Gradio for ML Models

TimeSeries Team

Ngày 22 tháng 9 năm 2025

Mục lục

1 Giới thiệu & Mục tiêu

Bức tranh tổng quan

Bạn có một mô hình hoặc đơn giản là một hàm suy luận (inference function) trong Python và muốn **cho người khác dùng thử ngay** mà không cần tự xây toàn bộ frontend/backend. Mục tiêu của bài viết là giúp bạn đưa mô hình đó thành **ứng dụng web demo** qua Gradio theo lộ trình rõ ràng, có thể tái sử dụng cho nhiều bài toán khác nhau (thị giác máy tính, NLP, tabular...).

Tư duy cốt lõi: Input – Model – Output

Trong ngữ cảnh suy luận, ta gom bài toán về ba mảnh ghép:

Input Người dùng sẽ cung cấp dữ liệu gì? (văn bản/số/ảnh/tệp/lựa chọn...). Xác định đúng giúp ta **chọn component Gradio** phù hợp (vd. `Textbox`, `Slider`, `Image`).

Model Hàm Python nhận input và trả về kết quả. Hãy **định dạng vào/ra** ngay từ đầu (PIL Image, NumPy array, `str`, `dict`...) để tránh chuyển đổi phức tạp về sau.

Output Ta muốn hiển thị gì cho người dùng? (label/text, ảnh đã vẽ, bảng, JSON, tệp tải về...). Với kết quả chỉ để xem, nên đặt `interactive=false` để tránh chỉnh nhầm.

Cách tư duy này giúp bạn tách bạch phần UI (component) khỏi phần logic (hàm suy luận), từ đó **giảm thời gian ghép nối** và **dễ mở rộng**.

Vì sao chọn Gradio cho demo nhanh?

- **Một ngôn ngữ (Python) cho cả UI lẫn model**: không cần HTML/CSS/JS hoặc framework web riêng.
- **Component dựng sẵn**: Textbox, Slider, Image, File, Dataframe, JSON...; có thể dùng cho cả input & output.
- **Sự kiện rõ ràng**: `.click()`, `.change()`, `.submit()`, `.release()` giúp điều khiển thời điểm gọi hàm chính xác.
- **Chia sẻ nhanh**: `launch(share=true)` tạo liên kết dùng tạm thời; có thể nâng cấp lên Hugging Face Spaces để có URL bền vững.

Bạn sẽ xây được gì sau bài này?

1. Hiểu và áp dụng khung **Input – Model – Output** để thiết kế giao diện tối thiểu mà hiệu quả.
2. Dựng demo với `Interface` (một hàm duy nhất) và nâng cấp lên `Blocks` (bố cục linh hoạt, nhiều sự kiện).
3. Tổ chức layout bằng `Row/Column`, thêm các nút `Sample/Clear`, và gắn sự kiện hợp lý (ưu tiên `.release` cho slider khi tác vụ nặng).
4. Xây một **demo Object Detection** gọn nhẹ (YOLOv8n) và hiển thị ảnh *đã annotate* kèm tóm tắt kết quả.
5. Đóng gói và **đưa demo "lên sóng"** bằng Hugging Face Spaces.

Đối tượng & yêu cầu tiên quyết

- Độc giả đã quen với Python cơ bản (hàm, `virtualenv`/conda), biết chạy notebook hoặc script.
- Có tối thiểu kiến thức nhập môn về mô hình ML đang sử dụng (cách nạp model, chạy suy luận).
- Môi trường đề xuất: Python 3.10+, `pip` cập nhật; GPU là *tùy chọn* (demo dùng bản YOLO nhỏ có thể chạy CPU).

Phạm vi & không nằm trong phạm vi

Phạm vi Dựng **demo suy luận** chạy được, có giao diện; không nhằm tối ưu training hay MLOps phức tạp.

Không phạm vi Không đi sâu huấn luyện, không chuẩn hóa CI/CD hay autoscaling; không bàn chi tiết tối ưu kiến trúc mạng.

Kiến trúc tinh gọn (tư duy hệ thống)

Mục tiêu: thay vì xây "FE ↔ API ↔ Model" đầy đủ, ta *gộp* vào một tiến trình Python:

1. Nạp trọng số/mô hình.
2. Định nghĩa hàm suy luận chuẩn hóa *đầu vào/đầu ra*.
3. Khai báo UI bằng Gradio; kết nối sự kiện tới hàm suy luận.
4. Khởi chạy server cục bộ và/hoặc chia sẻ liên kết.

Cách làm này giúp **rút ngắn thời gian** từ notebook tới ứng dụng có thể tương tác, đủ tốt cho việc truyền thông nội bộ, trình bày, thu thập phản hồi.

Lộ trình nội dung của bài

1. Vì sao Gradio: lợi ích thực tế khi làm demo nhanh.
2. Khung **Input** – **Model** – **Output** và cách ánh xạ sang component.
3. `Interface` vs `Blocks`: khi nào dùng cái nào; ví dụ tối thiểu.
4. Bố cục với `Row`/`Column`, bổ sung Tab/Accordion khi cần.
5. Event listeners và các lựa chọn: `.click`, `.change`, `.submit`, `.release`.
6. Thực hành: demo Object Detection (YOLOv8n) + Gradio.
7. Đưa demo lên Hugging Face Spaces.

Tiêu chí đánh giá một bản demo "đủ tốt"

- **Rõ ràng**: hiển thị Input & Output mạch lạc; có mô tả ngắn cách dùng.
- **Ổn định**: thao tác phổ biến (upload, kéo slider, bấm nút) không gây lỗi; có nút `Clear`.
- **Nhẹ**: chạy được trên CPU/GPU phổ thông; nếu ảnh lớn, có bước resize.
- **Chia sẻ được**: `share=true` hoặc deploy lên Spaces; ghi lại `requirements.txt`.

"Hello, Gradio!" tối thiểu (tùy chọn chạy thử)

Đoạn mã cực ngắn dưới đây giúp bạn kiểm tra nhanh môi trường trước khi đi sâu:

```python
import gradio as gr

def echo(x: str) -> str:
    return f"Ban vua nhap: {x}"

demo = gr.Interface(
    fn=echo,
    inputs=gr.Textbox(label="Nhap gi do"),
    outputs=gr.Textbox(label="Phan hoi", interactive=False),
    title="Hello Gradio",
    description="Demo toi thieu de kiem tra moi truong."
)

if __name__ == "__main__":
    demo.launch(share=False)  # chuyen sang True neu muon co link tam cong khai
```

Listing 1: Hello Gradio: kiểm tra môi trường & cách launch tối thiểu

Checklist thiết lập nhanh trước khi thực hành

- Python $\geq$ 3.10, `pip` cập nhật: `python -V`, `pip -V`.
- Tạo môi trường ảo (khuyến nghị) và `pip install gradio`.
- Chạy "Hello Gradio" ở trên để xác nhận giao diện mở được.
- Với bài thực hành Object Detection: cài thêm `ultralytics`, `opencv-python`, `pillow`, `torch`, `torchvision`.

Kết nối sang các phần tiếp theo

Sau phần mở đầu, ta sẽ lần lượt:

1. Chọn `Interface` hay `Blocks` cho bài toán cụ thể.
2. Bố cục UI mạch lạc với `Row`/`Column` và các tiện ích.
3. Gắn sự kiện hợp lý để tối ưu trải nghiệm và tài nguyên.
4. Áp dụng tất cả vào demo Object Detection và triển khai "lên sóng".

2 Vì sao dùng Gradio?

Bài toán thực tế khi đưa mô hình ra dùng thử

Khi có một mô hình hoặc hàm suy luận, bạn thường muốn:

1. **Cho người khác dùng ngay** (đồng đội, giảng viên, khách hàng) mà không cần cài môi trường.
2. **Thu thập phản hồi sớm**: dữ liệu đầu vào đa dạng, góc nhìn người dùng thật, ghi nhận vấn đề và cải tiến.
3. **Trình diễn nhanh** trong workshop, tiết giảng, hoặc buổi demo nội bộ.

Nếu đi theo hướng truyền thống (FE–BE–Model), thời gian dựng hạ tầng sẽ lấn át thời gian tinh chỉnh mô hình. Gradio giải quyết nút thắt này bằng cách **gom UI + server vào Python**.

Lợi ích cốt lõi của Gradio

- **Một ngôn ngữ duy nhất (Python)**: bạn vừa gọi model, vừa dựng UI, vừa chạy server — tất cả trong một script.
- **Component UI dựng sẵn**: `Textbox`, `Slider`, `Image`, `File`, `Dataframe`, `JSON`, ... giúp mô tả *đúng* kiểu dữ liệu.
- **Event rõ ràng**: `.click()`, `.change()`, `.submit()`, `.release()` cho phép điều khiển thời điểm chạy suy luận chính xác, tránh gọi thừa.
- **Chia sẻ tức thì**: `launch(share=true)` tạo liên kết tạm công khai; dễ *mời người khác dùng thử* không cần cài đặt.
- **Triển khai tối giản**: có thể đẩy lên Hugging Face Spaces để có URL bền vững, đóng vai trò như "sandbox" để demo/POC.

