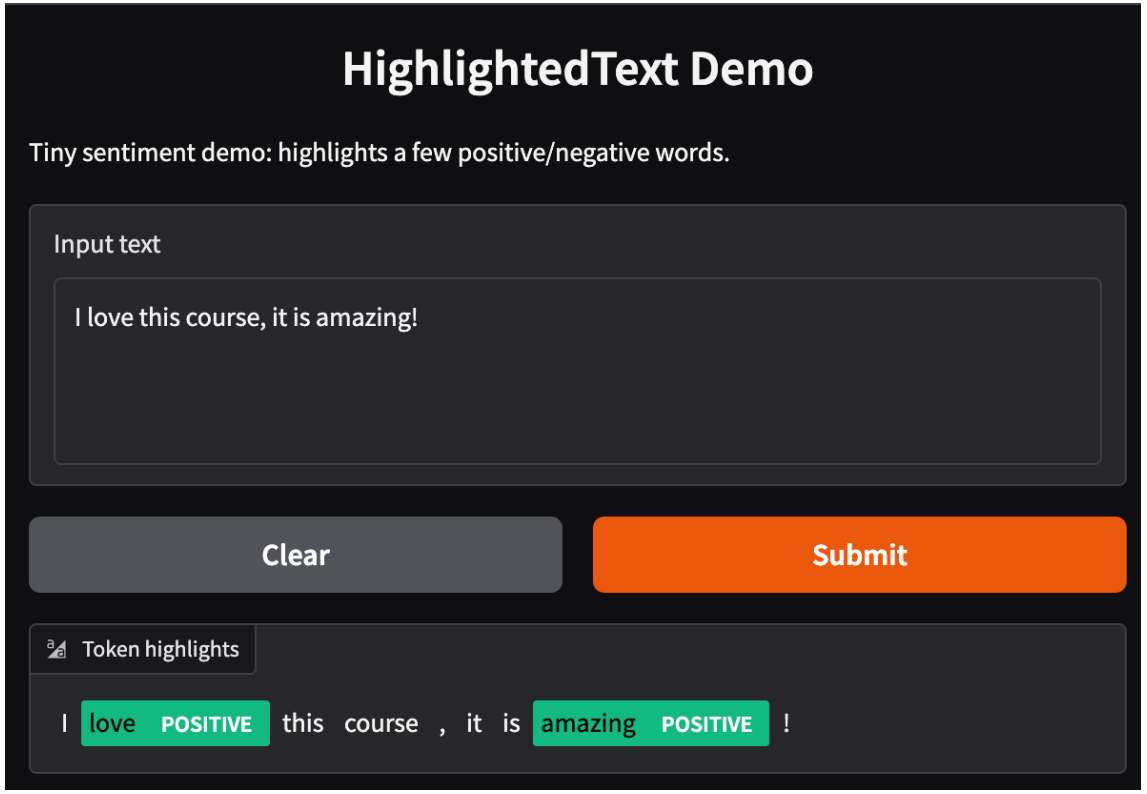
❖ Inputs: Text

HighlightedText

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

POS

NER



8_hightlight_text_input.py

```python
demo = gr.Interface(
    fn=highlight_sentiment,
    inputs=gr.Textbox(
        label="Input text",
        placeholder="I love this amazing course.",
        lines=4,
    ),
    outputs=gr.HighlightedText(
        label="Token highlights",
        color_map={
            "positive": "#10b981",
            "negative": "#ef4444",
        },
    ),
    title="HighlightedText Demo",
    description="Tiny sentiment demo: " \
                "highlights a few positive/negative words.",
    examples=[
        "I love this course, it is amazing!",
        "The movie was good but the ending was terrible.",
        "What a wonderful day, I'm so happy.",
        "The service was bad and the food was awful.",
    ],
)
demo.launch()
```

https://www.gradio.app/docs/gradio/highlightedtext

Gradio Basics

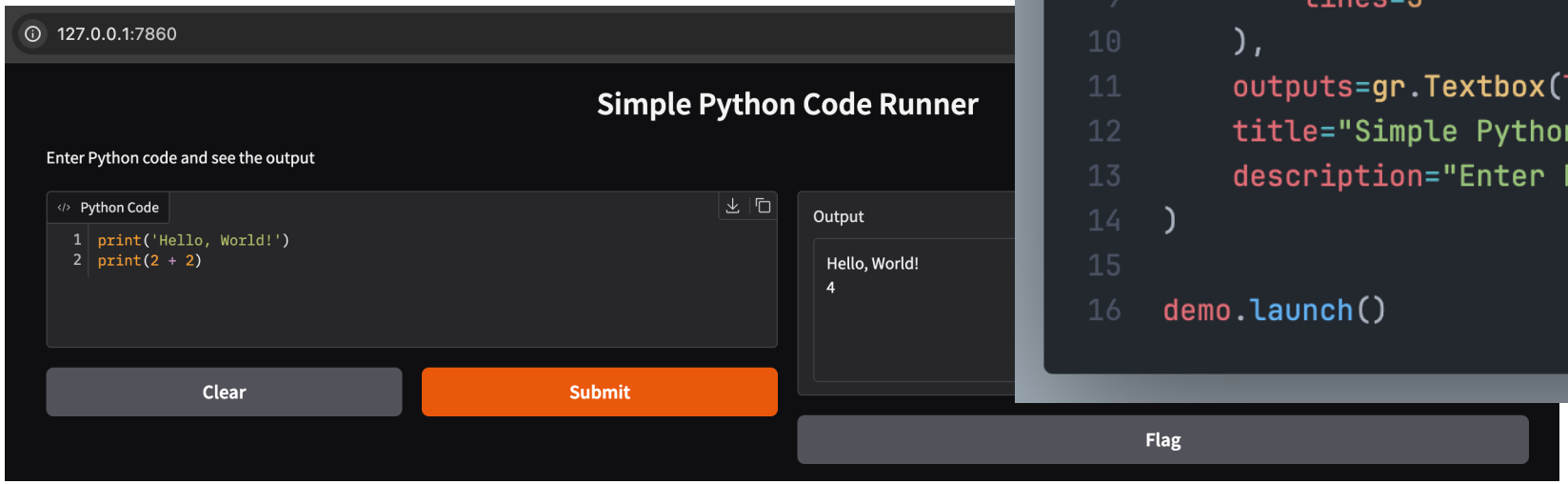
❖ Inputs: Text

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

```python
import gradio as gr

demo = gr.Interface(
    fn=execute_python,
    inputs=gr.Code(
        label="Python Code",
        language="python",
        value="print('Hello, World!')\nprint(2 + 2)",
        lines=5
    ),
    outputs=gr.Textbox(label="Output", lines=5),
    title="Simple Python Code Runner",
    description="Enter Python code and see the output"
)

demo.launch()
```



https://www.gradio.app/docs/gradio/code

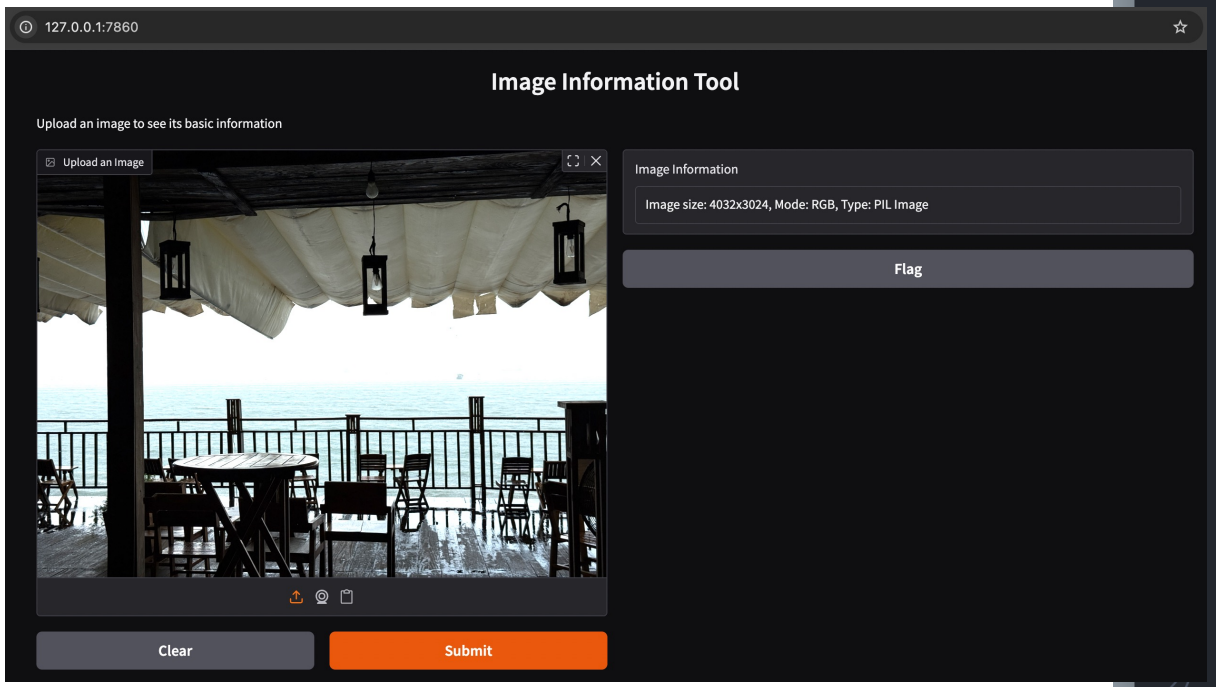
AI VIETNAM
All-in-One Course
(TA Session)

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

```python
import gradio as gr
import numpy as np
from PIL import Image

def get_image_info(image):
    """Get basic information about an image"""
    if image is None:
        return "No image provided"

    # Handle different image types
    if isinstance(image, np.ndarray):
        height, width = image.shape[:2]
        channels = image.shape[2] if len(image.shape) == 3 else 1
        return f"Image size: {width}x{height}, Channels: {channels}, Type: NumPy array"
    elif isinstance(image, Image.Image):
        width, height = image.size
        return f"Image size: {width}x{height}, Mode: {image.mode}, Type: PIL Image"
    else:
        return f"Image type: {type(image)}"

demo = gr.Interface(
    fn=get_image_info,
    inputs=gr.Image(label="Upload an Image", type="pil"),
    outputs=gr.Textbox(label="Image Information"),
    title="Image Information Tool",
    description="Upload an image to see its basic information"
)

demo.launch()
```

