

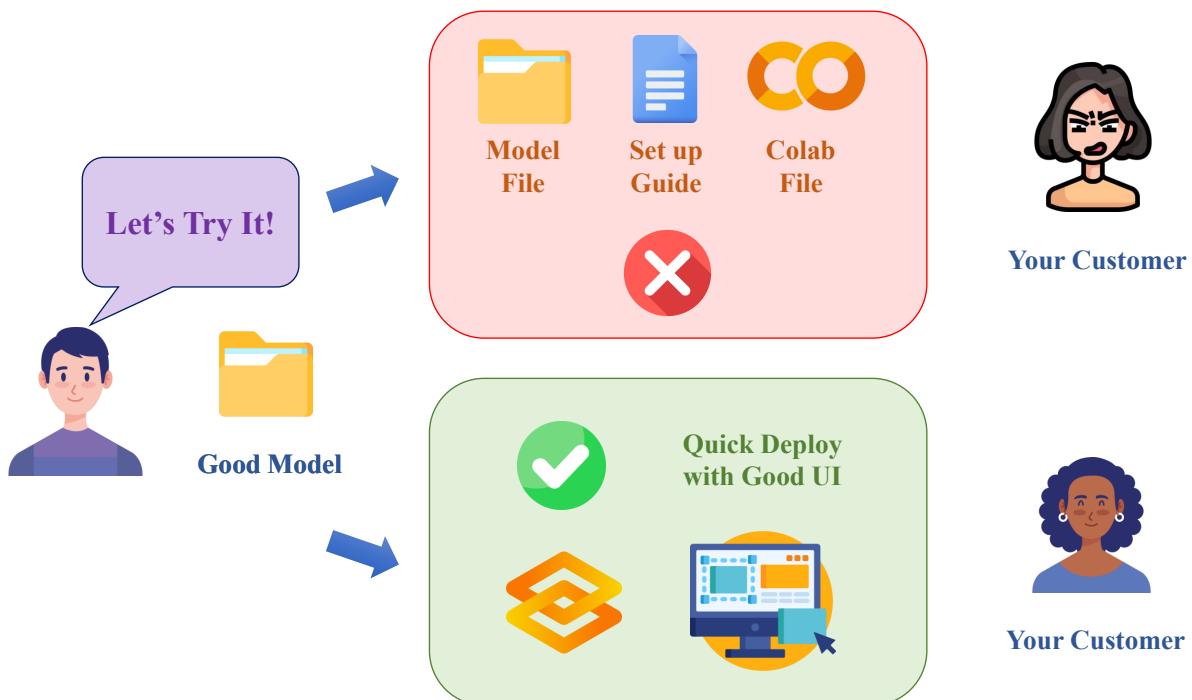
AI VIET NAM – AI COURSE 2025

Tutorial: Gradio

Tạ Hữu Anh Dương, Nguyễn Đăng Nhã, và Đinh Quang Vinh

I. Giới thiệu

Nếu bạn là một nhà khoa học dữ liệu, một kỹ sư máy học, hay đơn giản là một người yêu thích AI, chắc hẳn đã có lúc bạn tạo ra một mô hình dự đoán với độ chính xác cao. Nhưng rồi một vấn đề mà chúng ta hay gặp phải sau khi đã tạo ra được một mô hình tốt: "Làm thế nào để chúng ta có thể cho người khác như sếp, đồng nghiệp, khách hàng, bạn bè hay cộng đồng dùng thử mô hình này một cách dễ dàng mà không cần họ phải cài đặt Python hay chạy code?"



Hình 1: Gradio hỗ trợ dễ dàng cho việc triển khai mô hình

Một trong những giải pháp hỗ trợ chúng ta giải quyết vấn đề trên chính là sử dụng **Gradio**. Trong bài viết này, chúng ta sẽ cùng nhau khám phá và biến những mô hình phức tạp thành các

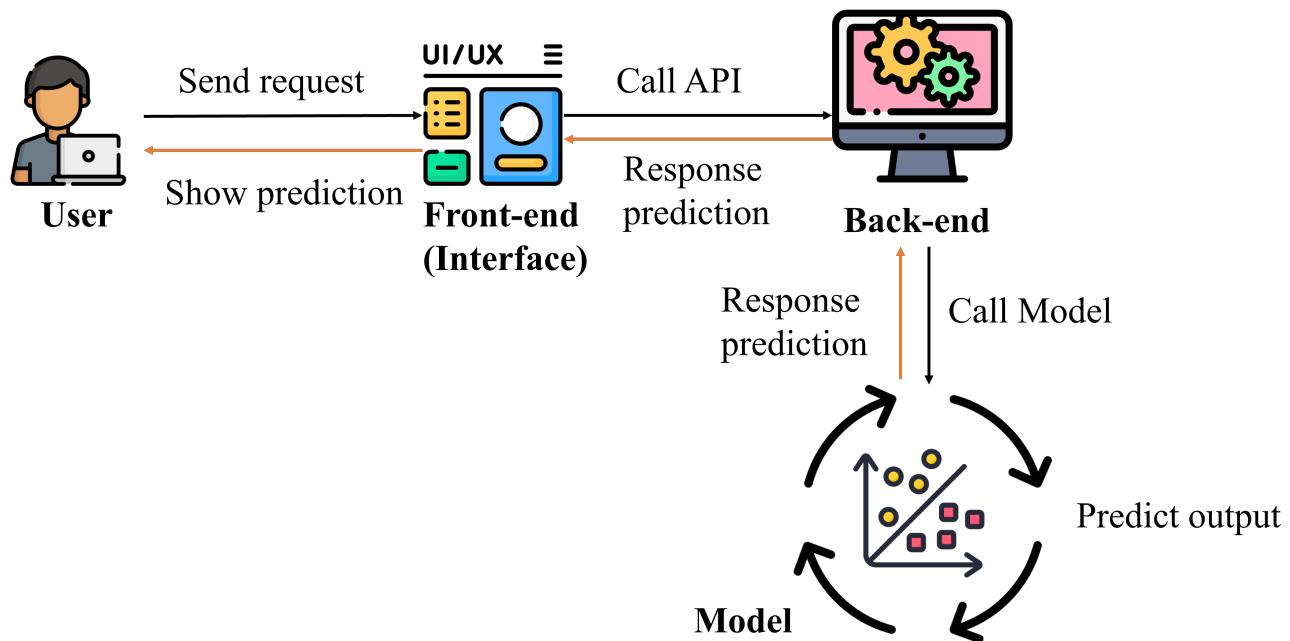
Ứng dụng web tương tác bằng Gradio chỉ với vài thao tác.