So sánh nhanh với các hướng tiếp cận khác

Tiêu chí	Gradio	FE–BE tự xây (VD: React + FastAPI)
Thời gian lên demo	**Rất nhanh** (vài chục dòng)	Chậm hơn (tách repo FE, BE, CORS, build ...)
Độ linh hoạt UI/UX	Vừa đủ cho demo, POC	**Rất cao** (thiết kế tuỳ ý, scale lớn)
Độ phức tạp vận hành	Thấp (một tiến trình)	Cao (deploy nhiều dịch vụ, CI/CD)
Kiểm soát bảo mật	Vừa phải (link tạm, Spaces)	**Toàn quyền** (auth, gateway, secrets)
Thích hợp cho	Học tập, demo nhanh, nội bộ	Sản phẩm production, SLA, traffic lớn

Khi nào nên dùng Gradio?

- **Demo nhanh/MVP**: một hàm suy luận, một vài tham số, cần link để người khác thử.
- **Bài giảng/workshop**: tập trung vào mô hình hơn là hạ tầng, sinh viên chạy được ngay.
- **Nội bộ/POC**: xác nhận tính hữu dụng trước khi đầu tư xây hạ tầng bài bản.

Khi nào chưa phù hợp?

- **Yêu cầu UI/UX phức tạp**: workflow nhiều bước, realtime đa thiết bị, brand guideline chặt.
- **Ràng buộc bảo mật/ngân hàng/y tế**: cần chuẩn hoá IAM, audit log, phân vùng mạng nghiêm ngặt.
- **SLA/traffic lớn**: cần auto-scaling, canary, quan sát hệ thống đầy đủ (metrics, tracing).

Mô hình kiến trúc tối giản (tư duy hệ thống)

Gradio hướng tới **đường đi ngắn nhất** từ mã Python tới app có thể tương tác:

1. **Nạp mô hình/trọng số** khi khởi động tiến trình.
2. **Định nghĩa hàm suy luận** (chuẩn hoá vào/ra, xử lý lỗi, giới hạn kích thước dữ liệu).
3. **Khai báo UI**: chọn component khớp kiểu dữ liệu; tạo bố cục bằng `Row`/`Column`/`Tab`.
4. **Gắn sự kiện**: kết nối component → hàm suy luận bằng `.click()`, `.release()`, ...
5. **Launch**: chạy server cục bộ; tuỳ chọn `share=true` để chia sẻ nhanh.

Ví dụ so sánh độ phức tạp (ý tưởng)

Không cần *routing*, *schema request/response*, *template HTML*. Thay vào đó:

```python
import gradio as gr

def predict(text, temp: float):
    # ... goi model NLP/LLM/Classifier ...
    return f"Output voi temp={temp:.2f}: {text[::-1]}"

with gr.Blocks(title="Demo nhanh") as demo:
    with gr.Row():
        with gr.Column(scale=1):
            inp = gr.Textbox(label="Nhap van ban")
            temp = gr.Slider(0.0, 1.0, value=0.5, step=0.05, label="Temperature")
            run = gr.Button("Chay")
        with gr.Column(scale=2):
            out = gr.Textbox(label="Ket qua", interactive=False, lines=6)
    run.click(predict, inputs=[inp, temp], outputs=out)

demo.launch(share=False)
```

Listing 2: Từ hàm Python đến app có giao diện trong vài dòng

Hiệu năng & trải nghiệm người dùng (UX)

- **Giảm số lần suy luận thừa**: dùng `.release` với `Slider` thay vì `.change` cho tác vụ nặng.
- **Hạn chế kích thước dữ liệu**: resize ảnh lớn trước khi suy luận; từ chối tệp quá dung lượng.
- **Phản hồi rõ ràng**: dùng `interactive=false` cho output; hiển thị thông điệp "đang xử lý" khi tác vụ lâu.
- **Hàng đợi**: bật `queue=True` nếu dự kiến nhiều yêu cầu đồng thời.

Bảo mật & chia sẻ

- **share=true**: liên kết tạm thời, tiện thử nghiệm; không nên dùng cho dữ liệu nhạy cảm.
- **Hugging Face Spaces**: URL công khai bền vững; có thể cấu hình *secrets* và quyền truy cập.
- **Dọn dữ liệu tạm**: đảm bảo không lưu trữ bản sao dữ liệu người dùng trái ý; xoá cache nếu cần.

Chi phí vận hành & bảo trì

- **Tài nguyên**: CPU/GPU tuỳ mô hình; dùng biến thể "nhẹ" cho demo để tối ưu chi phí.
- **Phụ thuộc gói**: ghi rõ phiên bản trong `requirements.txt` để tránh lỗi build khi triển khai.
- **Giới hạn**: demo không thay thế hạ tầng production (monitoring, canary, rollback).

Checklist ra quyết định "có dùng Gradio không"

Mục tiêu là **demo/POC**, không phải sản phẩm production dài hạn.

Có thể mô tả bài toán với **Input–Model–Output** rõ ràng.

UI chỉ cần **form đơn giản** (upload, slider, nút chạy) và vùng hiển thị kết quả.

Chấp nhận cơ chế **chia sẻ tạm** hoặc deploy nhẹ (Spaces).

Kết nối sang phần kế tiếp

Sau khi nắm lý do sử dụng Gradio, phần tiếp theo sẽ trình bày **khung tư duy Input–Model–Output** chi tiết hơn và cách ánh xạ trực tiếp sang các component của Gradio để bạn bắt tay vào dựng UI đúng ngay từ đầu.

3 Khung tư duy: Input – Model – Output

Mục tiêu của phần này

Xây một demo vững chắc bắt đầu từ việc **định nghĩa hợp đồng dữ liệu** (data contract) giữa UI và hàm suy luận. Phần này giúp bạn:

- Xác định chính xác **Input** người dùng đưa vào.
- Đặt chuẩn **Model function** (chữ ký hàm, kiểu dữ liệu, xử lý lỗi).
- Thiết kế **Output** dễ đọc, dễ kiểm thử, ít bất ngờ.

3.1 Đặt "hợp đồng dữ liệu" ngay từ đầu

Một hợp đồng dữ liệu nên chỉ rõ:

1. **Loại dữ liệu** (text, số, ảnh, tệp, danh sách...).
2. **Miền giá trị hợp lệ** (VD: temperature $\in [0, 1]$, kích thước ảnh $\leq$ 2k×2k).
3. **Ràng buộc** (bắt buộc/tuỳ chọn, số lượng tối đa tệp).
4. **Định dạng trả về** (chuỗi, JSON, ảnh đã vẽ, tệp ZIP...).

Lợi ích: giúp bạn chọn đúng component của Gradio, và viết mã suy luận *ít "if-else" chữa cháy.*

3.2 Ánh xạ Input ↔ Component

Kiểu input	Component gợi ý	Ghi chú triển khai
Văn bản ngắn/dài	`Textbox` / `TextArea`	`lines` phù hợp; hỗ trợ `.submit()`
Số, tham số liên tục	`Number`, `Slider`	Dùng `Slider` kèm `.release()` cho tác vụ nặng
Lựa chọn rời rạc	`Radio`, `Dropdown`	Thiết lập `choices`, `value` mặc định
Ảnh	`Image`	`type="pil"` hoặc `"numpy"`; resize trước khi suy luận
Tệp bất kỳ	`File`	Kiểm tra kích thước/định dạng; dọn tệp tạm
Nhiều mục	`CheckboxGroup`, `Gallery`	Chú ý ép kiểu sang danh sách

3.3 Thiết kế hàm `model_fn`: chữ ký, kiểm tra, trả về

Mẫu chữ ký hàm khuyến nghị:

```
from typing import Tuple, Dict, Any, Optional
from PIL import Image
import numpy as np

def model_fn(
    text: Optional[str] = None,
    image: Optional[Image.Image] = None,
    threshold: float = 0.5,
    options: Optional[Dict[str, Any]] = None
) -> Tuple[Any, Dict[str, Any]]:
    """
```

```
    Tra ve (primary_output, meta_info).
    - primary_output: du lieu chinh de hien thi (str/Image/ndarray/JSON).
    - meta_info: thong tin bo sung (thoi gian xu ly, tham so, canh bao...).
    """
    # 1) Validate
    if (text is None) and (image is None):
        return "Vui long nhap van ban hoac upload anh.", {"ok": False}

    if not (0.0 <= threshold <= 1.0):
        threshold = max(0.0, min(1.0, threshold))

    # 2) Chuyen doi kieu (neu can)
    if image is not None and not isinstance(image, Image.Image):
        image = Image.fromarray(np.asarray(image))

    # 3) Goi mo hinh / suy luan
    # ... your inference here ...
    result = {"message": "OK", "threshold": threshold}

    # 4) Tao output chinh + meta
    primary_output = str(result)  # vi du: tra JSON dang chuoi
    meta = {"ok": True, "params": {"threshold": threshold}}
    return primary_output, meta
```

Listing 3: Chữ ký hàm suy luận và xử lý lỗi nhất quán

Nguyên tắc: luôn trả về cùng một *hình dạng dữ liệu* để UI hiển thị ổn định.

3.4 Tiền xử lý (preprocess) & Hậu xử lý (postprocess)

Tiền xử lý

- Chuẩn hoá text (strip, lower, bỏ ký tự lạ nếu cần).
- Với ảnh: convert sang RGB, resize tối đa, chuẩn hoá dtype.
- Kiểm tra kích thước tệp, số lượng tệp.