https://www.gradio.app/docs/gradio/image

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

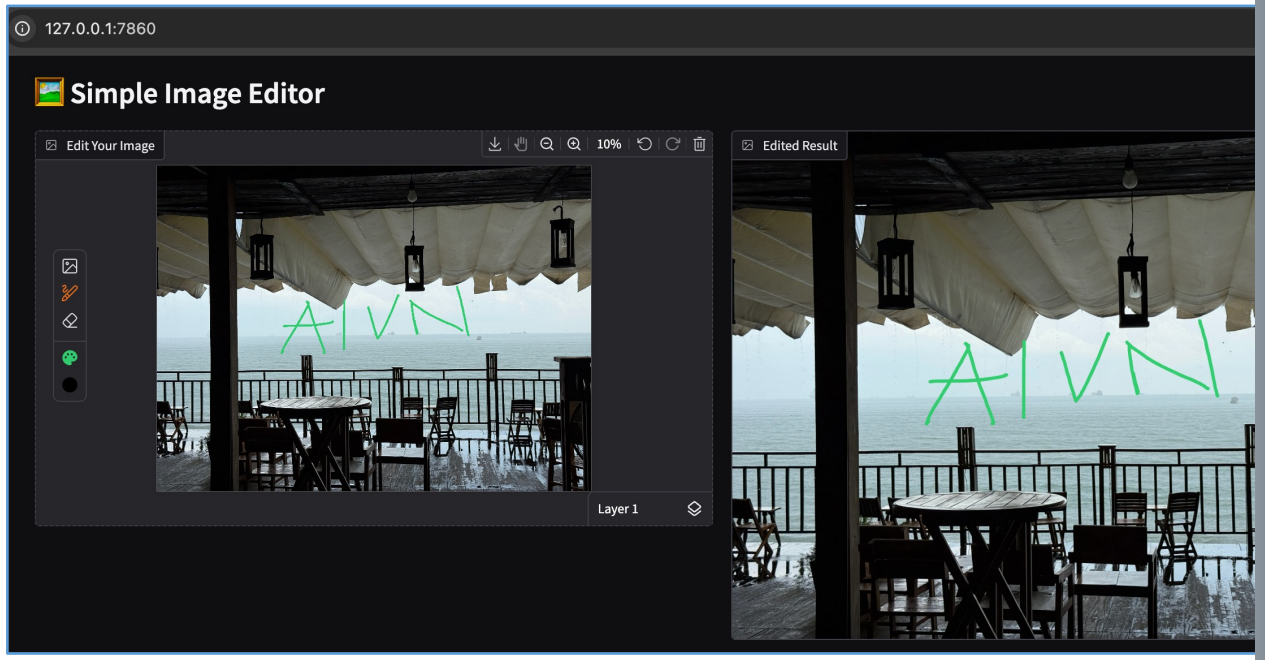
Gradio Basics

❖ 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

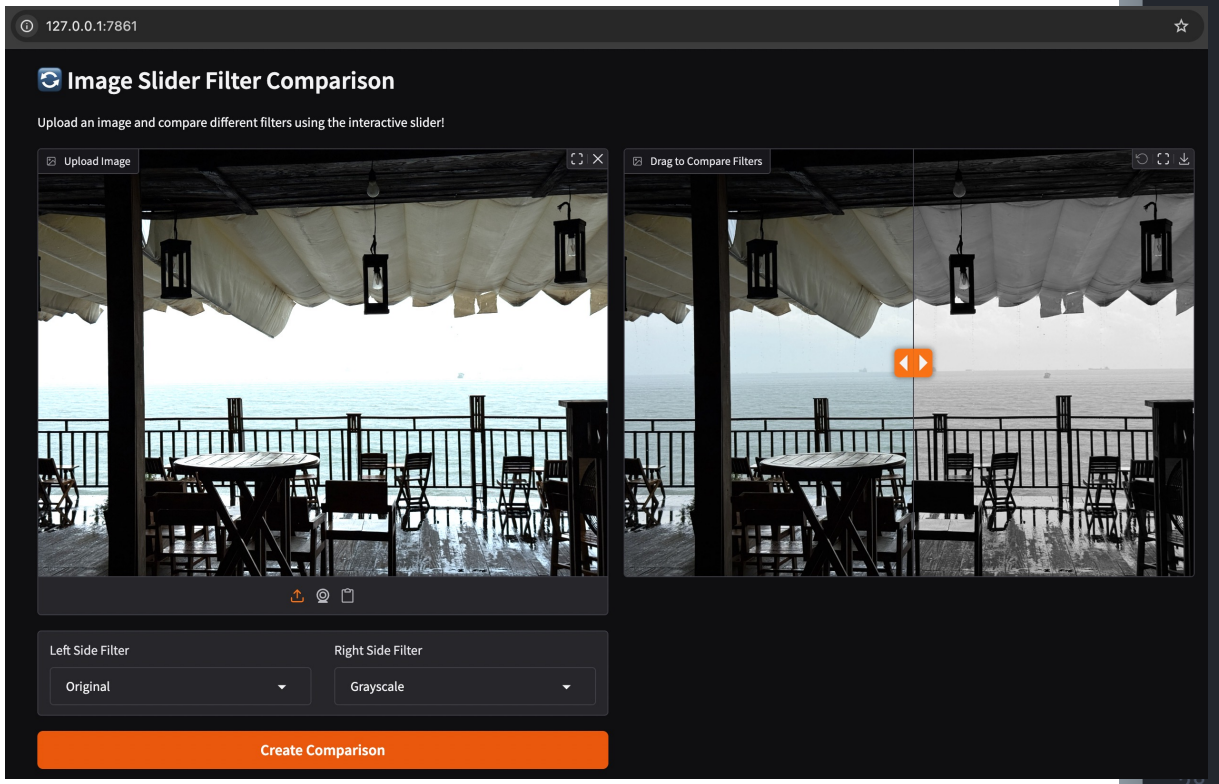
```python
import gradio as gr

def process_image(image_data):
    if image_data is None:
        return None
    return image_data.get('composite', None)

with gr.Blocks(title="Simple Image Editor") as demo:
    gr.Markdown("# 🖼️ Simple Image Editor")

    with gr.Row():
        image_editor = gr.ImageEditor(
            label="Edit Your Image",
            type="pil",
            sources=["upload", "webcam"],  # Allow upload and webcam
            transforms=["crop"],           # Enable cropping
            brush=gr.Brush(default_size=10),  # Drawing brush
            eraser=gr.Eraser(default_size=10) # Eraser tool
        )
        output_image = gr.Image(label="Edited Result", type="pil")

    image_editor.change(
        fn=process_image,
        inputs=[image_editor],
        outputs=[output_image]
    )

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

```python
choices = ["Original", "Grayscale", "Blur", "Sharpen",
           "Brightness", "Contrast", "Sepia", "Vintage"]

with gr.Blocks(title="Image Slider Demo") as demo:
    gr.Markdown("# 🔄 Image Slider Filter Comparison")

    with gr.Row():
        with gr.Column():
            input_image = gr.Image(label="Upload Image", type="pil")
            with gr.Row():
                filter1_dropdown = gr.Dropdown(
                    choices=choices,
                    value="Original",
                    label="Left Side Filter"
                )
                filter2_dropdown = gr.Dropdown(
                    choices=choices,
                    value="Grayscale",
                    label="Right Side Filter"
                )

            compare_btn = gr.Button("Create Comparison", variant="primary")

        with gr.Column():
            comparison_slider = gr.ImageSlider(
                label="Drag to Compare Filters",
                elem_id="comparison",
            )
```

Gradio Basics

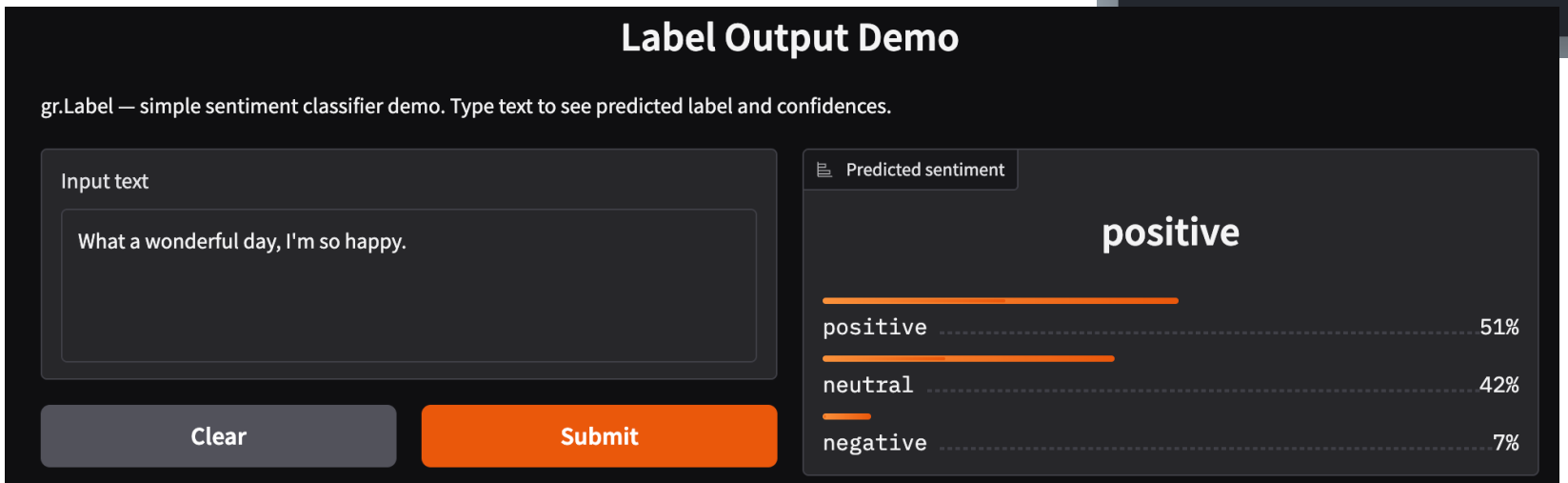
❖ Output: Label

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

```python
demo = gr.Interface(
    fn=classify_sentiment,
    inputs=gr.Textbox(
        label="Input text",
        placeholder="e.g. I love this course, it is amazing!",
        lines=4,
    ),
    outputs=gr.Label(label="Predicted sentiment"),
    title="Label Output Demo",
    description="gr.Label — simple sentiment classifier demo. " \
                "Type text to see predicted label and confidences.",
    examples=[
        "I love this amazing course but the pacing is bad.",
        "The movie was good but the ending was terrible.",
        "What a wonderful day, I'm so happy.",
        "The service was bad and the food was awful.",
    ],
)
demo.launch()
```



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

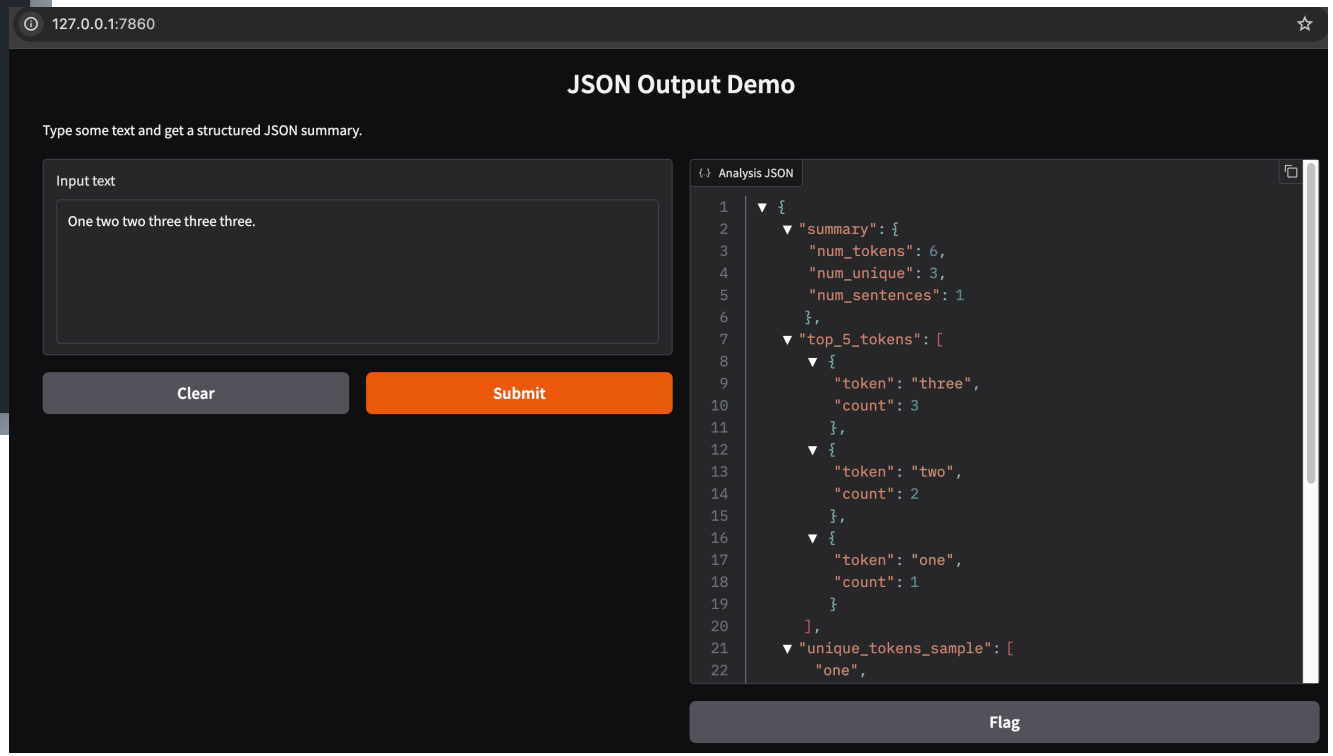
Gradio Basics

❖ Output: JSON

9_json_output.py