Để giúp dễ dàng tiếp cận và nắm bắt nội dung bài đọc, bài viết được tổ chức thành các phần chính như sau:

- I. Giới thiệu nội dung bài viết
- II. Bối cảnh - Tại sao Gradio lại cần thiết?
- III. Tư duy với Gradio - Ba Trụ cột: Input, Model, Output
- IV. Làm quen với Gradio
- V. Triển khai Object Detection cơ bản với Gradio trên Colab

II. Tại sao Gradio lại cần thiết?

Hãy cùng xem xét luồng hoạt động tiêu chuẩn của một ứng dụng AI khi nó được triển khai cho người dùng cuối:



Hình 2: Luồng hoạt động của một ứng dụng AI từ người dùng đến mô hình

Quy trình này có thể được tóm tắt qua các bước sau:

1. Người dùng tương tác với Giao diện (Front-end) trên trình duyệt (ví dụ: điền form, tải ảnh).
2. Khi nhấn "Gửi", giao diện đóng gói dữ liệu và gửi yêu cầu đến Máy chủ (Back-end).
3. Máy chủ nhận yêu cầu, xử lý, và "gọi" đến Mô hình Machine Learning.
4. Mô hình thực hiện dự đoán và trả kết quả về cho máy chủ.
5. Máy chủ gửi trả kết quả về lại cho giao diện.
6. Giao diện hiển thị kết quả cho người dùng một cách trực quan.

Quy trình này rất tốt, nhưng để xây dựng được nó đòi hỏi một đội ngũ với nhiều kỹ năng: chuyên gia Front-end (HTML, CSS, JavaScript), chuyên gia Back-end (Flask, FastAPI), và thậm chí có thể phải có cả kỹ sư DevOps để triển khai. Đây thực sự là một rào cản lớn.

Vậy vai trò của Gradio là gì?

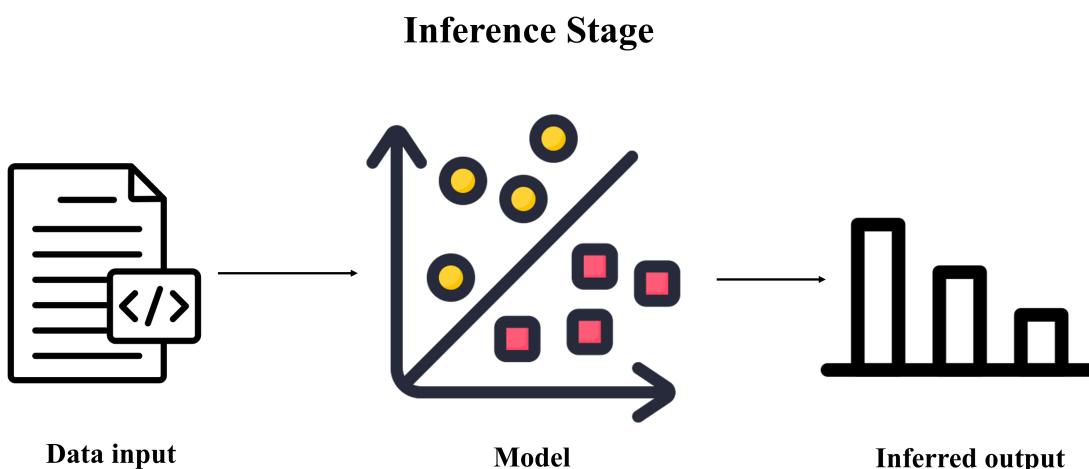
Gradio đóng vai trò tối giản những thứ mà bạn cần phải làm: "Hãy quên đi sự phức tạp đó. Bạn

chỉ cần đưa cho tôi 'bộ não' (mô hình AI của bạn), tôi sẽ lo phần còn lại."

Gradio sẽ tự động tạo ra cả giao diện Front-end và máy chủ Back-end cho bạn. Với tư cách là người tạo ra mô hình, bạn chỉ cần tập trung vào đúng một việc: gói gọn mô hình của mình vào một hàm Python.

III. Ba Trụ cột: Input, Model, Output

Trước khi bắt tay vào xây dựng, chúng ta cần hiểu nguyên tắc cơ bản nhất: mọi mô hình đều có đầu vào và đầu ra. Hãy nhìn vào sơ đồ xây dựng một ứng dụng cho người dùng ở giai đoạn Inference:



Hình 3: Ba thành phần cốt lõi của giai đoạn Inference

Trong giai đoạn này, có ba thành phần cốt lõi mà bạn phải xác định rõ:

- Input (Đầu vào): Đây là dữ liệu mà người dùng sẽ cung cấp cho ứng dụng của bạn. Nó có thể là một đoạn văn bản, một con số, một hình ảnh, hoặc nhiều thứ khác.
- Model (Mô hình): Đây là bộ não đã được huấn luyện. Trong Gradio, nó thường là một hàm Python nhận Input làm tham số và thực hiện logic dự đoán.
- Output (Đầu ra): Đây là kết quả mà mô hình của bạn trả về sau khi xử lý Input.

Nhiệm vụ của bạn khi dùng Gradio khi triển khai một mô hình nào đó là định nghĩa rõ ràng ba yếu tố này. Hãy xem cách chúng hoạt động qua một ví dụ đó là dự đoán lương (output) dựa trên tuổi (input) và chúng ta sẽ làm gói gọn chỉ trong 2 bước:

Bước 1: Chuẩn bị hàm dự đoán

Ở đây thay vì huấn luyện một mô hình hồi quy phức tạp, chúng ta sẽ giả lập nó bằng một hàm Python đơn giản.

Xây dựng hàm dự đoán lương dựa trên độ tuổi

```

1 # Hàm này nhận một tham số là 'age' và trả về một chuỗi kết quả.
2 def predict_salary(age):
3     """
4         Hàm dự đoán lương dựa trên tuổi.
5         Công thức giả lập: Lương khởi điểm 10 triệu, mỗi năm kinh nghiệm tăng 1.5 triệu.
6     """
7     if age < 18:
8         return "Tuổi không hợp lệ, vui lòng nhập tuổi lớn hơn hoặc bằng 18."
9
10    # Giả sử kinh nghiệm bắt đầu được tính từ năm 18 tuổi
11    experience_years = age - 18
12    base_salary = 10 # triệu VND
13    salary_increase_per_year = 1.5 # triệu VND
14
15    predicted_salary = base_salary + (experience_years * salary_increase_per_year)
16
17    return f"Với {age} tuổi, mức lương dự đoán là: {predicted_salary:.2f} triệu VND."

```

Bước 2: Triển khai ví dụ dự đoán lương dựa trên độ tuổi bằng Gradio

Chúng ta sẽ sử dụng gr.Interface và chỉ định rõ 3 trụ cột: fn (Model), inputs (Input), và outputs (Output).

Xây dựng giao diện với Gradio

```

1 # Đầu tiên, hãy chắc chắn bạn đã cài đặt Gradio
2 # Mở terminal và chạy: pip install gradio
3
4 import gradio as gr
5
6 # Khai báo một đối tượng Interface; kết nối hàm logic với các thành phần giao diện.
7 salary_app = gr.Interface(
8     fn=predict_salary, # Dùng hàm predict_salary
9     # Đầu vào là một con số, nên ta dùng component gr.Number.
10    inputs=gr.Number(label="Nhập số tuổi của bạn"),
11    # Đầu ra là văn bản, nên ta dùng gr.Textbox.
12    outputs=gr.Textbox(label="Kết quả dự đoán lương")
13 )
14
15 # Khởi chạy ứng dụng
16 salary_app.launch()

```

Khi bạn chạy file Python này, terminal sẽ hiển thị một địa chỉ URL, thường là <http://127.0.0.1:7860>. Hãy sao chép và dán nó vào trình duyệt web. Một giao diện website sẽ hiện lên với các thông tin:

- Một ô nhập liệu có nhãn "Nhập tuổi của bạn".
- Hai nút “Submit” và “Clear”.
- Một ô kết quả có nhãn "Kết quả dự đoán".