Hậu xử lý

- Định dạng điểm số, nhãn (`f"score:.2f"`).
- Vẽ bbox/overlay lên ảnh (nếu là CV).
- Kèm **meta info**: thời gian xử lý, tham số đã dùng.

3.5 Thiết kế Output: dễ đọc, kiểm thử, và nhất quán

Một số mẫu output phổ biến:

1. **Chữ (textbox/markdown)**: ngắn gọn, có gạch đầu dòng, nêu điểm chính.
2. **Ảnh** đã vẽ sẵn: trả về PIL Image/ndarray đã annotate.
3. **Bảng/JSON**: dùng `Dataframe`/`JSON` để hiển thị cấu trúc.
4. **Tệp tải về**: zip/csv kết quả hàng loạt.

Quy tắc vàng:

- `interactive=False` cho phần chỉ để xem.
- Trả về **meta** (dictionary) để debug/test A/B.
- Giao diện **không thay đổi hình dạng** khi giá trị biên (VD: "không phát hiện gì") — luôn hiển thị khung kết quả ổn định.

3.6 Ví dụ 1: NLP đơn giản (text + slider → text)

```
import gradio as gr
import time

def nlp_demo(text: str, temperature: float = 0.5):
    start = time.time()
    if not text or text.strip() == "":
        return "Vui long nhap van ban.", {"ok": False}
    # Fake generation: dao chuoi lam vi du
    out = f"[temp={temperature:.2f}] " + text[::-1]
    meta = {"ok": True, "elapsed": round(time.time()-start, 3), "temp": temperature}
    return out, meta

with gr.Blocks(title="NLP Demo") as demo:
    with gr.Row():
        with gr.Column(scale=1):
            inp = gr.Textbox(label="Nhap van ban")
            temp = gr.Slider(0.0, 1.0, value=0.5, step=0.05, label="Temperature")
            run  = gr.Button("Chay")
        with gr.Column(scale=2):
            out = gr.Textbox(label="Ket qua", interactive=False, lines=6)
            meta= gr.JSON(label="Meta")
    run.click(nlp_demo, inputs=[inp, temp], outputs=[out, meta])
    temp.release(nlp_demo, inputs=[inp, temp], outputs=[out, meta])  % dung .release
    tranh goi lien tuc
demo.launch()
```

Listing 4: Input text, tham so slider; Output text + meta

3.7 Ví dụ 2: Ảnh (image + threshold → image + text)

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

def fake_detect(image: Image.Image, thr: float = 0.3):
    if image is None:
        return None, "Vui long upload anh."
    # Vi du: ''vach'' mot strip mau len anh de minh hoa hau xu ly
    arr = np.array(image.convert("RGB"))
    h, w, _ = arr.shape
    arr[:max(5, int(h*thr)), :, :] = 255  # ke mot vach trang theo threshold
    annotated = Image.fromarray(arr)
    info = f"Da annotate: strip={int(h*thr)}px (thr={thr:.2f})"
    return annotated, info

with gr.Blocks(title="Image Demo") as app:
    with gr.Row():
        with gr.Column(scale=1):
            img = gr.Image(label="Anh dau vao", type="pil")
            thr = gr.Slider(0.05, 0.95, value=0.3, step=0.05, label="Threshold")
```

```
            run = gr.Button("Annotate")
        with gr.Column(scale=2):
            out_img = gr.Image(label="Anh da annotate", interactive=False)
            out_txt = gr.Textbox(label="Thong tin", interactive=False)
    run.click(fake_detect, inputs=[img, thr], outputs=[out_img, out_txt])
    thr.release(fake_detect, inputs=[img, thr], outputs=[out_img, out_txt])
app.launch()
```

Listing 5: Input PIL Image; Output Image da annotate + thong tin tom tat

3.8 Nhiều Input/Output: quy ước thứ tự và kiểu

Khi có nhiều đầu vào/đầu ra, hãy đảm bảo:

- Thứ tự `inputs=[...]` và `outputs=[...]` khớp chữ ký hàm.
- Kiểu trả về đúng (VD: `Tuple[Image, str]`).
- **Không thay đổi số lượng** output theo trường hợp (kể cả khi lỗi).

```
def multi_io(text: str, image: Image.Image, k: int = 3):
    if image is None or not text:
        return None, "Thieu du lieu dau vao.", {"ok": False}
    # ... suy luan ...
    annotated = image  # vi du
    summary = f"Text={text[:20]}..., topK={k}"
    meta = {"ok": True, "k": k}
    return annotated, summary, meta

# outputs phai nhan 3 muc tuong ung
btn.click(multi_io, inputs=[inp_text, inp_img, inp_k], outputs=[out_img, out_txt,
    out_meta])
```

Listing 6: Nhieu input/output va tra ve on dinh

3.9 Xử lý lỗi và trạng thái biên (edge cases)

- **Loại dữ liệu sai**: hiển thị thông báo thay vì raise exception lan ra UI.
- **Dữ liệu trống/quá lớn**: trả thông điệp hướng dẫn (kích thước tối đa, định dạng hợp lệ).
- **Không có kết quả**: vẫn trả về khung output cố định (VD: ảnh gốc + "không phát hiện").

3.10 Hiệu năng và ổn định

- Tiền xử lý nhẹ (resize ảnh), `.release` cho slider, `queue=True` khi nhiều người dùng.
- Tách phần nặng ra ngoài hàm sự kiện (load model một lần ở global scope).
- Cache kết quả trung gian nếu hợp lý (nhưng cân nhắc bộ nhớ).

3.11 Bảo mật dữ liệu (ở mức demo)

- Hạn chế lưu trữ dữ liệu người dùng; dọn tệp tạm sau khi xử lý.
- Không log thông tin nhạy cảm; nêu rõ phạm vi sử dụng ở phần mô tả.
- Khi chia sẻ liên kết, nhắc người dùng không đưa dữ liệu cá nhân.

Tóm tắt

"Input – Model – Output" là xương sống của mọi demo Gradio. Khi hợp đồng dữ liệu rõ ràng, bạn sẽ:

- Chọn component chính xác, tránh đổi kiểu nhiều lần.
- Viết hàm suy luận sạch, dễ kiểm thử.
- Hiển thị kết quả nhất quán, thân thiện với người dùng.

Phần tiếp theo sẽ áp dụng khung này để so sánh và chọn giữa `Interface` và `Blocks`.

4 Các component hay dùng

Mục tiêu của phần này

Chọn đúng component giúp:

- Giảm thời gian ghép nối giữa UI và hàm suy luận.
- Hạn chế lỗi ép kiểu (PIL → NumPy, `str` → `float` ...).
- Tạo trải nghiệm sử dụng tự nhiên với **ít tuỳ chọn nhưng đủ**.

4.1 Nguyên tắc chọn component

1. **Trung thực với kiểu dữ liệu**: nhập số dùng `Slider`/`Number`; ảnh dùng `Image`; tệp dùng `File`.
2. **Giới hạn sớm**: đặt ràng buộc tại UI (min/max, step, choices) để giảm nhánh kiểm tra ở backend.
3. **Giảm tự do nếu không cần**: dùng `Radio`/`Dropdown` thay vì `Textbox` tự do cho tham số phân loại.
4. **Output chỉ để xem**: thêm `interactive=false` để tránh người dùng chỉnh nhầm.

4.2 Nhóm Input cơ bản

Textbox / TextArea

- Dùng cho văn bản tự do (prompt, câu hỏi, mô tả).
- Thuộc tính gợi ý: `lines`, `max_lines`, `placeholder`.
- Gợi ý sự kiện: `.submit()` để nhấn Enter chạy luôn.

Number / Slider

- Dùng cho số/tham số liên tục (confidence, temperature).
- Thuộc tính: `minimum`, `maximum`, `step`, `value` (mặc định).
- Với tác vụ nặng: ưu tiên `.release()` thay vì `.change()` khi kéo `Slider`.

Radio / Dropdown / Checkbox / CheckboxGroup

- Thích hợp cho lựa chọn ràng buộc (một hoặc nhiều).
- Đặt `choices` (danh sách), `value` (mặc định).
- Tương ứng ở back-end: map sang `Enum` hoặc `set` để xử lý rõ ràng.

File

- Dùng khi loại tệp không cố định (CSV, ZIP, PDF...).
- Kiểm tra kích thước/đuôi tệp trong hàm suy luận; dọn tệp tạm nếu có ghi ra đĩa.

Image

- Chọn `type="pil"` để nhận `PIL.Image` hoặc `type="numpy"` để nhận `ndarray`.
- Nên **resize** ảnh lớn (ví dụ cạnh dài ≤ 1280) trước khi suy luận trên CPU.

Audio / Video

- Trả về đường dẫn tạm hoặc mảng mẫu tuỳ chế độ; cân nhắc thời lượng tối đa.
- Với Video, xem xét trích keyframe trước khi đưa vào mô hình nếu inference nặng.

4.3 Nhóm Output phổ biến

Textbox / Label / Markdown

- Dùng cho kết quả ngắn gọn, chú thích, hoặc định dạng nhẹ (Markdown).
- Đặt `interactive=false` cho output để tránh sửa nhầm.

Image / Gallery

- Cho thị giác máy tính (ảnh đã annotate, segmentation overlay).
- `Gallery` phù hợp hiển thị nhiều ảnh (top-k, trước/sau).

JSON / Dataframe / HTML

- `JSON`: debug thông số, trả cấu trúc kết quả.
- `Dataframe`: bảng kết quả, thống kê.
- `HTML`: mô tả giàu định dạng; thận trọng với nội dung người dùng nhập vào (escape).

File (tải về)

- Gộp kết quả hàng loạt thành ZIP/CSV cho người dùng tải.
- Đặt tên tệp rõ ràng, có timestamp để phân biệt phiên chạy.

4.4 Mẫu ánh xạ nhanh Input → Output

Bài toán	**Input**	**Output**
Phân tích cảm xúc (NLP)	`Textbox` (câu)	`Label` + `JSON` (score)
Tóm tắt văn bản	`Textbox` (đoạn)	`Textbox`/`Markdown` (tóm tắt)
Object Detection	`Image` + `Slider` (conf)	`Image` (annotate) + `Textbox` (tóm tắt)
Phân lớp ảnh	`Image`	`Label` + `JSON` (top-k)
OCR	`Image`/`File` (PDF)	`Textbox` (văn bản) + `File` (.txt)
Batch inference	`File` (CSV/ZIP)	`File` (CSV/ZIP kết quả)

4.5 Ví dụ 1: Textbox + Slider → Textbox + JSON

