# Easy UI for ML Demo

**Streamlit  Gradio**

# Lecture Overview

We'll explore how to rapidly build interactive UIs for machine learning projects using Streamlit and Gradio. These frameworks allow you to transform models into interactive demos with minimal effort.

### First Half

Streamlit fundamentals, component architecture, deployment options

### Second Half

Gradio interface design, media components and deployment on Hugging Face Spaces

### Final Segment

Comparison guide, performance tips

By the end, you'll be able to create app and understand deployment options

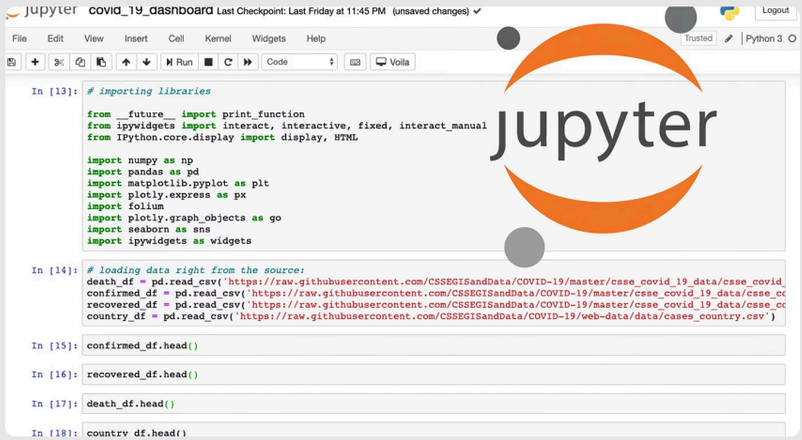

# Session Goals

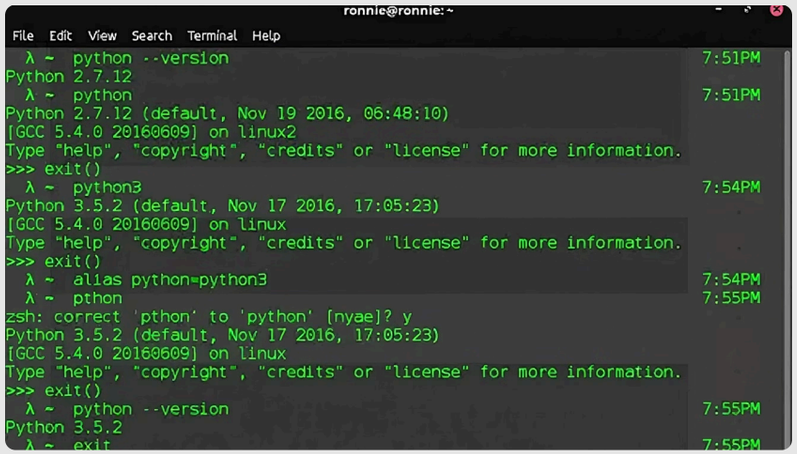
1. Understand why a UI speeds up an ML project
2. Ship a working prototype in ≤15 minutes from a single Python file
3. Streamlit first, then Gradio
4. Decide which tool fits your case