Hãy thử nhập một số tuổi và nhấn "Submit". Kết quả từ hàm predict_salary của bạn sẽ ngay lập tức sẽ hiện ra như này:

Nhập số tuổi của bạn

30

Clear Submit

Kết quả dự đoán lương

Với 30 tuổi, mức lương dự đoán của bạn là: 28.00 triệu VND.

Flag

Như vậy chúng ta vừa tạo ra một ứng dụng web đầy đủ chức năng mà không cần viết một dòng HTML hay JavaScript nào.

IV. Làm quen với Gradio

Sau khi chúng ta đã hiểu lí do tại sao cần Gradio cho việc triển khai mô hình học máy và sự tiện nghi khi chúng ta sử dụng Gradio thay vì phương pháp truyền thống, thì bây giờ chúng ta sẽ tìm hiểu sâu hơn về cách hoạt động của Gradio, vì vậy chúng ta sẽ cần nắm bắt được các thành phần chính mà Gradio cung cấp, bao gồm Input Elements, Output Elements, Interface, và Blocks.

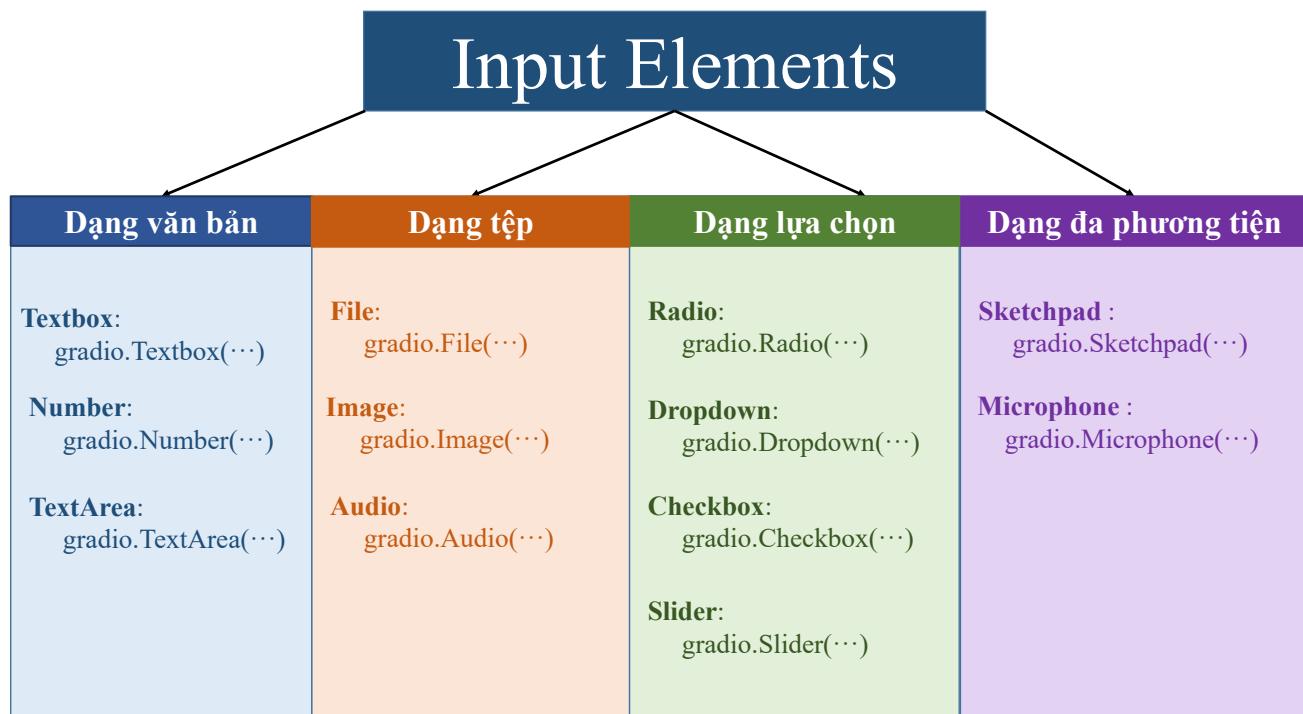
IV.1. Input Elements

Như đã đề cập ở mục III, một trong những yếu tố cần được xác định rõ ràng để có thể tương tác với mô hình AI. Gradio hỗ trợ cho chúng ta việc sử dụng dữ liệu đầu vào rất đa dạng và ở đây sẽ phân thành bốn nhóm phổ biến như sau:

- Các phần tử nhập liệu dạng văn bản
 - TextBox: Một hộp văn bản đơn giản để nhập chuỗi ký tự.
 - Number: Nhập liệu giá trị số (số nguyên hoặc số thập phân).
 - TextArea: Một trường văn bản nhiều dòng để nhập các văn bản dài hơn.
- Các phần tử nhập liệu dạng tệp

- File: Cho phép người dùng tải lên bất kỳ loại tệp nào.
- Image: Nhập liệu cho tệp hình ảnh, hỗ trợ nhiều định dạng hình ảnh như JPEG hay PNG.
- Audio: Cho phép người dùng tải lên tệp âm thanh.
- Các phần tử lựa chọn
 - Radio: Nhóm các nút lựa chọn cho phép người dùng chọn một trong nhiều tùy chọn.
 - Dropdown: Một menu thả xuống cho phép người dùng chọn một tùy chọn từ danh sách.
 - Checkbox: Một hộp kiểm cho phép người dùng nhập giá trị đúng/sai (true/false).
 - Slider: Một thanh trượt cho phép chọn một số trong một phạm vi nhất định.
- Các phần tử nhập liệu dạng đa phương tiện
 - Sketchpad: Cho phép người dùng vẽ trực tiếp trên canvas.
 - Microphone: Ghi âm từ micro của người dùng.