```python
import gradio as gr, time, random

def nlp_infer(text: str, temperature: float = 0.5):
    t0 = time.time()
    if not text or text.strip()=="":
        return "Vui long nhap van ban.", {"ok": False}
    # minh hoa: dao chu va chen nhieu tu theo temperature
    words = text.split()[::-1]
    k = max(1, int(len(words)*temperature))
    out = " ".join(words[:k])
    meta = {"ok": True, "elapsed": round(time.time()-t0,3), "temp": temperature, "len"
    : len(words)}
    return out, meta

with gr.Blocks(title="NLP Example") as demo:
    with gr.Row():
        with gr.Column(scale=1):
            inp = gr.Textbox(label="Nhap van ban", lines=4, placeholder="Vi du: Xin
    chao ...")
            temp = gr.Slider(0.0, 1.0, value=0.5, step=0.05, label="Temperature")
            run = gr.Button("Chay")
        with gr.Column(scale=2):
            out = gr.Textbox(label="Ket qua", interactive=False, lines=6)
            meta= gr.JSON(label="Meta")
    run.click(nlp_infer, inputs=[inp, temp], outputs=[out, meta])
    temp.release(nlp_infer, inputs=[inp, temp], outputs=[out, meta])
demo.launch()
```

Listing 7: NLP: tham so temperature va ket qua dang text + JSON

4.6 Ví dụ 2: Image + Slider → Image + Textbox

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

def annotate(image: Image.Image, thr: float = 0.3):
    if image is None:
        return None, "Vui long upload anh."
    arr = np.array(image.convert("RGB"))
    h, w, _ = arr.shape
    # ve 1 thanh ngang ti le theo thr -> minh hoa hau xu ly
    hh = max(6, int(h*thr))
    arr[:hh,:,:] = [255, 255, 255]
    return Image.fromarray(arr), f"Da ve strip {hh}px (thr={thr:.2f})"

with gr.Blocks(title="Image Example") as app:
    with gr.Row():
        with gr.Column(scale=1):
            img = gr.Image(label="Anh dau vao", type="pil")
            thr = gr.Slider(0.05, 0.95, value=0.3, step=0.05, label="Threshold")
            run = gr.Button("Annotate")
        with gr.Column(scale=2):
            out_img = gr.Image(label="Anh da annotate", interactive=False)
            out_txt = gr.Textbox(label="Thong tin", interactive=False, lines=4)
    run.click(annotate, inputs=[img, thr], outputs=[out_img, out_txt])
    thr.release(annotate, inputs=[img, thr], outputs=[out_img, out_txt])
```

```
app.launch()
```

Listing 8: CV: annotate gia lap theo threshold

4.7 Xử lý file và media an toàn

- **Giới hạn kích thước**: từ chối tệp quá lớn (thông báo rõ ràng).
- **Kiểm tra đuôi tệp**: chỉ nhận các định dạng bạn hỗ trợ (CSV/JPG/PNG ...).
- **Dọn tệp tạm**: nếu lưu ra ổ đĩa, xóa tệp sau khi trả kết quả.
- **Resize/convert sớm**: chuẩn hoá ảnh (RGB, kích thước) để giảm lỗi downstream.

4.8 Mẹo UI/UX nhỏ nhưng hiệu quả

- **Placeholder** mô tả rõ *đầu vào hợp lệ*.
- **Giá trị mặc định** hợp lý: `Slider` bắt đầu ở giá trị thường dùng.
- **Nút Clear/Sample**: cho phép thử nhanh và reset trạng thái.
- **Trạng thái rỗng**: khi chưa có kết quả, vẫn hiển thị khung output để tránh "nhảy UI".

Tóm tắt

Chọn component đúng giúp demo mạch lạc, ít lỗi ép kiểu và tối ưu chi phí tính toán. Hãy:

1. Bắt đầu từ kiểu dữ liệu **thật** mà mô hình cần.
2. Giới hạn sớm bằng `choices`, `min/max`, `step`.
3. Sử dụng `interactive=false` cho output và thêm **Clear/Sample**.

Phần kế tiếp sẽ so sánh cụ thể giữa `Interface` và `Blocks`, kèm tiêu chí chọn và ví dụ "đổi cỡ" từ bản đơn giản sang bản linh hoạt.

5 Hai cách dựng giao diện: Interface vs. Blocks

Mục tiêu của phần này

Làm rõ khi nào dùng `Interface` để *lên demo thật nhanh*, khi nào dùng `Blocks` để *kiểm soát UI/UX và luồng tương tác phức tạp*. Đồng thời, đưa ra lộ trình **nâng cấp** (migrate) từ Interface → Blocks *mà không phải viết lại logic suy luận.*

5.1 Interface: đường tắt cho MVP/POC

Khi nào dùng?

- Bài toán có **một hàm suy luận trung tâm** (input → output).
- Tham số ít, thao tác tuyến tính: nhập → bấm nút → xem kết quả.
- Mục tiêu là **có link chạy được ngay** để trình diễn/thu phản hồi.

Đặc trưng

- Khai báo ngắn gọn: `fn`, `inputs`, `outputs`.
- Tự sinh UI với layout tối thiểu; ít phải quan tâm bố cục.
- Dễ đọc mã, ít "keo dán" giữa UI và backend.

```python
import gradio as gr

def score_sentiment(text: str, threshold: float = 0.5):
    if not text or text.strip()=="":
        return "Vui long nhap van ban."
    # minh hoa: diem > threshold -> "pos" nguoc lai "neg"
    s = (sum(map(ord, text)) % 101) / 100.0
    label = "positive" if s >= threshold else "negative"
    return f"score={s:.2f} -> {label} (thr={threshold:.2f})"

demo = gr.Interface(
    fn=score_sentiment,
    inputs=[gr.Textbox(label="Van ban"), gr.Slider(0,1,0.5,0.01,label="Threshold")],
    outputs=gr.Textbox(label="Ket qua", interactive=False),
    title="Sentiment MVP",
    description="Nhap van ban, dieu chinh threshold, nhan ket qua."
)
demo.launch()
```

Listing 9: Interface: demo một-hàm

Ưu/nhược

- **Ưu**: Nhanh, ít mã, gần như không thể sai layout; rất hợp để dạy/hackathon/POC.
- **Nhược**: Khó làm **nhiều luồng** (multi-step), bố cục phức tạp, hoặc nhiều sự kiện tinh chỉnh.

5.2 Blocks: "lego" để mở rộng UI/UX và luồng

Khi nào dùng?

- Cần **bố cục** (hàng/cột/tab/accordion), widget phụ trợ (Clear, Sample, Download).
- Có **nhiều sự kiện** (click, change, submit, release) và **nhiều hàm** xử lý.
- Muốn giữ **trạng thái** giữa các thao tác, hoặc tách nhiều pipeline.

Đặc trưng

- Dựng UI theo ngữ nghĩa: `with gr.Blocks(): with gr.Row(): with gr.Column(): ...`
- **Kết nối sự kiện** chi tiết: `.click()`, `.change()`, `.submit()`, `.release()` ...
- Hỗ trợ tốt việc **tổ chức nhiều hàm suy luận** và phối hợp kết quả.

```python
import gradio as gr

def preprocess(x):
    return x.strip()

def infer(x, t):
    if not x: return "Trong.", {"ok": False}
    score = (sum(map(ord, x)) % 101) / 100.0
    label = "positive" if score >= t else "negative"
    return f"score={score:.2f} -> {label}", {"ok": True, "thr": t, "len": len(x)}

def clear_all():
    return "", "", {}

with gr.Blocks(title="Sentiment Advanced", queue=True) as app:
    gr.Markdown("## Sentiment (Blocks)")
    with gr.Row():
        with gr.Column(scale=1):
            raw = gr.Textbox(label="Van ban", lines=4)
            thr = gr.Slider(0,1,0.5,0.01,label="Threshold")
            with gr.Row():
                btn_clean = gr.Button("Preprocess")
                btn_run   = gr.Button("Infer")
                btn_clear = gr.Button("Clear")
        with gr.Column(scale=2):
            cleaned = gr.Textbox(label="Sau preprocess", interactive=False)
            output  = gr.Textbox(label="Ket qua", interactive=False, lines=4)
            meta    = gr.JSON(label="Meta")

    btn_clean.click(preprocess, inputs=raw, outputs=cleaned)
    btn_run.click(infer, inputs=[cleaned, thr], outputs=[output, meta])
    thr.release(infer, inputs=[cleaned, thr], outputs=[output, meta])
    btn_clear.click(clear_all, outputs=[cleaned, output, meta])

app.launch()
```

Listing 10: Blocks: bo cuc + nhieu su kien

Ưu/nhược

- **Ưu**: Rất linh hoạt; dễ tạo **trải nghiệm giàu tương tác** và quản lý nhiều bước.
- **Nhược**: Nhiều mã hơn Interface; cần suy nghĩ **kiến trúc sự kiện** để tránh rối.