```python
demo = gr.Interface(
    fn=analyze,
    inputs=gr.Textbox(
        label="Input text",
        placeholder="e.g. Gradio is great for prototyping ML apps!",
        lines=6,
    ),
    outputs=gr.JSON(label="Analysis JSON"),
    title="JSON Output Demo",
    description="Type some text and get a structured JSON summary.",
    examples=[
        "Hello world. Hello Gradio. This is a simple JSON output demo.",
        "One two two three three three.",
        "Gradio makes it easy to build demos. JSON shows structured data.",
    ],
)

demo.launch()
```

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



https://www.gradio.app/docs/gradio/json

Gradio Basics

❖ Multi-Input / Multi-Output

10_multi_inputs_outputs.py

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

Gradio Basics

❖ Inputs: File

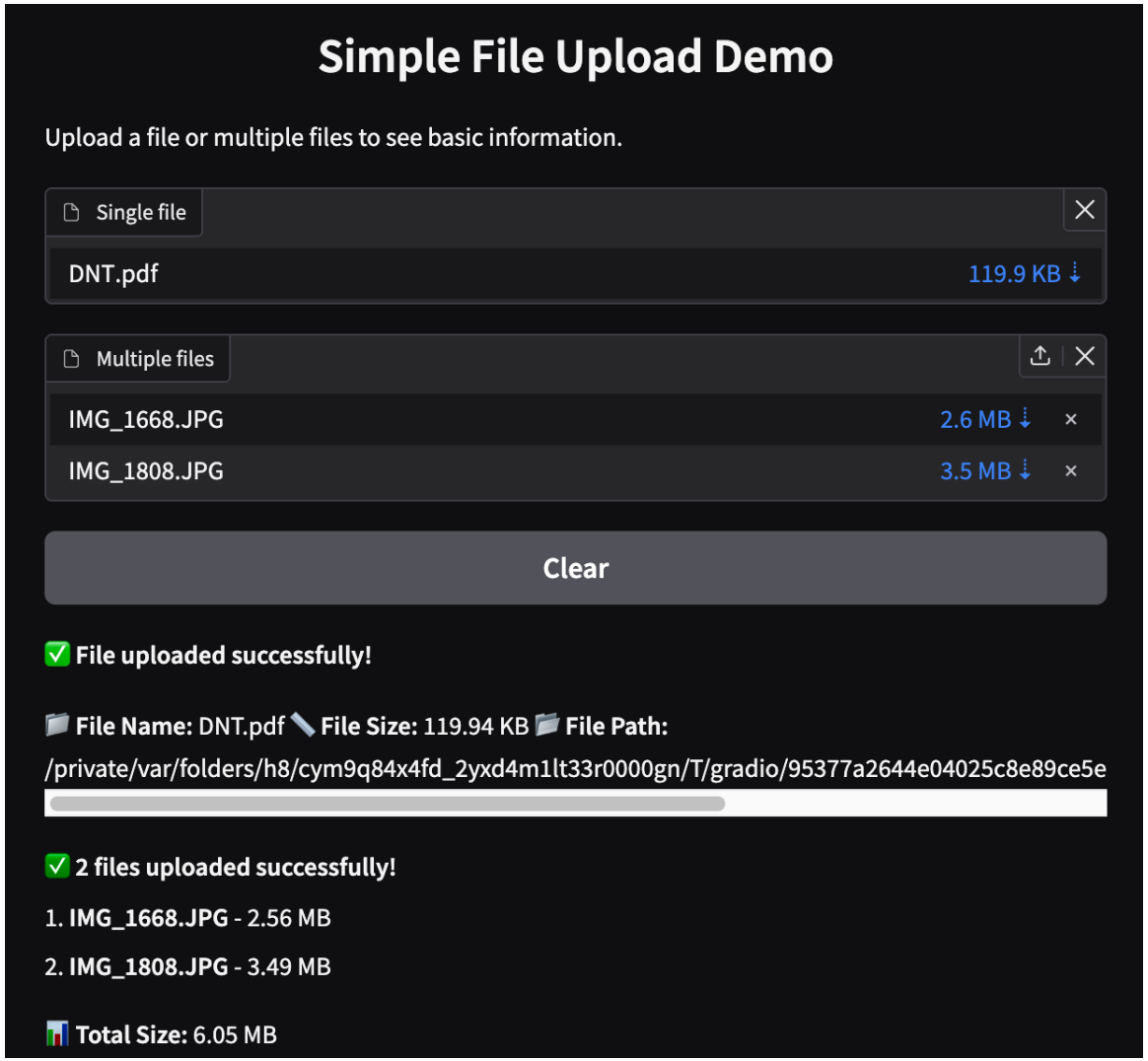
Simple & Multi Files

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

```python
demo = gr.Interface(
    fn=process_pair,
    inputs=[
        gr.File(label="Single file", file_count="single"),
        gr.File(label="Multiple files", file_count="multiple"),
    ],
    outputs=[
        gr.Markdown(label="Single file info"),
        gr.Markdown(label="Multiple files info"),
    ],
    title="Simple File Upload Demo",
    description="Upload a file or multiple files to see basic information.",
    live=True,
)