Để giúp mọi người có thêm cái nhìn tổng quát hơn thì dưới đây là tóm tắt và cú pháp của các loại phần tử nhập liệu:



Hình 4: Các loại phần tử nhập liệu và cú pháp

IV.2. Output Elements

Output elements là các thành phần mà qua đó Gradio hiển thị kết quả từ mô hình AI. Tương tự như Input Elements thì Output Elements cũng là một thành phần rất quan trọng cần phải xác định rõ ràng. Bốn nhóm output phổ biến bao gồm:

- Các phần tử đầu ra dạng văn bản
 - Label: Hiển thị văn bản hoặc nhãn cho kết quả đầu ra.
 - Textbox: Hiển thị văn bản hoặc kết quả từ quá trình xử lý
- Các phần tử đầu ra dạng tệp và đa phương tiện
 - File: Cung cấp tệp để tải xuống sau khi xử lý.
 - Image: Hiển thị hình ảnh dưới dạng kết quả đầu ra.
 - Video: Hiển thị video đã tải lên hoặc được tạo ra từ quá trình xử lý.
 - Audio: Phát âm thanh đầu ra từ một tệp hoặc một đoạn ghi âm.
 - Gallery: Hiển thị một bộ sưu tập hình ảnh hoặc đa phương tiện.
- Các phần tử đầu ra dạng dữ liệu
 - Dataframe: Hiển thị dữ liệu dưới dạng bảng dữ liệu (DataFrame).
 - JSON: Hiển thị dữ liệu JSON được định dạng.
- Các phần tử đầu ra dạng khác
 - HighlightedText: Hiển thị văn bản với các từ được tô sáng.
 - HTML: Hiển thị nội dung HTML, có thể bao gồm văn bản, hình ảnh hoặc video.
 - Plot: Hiển thị biểu đồ hoặc hình ảnh đồ thị.

Lưu ý về tính hai chiều: Nhiều thành phần của Gradio (như Textbox, Image, Video, DataFrame) có thể hoạt động như cả Input và Output. Vai trò của chúng được quyết định bởi việc bạn đặt chúng vào danh sách inputs hay outputs.

IV.3. Các cách để xây dựng giao diện với Gradio

Ở đây chúng ta sẽ có hai cách chính để xây dựng giao diện với Gradio với mức độ kiểm soát khác nhau đó là **Interface (Đơn giản, nhanh chóng)** và **Blocks (Linh hoạt, tùy biến cao)**

IV.3.1. Interface

Interface là cách nhanh chóng và đơn giản nhất để triển khai giao diện cho mô hình AI trong Gradio. Với Interface, ta chỉ cần định nghĩa hàm xử lý, chỉ định các thành phần input và output là Gradio sẽ tự động tạo ra một giao diện cho người dùng sử dụng. Điều này rất tiện lợi cho các ứng dụng nhỏ hoặc thử nghiệm nhanh mô hình AI.

Ví dụ dự đoán lương (output) dựa trên tuổi (input) ở mục III chính là ví dụ về triển khai giao diện bằng Interface để tạo giao diện cụ thể như sau:

Ví dụ về triển khai giao diện bằng Interface

```

1 salary_app = gr.Interface( # Khởi tạo Interface để triển khai giao diện
2     fn=predict_salary,           # Dùng hàm predict_salary làm bộ não.
3     inputs=gr.Number(label="Nhập số tuổi của bạn"), # Đầu vào là một con số, nên ta dùng
4                                         # component gr.Number.
5     outputs=gr.Textbox(label="Kết quả dự đoán lương") # Đầu ra là văn bản, nên ta dùng
6                                         # gr.Textbox.
7 )

```

IV.3.2. Blocks

Mặt khác, lớp Blocks sẽ giúp tăng tính linh hoạt và khả năng kiểm soát khi xây dựng giao diện. Với Blocks, bạn có thể sắp xếp nhiều thành phần, nhóm chúng theo bố cục tùy chỉnh và xác định tương tác giữa chúng, khiến nó trở thành lựa chọn hàng đầu cho các ứng dụng phức tạp hơn. Sau đây là một ví dụ về việc in ra lời chào mừng cho người dùng để chứng minh cách Blocks cho phép tùy chỉnh bố cục và chức năng:

Ví dụ triển khai giao diện bằng Blocks

```

1 import gradio as gr
2 def greet(name):
3     return f"Hello, {name}!"
4 # Sử dụng Khối để tạo bố cục tùy chỉnh
5 with gr.Blocks() as demo:
6     name_input = gr.Textbox(label="Nhập tên của bạn") # Input dạng textbox
7     greet_button = gr.Button("Gửi lời chào") # Button dạng button
8     greeting_output = gr.Textbox(label="Lời chào") # Output dạng textbox
9     # Button sẽ kích hoạt hàm greet, với input là name_input và output là
10    # greeting_output
11    greet_button.click(greet, inputs=name_input, outputs=greeting_output)
12 demo.launch()

```



Cụ thể ở đây với Blocks bạn có thể:

- Kiểm soát bố cục, định vị các phần tử name_input, greet_button và greeting_output theo ý muốn.
- Button greet_button chỉ kích hoạt chức năng chào hỏi khi được nhập vào, mang đến một luồng tương tác.
- Linh hoạt trong việc thiết kế các quy trình làm việc phức tạp, điều mà chỉ riêng Interface không thể đạt được.

IV.3.3. Bố trí bố cục đơn giản với Blocks

Bây giờ chúng ta sẽ đi sâu hơn về cách bố trí bố cục với Blocks qua việc kết hợp với hai thành phần cơ bản đó là Row, Column (Ngoài ra chúng ta còn có thể kết hợp với các thành phần khác như Tab, Accordion).

Quy ước ví dụ: Chúng ta sẽ thống nhất ví dụ cụ thể để áp dụng cho các trường hợp cần xây dựng bố cục như sau: Nhập tên người dùng và chọn độ tuổi, sau đó in ra thông tin đã nhập qua một nút Button.

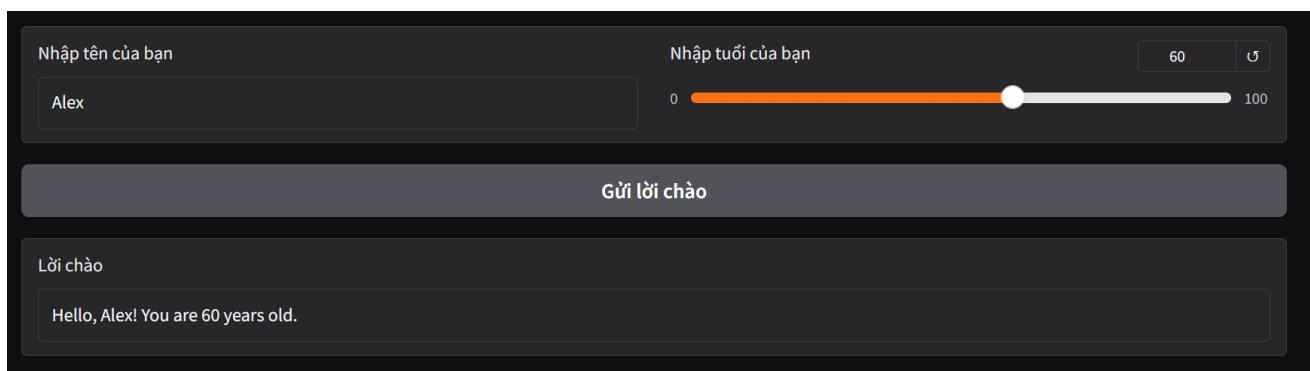
Đầu tiên chính là sử dụng gradio.Row() cho phép sắp xếp các phần tử nằm ngang trên cùng một hàng.