5.3 Tiêu chí chọn nhanh

Tình huống	Nên chọn	Lý do
Một hàm suy luận, 1–2 tham số	Interface	Lên demo tức thì, ít mã, khó sai
Cần bố cục 2 cột + nút Clear/Sample	Blocks	Control layout + nhiều sự kiện
Nhiều pipeline/luồng (tiền xử lý, suy luận, hậu xử lý)	Blocks	Tách hàm, kết nối sự kiện minh bạch
Prototype cực nhanh để xin ý kiến	Interface (sau đó nâng cấp)	Ra link nhanh, rồi migrate dần

5.4 Mẫu nâng cấp: từ Interface → Blocks

```
demo = gr.Interface(fn=infer, inputs=[gr.Textbox(), gr.Slider(0,1,0.5,0.01)],
                    outputs=gr.Textbox())
demo.launch()
```

```
def preprocess(x): return x.strip()
def infer(x, t):  # giu nguyen chu ky
    # ... suy luan ...
    return "ket qua"
```

```
with gr.Blocks() as app:
    with gr.Row():
        with gr.Column(scale=1):
            raw = gr.Textbox()
            thr = gr.Slider(0,1,0.5,0.01)
            clean = gr.Button("Preprocess")
            run   = gr.Button("Infer")
        with gr.Column(scale=2):
            x     = gr.Textbox(label="Sau preprocess", interactive=False)
            y     = gr.Textbox(label="Ket qua", interactive=False)
    clean.click(preprocess, raw, x)
    run.click(infer, [x, thr], y)
app.launch()
```

Điểm mấu chốt: *không thay đổi* chữ ký `infer`; chỉ thay UI gọi vào đúng chỗ. Nhờ đó, việc nâng cấp không đụng tới logic mô hình.

5.5 Các mẫu thiết kế (patterns) thường gặp

Bước 3: Blocks với bố cục + sự kiện

Pattern 1: "Form trái – Kết quả phải"

- Trái: input + tham số + nút hành động.
- Phải: kết quả (image/text) + meta (JSON/log).

Pattern 2: "Tab theo chức năng"

- Tab 1: Upload/nhập liệu.
- Tab 2: Kết quả/biểu đồ.
- Tab 3: Cấu hình nâng cao/Logs.

Pattern 3: "Pipeline nhiều bước"

- Nút `Preprocess` → `Infer` → `Postprocess/Export`.
- Mỗi bước một hàm; kết quả bước trước là input bước sau.

5.6 Bẫy thường gặp và cách tránh

- **Gọi model quá nhiều lần**: tránh `.change` với Slider cho tác vụ nặng, dùng `.release`.
- **Không giữ định dạng output**: luôn trả về cùng kiểu/shape để UI không "nhảy".
- **Trộn logic vào UI**: tách hàm `preprocess/infer/postprocess` riêng, UI chỉ "điều phối".
- **Thiếu nút Clear/Sample**: khó kiểm thử nhanh, khó reset trạng thái.

5.7 Hiệu năng & khả năng mở rộng

- Đặt tải mô hình (`load weights`) ở *global scope* để không nạp lại mỗi lần.
- Dùng `queue=True` khi chờ nhiều người dùng; cân nhắc batch nếu phù hợp.
- Với ảnh/video lớn: resize/giới hạn độ dài để giảm độ trễ.

Tóm tắt

`Interface` cho tốc độ và sự đơn giản; `Blocks` cho kiểm soát và mở rộng. Lộ trình hợp lý là:

1. **Bắt đầu** với Interface để có demo chạy ngay.
2. **Giữ** logic suy luận độc lập với UI.
3. **Nâng cấp** lên Blocks khi cần bố cục/sự kiện phức tạp, mà không đổi chữ ký hàm.

Ở phần tiếp theo, ta sẽ **bố cục UI** bằng Row/Column, Tab, Accordion để tạo trải nghiệm gọn gàng và dễ dùng.

6 Bố cục UI: Row/Column, Tab, Accordion

Mục tiêu của phần này

Tổ chức giao diện sao cho:

- **Nhìn thấy ngay** các thao tác chính (nhập liệu, tham số, nút chạy).
- **Kết quả nổi bật**, không bị lẫn trong phần cấu hình.
- Có **không gian mở rộng** (nhiều chức năng, nhật ký, cấu hình nâng cao) mà không làm rối mắt.

6.1 Nguyên tắc bố cục tổng quát

1. **1 màn hình = 1 mục tiêu**: trang chính chỉ nên phục vụ một *luồng sử dụng* ưu tiên.
2. **Tách khu nhập liệu và kết quả**: bố trí "*Form bên trái – Kết quả bên phải*" cho hầu hết demo suy luận.
3. **Giảm chuyển động UI**: hạn chế thay đổi kích thước/thứ tự phần tử khi có kết quả mới; giữ khung kết quả cố định.
4. **Tối thiểu cuộn**: ưu tiên `Row/Column` thay vì xếp dọc quá dài; dùng Tab/Accordion cho phần phụ.

6.2 Row/Column: xương sống của layout

Khái niệm `Row` sắp xếp phần tử theo hàng ngang; `Column` sắp xếp theo cột dọc. Có thể lồng nhau để tạo bố cục hai cột, nhiều hàng.

Tỉ lệ cột với `scale` `scale` xác định tỉ lệ phân chia không gian trong một `Row`. Ví dụ: trái 1, phải 2 (1 : 2).

```
import gradio as gr

with gr.Blocks(title="Layout Co Ban") as demo:
    gr.Markdown("## Demo: Form trai -- Ket qua phai")
    with gr.Row():
        #  C t  tr i (scale=1):  n h p  l i u  v  tham  s
        with gr.Column(scale=1):
            inp = gr.Textbox(label="Nhap van ban", lines=4, placeholder="Vi du...")
            thr = gr.Slider(0, 1, value=0.5, step=0.05, label="Threshold")
            with gr.Row():
                btn_run   = gr.Button("Chay")
                btn_clear = gr.Button("Clear")
        #  C t  p h i  (scale=2):  k ỉt  q u  v  meta
        with gr.Column(scale=2):
            out = gr.Textbox(label="Ket qua", interactive=False, lines=8)
            meta= gr.JSON(label="Meta")
    #  G n  s  k i n  minh  h o
    btn_clear.click(lambda: ("", {"note": "cleared"}), outputs=[out, meta])
demo.launch()
```

Listing 11: Pattern cơ bản: Form trái – Kết quả phải

Chồng nhiều hàng trong một cột Khi cần nhóm tham số, đặt chúng trong một `Column` riêng để tránh dồn thành "dải nút".

```
with gr.Blocks(title="Rows trong Column") as app:
    with gr.Row():
        with gr.Column(scale=1):
            with gr.Row():
                inp = gr.Textbox(label="Nhap 1", lines=3)
            with gr.Row():
                thr = gr.Slider(0,1,0.5,0.05,label="Threshold")
            with gr.Row():
                btn = gr.Button("Run")
        with gr.Column(scale=2):
            out = gr.Textbox(label="Ket qua", interactive=False, lines=6)
    btn.click(lambda x, t: f"Input len={len(x)}; thr={t}", [inp, thr], out)
app.launch()
```

Listing 12: Nhiều hàng trong cột trái

6.3 Tab: phân tách chức năng theo ngữ nghĩa

Dùng `Tab` khi có nhiều *nhóm* chức năng: ví dụ Upload/Infer, Phân tích kết quả, Cấu hình nâng cao, Logs.

```
with gr.Blocks(title="Tab Layout") as demo:
    with gr.Tab("Chuc nang"):
        with gr.Row():
            with gr.Column(scale=1):
                img = gr.Image(label="Anh dau vao", type="pil")
                conf= gr.Slider(0.05, 0.95, value=0.25, step=0.05, label="Confidence")
                btn = gr.Button("Detect")
            with gr.Column(scale=2):
                out_img = gr.Image(label="Anh da annotate", interactive=False)
                out_txt = gr.Textbox(label="Thong tin", interactive=False, lines=8)
        btn.click(lambda im, c: (im, f"conf={c:.2f}"), [img, conf], [out_img, out_txt
    ])

    with gr.Tab("Ket qua phan tich"):
        gr.Markdown("### Bieu do/Thong ke (placeholder)")
        stats = gr.JSON(label="Thong ke")

    with gr.Tab("Cau hinh nang cao"):
        adv = gr.CheckboxGroup(choices=["A","B","C"], label="Tuy chon")
        save = gr.Button("Luu cau hinh")
        save.click(lambda x: {"saved": x}, adv, stats)

    with gr.Tab("Logs"):
        logs = gr.Textbox(label="Nhat ky", lines=12, interactive=False)
demo.launch()
```