# Why UI is useful for demos

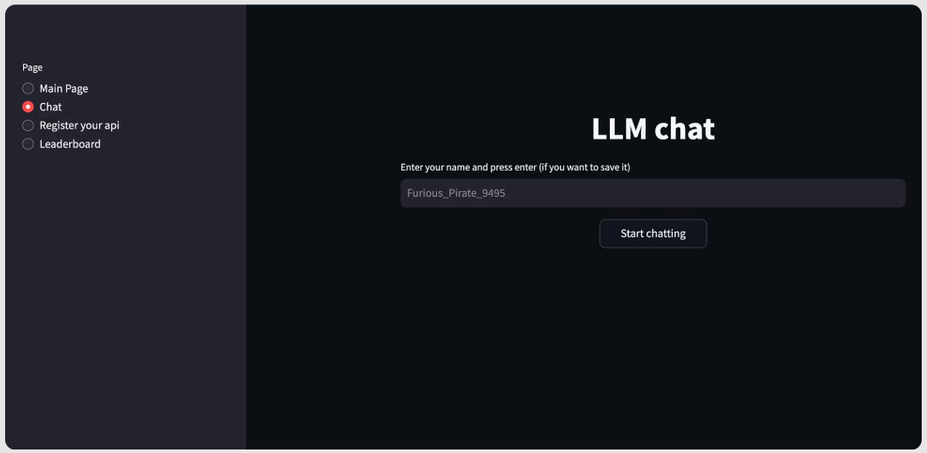
- 5-second interactive demo or 3-page notebook
- UI turns offline metrics → online feedback early in development, reducing iterations
- Visible errors accelerate error analysis
- Take-away: show working behavior as soon as possible

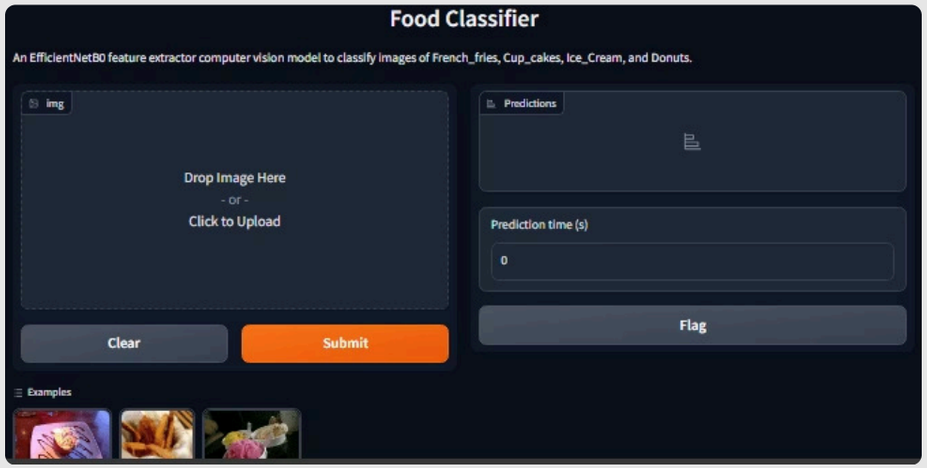
Jupyter Notebook and Terminal




Streamlit/Gradio App



# Python UI Landscape in One Table

| Tier | Typical tool | Primary use-case |
|---|---|---|
| Low-code dashboards | Streamlit | BI / EDA etc. |
| ML-centric demos | Gradio | Model showcase |
| Full-stack JS | React, Vue | Consumer-grade prod apps |

Has anyone worked with any UI frameworks?

# How Streamlit Works Under the Hood

```
import streamlit as st
import numpy as np

st.title("Streamlit Hello Demo")

# Short note: script reruns on every widget interaction
st.write(
    "Move the slider — the script reruns automatically "
    "and computes the square of the selected number."
)

value = st.slider("Number (0–100)", 0,100,25)
st.write(f"Square: {value ** 2}")
st.write(f"Random number for this run: {np.random.randint(0,
1000)}")
```

- Each browser tab gets its own Python thread
- Every widget change triggers a top-to-bottom rerun

# Install & First Run

| Step | Command / Code |
| --- | --- |
| 1. Install | pip install streamlit |
| 2. Verify | streamlit hello # opens showcase app |
| 3. Your first app | import streamlit as st<br>st.title("Hello") |

# Core Widgets You'll Use 80% of the Time

| Category | Most-used APIs | What they solve |
|---|---|---|
| Display | st.write, st.metric, st.data_editor | Show numbers & tables |
| Charts | st.line_chart, st.bar_chart, st.map | Instant visualization |
| Inputs | st.text_input, st.slider, st.file_uploader | Collect parameters |

Tip: st.data_editor lets users edit a DataFrame inline .