Ví dụ sử dụng gradio.Row()

```

1 import gradio as gr
2 def greet(name, age):
3     return f"Hello, {name}! You are {age} years old."
4 # Sử dụng Khối để tạo bố cục tùy chỉnh
5 with gr.Blocks() as demo:
6     with gr.Row(): # Dòng 1
7         name_input = gr.Textbox(label="Nhập tên của bạn") # Tên dạng textbox
8         age_slider = gr.Slider(label="Nhập tuổi của bạn", minimum=0, maximum=100, step=
9             1) # Tuổi dạng slider
10    with gr.Row(): # Dòng 2
11        greet_button = gr.Button("Gửi lời chào") # Button dạng button
12        greeting_output = gr.Textbox(label="Lời chào") # Output dạng textbox
13        # Button sẽ kích hoạt hàm greet, với input là name_input và output là
14        # greeting_output
15        greet_button.click(greet, inputs=[name_input, age_slider], outputs=greeting_output)
16    demo.launch()

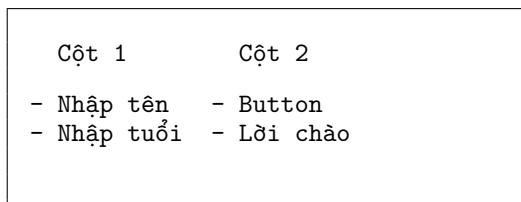
```



Kết quả phần Input là "Nhập tên", "Nhập tuổi" đều đang được sắp xếp trên dòng thứ nhất và nút "Gửi lời chào" được hiển thị trên dòng thứ hai.

Thứ hai chính là sử dụng `gr.Column()` cho phép sắp xếp các thành phần giao diện theo chiều dọc, nghĩa là các phần tử sẽ được xếp chồng lên nhau theo thứ tự:

Ở đây phần này chúng ta sẽ tạo một hàng chứa hai cột với cấu trúc như sau:

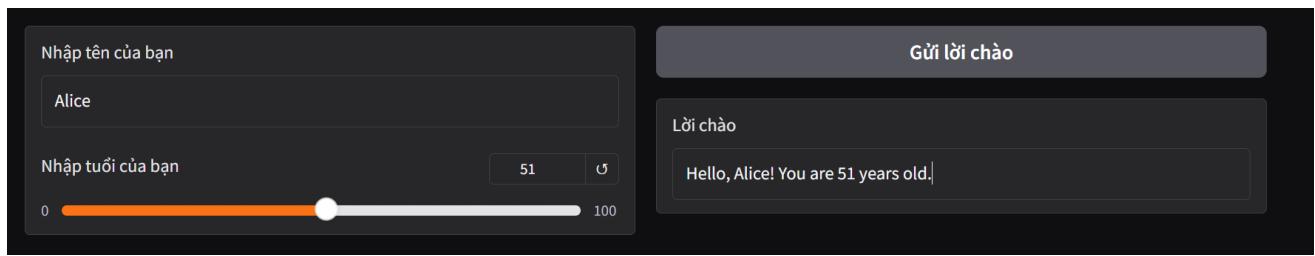


Ví dụ sử dụng `gradio.Column()` kết hợp với `gradio.Row()`

```

1 import gradio as gr
2 def greet(name, age):
3     return f"Hello, {name}! You are {age} years old."
4 # Sử dụng Khối để tạo bố cục tùy chỉnh
5 with gr.Blocks() as demo:
6     with gr.Row(): # Tạo hàng chứa 2 cột
7         with gr.Column(): # Cột 1: Input
8             name_input = gr.Textbox(label="Nhập tên của bạn") # Tên dạng textbox
9             age_slider = gr.Slider(label="Nhập tuổi của bạn", minimum=0, maximum=100,
10                                     step=1) # Tuổi dạng slider
11         with gr.Column(): # Cột 2: Button và Output
12             greet_button = gr.Button("Gửi lời chào") # Button dạng button
13             greeting_output = gr.Textbox(label="Lời chào") # Output dạng textbox
14             # Button sẽ kích hoạt hàm greet, với input là name_input và output là
15             # greeting_output
16             greet_button.click(greet, inputs=[name_input, age_slider], outputs=greeting_output)
17 demo.launch()

```



IV.3.4. Event Listeners

Thông thường khi triển khai giao diện bằng Blocks chúng ta sẽ cần chú ý thêm về Event Listeners. Chúng tuân theo một quy tắc đơn giản nhưng mạnh mẽ: "**Khi [sự kiện A] xảy ra trên [thành phần X], hãy thực thi [hàm Y].**"

Ví dụ: "Khi [sự kiện click] xảy ra trên [thành phần nút Gửi], hãy thực thi [hàm dự đoán]."

Hầu hết các Event Listener đều có cú pháp tương tự nhau:

Cú pháp chung của Event Listener

```

1 component.event_name( # component: Đối tượng component mà bạn muốn "lắng nghe"
2   # event_name: Tên của sự kiện bạn muốn bắt (ví dụ: click, change).
3   fn=your_function, # Hàm Python sẽ được thực thi khi sự kiện xảy ra
4   inputs=[input_component_1, input_component_2], # Một component hoặc một danh sách c
      ác component sẽ cung cấp giá trị đầu vào
      cho hàm fn.
5   outputs=[output_component_1] # Một component hoặc một danh sách các component sẽ đư
      ợc cập nhật bởi giá trị trả về của hàm fn.
6 )

```

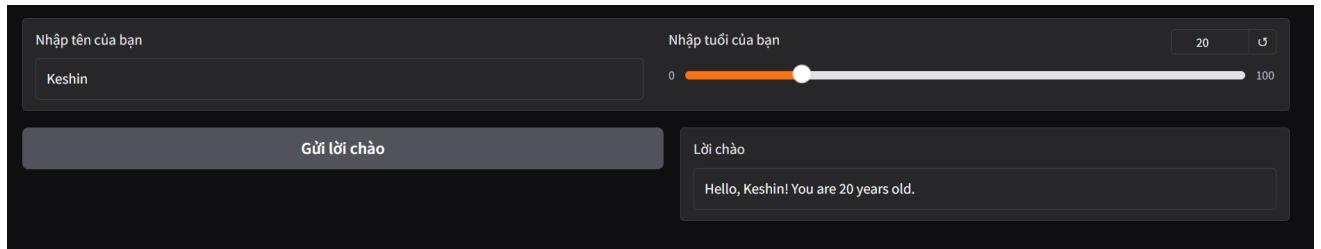
Bây giờ, hãy cùng đi sâu vào các event listener phổ biến nhất.

- button.click(): Kích hoạt khi người dùng nhấn chuột vào một component (Phổ biến nhất).
- component.change(): Được kích hoạt mỗi khi giá trị của component thay đổi.
- component.submit(): Kích hoạt khi người dùng nhấn phím Enter bên trong một component có thẻ nhập liệu.
- slider.release(): Kích hoạt khi người dùng thả chuột ra sau khi kéo một slider, một câu hỏi sẽ đặt ra ở đây: Vậy tại sao không dùng .change() ?, Đó là vì .change() sẽ kích hoạt liên tục khi người dùng đang kéo, nếu hàm xử lý của bạn nặng (ví dụ: tạo ảnh từ AI), nó sẽ gây lag và tốn tài nguyên và .release() đảm bảo hàm chỉ chạy một lần khi người dùng đã chọn xong giá trị cuối cùng.