Listing 13: Bố cục theo Tab: Chuc nang, Ket qua, Cau hinh

6.4 Accordion: giấu bớt phần ít dùng

`Accordion` phù hợp với phần cấu hình hiếm khi đụng tới (ví dụ: seed, batch size, định dạng xuất).

```
with gr.Blocks(title="Accordion") as demo:
    with gr.Row():
        with gr.Column(scale=1):
            text = gr.Textbox(label="Nhap van ban", lines=3)
            run  = gr.Button("Chay")
```

```
            with gr.Accordion("Cau hinh nang cao", open=False):
                seed  = gr.Number(value=42, label="Seed")
                top_k = gr.Slider(1, 20, value=5, step=1, label="Top-K")
        with gr.Column(scale=2):
            out = gr.Textbox(label="Ket qua", interactive=False, lines=8)

    def infer(x, s, k):
        return f"text='{x[:20]}...', seed={int(s)}, top_k={int(k)}"

    run.click(infer, [text, seed, top_k], out)
demo.launch()
```

Listing 14: Accordion cho cau hinh hiem dung

6.5 Tổ chức "sidebar" tham số

Với nhiều tham số, có thể mô phỏng một "sidebar" bằng cột trái hẹp (`scale` nhỏ) chứa toàn bộ slider/checkbox.

```
with gr.Blocks(title="Sidebar") as demo:
    with gr.Row():
        with gr.Column(scale=1, min_width=280):
            gr.Markdown("### Tham so")
            temp = gr.Slider(0,1,0.5,0.01,label="Temperature")
            top_p= gr.Slider(0,1,0.9,0.01,label="Top-p")
            rep  = gr.Slider(0,2,1.0,0.05,label="Repetition penalty")
            run  = gr.Button("Generate")
        with gr.Column(scale=2):
            prompt = gr.Textbox(label="Prompt", lines=6)
            output = gr.Textbox(label="Output", interactive=False, lines=12)
    run.click(lambda p, t, tp, r: f"[t={t:.2f}, tp={tp:.2f}, r={r:.2f}] {p}",
              [prompt, temp, top_p, rep], output)
demo.launch()
```

Listing 15: Sidebar tham so ben trai

6.6 Chia nhóm "kết quả chính" và "siêu dữ liệu"

Hiển thị kết quả trực quan (ảnh/nhãn) ở vị trí nổi bật; phần *siêu dữ liệu* (JSON, thời gian xử lý, thông số) ở khu vực phụ để không làm nhiễu.

```
with gr.Blocks(title="Result + Meta") as app:
    with gr.Row():
        with gr.Column(scale=2):
            main = gr.Image(label="Ket qua chinh", interactive=False)
        with gr.Column(scale=1):
            meta = gr.JSON(label="Thong tin bo sung")
    # minh hoa
    app.load(lambda: ("https://picsum.photos/800", {"elapsed": 0.12, "version": "demo"
    }),
             inputs=None, outputs=[main, meta])
app.launch()
```

Listing 16: Ket qua chinh + sieu du lieu

6.7 Anti-patterns cần tránh

- **Một cột dọc rất dài**: người dùng phải cuộn liên tục để tìm nút/kết quả; thay vào đó chia 2 cột hoặc dùng Tab.
- **Kết quả chen giữa form**: làm người dùng mất định hướng; nên để kết quả ở cột/phần riêng.

- **UI nhảy kích thước**: khi không có kết quả thì trống hoàn toàn, có kết quả thì chiếm chỗ lớn; nên đặt container cố định (ví dụ `lines` cho Textbox, hoặc sẵn khung `Image`).
- **Quá nhiều tham số cấp 1**: nhét hết ra ngoài khiến form rối; gom nhóm vào Accordion/Tab.

6.8 Mẹo vi mô để UI gọn gàng

- Dùng `min_width` cho cột chứa nhiều slider để tránh co cụm.
- Giới hạn `lines` và `max_lines` cho Textbox để giữ chiều cao ổn định.
- Nhãn (`label`) ngắn gọn, nhất quán kiểu chữ; thêm `placeholder` để hướng dẫn.
- Nhóm các nút liên quan trong cùng `Row` (VD: `Run`/`Clear`/`Sample`).

Tóm tắt

Bố cục tốt giúp người dùng:

1. **Biết phải làm gì** (khu nhập liệu rõ ràng).
2. **Thấy được điều gì** (kết quả nổi bật, ổn định).
3. **Không bị nhiễu** (cấu hình/nhật ký ẩn trong Tab/Accordion).

Ở phần tiếp theo, ta sẽ đi sâu vào **Event listeners** (`.click`, `.change`, `.submit`, `.release`) và cách chọn đúng để app mượt, tránh gọi suy luận thừa.

7 Event listeners: chọn đúng để app mượt

Mục tiêu của phần này

Chọn đúng *sự kiện* để:

- Tránh gọi suy luận **thừa** (giảm độ trễ, tiết kiệm tài nguyên).
- Tăng **độ phản hồi** và tính dự đoán được của UI.
- Giữ **hình dạng output ổn định** trong mọi tình huống.

7.1 Bốn sự kiện cốt lõi

`.click()` Kích hoạt khi người dùng bấm nút. Phù hợp nhất cho "chạy suy luận" chủ động.

`.change()` Kích hoạt *mỗi khi* giá trị component thay đổi (nhập ký tự, kéo slider). Dễ gây *lặp gọi* nếu tác vụ nặng.

`.submit()` Kích hoạt khi nhấn Enter trong `Textbox/TextArea`. Trải nghiệm tự nhiên cho tác vụ text.

`.release()` Kích hoạt khi **thả chuột** khỏi `Slider`. Lý tưởng cho tham số ảnh hưởng mạnh đến chi phí tính toán (confidence, temperature...).

7.2 Quy tắc chọn nhanh

Tình huống	Sự kiện khuyến nghị
Nút "Chạy", "Detect", "Generate"	`.click()`
Nhập văn bản xong và nhấn Enter	`.submit()`
Điều chỉnh `Slider` cho tác vụ nặng (OD, sinh ảnh, LLM)	`.release()` (tránh `.change()`)
Những thay đổi nhẹ, chi phí thấp (lọc danh sách, hiển thị tức thì)	`.change()`

7.3 Mẫu kết nối sự kiện chuẩn

```python
import gradio as gr

def run_infer(x, t):
    # ... goi mo hinh ...
    return f"done: len={len(x)}, thr={t:.2f}"

with gr.Blocks(title="Events Basic") as app:
    txt = gr.Textbox(label="Nhap van ban", lines=3)
    thr = gr.Slider(0,1,0.5,0.01,label="Threshold")
    btn = gr.Button("Chay")
    out = gr.Textbox(label="Ket qua", interactive=False, lines=4)

    # Chay bang nut
    btn.click(run_infer, inputs=[txt, thr], outputs=out)
    # Chay bang Enter
    txt.submit(run_infer, inputs=[txt, thr], outputs=out)
    # Chay khi tha slider
    thr.release(run_infer, inputs=[txt, thr], outputs=out)
```

```

app.launch()
```

Listing 17: Kết nối sự kiện cơ bản cho một demo

7.4 Tránh lặp gọi với `Slider`

Vấn đề `.change()` trên `Slider` sẽ gọi hàm *liên tục* khi người dùng kéo — tệ cho tác vụ nặng (Object Detection, diffusion, LLM).

Giải pháp Dùng `.release()` để gọi *một lần* sau khi người dùng chọn xong giá trị.

```
conf = gr.Slider(0.05, 0.95, value=0.25, step=0.05, label="Confidence")
out_img = gr.Image()
out_txt = gr.Textbox()

def infer(image, conf):
    # ... object detection ...
    return image, f"conf={conf:.2f}"

conf.release(infer, inputs=[img, conf], outputs=[out_img, out_txt])
```

Listing 18: Ưu tiên .release cho tác vụ nặng

7.5 Ghép nhiều sự kiện cho một luồng

Có thể gắn **nhiều sự kiện** vào cùng một hàm để hỗ trợ nhiều cách kích hoạt:

```
def pipeline(text, temp):
    # ... inference ...
    return f"[temp={temp:.2f}] {text[::-1]}"

btn.click(pipeline, [txt, temp], out)
txt.submit(pipeline, [txt, temp], out)
temp.release(pipeline, [txt, temp], out)
```

Listing 19: .click + .submit + .release tới cùng một hàm

7.6 Tách *preprocess/infer/postprocess* bằng sự kiện

```
def preprocess(x):
    return x.strip()

def infer(x, t):
    if not x: return "Trong."
    # ... heavy model ...
    return f"OK({t:.2f}): {x[:20]}..."

def postprocess(y):
    return y.upper()

with gr.Blocks() as app:
    raw  = gr.Textbox(label="Raw", lines=3)
    temp = gr.Slider(0,1,0.5,0.05,label="Temp")
    btnP = gr.Button("Preprocess")
    btnI = gr.Button("Infer")
    btnO = gr.Button("Postprocess")

    cleaned = gr.Textbox(label="Cleaned", interactive=False)
    out1    = gr.Textbox(label="Infer out", interactive=False)
```

```
    out2    = gr.Textbox(label="Post out", interactive=False)

    btnP.click(preprocess, raw, cleaned)
    btnI.click(infer, [cleaned, temp], out1)
    btnO.click(postprocess, out1, out2)
app.launch()
```