# Clean Layout in Three Calls

```
import streamlit as st
import pandas as pd
import numpy as np

st.set_page_config(layout="wide")
st.title("Editable Table + Multiplier")

# --- 2. Multiplier widget ---
multiplier = st.sidebar.number_input("Multiplier (1–10)", 1, 10,
2)

# --- 5. Metrics & chart ---
st.sidebar.metric("Rows", len(st.session_state.df))

st.subheader("B × k Line")
st.line_chart(st.session_state.df.set_index("A")["B"] * multiplier)
```


- st.sidebar – global settings
- st.columns – responsive grid
- st.tabs – view switching

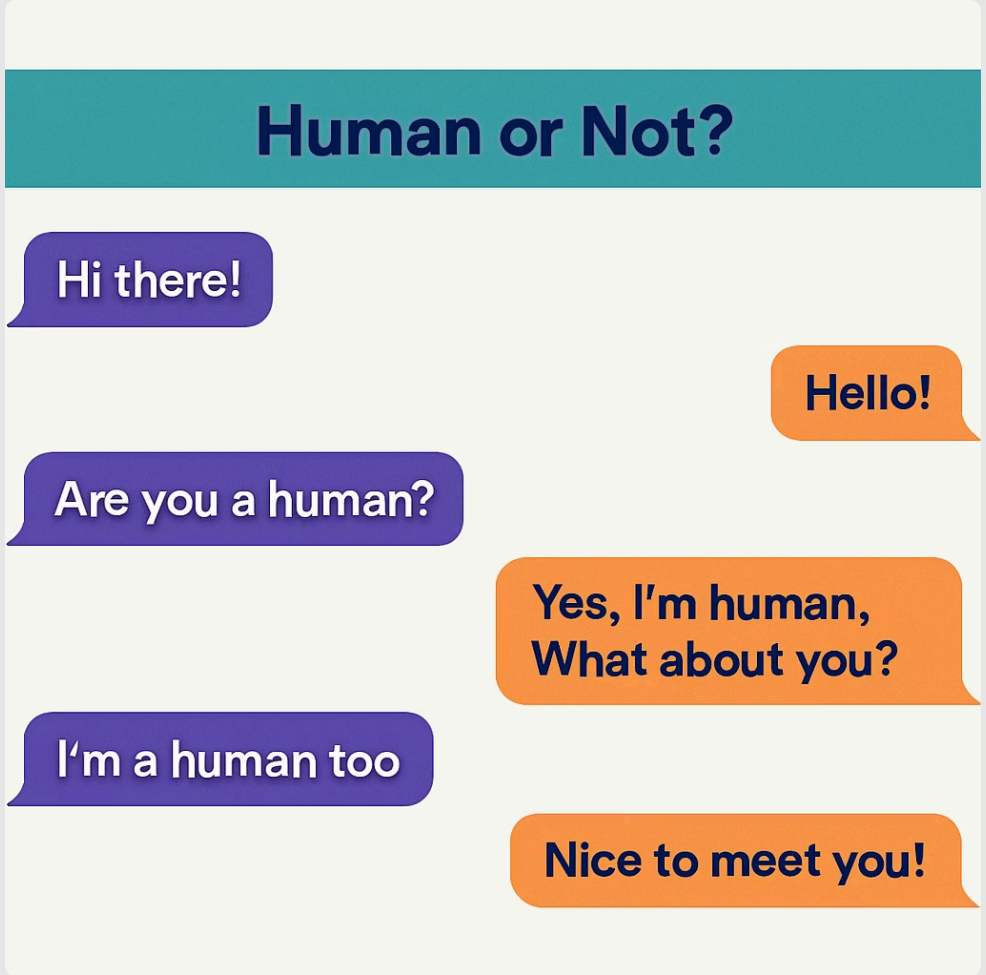
# State & Chat Components

```
import streamlit as st

if "hits" not in st.session_state:
    st.session_state.hits = 0
st.session_state.hits += 1
st.write("Page views:", st.session_state.hits)

# Simple echo chat
for m in st.session_state.get("dialog", []):
    st.chat_message(m["role"]).write(m["text"])

if q := st.chat_input("Say something"):
    st.session_state.dialog = st.session_state.get("dialog", [])
    st.session_state.dialog.append({"role":"user", "text": q})
    st.chat_message("assistant").write(q[::-1])
```




- st.session_state persists per tab across reruns
- st.chat_message + st.chat_input give ready-made chat UX

# Forms: Batch Input, Zero Noise

## Problem

Every keystroke in a text area normally triggers a rerun → heavy models re-load.