Bây giờ chúng ta sẽ cùng thử một ví dụ đầy chính là ở mục IV.3.3 chúng ta sẽ thay đổi một chút đó là thay vì lắng nghe sự kiện .click() thì chúng ta sẽ sử dụng .release() trên thanh Slider "Nhập tuổi" với code như sau:

Ví dụ sử dụng slider.release(...)

```
1 import gradio as gr
2 def greet(name, age):
3     return f"Hello, {name}! You are {age} years old."
4 # Sử dụng Khối để tạo bối cảnh tùy chỉnh
5 with gr.Blocks() as demo:
6     with gr.Row(): # Dòng 1
7         name_input = gr.Textbox(label="Nhập tên của bạn") # Tên dạng textbox
8         age_slider = gr.Slider(label="Nhập tuổi của bạn", minimum=0, maximum=100, step=
9             1) # Tuổi dạng slider
10    with gr.Row(): # Dòng 2
11        greet_button = gr.Button("Gửi lời chào") # Button dạng button
12        greeting_output = gr.Textbox(label="Lời chào") # Output dạng textbox
13        # Slider sẽ kích hoạt hàm greet khi release, với input là name_input và output là
14        # greeting_output
15        age_slider.release(greet, inputs=[name_input, age_slider], outputs=greeting_output)
16 demo.launch()
```



V. Triển khai Object Detection cơ bản với Gradio trên Colab

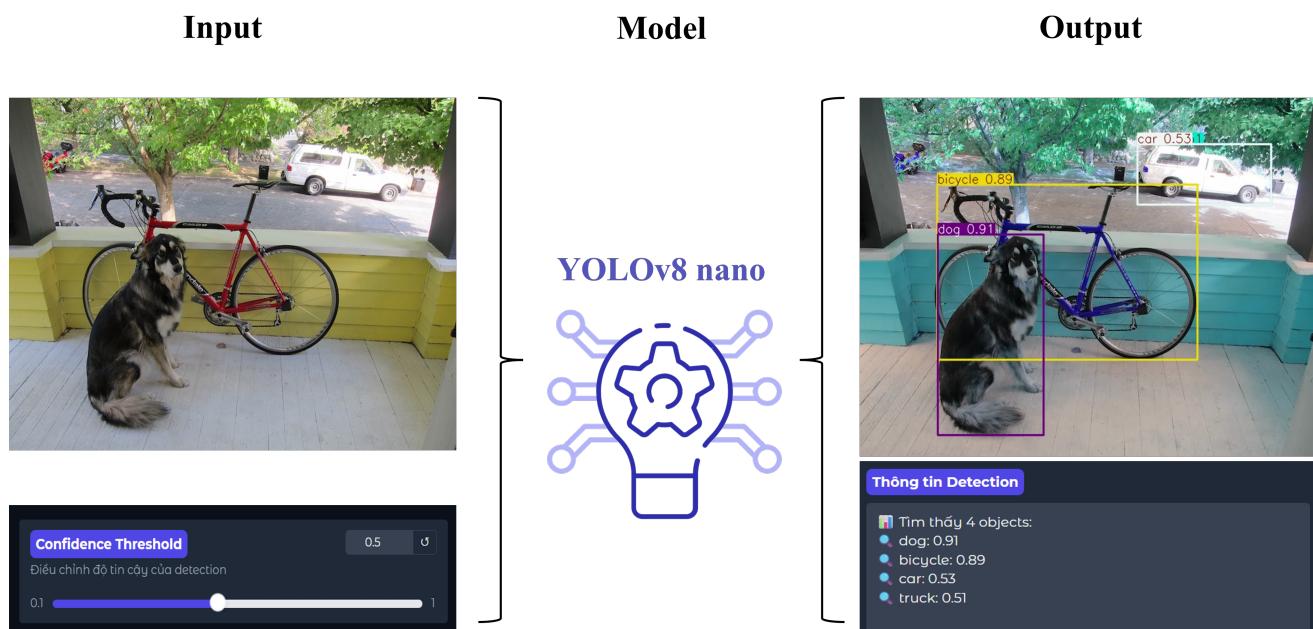
Sau khi tìm hiểu về các kiến thức cơ bản trong Gradio, bây giờ chúng ta sẽ thực hành một ví dụ thực tế hơn triển khai ứng dụng Object Detection lên website với Gradio.

V.1. Giới thiệu bài toán

Object Detection (Phát hiện đối tượng) là một bài toán cốt lõi trong lĩnh vực Computer Vision, có nhiệm vụ phát hiện, phân loại và định vị các đối tượng trong hình ảnh. Đây là nền tảng cho nhiều ứng dụng thực tế như hệ thống giám sát an ninh, xe tự lái, chẩn đoán y tế, và nhiều lĩnh vực khác. Object Detection yêu cầu hệ thống AI thực hiện đồng thời ba nhiệm vụ chính:

- Phát hiện (Detection): Tìm ra các vùng trong hình ảnh có chứa đối tượng quan tâm.
- Phân loại (Classification): Xác định đối tượng đó thuộc class nào (người, xe, động vật, v.v.).
- Định vị (Localization): Vẽ bounding box bao quanh đối tượng để chỉ rõ vị trí.

V.2. Các thành phần chính cần xác định



Hình 5: Các thành phần cần xác định

Ba thành phần chính chúng ta cần xác định trước bao gồm:

- Input sẽ bao gồm File ảnh ở các định dạng phổ biến (JPEG, PNG) và Ngưỡng tin cậy (Confidence threshold) từ 0.1 đến 1.0 để lọc kết quả.
- Về model sẽ sử dụng YOLOv8 nano với kích thước mô hình rất nhẹ và phù hợp ứng dụng do các nhu cầu triển khai trên các thiết bị đòi hỏi khả năng real-time. Bạn có thể xem chi tiết về mô hình [tại đây](#).
- Output sẽ bao gồmẢnh đã được annotate: Ảnh gốc với các bounding boxes và labels và Thông tin chi tiết bao gồm: Danh sách các đối tượng được phát hiện, Điểm tin cậy cho từng phát hiện.