demo.launch()
```

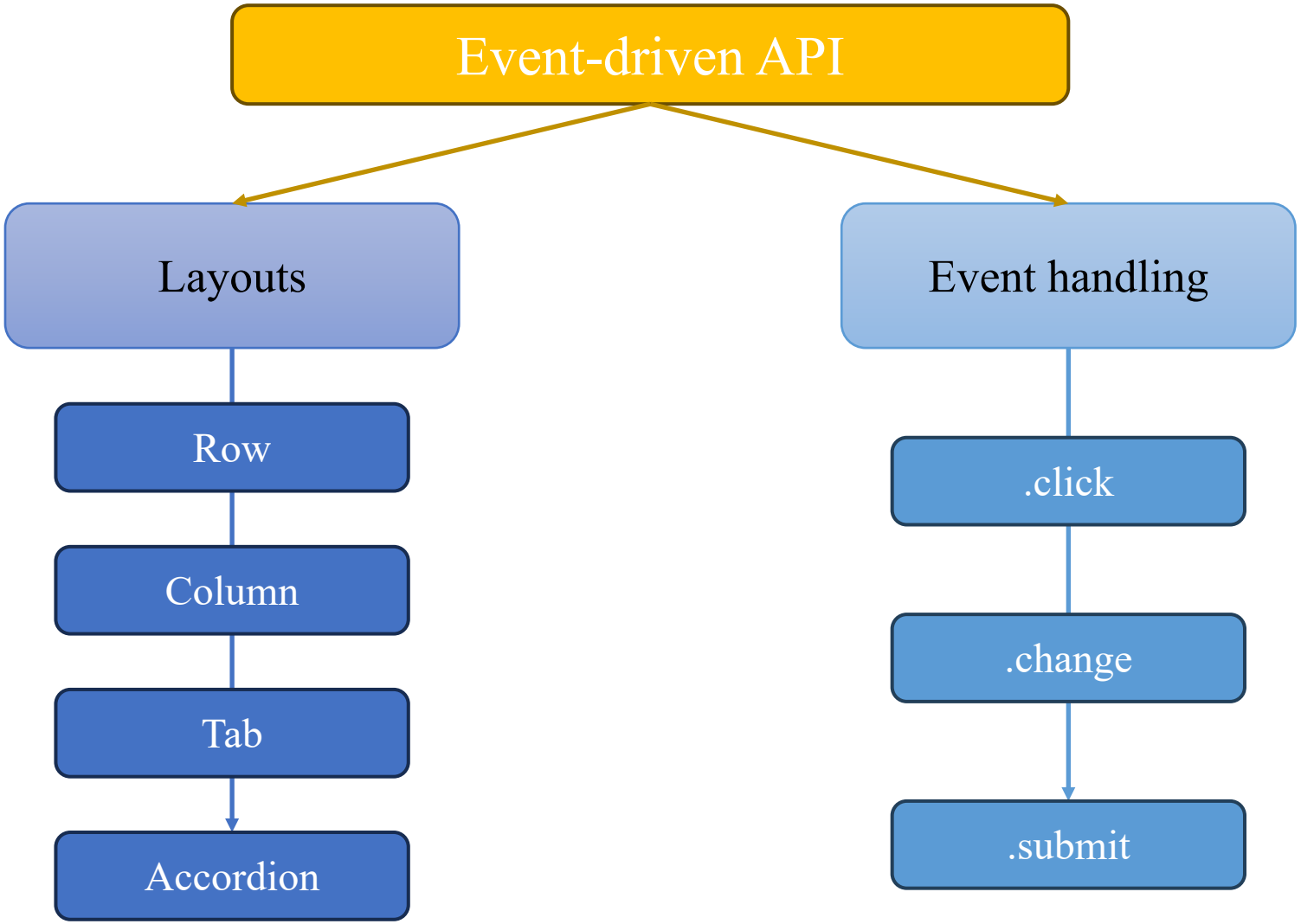


https://www.gradio.app/docs/gradio/file

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.

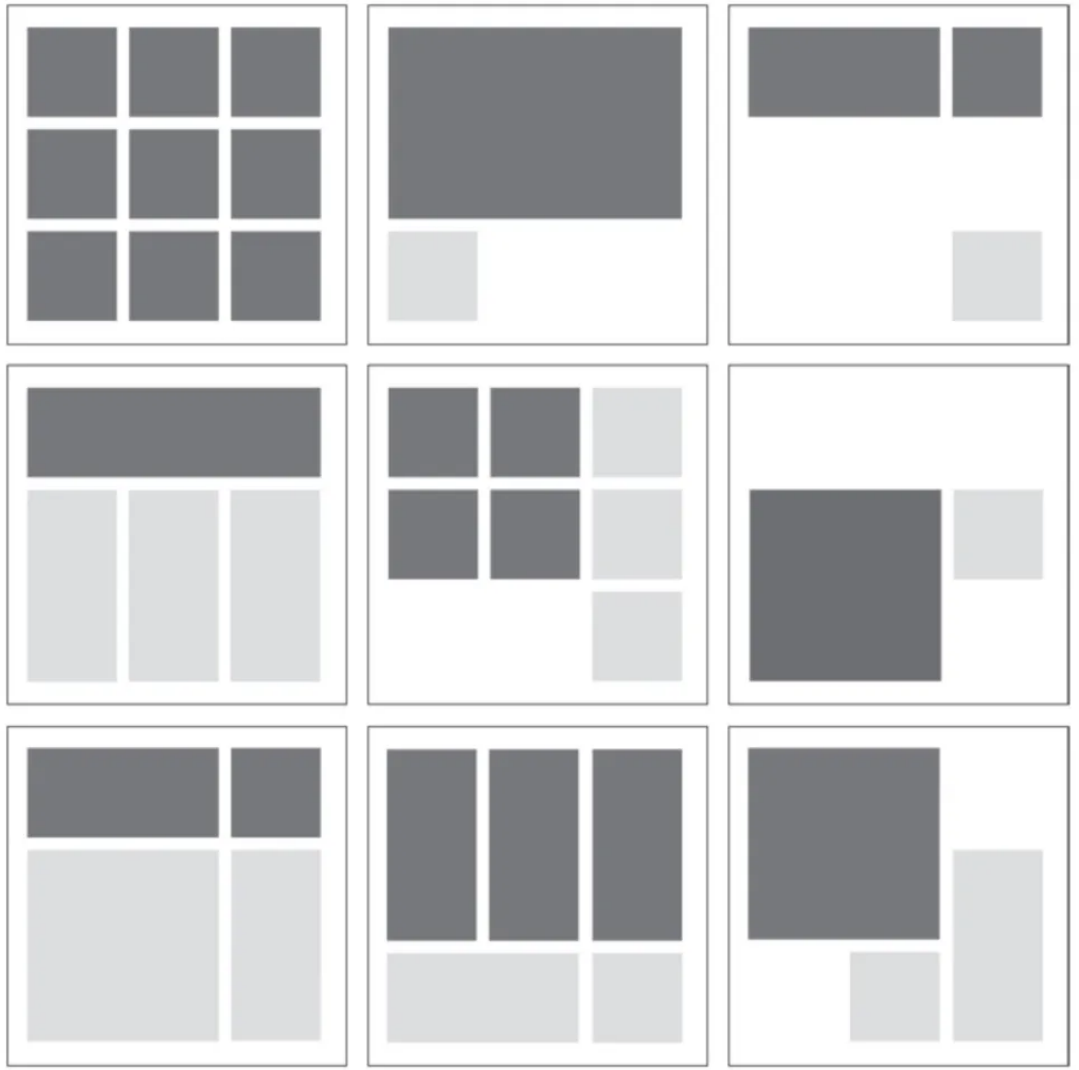
Feature	gr.Interface	gr.Interface
Ease of use	Very easy, minimal code	More verbose
Customization	Limited	Very flexible
Layout control	Layout control	Layout control
Best for	Simple model demos	Complex apps (chatbots, dashboards)
Events	Only function binding	Multiple event handlers

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.).



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

Blocks

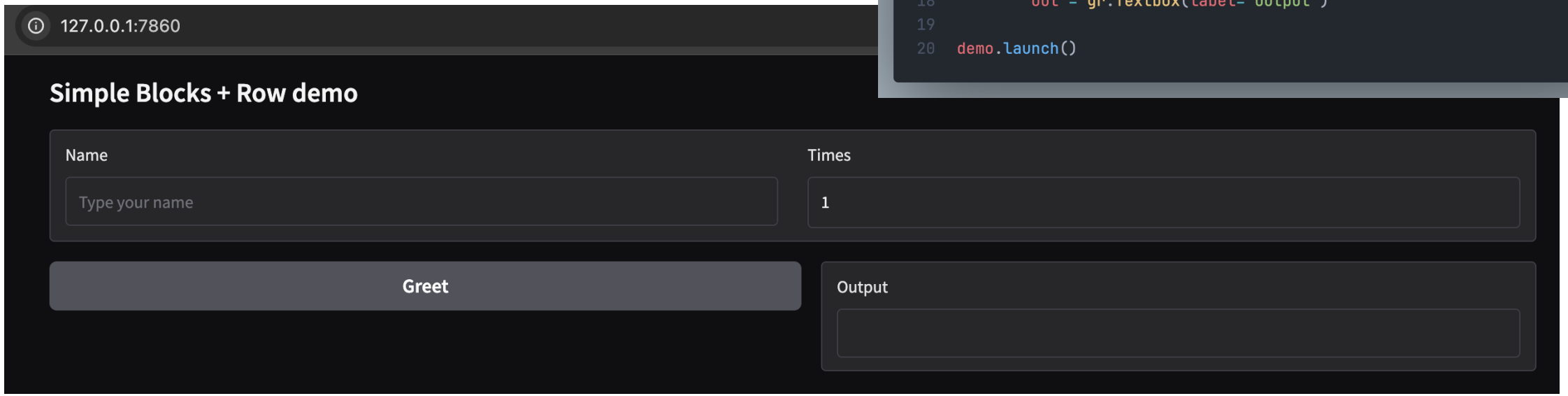
❖ Layout: Row

Arrange components side by side.

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

12_blocks_row.py

```python
import gradio as gr

def greet(name: str, times: float) → str:
    n = int(times or 1)
    name = name or "there"
    return "\n".join([f"Hello, {name}!" for _ in range(max(n, 1))])


with gr.Blocks(title="Blocks + Row: simplest") as demo:
    gr.Markdown("## Simple Blocks + Row demo")

    with gr.Row():
        name = gr.Textbox(label="Name", placeholder="Type your name")
        times = gr.Number(value=1, label="Times")

    with gr.Row():
        btn = gr.Button("Greet")
        out = gr.Textbox(label="Output")

demo.launch()
```



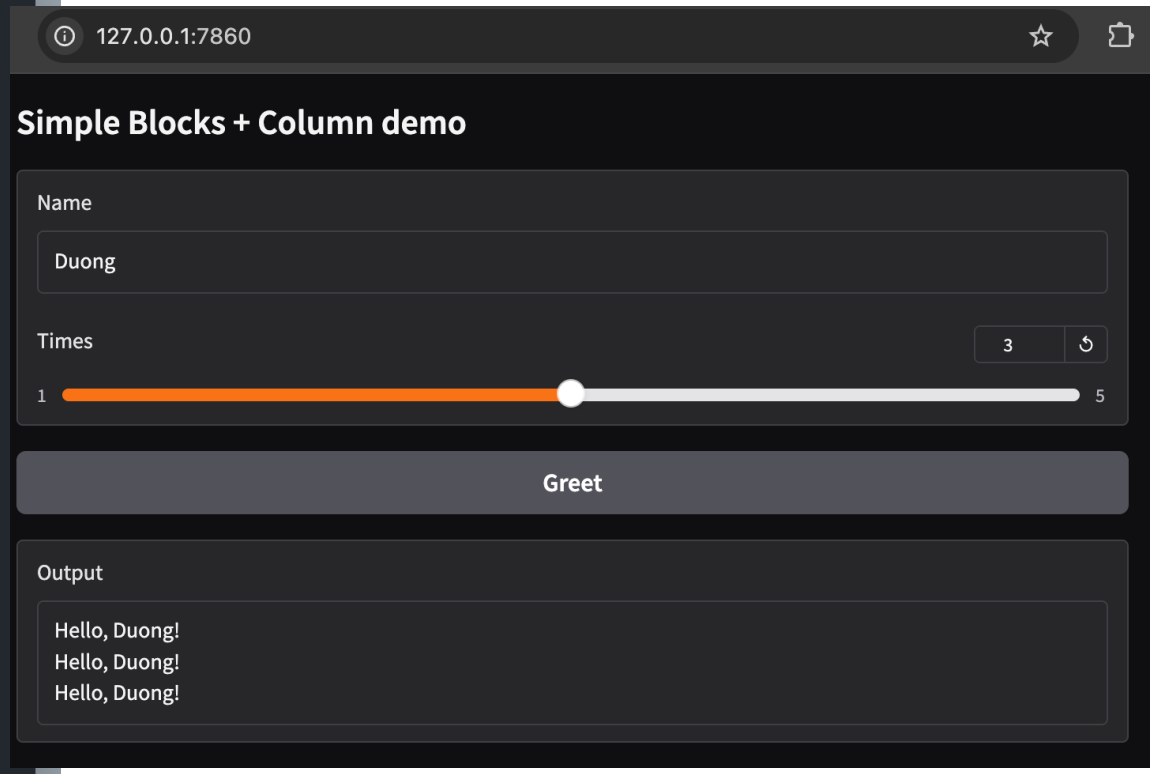
Blocks

❖ Layout: Column

Stack components vertically.

13_blocks_column.py

```python
import gradio as gr

def greet(name: str, times: float) -> str:
    n = int(times or 1)
    name = name or "there"
    return "\n".join([f"Hello, {name}!" for _ in range(max(n, 1))])


with gr.Blocks(title="Blocks + Column: simplest") as demo:
    gr.Markdown("## Simple Blocks + Column demo")

    with gr.Column():
        name = gr.Textbox(label="Name", placeholder="Type your name")
        times = gr.Slider(1, 5, value=1, step=1, label="Times")
        btn = gr.Button("Greet")
        out = gr.Textbox(label="Output", interactive=False)

demo.launch()
```

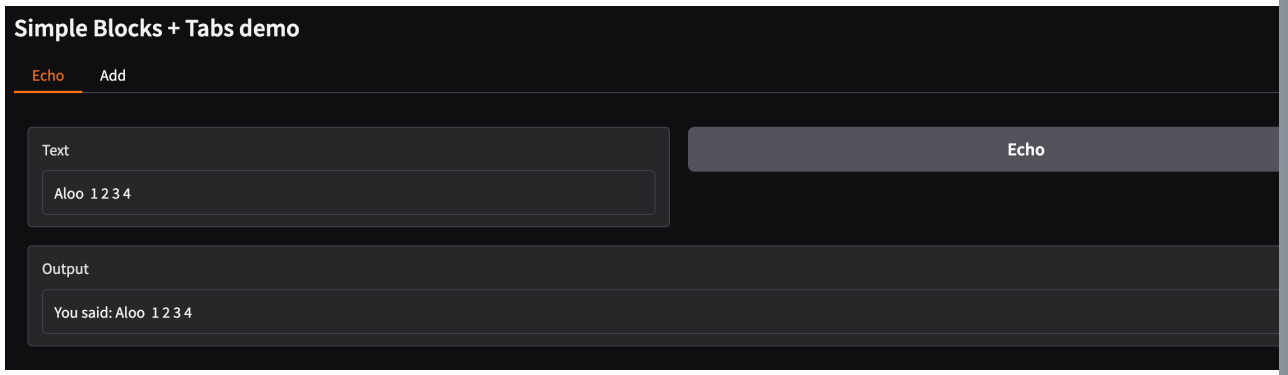


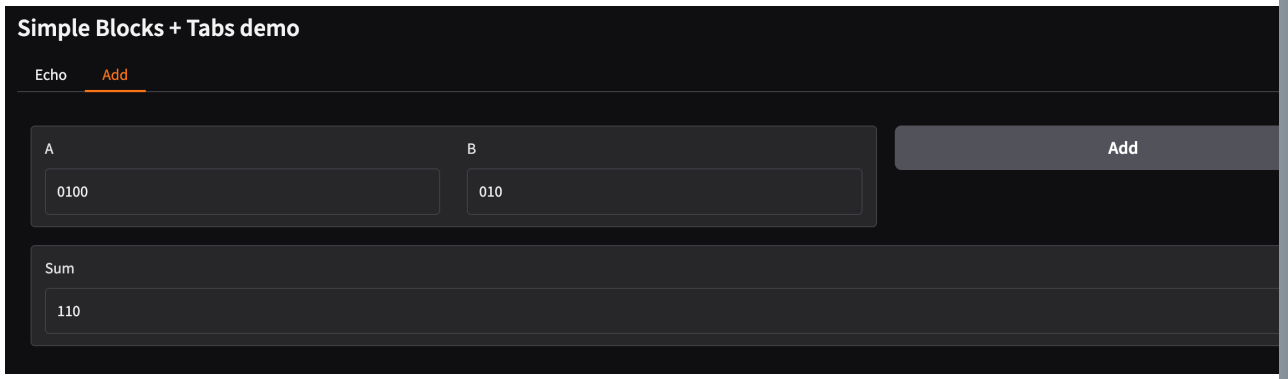
https://www.gradio.app/docs/gradio/column

Blocks

❖ Layout: Tab

Organize components in different tabs





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

14_blocks_tabs.py