## Solution code

```
with st.form(key="batch"):
    txts = st.text_area("Paste many sentences", height=120)
    topk = st.slider("Top-K labels", 1, 5, 3)
    submitted = st.form_submit_button("Run inference")

if submitted:
    st.success("Processing...")
    st.write(model.predict(txts.splitlines(), k=topk))
```

## Before / After

| Without form | With st.form |
|---|---|
| every keypress reruns the whole script | ▶️ one rerun only when "Submit" is clicked |

Forms batch user input into a single rerun.

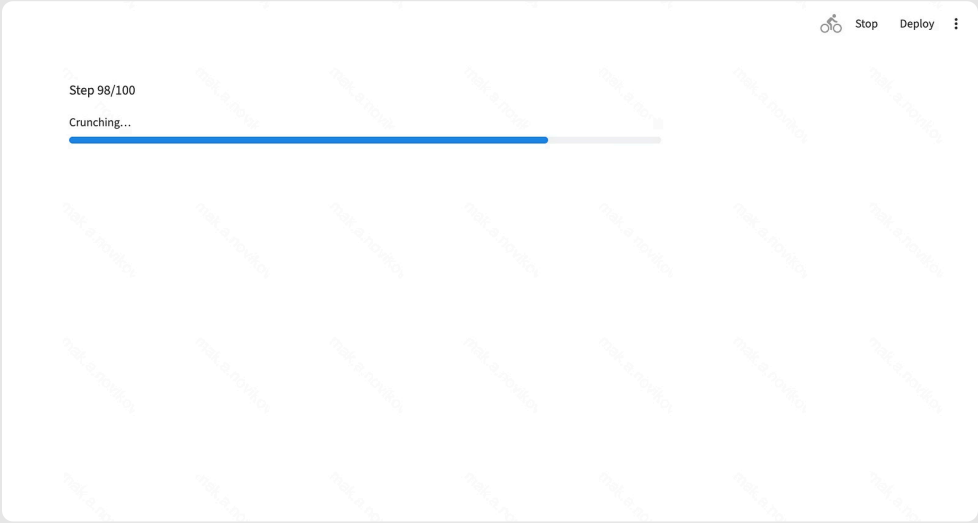
# Progress Bars & Placeholders

```
import streamlit as st, time

slot = st.empty() # reserve spot
progress = st.progress(0)

for i in range(100):
    time.sleep(0.03)
    progress.progress(i + 1, text="Crunching...")
    slot.write(f"Step {i+1}/100")

progress.empty() # remove bar
st.success("Done!")
```




**Why st.empty() first?** Reserving the slot avoids layout shift when the element appears later.

# Mini Dashboard

## Checklist

1. Load data with @st.cache_data
2. Sidebar filters → range slider
3. Map with st.map() to plot lat/lon points

# Performance & Resource Tips

| Problem | Quick fix | Why it helps |
|---|---|---|
| First load slow | Wrap heavy load in @st.cache_resource | Only once per process |
| 1 GB RAM limit on free Cloud | Prune unused libs or upgrade plan | Prevent Out-of-Memory kill |
| Global variable mutation | Copy data before edit | Avoid cache invalidation |
| Long CPU loop blocks UI | Use st.progress + time.sleep() or move heavy work to separate thread | Keeps user informed |