Listing 20: Thiết kế pipeline nhiều bước qua sự kiện

7.7 Xử lý lỗi & trạng thái biên qua sự kiện

- **Luôn trả về cùng cấu trúc**: dù lỗi hay thành công (`(output, meta)`), UI sẽ không "nhảy".
- **Hiển thị thông điệp thân thiện**: thay vì raise exception ra UI, trả chuỗi hướng dẫn ("Vui lòng upload ảnh.").
- **Nút Clear**: gắn sự kiện trả về giá trị rỗng cho tất cả output liên quan.

```
def clear_all():
    return "", "", {}  # vi du: out_text, out_log, meta

btn_clear.click(clear_all, outputs=[out_text, out_log, out_meta])
```

Listing 21: Nút Clear trả về trạng thái rỗng ổn định

7.8 Hàng đợi (`queue=True`) và phản hồi người dùng

- Bật `queue=True` khi dự kiến nhiều yêu cầu song song để điều phối lượt chạy.
- Cân nhắc hiển thị "đang xử lý" (progress/message) cho tác vụ dài (`debug=True` để thấy traceback khi lỗi).

```
with gr.Blocks(title="Queued App", queue=True) as demo:
    # ... UI ...
    pass
demo.launch(debug=True)
```

Listing 22: Bật hàng đợi cho app nhiều người dùng

7.9 Anti-patterns thường gặp

- **Gắn mọi thứ vào `.change()`**: kéo/thả/nhập đều kích hoạt suy luận liên tục, gây lag.
- **Không ràng buộc tham số**: slider không có `step`/`min`/`max` dẫn tới giá trị "lạ".
- **Output lúc có lúc không**: trả về số lượng output khác nhau giữa các nhánh (làm vỡ layout).

Tóm tắt

1. Dùng `.click()` cho hành động *có chủ đích*, `.submit()` cho text, `.release()` cho slider *tác vụ nặng*.
2. Tránh `.change()` khi inference nặng; chỉ dùng cho thao tác nhẹ, tức thì.
3. Giữ cấu trúc trả về nhất quán và có **Clear** để reset nhanh.

Phần tiếp theo sẽ áp dụng toàn bộ vào một demo **Object Detection (YOLOv8n)** hoàn chỉnh, từ nạp model, viết hàm suy luận, dàn layout đến nối sự kiện.

8 Thực hành: Object Detection (YOLOv8n) với Gradio

Mục tiêu của phần này

Xây một ứng dụng demo **Object Detection** hoàn chỉnh:

- Người dùng *upload ảnh* hoặc *tải ảnh mẫu*.
- Điều chỉnh `confidence` (Slider) và chạy suy luận.
- Nhận **ảnh đã annotate** (vẽ bounding box + nhãn) và **tóm tắt phát hiện**.

Ta sẽ dùng **YOLOv8n** (bản nano) từ thư viện `ultralytics` vì nhẹ, phù hợp demo trên CPU.

8.1 Chuẩn bị môi trường

Cài đặt gói cần thiết (Colab hoặc máy cá nhân):

```
pip install --upgrade gradio ultralytics opencv-python pillow torch torchvision
```

Listing 23: Cài đặt thư viện

Ghi chú

- Nếu GPU có sẵn (CUDA), `torch` sẽ tự nhận; nếu không, demo vẫn chạy trên CPU với ảnh vừa phải.
- Với mạng chậm, lần đầu tải trọng số `yolov8n.pt` có thể mất thời gian.

8.2 Cấu trúc dự án gợi ý

```
.
 app.py                # mã Gradio chính
 requirements.txt      # gói bắt buộc khi deploy (HF Spaces)
 assets/               # (tùy chọn) ảnh minh hoạ, icon...
```

8.3 Hằng số và tiện ích xử lý ảnh

Ta viết một số tiện ích nhỏ để mã gọn, ổn định trên CPU:

```
import io, time, requests
from typing import Tuple, List, Optional, Dict, Any

import numpy as np
from PIL import Image

MAX_SIDE = 1280  # giới hạn cạnh của ảnh gửi vào mô hình để tránh quá tải CPU
SAMPLE_URL = "https://ultralytics.com/images/bus.jpg"

def load_sample_image() -> Image.Image:
    resp = requests.get(SAMPLE_URL, timeout=10)
    im = Image.open(io.BytesIO(resp.content)).convert("RGB")
    return im

def ensure_rgb(im: Image.Image) -> Image.Image:
    return im.convert("RGB") if im.mode != "RGB" else im

def resize_long_side(im: Image.Image, max_side: int = MAX_SIDE) -> Image.Image:
    w, h = im.size
```

```
    m = max(w, h)
    if m <= max_side:
        return im
    scale = max_side / float(m)
    new_w, new_h = int(round(w * scale)), int(round(h * scale))
    return im.resize((new_w, new_h), Image.Resampling.LANCZOS)
```

Listing 24: Hằng số và tiện ích

8.4 Nạp mô hình một lần ở phạm vi toàn cục

Để tránh nạp lại mỗi lần người dùng bấm nút, ta đặt việc nạp model ở *global scope*.

```
from ultralytics import YOLO

#  T i  model ( l n     u     s í   t i   t r ng   s     v   cache  n g   Íi  d ng)
MODEL_NAME = "yolov8n.pt"
model = YOLO(MODEL_NAME)

# T y  c h n :  k i m  tra  t h i ỉt  b
try:
    import torch
    DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
except:
    DEVICE = "cpu"
```

Listing 25: Nạp YOLOv8n một lần

8.5 Hàm suy luận: annotate ảnh + tóm tắt kết quả

Nguyên tắc:

- **Validate** đầu vào (ảnh trống).
- Chuẩn hoá: `RGB`, `resize` để giới hạn kích thước.
- Gọi `model.predict(..., conf=...)` rồi `plot()` để lấy ảnh annotate.
- Tạo **bản tóm tắt** từ `boxes` (class_id, score, toạ độ).
- Trả về `(annotated_image, info_text, meta)` ổn định.

```
def detect_objects(image: Optional[Image.Image], conf: float = 0.25):
    t0 = time.time()
    if image is None:
        return None, "Vui  l  ng  upload   nh  .", {"ok": False}

    #  T i n   x   l í
    image = ensure_rgb(image)
    image = resize_long_side(image, MAX_SIDE)
    arr = np.array(image)  # HWC RGB

    # Suy  l u n
    results = model.predict(arr, conf=conf, verbose=False, device=DEVICE)
    res = results[0]

    #  V í  annotate
    annotated = res.plot()  # ndarray RGB
    annotated_pil = Image.fromarray(annotated)

```

```
    # T m  t t  k ỉt  q u
    lines: List[str] = []
    n = 0
    if getattr(res, "boxes", None) is not None and len(res.boxes) > 0:
        for b in res.boxes:
            cls_id = int(b.cls[0])
            score = float(b.conf[0])
            x1, y1, x2, y2 = [float(x) for x in b.xyxy[0].tolist()]
            lines.append(
                f"- id={cls_id}, score={score:.2f}, "
                f"box=({x1:.1f}, {y1:.1f}, {x2:.1f}, {y2:.1f})"
            )
        n = len(res.boxes)
        info = "Ph t  h i n :\n" + "\n".join(lines)
    else:
        info = "Kh ng  ph t  h i n    i    t   ng  n  o."

    meta = {
        "ok": True,
        "elapsed_sec": round(time.time() - t0, 3),
        "num_dets": n,
        "conf": conf,
        "device": DEVICE,
        "input_size": list(arr.shape)  # [H, W, C]
    }
    return annotated_pil, info, meta
```

Listing 26: Hàm suy luận Object Detection

8.6 Hàm tiện ích: Clear và tải ảnh mẫu

```
def clear_all():
    #  T r   v   t r ng  th i  r ng  cho  c  c  output
    return None, "", {}

def load_sample():
    return load_sample_image()
```

Listing 27: Clear và Load Sample

8.7 Dựng UI bằng `Blocks`

Bố cục "Form trái – Kết quả phải", có `Sample`/`Detect`/`Clear`, và `confidence` dùng `.release()` để tránh gọi liên tục.

```
import gradio as gr

with gr.Blocks(title="YOLOv8n Object Detection", queue=True) as demo:
    gr.Markdown("## Object Detection Demo (YOLOv8n)")

    with gr.Row():
        #  C t  tr i: input + tham  s  + n  t
        with gr.Column(scale=1):
            inp_image = gr.Image(label="  nh    u   v  o", type="pil")
            conf = gr.Slider(0.05, 0.95, value=0.25, step=0.05, label="Confidence")
            with gr.Row():
                btn_sample = gr.Button("Load Sample")
                btn_detect = gr.Button("Detect")
                btn_clear  = gr.Button("Clear")

```

```
        # C t  p h i :  k ỉt  q u  + meta
        with gr.Column(scale=2):
            out_image = gr.Image(label="  nh       annotate", interactive=False)
            out_info  = gr.Textbox(label="Th ng tin ph t  h i n ", interactive=False,
    lines=10)
            out_meta  = gr.JSON(label="Meta")

    #  S   k i n
    btn_sample.click(load_sample, outputs=inp_image)
    btn_detect.click(detect_objects, inputs=[inp_image, conf], outputs=[out_image,
    out_info, out_meta])
    btn_clear.click(clear_all, outputs=[out_image, out_info, out_meta])

    # Tr nh  g i  suy  l u n  li n  t c  khi k o slider:
    conf.release(detect_objects, inputs=[inp_image, conf], outputs=[out_image,
    out_info, out_meta])

if __name__ == "__main__":
    demo.launch(share=True, debug=True)
```

Listing 28: Gradio Blocks cho Object Detection

8.8 Kịch bản kiểm thử nhanh

- **Không upload ảnh**: bấm `Detect` → thông báo "Vui lòng upload ảnh." (không lỗi).
- **Ảnh lớn (4K)**: UI vẫn phản hồi, thời gian dài hơn; ảnh được `resize` về `MAX_SIDE`.
- **Confidence thấp/cao**: so sánh số lượng box và chất lượng phát hiện.
- **Nhấn Clear**: mọi output về trạng thái rỗng, layout ổn định.