```python
import gradio as gr

def echo(text: str) -> str:
    text = text or ""
    return f"You said: {text}"


def add(a: float, b: float) -> float:
    return (a or 0) + (b or 0)


with gr.Blocks(title="Blocks + Tabs: simplest") as demo:
    gr.Markdown("## Simple Blocks + Tabs demo")

    with gr.Tabs():
        with gr.TabItem("Echo"):
            with gr.Row():
                inp = gr.Textbox(label="Text", placeholder="Type something")
                btn1 = gr.Button("Echo")
            out1 = gr.Textbox(label="Output", interactive=False)
            btn1.click(echo, inputs=inp, outputs=out1)

        with gr.TabItem("Add"):
            with gr.Row():
                a = gr.Number(value=0, label="A")
                b = gr.Number(value=0, label="B")
                btn2 = gr.Button("Add")
            out2 = gr.Number(label="Sum", interactive=False)
            btn2.click(add, inputs=[a, b], outputs=out2)

demo.launch()
```

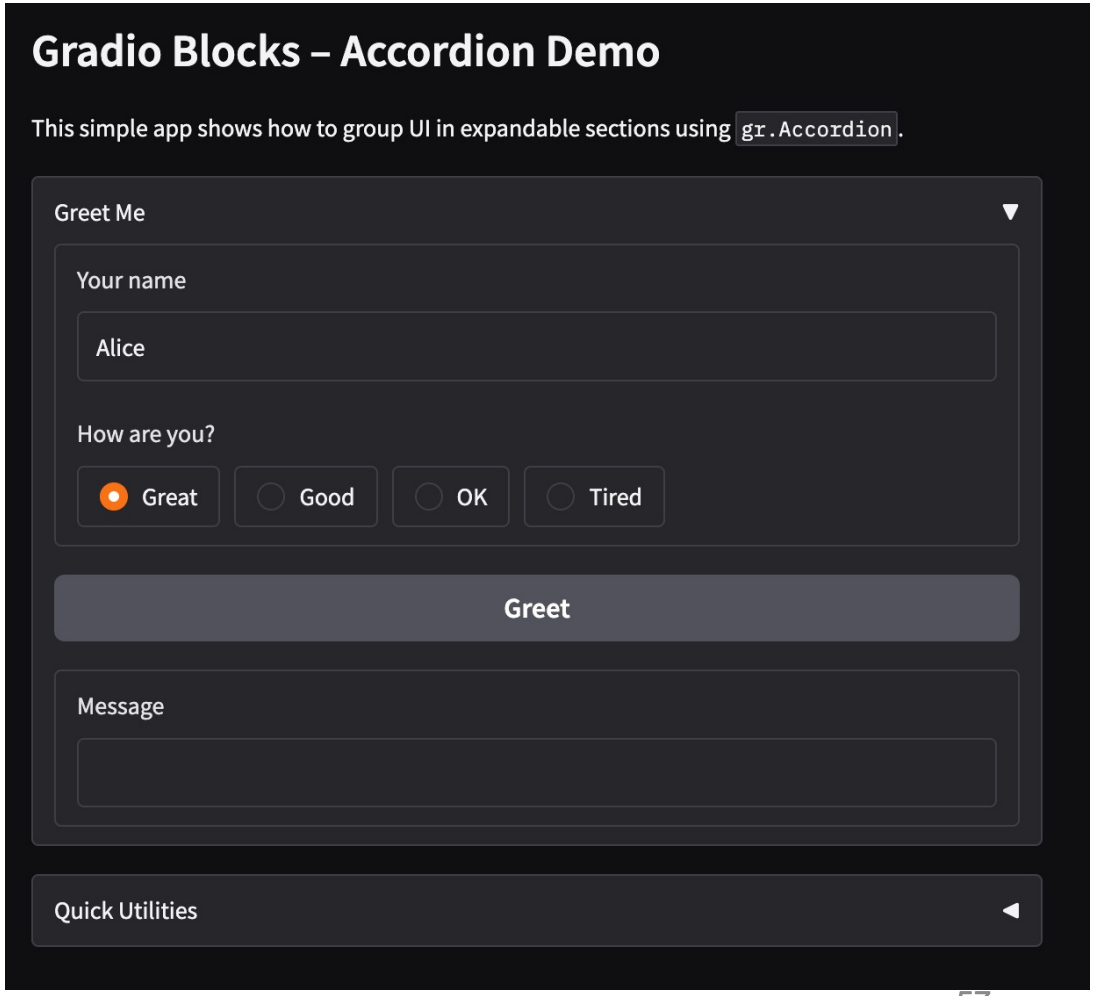
Blocks

❖ Layout: Tab

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

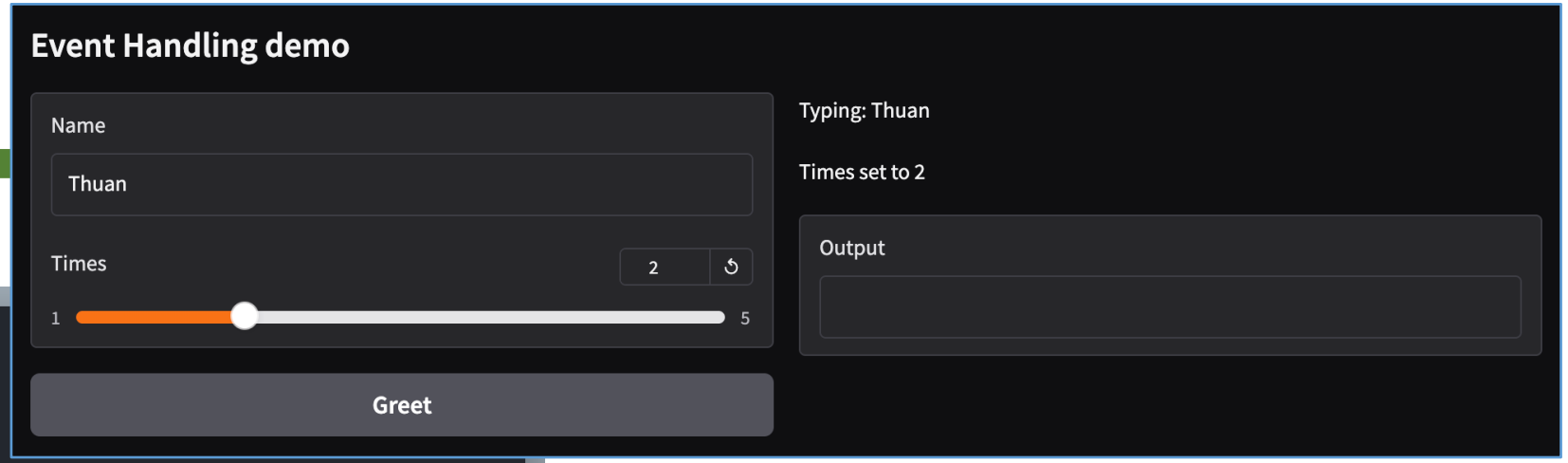
```python
with gr.Blocks(title="Accordion Demo") as demo:
    gr.Markdown("# Gradio Blocks - Accordion Demo")
    gr.Markdown("This simple app shows how to group UI "
                "in expandable sections using `gr.Accordion`.")

    with gr.Accordion("Greet Me", open=True):
        name = gr.Textbox(label="Your name",
                          value="Alice",
                          placeholder="Type your name…")
        mood = gr.Radio(["Great", "Good", "OK", "Tired"],
                        label="How are you?",
                        value="Great")
        greet_btn = gr.Button("Greet")
        greet_out = gr.Textbox(label="Message", interactive=False)
        greet_btn.click(greet, [name, mood], greet_out)

    with gr.Accordion("Quick Utilities", open=False):
        with gr.Row():
            num = gr.Number(label="Number to square", value=10)
            square_btn = gr.Button("Square it")
        result = gr.Number(label="Result", interactive=False)
        square_btn.click(square, num, result)