# Deploy & Share Your Streamlit App

| Scenario | Command / Action | Result |
|---|---|---|
| Local dev | streamlit run app.py | Opens localhost:8501 |
| LAN share | streamlit run app.py --server.address 0.0.0.0 | Any device on Wi-Fi can open :8501 |
| Docker | docker build -t myapp .<br>docker run -p 8501:8501 myapp | Reproducible container |
| Streamlit Cloud (free) | Click Deploy on GitHub repo | Public URL + CI (1 vCPU / 1 GB RAM) |

# Streamlit Recap – 4 Things to Remember

1

### Notebook-style coding

Transform from simple Python scripts to production-looking apps in minutes

2

### Full-script rerun

Simple code structure with caching to fix performance bottlenecks

3

### Rich layout & widgets

Columns, tabs, editable tables, chat interfaces for varied presentations

4

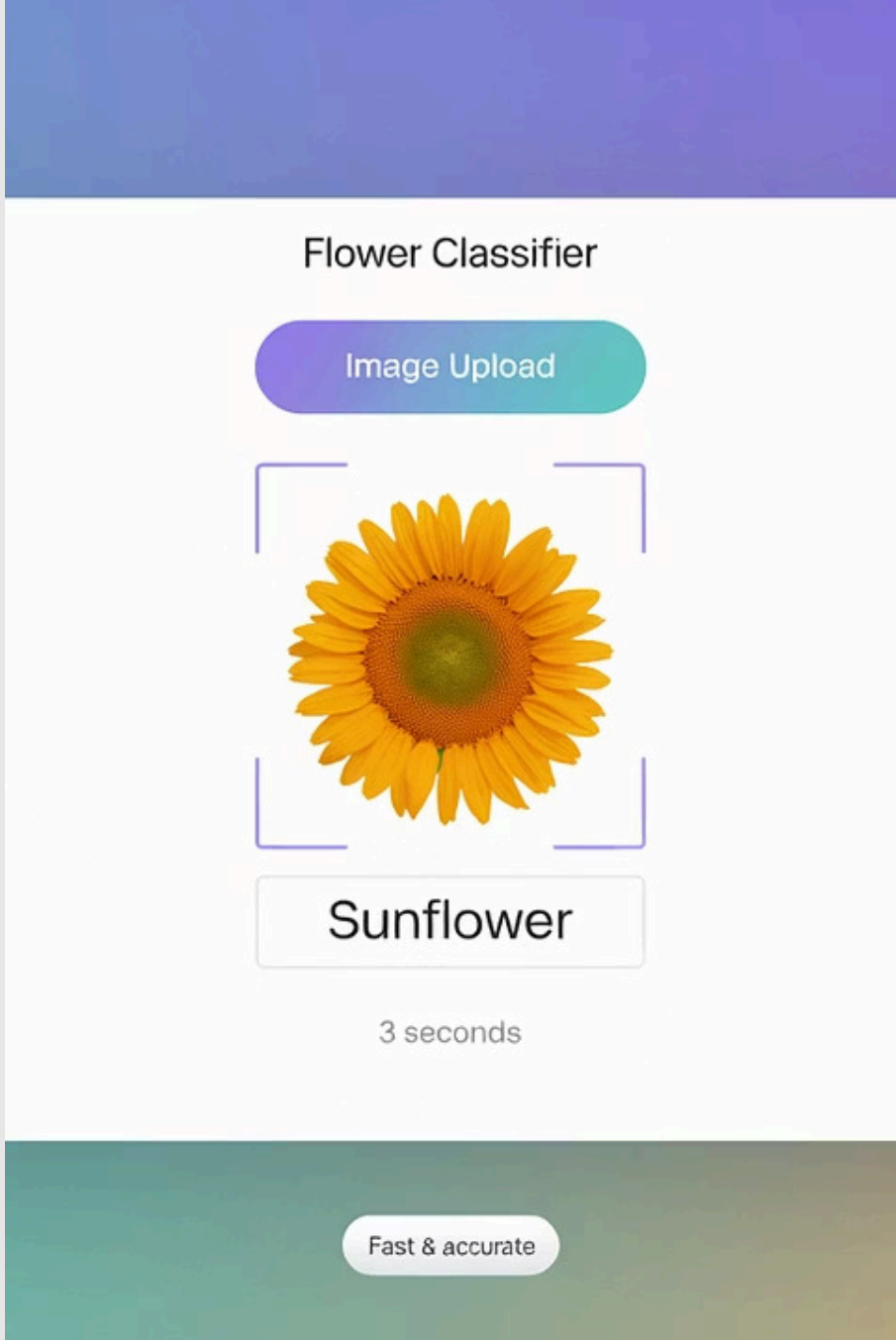
### One-click share

Deploy via Streamlit Cloud or package with Docker for flexible hosting

# Transition – Why Look at Gradio?

> "From one function to a public demo in five lines."

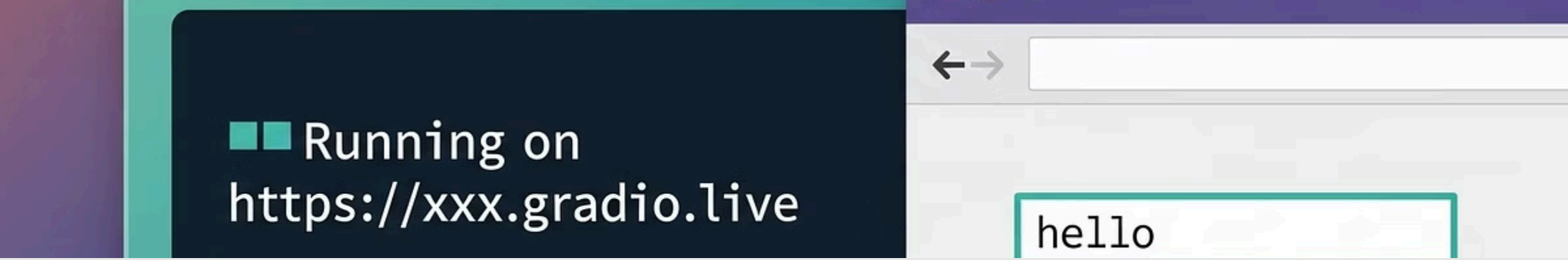
Streamlit shines for data dashboards; **Gradio** is built for media-rich model demos and instant share-links. Let's see how.



# Gradio in One Sentence

> "Set share=True, and Gradio gives you a public HTTPS link... in under 30 seconds."

Share links last 72h; disable with launch(share=False).




# Hello World in 4 Lines

```