V.3. Triển khai code

Bước 1: Cài đặt và thêm thư viện

Cài đặt và thêm thư viện

```

1 # Cài đặt các thư viện cần thiết
2 !pip install gradio ultralytics opencv-python pillow torch torchvision
3
4 # Import các thư viện
5 import gradio as gr
6 import cv2
7 import numpy as np
8 from PIL import Image
9 import torch
10 import requests
11 from io import BytesIO

```

Bước 2: Tải model YOLO và kiểm tra hardware

Tải model YOLO và kiểm tra hardware

```

1 device = "cuda" if torch.cuda.is_available() else "cpu"
2 print(f"Đang sử dụng: {device}")
3
4 # Load YOLO model
5 try:
6     from ultralytics import YOLO
7     model = YOLO('yolov8n.pt') # YOLOv8 nano
8     print("YOLO model đã được load thành công!")
9 except ImportError:
10     print("Cần cài đặt ultralytics: pip install ultralytics")
11     model = None

```

Bước 3: Xây dựng hàm xử lý detect objects

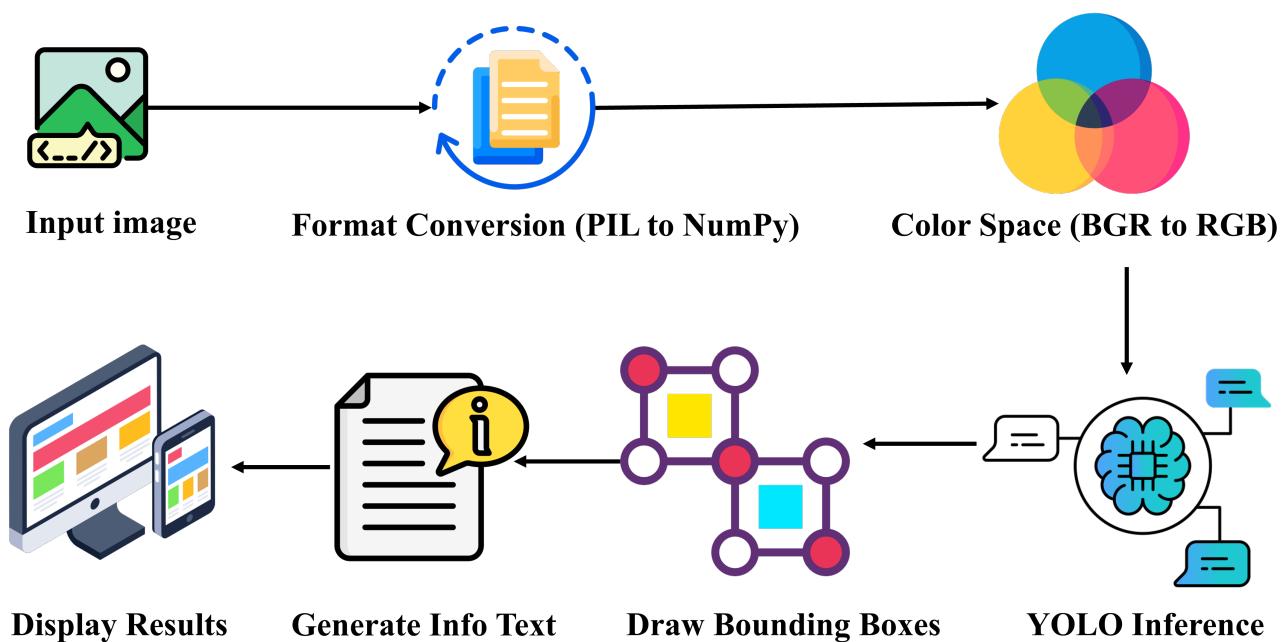
Xây dựng hàm xử lý detect objects

```

1 def detect_objects(image, confidence_threshold=0.5):
2     if model is None:
3         return image, "Model chưa được load (pip install ultralytics)"
4
5     try:
6         # Chuyển đổi PIL Image sang numpy array
7         if isinstance(image, Image.Image):
8             image_np = np.array(image)
9         else:
10            image_np = image
11
12         # Chuyển đổi BGR sang RGB nếu cần
13         if len(image_np.shape) == 3 and image_np.shape[2] == 3:
14             image_rgb = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
15         else:
16             image_rgb = image_np
17
18         # Chạy detection
19         results = model(image_rgb, conf=confidence_threshold)
20
21         # Vẽ bounding boxes
22         annotated_image = results[0].plot()
23
24         # Lấy thông tin detection
25         detections = results[0].boxes
26         detection_info = []
27
28         if detections is not None:
29             for i, box in enumerate(detections):
30                 # Lấy class name và confidence
31                 class_id = int(box.cls[0])
32                 confidence = float(box.conf[0])
33                 class_name = model.names[class_id]
34                 detection_info.append(f"{class_name}: {confidence:.2f}")
35
36         # Tạo thông tin tổng hợp
37         if detection_info:
38             info_text = f"Tim thấy {len(detection_info)} objects:\n" + "\n".join(
39                             detection_info)
40         else:
41             info_text = "Không tìm thấy object nào với confidence threshold này."
42
43         return annotated_image, info_text
44
45     except Exception as e:
46         return image, f"Loi: {str(e)}"

```

Hình số 6 sẽ giúp mọi người có thêm cái nhìn tổng quan về các quy trình trong bước số 3:



Hình 6: Detect objects pipeline

Bước 4 (Optional): Xây dựng hàm tải ảnh mẫu với mục đích tạo ảnh mẫu để test

Xây dựng hàm tải ảnh mẫu

```

1  def load_sample_image():
2      """
3          Load ảnh mẫu từ internet
4
5      Returns:
6          PIL.Image: Ảnh mẫu để test
7      """
8      try:
9          # URL ảnh mẫu từ Unsplash
10         url = "https://images.unsplash.com/photo-1552053831-71594a27632d?w=500"
11         response = requests.get(url)
12         image = Image.open(BytesIO(response.content))
13         return image
14     except:
15         # Nếu không load được ảnh từ internet, tạo ảnh mẫu
16         return Image.new('RGB', (500, 300), color='lightblue')

```

Bước 5: Tạo giao diện Gradio (Phần chính)

Đây là nơi chúng ta xây dựng giao diện người dùng và kết nối tất cả các thành phần lại với nhau. Hãy phân tích từng phần một cách chi tiết:

Khởi tạo giao diện chính

```
1 with gr.Blocks(title="Object Detection - Simple", theme=gr.themes.Soft()) as demo:
```

Giải thích:

- gr.Blocks(): Đây là container chính của Gradio, cho phép tạo layout tùy chỉnh phức tạp. Khác với gr.Interface() (đơn giản hơn), gr.Blocks() cho phép kiểm soát hoàn toàn layout và tương tác.
- title="Object Detection - Simple": Thiết lập tiêu đề cho tab browser khi mở ứng dụng.
- theme=gr.themes.Soft(): Áp dụng theme "Soft".
- as demo: Gán toàn bộ giao diện vào biến 'demo' để có thể khởi chạy sau này.

Thiết kế layout cơ bản

```
1 gr.Markdown("#Object Detection với YOLO")
2 gr.Markdown("Upload ảnh để detect objects!")
3
4 with gr.Row():
5     with gr.Column(scale=1):
6         # Input section
7         gr.Markdown("##Input")
```

Giải thích:

- gr.Markdown(): Hiển thị text với định dạng Markdown.
- gr.Row(): Tạo một hàng ngang chứa các thành phần. Mọi thứ bên trong sẽ được sắp xếp theo chiều ngang.
- gr.Column(scale=1): Tạo một cột với tỷ lệ 1. Khi có nhiều cột, scale quyết định độ rộng tương đối. Ví dụ: scale=1 và scale=2 có nghĩa cột thứ 2 rộng gấp đôi cột thứ 1.