demo.launch()
```

15_blocks_accordion.py



https://www.gradio.app/docs/gradio/accordion

❖ Event handling



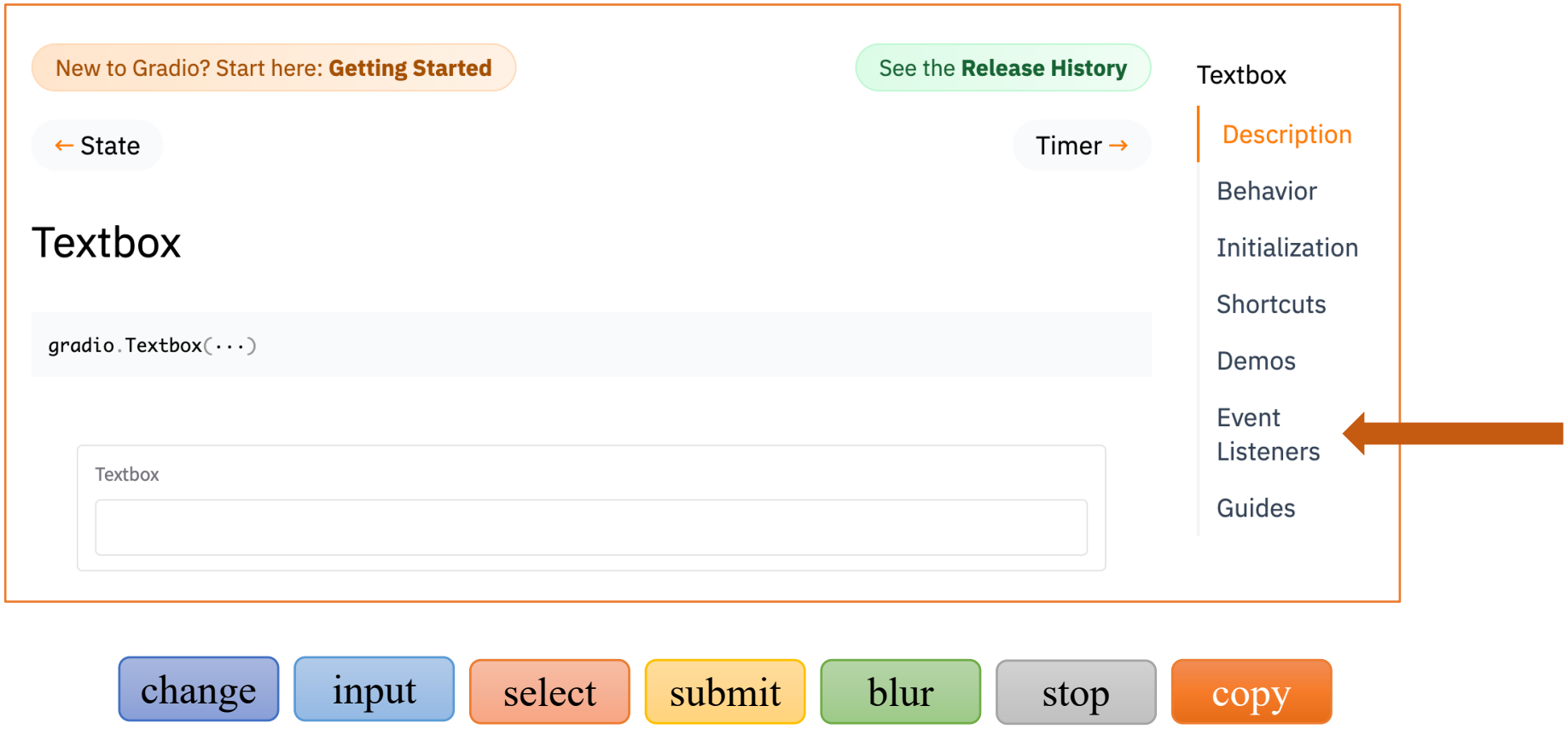
16_event_handling.py

```python
def greet(name: str, times: float) -> str:
    n = int(times or 1)
    name = name or "there"
    return "\n".join([f"Hello, {name}!" for _ in range(max(n, 1))])

def live_preview(name: str) -> str:
    return f"Typing: {name or ''}"

def on_times_change(times: float) -> str:
    return f"Times set to {int(times or 1)}"

with gr.Blocks(title="Event Handling: simplest") as demo:
    gr.Markdown("## Event Handling demo")
    with gr.Row():
        with gr.Column():
            name = gr.Textbox(label="Name", placeholder="Type your name and press Enter")
            times = gr.Slider(1, 5, value=1, step=1, label="Times")
            greet_btn = gr.Button("Greet")
        with gr.Column():
            live_md = gr.Markdown("(Start typing to see live preview)")
            times_md = gr.Markdown("Times set to 1")
            out = gr.Textbox(label="Output", interactive=False)

    # Events
    name.input(live_preview, inputs=name, outputs=live_md)
    name.submit(greet, inputs=[name, times], outputs=out)
    greet_btn.click(greet, inputs=[name, times], outputs=out)
    times.change(on_times_change, inputs=times, outputs=times_md)

demo.launch()
```

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



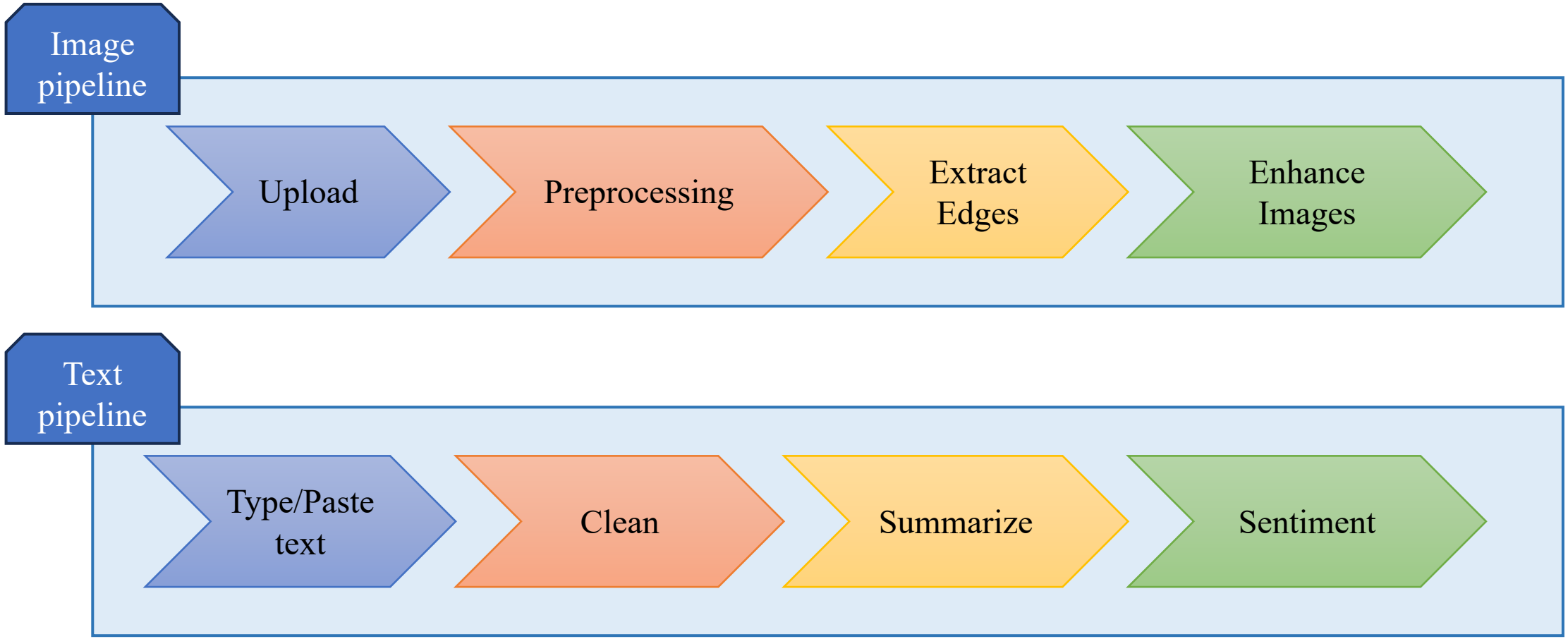
change | input | select | submit | blur | stop | copy

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

```python
with gr.TabItem("Image Pipeline"):
    with gr.Row():
        with gr.Column(scale=1):
            image_in = gr.Image(label="Upload Image", type="pil")
            strength = gr.Slider(0.1, 3.0, value=1.0, step=0.1, label="Edge Strength")
            # Removed Demo Delay slider
            with gr.Row():
                btn_pre = gr.Button("Step 1: Preprocess")
                btn_edge = gr.Button("Step 2: Detect Edges")
                btn_enh = gr.Button("Step 3: Enhance")
            with gr.Row():
                btn_run_all = gr.Button("Run All", variant="primary")
                btn_reset_img = gr.Button("Reset")

            # Internal states to pass between steps
            st_pre = gr.State()
            st_edge = gr.State()

        with gr.Column(scale=1):
            out_pre = gr.Image(label="Preprocessed", interactive=False)
            out_edge = gr.Image(label="Edges", interactive=False)
            out_enh = gr.Image(label="Enhanced", interactive=False)
```

Each step behind requires the previous step to be completed.

❖ Complex Apps

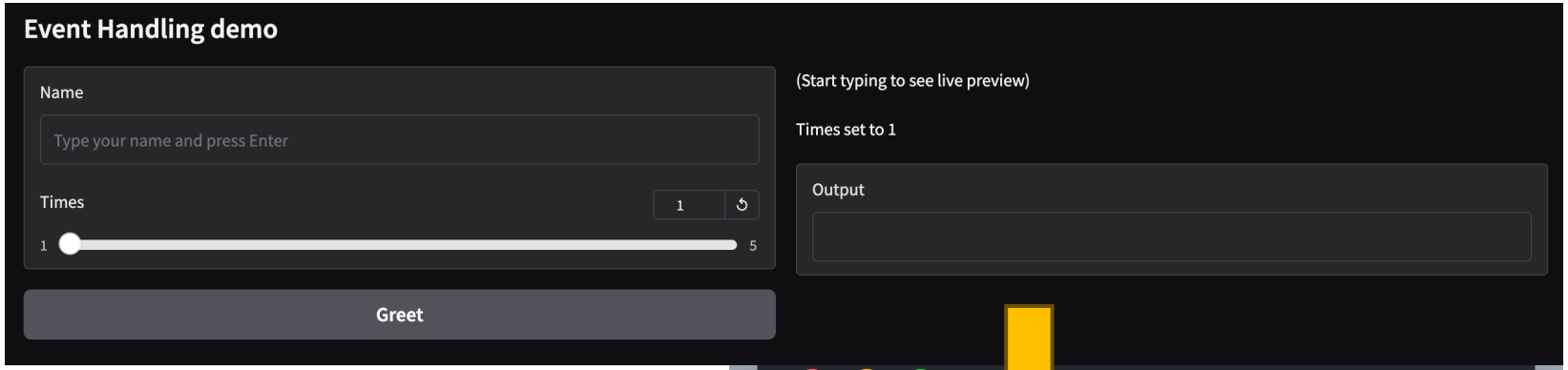
17_complex_app.py

```python
# Wiring events for image pipeline
def _preprocess_and_store(img, progress=gr.Progress(track_tqdm=True)):
    p = preprocess_image(img, progress=progress)
    return p, p

btn_pre.click(_preprocess_and_store, inputs=[image_in], outputs=[out_pre, st_pre])

def _edge_and_store(img_pre, k, progress=gr.Progress(track_tqdm=True)):
    if img_pre is None:
        gr.Warning("Run Step 1 first.")
        return None, None
    e = detect_edges(img_pre, strength=k, progress=progress)
    return e, e

btn_edge.click(_edge_and_store, inputs=[st_pre, strength], outputs=[out_edge, st_edge])

def _enhance(img_edge, progress=gr.Progress(track_tqdm=True)):
    if img_edge is None:
        gr.Warning("Run Step 2 first.")
        return None
    return enhance_image(img_edge, progress=progress)

btn_enh.click(_enhance, inputs=[st_edge], outputs=out_enh)

def _run_all(img, k, progress=gr.Progress(track_tqdm=True)):
    p, e, h = run_all_image(img, k, progress=progress)
    # Also store states for continuity
    return p, e, h, p, e

btn_run_all.click(_run_all, inputs=[image_in, strength], outputs=[out_pre, out_edge, out_enh, st_pre, st_edge])