demo = gr.Interface(
    fn=greet,              # Python function to invoke
    inputs="text",         # single text input
    outputs="text",        # single text output
    title="Gradio Hello Demo",
    description="Enter your name to receive a greeting.",
)
```

# Gradio Component Catalog: the 80% you'll actually use

| Media type | Input / Output components | Typical use |
|---|---|---|
| Text | Textbox, Markdown, Label | Prompts, logs |
| Numeric | Slider, Number, Checkbox | Hyper-params |
| Visual | Image, Video, Plot | CV models |
| Audio | Audio, Microphone | ASR / TTS |
| Data | Dataframe, JSON, HighlightText | Tabular output |

```
import gradio as gr
demo = gr.Interface(
 fn=lambda img: img, # identity
 inputs="image", # ← swap "text" to "image"
 outputs="image"
).launch()
```

# Interface vs Blocks – When do you level up?

| Criterion | Interface | Blocks |
|---|---|---|
| Lines of code | ≤5 | 10-30 |
| Custom layout | ➖ | ✅ |
| Multiple event flows | ➖ | ✅ |
| Recommended for | Quick prototype | Production demo |

```
# Interface (5 lines)
import gradio as gr
gr.Interface(lambda t: t[::-1],
             "text", "text").launch()

# Blocks (15 lines, custom layout)
with gr.Blocks() as demo:
    gr.Markdown("### Reverse!")
    txt = gr.Textbox()
    out = gr.Textbox()
    btn = gr.Button("Run")
    btn.click(lambda s: s[::-1], txt, out)
    demo.launch()
```