Các thành phần Input

```
1 # Upload ảnh
2 input_image = gr.Image(
3     label="Upload ảnh",
4     type="pil",
5     height=300
6 )
7     # Confidence threshold
8 confidence_slider = gr.Slider(
9     minimum=0.1, maximum=1.0,
10    value=0.5, step=0.1,
11    label="Confidence Threshold",
12    info="Điều chỉnh độ tin cậy của detection"
13 )
```

Giải thích:

- gr.Image(): Component để upload và hiển thị hình ảnh, và type="pil": Chỉ định định dạng dữ liệu trả về là PIL Image object. Các tùy chọn khác: "numpy", "filepath".
- gr.Slider(): Thanh trượt để người dùng chọn giá trị trong một khoảng với giá trị từ 0.1 đến 1.0 (tương ứng 10% đến 100% confidence).

Cấu hình các nút cần thiết

```

1 # Buttons
2 with gr.Row():
3     detect_btn = gr.Button("Detect Objects", variant="primary", size="lg")
4     sample_btn = gr.Button("Load Sample", variant="secondary")
5     clear_btn = gr.Button("Clear", variant="secondary")

```

Giải thích:

- Sẽ bao gồm ba nút chính "Detect Objects", "Load Sample", "Clear" và được sắp xếp trên cùng một hàng.

Các thành phần Output

```

1 with gr.Column(scale=1):
2     # Output section
3     gr.Markdown("##Output")
4
5     # Kết quả detection
6     output_image = gr.Image(
7         label="Kết quả Detection",
8         height=300
9     )
10
11    # Thông tin detection
12    detection_info = gr.Textbox(
13        label="Thông tin Detection",
14        lines=8,
15        max_lines=10,
16        interactive=False
17    )

```

Giải thích

- gr.Image() cho output: Hiển thị ảnh đã được xử lý với bounding boxes.
- gr.Textbox(): Hiển thị text thông tin detection với interactive=False: Người dùng không thể chỉnh sửa textbox này, chỉ để hiển thị.

Xử lý sự kiện (Event Handlers)

```

1 # Event handlers
2 def process_detection(image, confidence):
3     if image is None:
4         return None, "Vui lòng upload ảnh trước!"
5     return detect_objects(image, confidence)
6
7 def clear_all():
8     return None, None, ""

```

Giải thích:

- Gọi hàm process_detection() để xử lý chính khi người dùng muốn detect objects.
- Gọi clear_all() để xóa xóa tất cả dữ liệu.

Kết nối Events

```

1 # Kết nối events
2 detect_btn.click(
3     process_detection,
4     inputs=[input_image, confidence_slider],
5     outputs=[output_image, detection_info]
6 )
7 sample_btn.click(
8     load_sample_image,
9     outputs=[input_image]
10 )
11 clear_btn.click(
12     clear_all,
13     outputs=[input_image, output_image, detection_info]
14 )
15 # Auto-detect khi upload ảnh
16 input_image.change(
17     process_detection,
18     inputs=[input_image, confidence_slider],
19     outputs=[output_image, detection_info]
20 )

```

Giải thích

- Ở detect_btn.click(...) sẽ gọi hàm process_detection() với giá trị input là từ 2 components input_image, confidence_slider, từ đó trả về kết quả và hiển thị ở 2 components output_image, detection_info.
- Ở sample_btn.click(...) sẽ gọi hàm load_sample_image() và hiển thị kết quả ở input_image.
- Khi click "Clear", gọi hàm clear_all() và xóa tất cả 3 components: input_image, output_image, detection_info.
- Ở input_image.change(), event được trigger khi người dùng upload ảnh mới.

Bước 6: Khởi chạy ứng dụng

Xử lý sự kiện (Event Handlers)

```

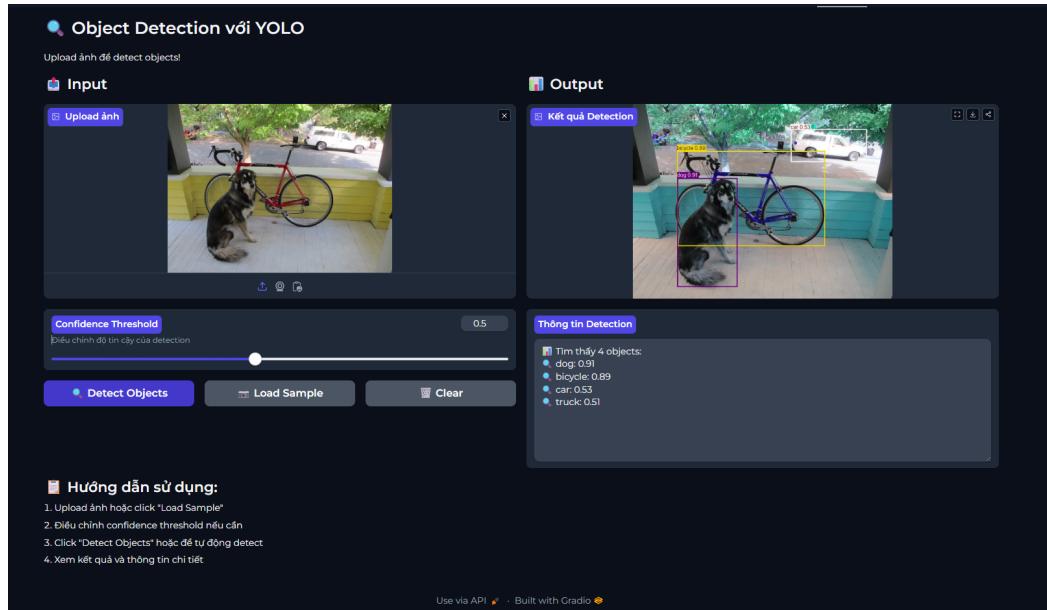
1 if __name__ == "__main__":
2     demo.launch(
3         share=True, # Tạo public link (hoạt động 72h)
4         server_name="0.0.0.0",
5         server_port=7860, # Port là 7860
6         show_error=True,
7         debug=True
8 )

```

Khi xong bước cuối cùng, chúng ta sẽ có một đường liên kết và có thể sử dụng để chia sẻ cho những người xung quanh để trải nghiệm ứng dụng mô hình AI của mình.

V.4. Triển khai ứng dụng đã xây dựng bằng Gradio lên Hugging Face Spaces

Sau khi sử dụng Gradio để tạo giao diện ứng dụng, chúng ta có thể sử dụng HuggingFace Spaces để triển khai ứng dụng này lên Internet, dễ dàng truy cập từ đường link online. Mọi người có thể truy cập bằng cách nhấn vào đường link sau đây: **Click Here** để truy cập ứng dụng OD-YOLOv8n tạo bằng Gradio đã được triển khai lên HuggingFace.



Hình 7: Giao diện của ứng dụng xây dựng bằng gradio sau khi triển khai lên HuggingFace

VI. Phụ lục

Source code: *Colab Link*

Nguồn tham khảo:

- <https://pyimagesearch.com/2025/02/03/introduction-to-gradio-for-building-interactive-applications/>
- <https://www.gradio.app/docs>
- <https://medium.com/@nimritakoul01/getting-started-with-gradio-python-library-49e59e363c66>
- <https://docs.ultralytics.com/vi/models/yolov8/#yolov8-usage-examples>