def _reset_img():
    return None, None, None, None, None

btn_reset_img.click(_reset_img, outputs=[image_in, out_pre, out_edge, out_enh, st_pre])

```

Advanced Usage

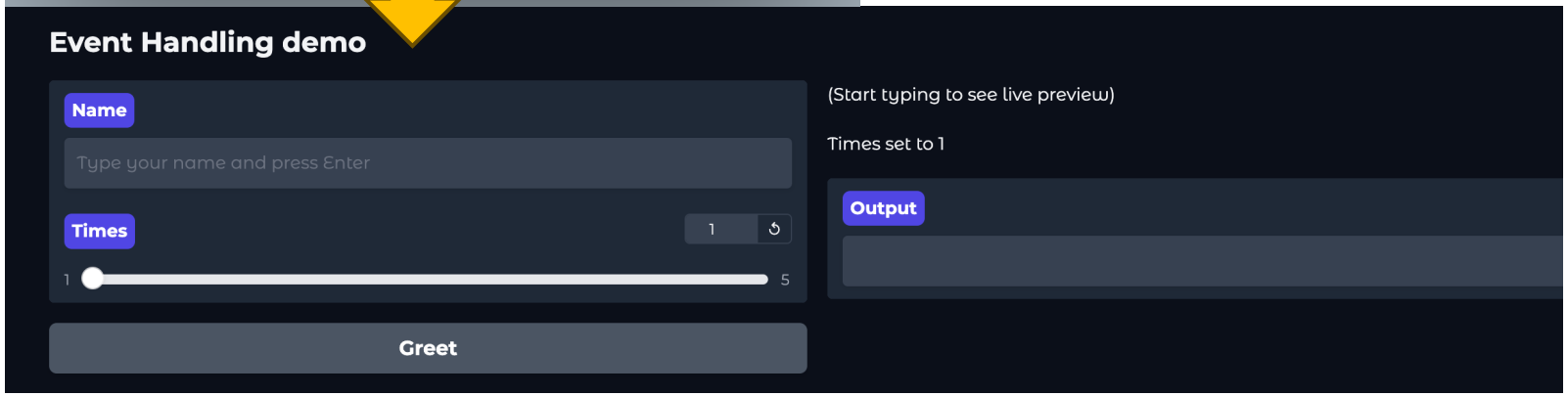
❖ Customization: Theming

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



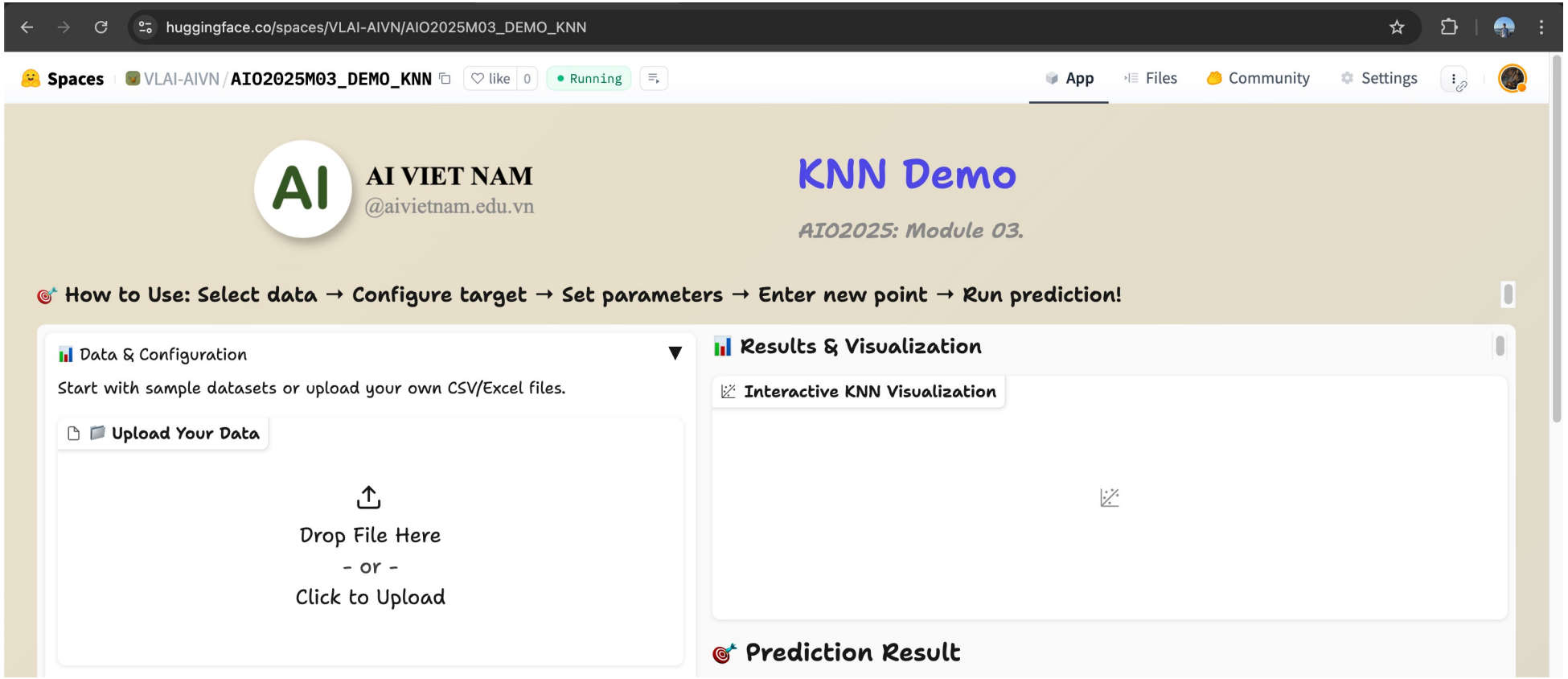
18_theming.py

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



Advanced Usage

❖ Customization: JS/HTML integration



https://huggingface.co/spaces/VLAI-AIVN/AIO2025M03_DEMO_KNN/blob/main/app.py

Advanced Usage

❖ Customization: JS/HTML integration

```
force_light_theme_js = """
() => {
  const params = new URLSearchParams(window.location.search);
  if (!params.has('__theme')) {
    params.set('__theme', 'light');
    window.location.search = params.toString();
  }
}
"""
```

```
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-
    <img src="{logo_base64_vlai}" alt="Logo" style="
    </a> from <a href="https://aivietnam.edu.vn/" target="_blank" style="c
  </div>
</div>
"""
```

```
custom_css = """

.gradio-container {
    min-height: 100vh !important;
    width: 100vw !important;
    margin: 0 !important;
    padding: 0px !important;
    background: linear-gradient(135deg, #F5EFE6 0%, #E8DFCA 50%, #AEBDCA 100%);
    background-size: 600% 600%;
    animation: gradientBG 7s ease infinite;
}

@keyframes gradientBG {
    0% {background-position: 0% 50%;}
    50% {background-position: 100% 50%;}
    100% {background-position: 0% 50%;}
}

/* Minimize spacing and padding */
.content-wrap {
    padding: 2px !important;
    margin: 0 !important;
}
```

EXAMPLES

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

Deployment & Sharing

Deployment & Sharing

❖ Public share links

```
(method) def launch(
    inline: bool | None = None,
    inbrowser: bool = False,
    share: bool | None = None,
    debug: bool = False,
    max_threads: int = 40,
    auth: ((str, str) → bool) | tuple[str, str] | list[tuple[str, str]] | None = None,
    auth_message: str | None = None,
    prevent_thread_lock: bool = False,
    show_error: bool = False,
    server_name: str | None = None,
    server_port: int | None = None,
```

Set True

```
demo.launch()
```

```
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()`.
```

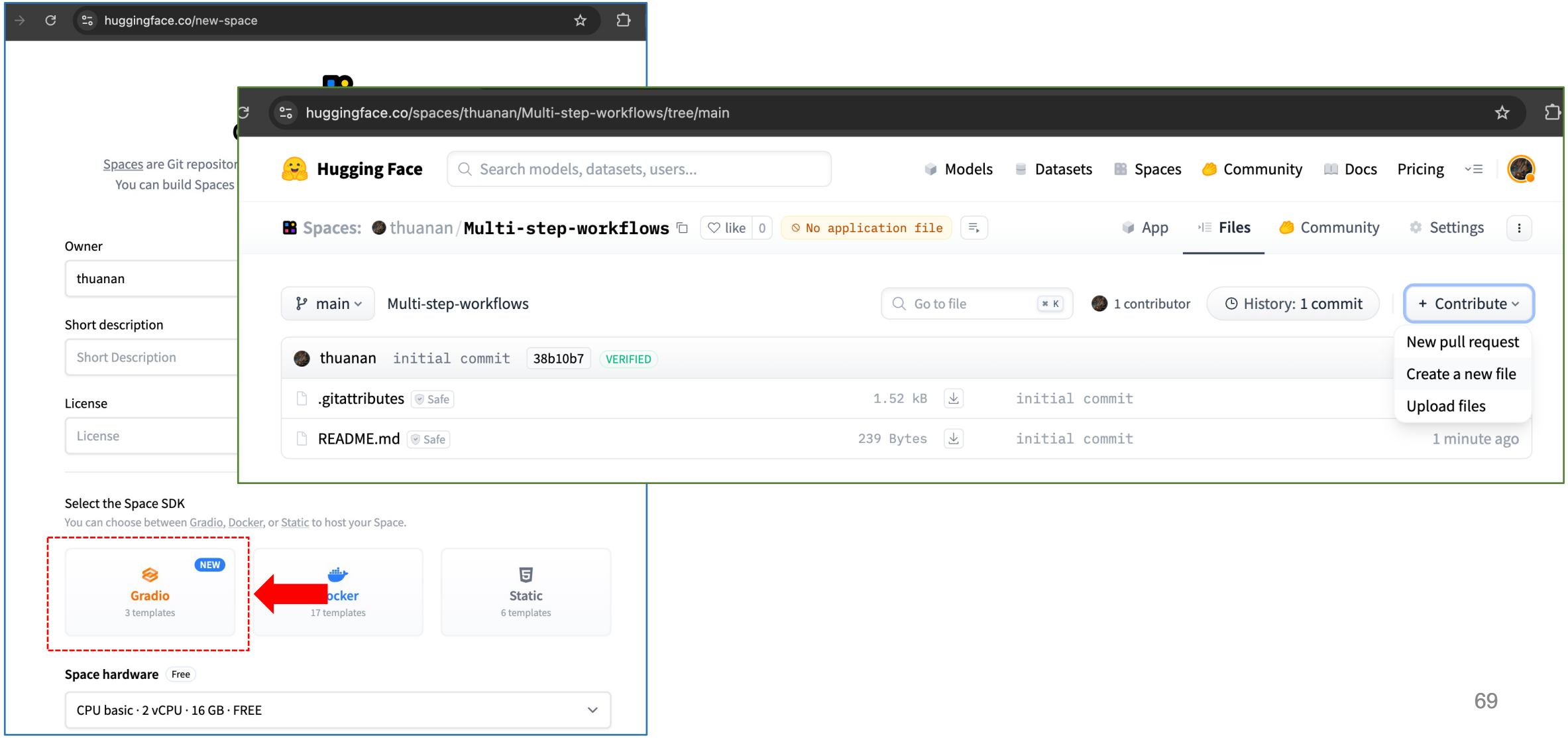
```
demo.launch(share=True)
```

```
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)