# Events & Session State

```
with gr.Blocks() as demo:
    counter = gr.State(0)

    def inc(n):
        return n + 1

    btn = gr.Button("Increment")
    txt = gr.Textbox(label="Clicks so far")

    btn.click(inc, counter, counter)  # mutate state
    btn.click(lambda n: str(n), counter, txt)

    demo.launch()
```

The event flow connects UI components to Python functions. When a button is clicked, its event triggers function calls.

gr.State stores data that persists between reruns but is isolated to each browser session. Perfect for:

- Chat history
- User preferences
- Multi-step workflows

State is deleted after a session times out.

# Queues & Concurrency: Keep the UI Responsive

| Pain point | Gradio fix | One-liner |
|---|---|---|
| Heavy model blocks UI | Activate a global queue | demo.queue() |
| Too many parallel requests | Limit workers per listener | concurrency_count=3 |
| Users flood with long prompts | Cap backlog size | max_size=25 |

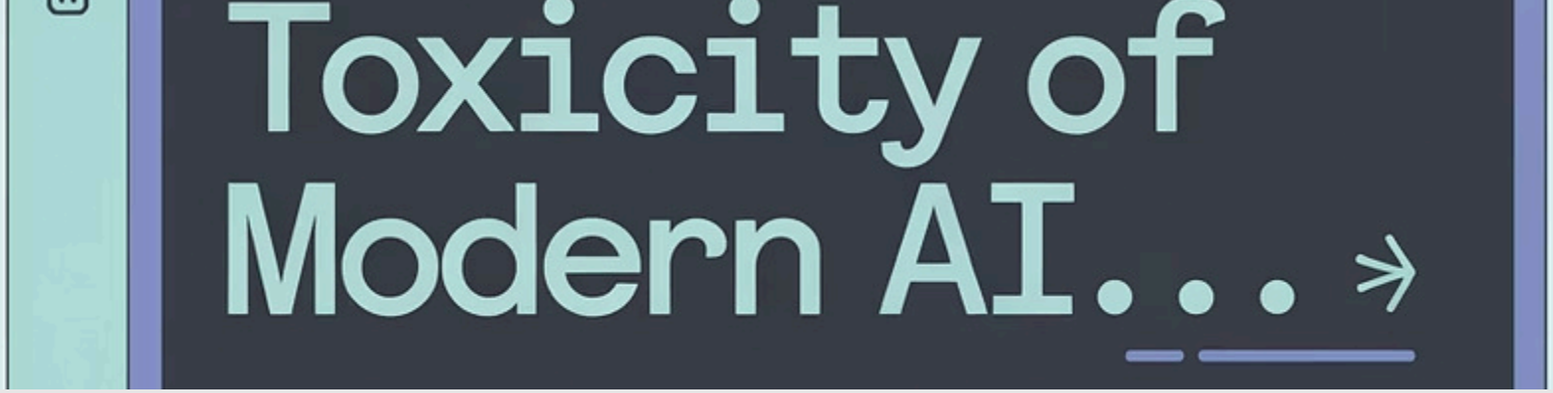
```
with gr.Blocks() as demo:
 ...
 demo.queue(concurrency_limit=3, max_size=25).launch(share=True)
```

# Deploy on Hugging Face Spaces

## 5-step checklist

1. Create new Space → "Gradio" template
2. Push repo (app.py, requirements.txt)
3. Click Settings → Hardware – default = CPU Basic (free)
4. Need GPU? Choose T4-small ($0.40/h)
5. Share the public URL with stakeholders

| Tier | CPU | GPU | Hourly price |
|---|---|---|---|
| CPU Basic | 2 vCPU / 16 GB | – | FREE |
| T4-small | 4 vCPU / 15 GB | 16 GB VRAM | $0.40 |