8.9 Tối ưu hiệu năng

- **Resize ảnh**: giới hạn cạnh dài (`MAX_SIDE`) để giảm độ trễ, nhất là trên CPU.
- **Load model một lần**: đặt ở *global scope*.
- **Giảm số lần gọi**: dùng `.release` thay vì `.change` với `Slider`.
- **Chọn bản nhỏ**: `yolov8n.pt` thay vì `m/l/x` khi demo.

8.10 Xử lý lỗi và tình huống biên

- **Ảnh không hợp lệ**: bao try/except khi `open()` hoặc `np.array()`, trả thông điệp thân thiện.
- **Không phát hiện**: vẫn trả `annotated_image` (ảnh gốc) và chuỗi "Không phát hiện..." để UI không nhảy.
- **Thiếu internet (ảnh mẫu)**: nếu tải `SAMPLE_URL` lỗi, thông báo "Không tải được ảnh mẫu.".

8.11 An toàn dữ liệu (mức demo)

- Không lưu ảnh người dùng trừ khi thật cần; nếu có, nêu rõ mục đích và xoá sau khi xử lý.
- Không ghi log chi tiết chứa dữ liệu nhạy cảm.

8.12 Checklist hoàn tất phần thực hành

Chạy được trên CPU/GPU với ảnh mẫu và ảnh người dùng.

Output **ổn định**: `(image, info, meta)` trong mọi nhánh.

Nút `Sample`/`Detect`/`Clear` hoạt động như kỳ vọng.

`Slider` dùng `.release` để tránh suy luận thừa.

Ảnh lớn được `resize` trước khi suy luận.

Kết nối sang phần tiếp theo

Ở phần kế tiếp, ta sẽ **đưa ứng dụng lên Hugging Face Spaces** để có *URL công khai bền vững*, kèm `requirements.txt` tối giản và mẹo tránh lỗi build.

9 Triển khai lên Hugging Face Spaces

Mục tiêu của phần này

Đưa ứng dụng Gradio (đã chạy ổn cục bộ/Colab) lên **Hugging Face Spaces** để nhận *URL công khai bền vững*, tiện chia sẻ cho giảng viên, đồng đội, hoặc khách hàng.

9.1 Kiến thức nền tảng nhanh

- **Space** là một repo đặc biệt trên Hugging Face chứa mã nguồn ứng dụng. Bạn có thể chọn runtime `Gradio`.
- App tự build và chạy mỗi khi bạn **commit** (push) thay đổi.
- Có thể cấu hình **Secrets** (biến môi trường ẩn) và **Hardware** (CPU/GPU).

9.2 Chuẩn bị dự án

Cấu trúc tối thiểu:

```
.
 app.py                # mã Gradio chính (điểm vào)
 requirements.txt      # phiên bản gói cần cài khi build
 assets/               # (tùy chọn) ảnh minh hoạ, icon...
```

```python
import gradio as gr

def echo(x: str) -> str:
    return f"Echo: {x}"

demo = gr.Interface(
    fn=echo,
    inputs=gr.Textbox(label="Nhap gi do"),
    outputs=gr.Textbox(label="Ket qua", interactive=False),
    title="Hello Gradio on Spaces",
    description="Demo toi thieu"
)

if __name__ == "__main__":
    demo.launch()
```

Listing 29: app.py tối giản cho Spaces

```
gradio>=4.0.0
ultralytics>=8.0.0
opencv-python>=4.8.0
pillow>=10.0.0
torch           # de Spaces tu chon ban phu hop voi phan  c ng
torchvision
requests
```

Listing 30: requirements.txt tham khảo

Mẹo: khi đã có bản chạy ổn định, cân nhắc *ghim chặt* phiên bản (ví dụ `gradio==4.44.0`) để tránh build hỏng do thay đổi upstream.

9.3 Tạo Space và đẩy mã

requirements.txt (ghim phiên bản để build ổn định)

1. Đăng nhập Hugging Face → `Create new Space` → `SDK: Gradio`.
2. Chọn **Public** (mặc định) hoặc **Private** (nếu cần).
3. Tạo repo, sử dụng `git lfs` nếu có file lớn.
4. Thêm `app.py`, `requirements.txt`, và (tuỳ chọn) thư mục `assets/`. Commit và push.
5. Chờ tiến trình **Build** hoàn tất (∼ vài phút). Khi thành công, Space sẽ hiển thị app và cấp URL công khai.

9.4 Cấu hình nâng cao

Hardware

- Mặc định: CPU. Với mô hình nặng (diffusion/LLM), chọn GPU phù hợp (tốn credit).
- Với OD dùng YOLOv8n, CPU thường đủ cho demo.

Secrets (biến môi trường)

- Dùng cho khóa API (ví dụ OpenAI, dịch vụ nội bộ).
- Truy cập ở phần `Settings → Repository secrets`; đọc trong mã qua `os.environ`.

Pin revision & cache

- Hugging Face có **cache** cho model/checkpoint (ví dụ `yolov8n.pt`).
- Có thể warm-up lần đầu lâu hơn; những lần sau nhanh hơn.

Startup script

- Với Gradio, không cần quy định port/tham số server; Spaces tự nhận.
- `demo.launch()` trong `app.py` là đủ.

9.5 Khắc phục sự cố thường gặp

- **Build fail do gói**: kiểm tra `requirements.txt`, ghim phiên bản cụ thể, loại bỏ gói không cần thiết.
- **Thiếu quyền mạng (tải mẫu)**: dùng try/except và thông báo "Không tải được ảnh mẫu."; chuẩn bị ảnh fallback nội bộ.
- **Timeout khi nạp model lớn**: dùng bản nhẹ hơn, hoặc chuyển sang GPU nếu bắt buộc.
- **Vỡ UI sau cập nhật gói**: khôi phục phiên bản trước (ghi rõ version trong `requirements.txt`).

9.6 Checklist triển khai thành công

`app.py` chạy cục bộ được trước khi đẩy lên.

`requirements.txt` tối giản, có phiên bản phù hợp.

App hiện **online**, URL public truy cập ổn định.

Thêm **README** ngắn ở Space mô tả cách dùng, dữ liệu mẫu, lưu ý bảo mật.

Gợi ý mở rộng

- Thêm ảnh/video/README mô tả kết quả đầu ra.
- Sử dụng Tabs/Accordion để chia khu vực "Hướng dẫn", "Giới thiệu mô hình", "FAQ".
- Bật `queue=True` cho app nhiều người dùng; thêm giới hạn vào/ra nếu cần.

10 Kết luận

Tóm lược

Bằng cách áp dụng khung **Input** – **Model** – **Output** và lựa chọn đúng giữa `Interface` / `Blocks`, bạn có thể đưa một hàm suy luận hoặc mô hình ML thành **ứng dụng web** trong thời gian rất ngắn. Bố cục hợp lý (Row/Column, Tab, Accordion) và **sự kiện** phù hợp (`.click`, `.submit`, `.release`) giúp giảm độ trễ, ổn định UI, và nâng cao trải nghiệm người dùng.

Giá trị thực tiễn

- **Demo nhanh/MVP**: chia sẻ đường link để lấy phản hồi sớm.
- **Giảng dạy/workshop**: giảm rào cản hạ tầng, tập trung vào logic học máy.
- **Nội bộ/POC**: kiểm chứng ý tưởng trước khi đầu tư xây dựng hệ thống production.

Hướng đi tiếp theo

- Tách rõ các hàm `preprocess`/`infer`/`postprocess`, chuẩn hoá đầu vào/ra để dễ bảo trì.
- Thử nghiệm thêm **Blocks nâng cao**: *Tabs cho đa chức năng*, *Accordion cho cấu hình hiếm dùng*.
- Nâng cấp lên mô hình lớn hơn (khi cần) và cân nhắc **GPU** trên Spaces.
- Viết **README** rõ ràng + thêm hình minh hoạ kết quả để người dùng hiểu ngay cách sử dụng.

Lời khuyên cuối

1. Bắt đầu nhỏ với `Interface`, có đường link **chạy được**.
2. Giữ logic mô hình *độc lập* với UI để **nâng cấp dần** lên `Blocks` mà không động đến lõi suy luận.
3. Luôn có **Clear**, **Sample**, và **thông báo lỗi thân thiện**.

Chúc bạn dựng demo thuận lợi và nhận được phản hồi hữu ích!