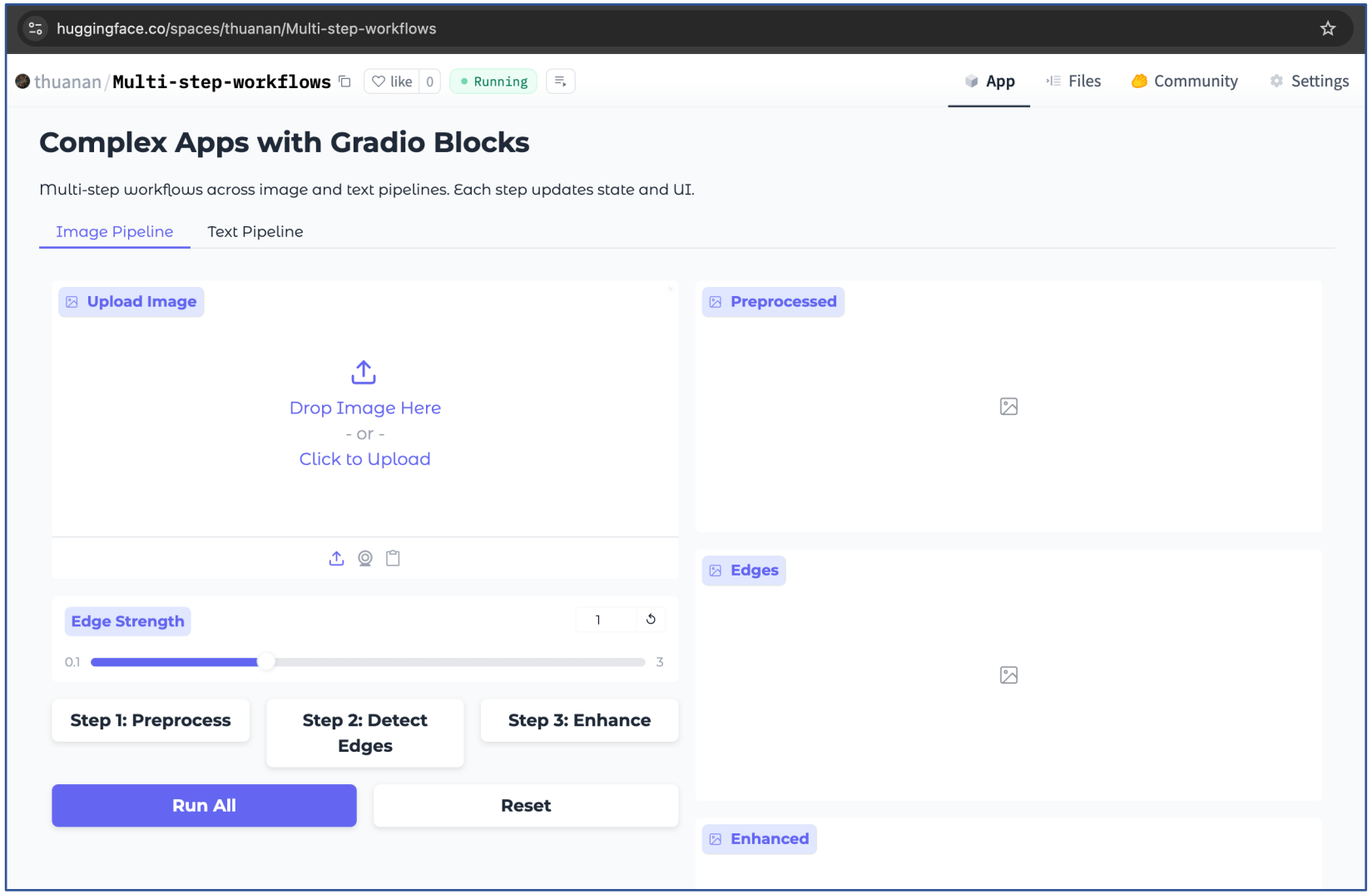
https://huggingface.co/spaces



Deployment & Sharing

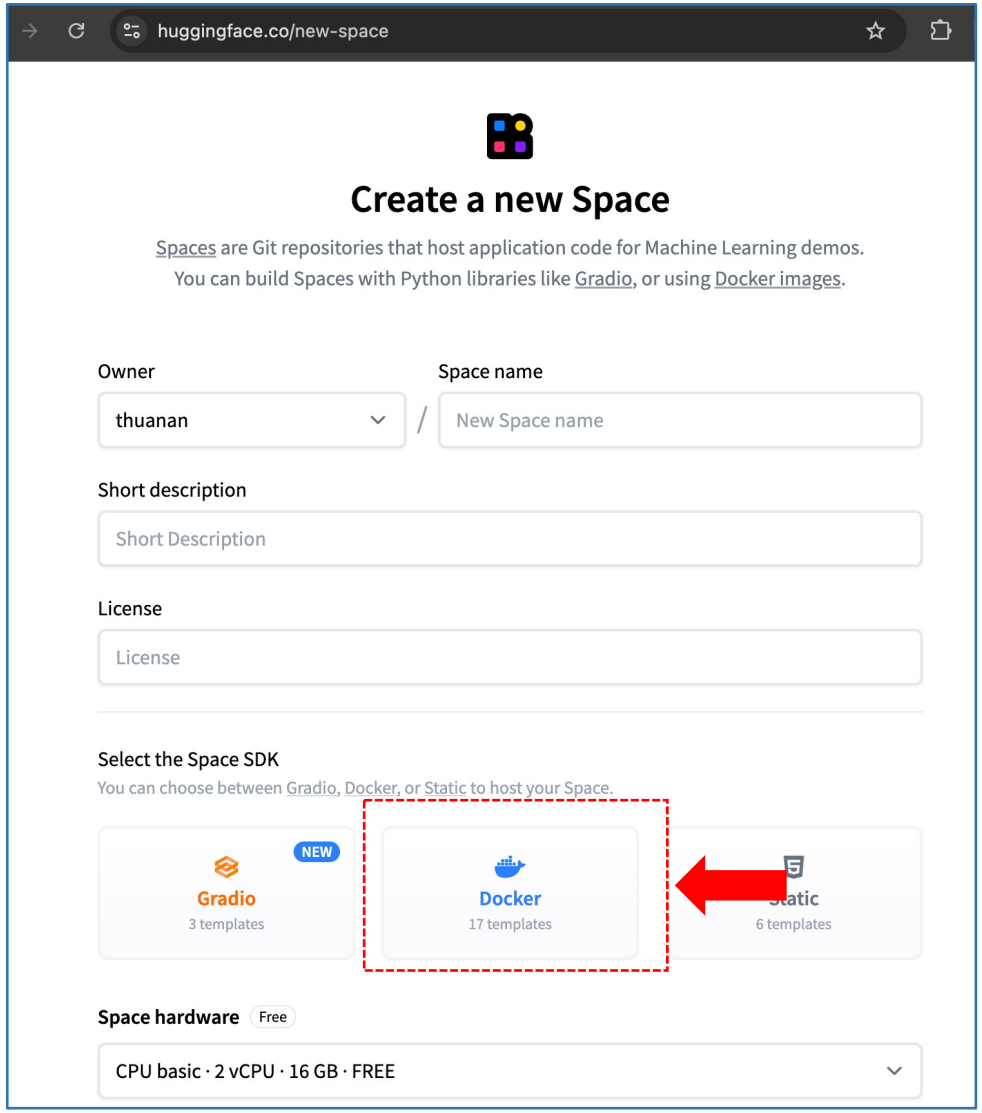
❖ Hugging Face Spaces (free hosting)

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



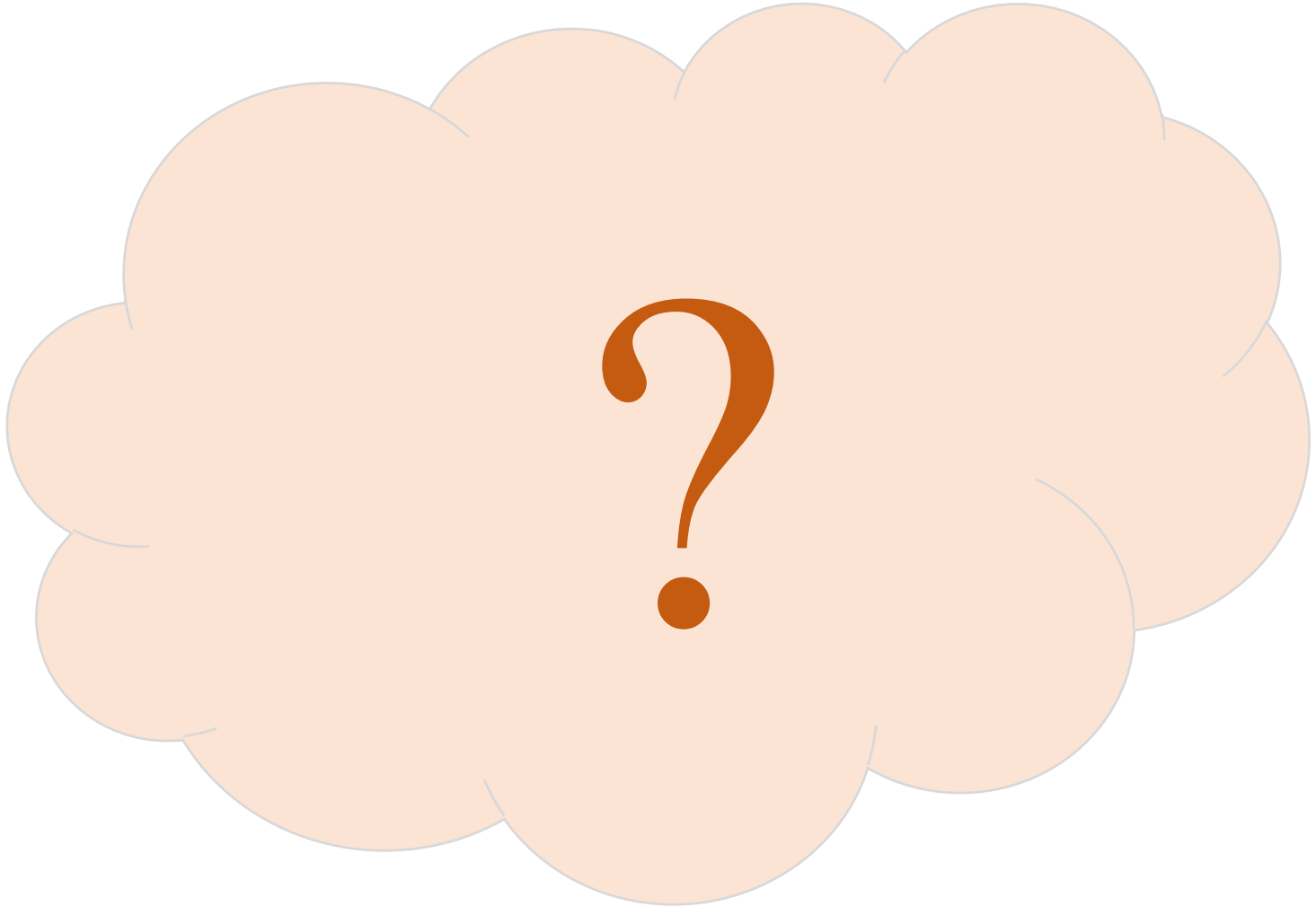
Deployment & Sharing

❖ Dockerize a Gradio app




```dockerfile
FROM python:3.11-slim

ARG PORT=7860
ENV PORT=${PORT} \
    GRADIO_SERVER_NAME=0.0.0.0 \
    GRADIO_SERVER_PORT=${PORT}

WORKDIR /app

COPY requirements.txt /app/requirements.txt
RUN pip install --upgrade pip && \
    pip install -r requirements.txt

COPY app.py /app/app.py

RUN useradd -m appuser && chown -R appuser /app
USER appuser

EXPOSE ${PORT}

# Run the Gradio app
CMD ["python", "app.py"]
```

Question




Thank you!