# Toxic Classifier

```
import gradio as gr

# ------------------ UI Definition ------------------
demo = gr.Interface(
    fn=classify,
    inputs=gr.Textbox(lines=3, placeholder="Enter text..."),
    outputs=gr.Label(num_top_classes=5),
    title="Text Toxicity Classifier (BERT)",
    description="Multi-label output: non-toxic, insult, obscenity, threat, dangerous",
)

if __name__ == "__main__":
    # share=True instantly exposes a public HTTPS tunnel
    demo.launch(share=True)
```

# Gradio Python Client = Free REST API

```
from gradio_client import Client

# Public URL of the Gradio app created with `launch(share=True)`
URL = "https://xxxxxxxx.gradio.live"
client = Client(URL)

# 2) Synchronous call: waits until the server returns a prediction
resp = client.predict(
    "You are a disgusting fool!",  # input text to classify
    api_name="/predict"            # endpoint path; omit if only one exists
)  # Example: {'toxic': 0.97, 'insult': 0.88, ...}
```

- Wraps all HTTP + JSON; behaves like a local function call
- Works with any public Space or self-hosted Gradio app

# Security Check-list: Ship a Demo, not an Open Door

| Risk | One-line fix | How to do it |
|---|---|---|
| Anonymous GPU abuse | Password-gate your Space | demo.launch(auth=("admin", "s3cret")) |
| Infinite queue backlog | Cap queue size | demo.queue(max_size=25) |
| Malicious file uploads | Allow only safe types | gr.File(file_types=[".png", ".jpg"]) |
| Secrets in code | Load from env variables | os.getenv("HF_TOKEN") |
| Prompt injection in LLMs | Sanitize & truncate input | text[:1024] before send |

Security first – a leaked GPU = $0.40/h money drain.

# Gradio Recap – Key Points

1

### 5-line prototype → public link in 30s

Add share=True for instant public sharing with no setup

2

### 70+ media components

Support for text, images, audio, data visualization in one package

3

### Queues & auth built-in

Manage concurrency and secure your demo with minimal configuration

4

### Themes + CSS in 2 lines

# Decision Grid: Streamlit vs Gradio

| Axis | Streamlit | Gradio |
|---|---|---|
| Layout granularity | Columns, tabs, containers | Basic rows; fine control via Blocks |
| Media widgets | Limited (image, audio) | Full catalog 70+ components |
| Built-in queue | via FastAPI / Celery | .queue() + concurrency_count |
| Auth out-of-box | Add-on modules | auth=() in launch() |
| One-click share | Streamlit Cloud / HF Spaces | share=True / HF Spaces |
| Best suited for | Dashboards, EDA, KPIs | Media-rich model demos |

# Cost Snapshot

| Platform | CPU free tier | GPU option | Hourly $ |
|---|---|---|---|
| Streamlit Cloud | 1 vCPU / 1 GB RAM | – | $0 |
| HF Spaces – CPU | 2 vCPU / 16 GB RAM | – | $0 |
| HF Spaces – T4-small | 4 vCPU / 15 GB RAM | 16 GB VRAM | $0.40 |
| Self-host | | | cheaper - more complicated |

# Performance Cheatsheet (Streamlit vs Gradio)

## Streamlit

- Cache slow data with @st.cache_data; heavy models with @st.cache_resource
- Preload models globally; they're shared across sessions
- 1 GB RAM cap on free Cloud – monitor memory

## Gradio

- Use queue to decouple UI; set concurrency_count to GPU batch size
- Limit max_size to stop DoS
- Use gr.State instead of globals for per-tab data
- HF Spaces auto-restarts when RAM>quota

# Best-Practice Checklist

**Cache heavy objects (@st.cache_resource)**

Cuts first request by 10×

**Use Session State**

Persist context per tab

**Enable queue (concurrency_count)**

Prevents GPU over-booking

**Add basic auth in public demos**

Blocks scrapers

**README + docs link**

Your colleagues onboard fast