1️⃣ Generative Models for Vision Synthesis

📘 Definition

Generative models learn patterns in data to create new, synthetic content. In vision: they generate images, transform styles, or reconstruct missing data.

🔹 GANs (Generative Adversarial Networks)

- **Concept:** Competing neural networks (generator vs discriminator) generate highly realistic samples.
- **Code Snippet: Basic DCGAN Generator**

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.ConvTranspose2d(100, 64, 4, 1, 0),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 1, 4, 2, 1),
            nn.Tanh()
        )
    def forward(self, x):
        return self.model(x)
```

🔹 VAEs (Variational Autoencoders)

- **Concept:** Learn latent space representations and reconstruct images while allowing smooth interpolation.
- **Use Case:** Facial morphing, anomaly detection

📘 **Project:** Create a synthetic fashion/image dataset using GAN or VAE
🔧 Tools: PyTorch, torchvision, numpy, TensorBoard

2️⃣ Vision Transformers (ViTs) and Attention Mechanisms

📘 Definition

Transformers model relationships between image patches via self-attention, allowing long-range dependencies in image understanding.

🔹 **Vision Transformer (ViT)**

- **Code Snippet: Load with timm**

```
import timm

model = timm.create_model("vit_base_patch16_224", pretrained=True)
```

📘 **Project:** Compare ViT vs. CNN accuracy on ImageNet subsets
🔧 Tools: PyTorch, timm, HuggingFace datasets

3️⃣ Multimodal Learning: Vision + Text/Audio

📘 Definition

Models that combine multiple input types—such as images and text—to answer queries, generate captions, or perform scene reasoning.

🔹 VQA (Visual Question Answering)

- **Concept:** System answers questions based on image input.
- **Architecture:** CNN (image features) + Transformer or RNN (text query)
- **Code Snippet (text tokenization + image embedding)**

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

tokens = tokenizer("What color is the cat?", return_tensors="pt")
```

📘 **Project:** Build a VQA model using Flickr or GQA datasets
🔧 Tools: PyTorch, HuggingFace, torchvision, OpenCLIP

4️⃣ Ethical AI, Fairness, Bias Mitigation

📘 Definition

Ensure models avoid discriminatory behavior by addressing dataset imbalance, harmful outputs, and inaccessible designs.

🔹 Concepts

- **Bias sources:** Label imbalance, cultural representation, resolution disparities
- **Techniques:** Balanced sampling, adversarial fairness training, explainability

📘 **Project:** Audit CV model for demographic bias on face detection
🔧 Tools: IBM AI Fairness 360, Captum, SHAP, custom dataset analysis

5️⃣ Publishing Research in Top-tier Journals

📘 Goals

Train students to structure research papers for conferences (CVPR, ICCV, NeurIPS). Emphasis on reproducibility, open science, and visual storytelling.

🔹 Module Includes:

- Writing abstracts, introductions, and impact statements
- Crafting method illustrations and diagrams
- Using LaTeX for academic formatting
- Open-source release documentation

📘 **Project:** Write and document a reproducible image synthesis pipeline
🔧 Tools: Overleaf, GitHub Pages, Jupyter Book, wandb

🧩 Markdown Template Example of Vision Transformer Module

📍 Overview

Vision Transformers segment images into patches and process them using self-attention layers. This enables contextual understanding across large spatial regions.

🍼 Dependencies

```
pip install timm torch torchvision
```

🔧 Basic Usage

```
import timm

model = timm.create_model('vit_base_patch16_224', pretrained=True)
```

🖼️ Visual Diagram

[Image] → [Patch Embedding] → [Transformer Encoder] → [Classification Head]

📣 Notes

ViTs require large datasets and compute but offer state-of-the-art results on classification tasks.

💻 Code Examples Galore

Task	Model	Library
Image synthesis	DCGAN, VAE	PyTorch
Classification	ViT, SwinTransformer	timm
VQA	ViLT, CLIP + BERT	HuggingFace
Bias analysis	FaceNet + fairness metrics	Captum, AI Fairness 360
Paper figures	Custom data plotter	matplotlib, LaTeX PGFPlots

Generative Models (GANs & VAEs) for Image Synthesis

```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt


# Generator (DCGAN-style)
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.ConvTranspose2d(100, 64, 4, 1, 0),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 1, 4, 2, 1),
            nn.Tanh()
        )
    def forward(self, x):
        return self.model(x)


# Sample noise & generate image
noise = torch.randn(1, 100, 1, 1)
gen = Generator()
img = gen(noise).detach().squeeze()
plt.imshow(img.numpy(), cmap="gray")
plt.title("Synthetic Image from GAN Generator")
plt.axis("off")
plt.show()
```

📓 **Vision Transformers & Attention with timm:**

```
!pip install timm

import timm

import torch


# Load pretrained ViT

model = timm.create_model('vit_base_patch16_224', pretrained=True)

# Summary of architecture

print(model)
```

```
# Use hooks or forward pass to extract embeddings later
```

🔍 *Optional: Add a visualization cell for patch embeddings over image using OpenCV or matplotlib.*

📓 **Multimodal VQA System**

```
# Visual Question Answering Prototype

from transformers import BertTokenizer, BertModel

import torch

import torchvision.models as models


# Text tokenization

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

tokens = tokenizer("What color is the cat?", return_tensors="pt")

# Image feature extractor

img_model = models.resnet18(pretrained=True)

img_model.fc = torch.nn.Identity()  # Remove classifier

# Example image tensor (load with torchvision or PIL)

# img_tensor = preprocess("cat.jpg")
```

Combine for fusion later: [Text tokens] + [Image features]

💡 *Extend this by fusing modalities for end-to-end prediction.*

📘 **Ethics and Bias Auditing > Bias Audit: Face Detection Example**

```
from aif360.datasets import BinaryLabelDataset

from aif360.metrics import BinaryLabelDatasetMetric


# Simulated dataset load

data = BinaryLabelDataset(

    favorable_label=1, unfavorable_label=0,

    df=your_dataframe, label_names=['prediction'], protected_attribute_names=['ethnicity']

)


metric = BinaryLabelDatasetMetric(data, privileged_groups=[{'ethnicity': 1}],
unprivileged_groups=[{'ethnicity': 0}])

print("Disparate impact:", metric.disparate_impact())
```

🔎 *You can analyze how fairness shifts with dataset balancing or adversarial correction.*

📓 **Research & Reproducibility**

```
# Publishing Pipeline: Figure Generation & Logging
    import matplotlib.pyplot as plt
    import wandb

    wandb.init(project="image-synthesis-pipeline")

    loss_vals = [0.5, 0.4, 0.3, 0.2]
    epochs = list(range(1, 5))

    plt.plot(epochs, loss_vals)
    plt.title("Training Loss Over Epochs")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.grid(True)
    plt.show()

    # Log to wandb
    wandb.log({"loss_chart": plt})
```

🔮 GAN Architecture (DCGAN-style)

📊 With `matplotlib`

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

fig, ax = plt.subplots(figsize=(10, 2))
# Generator
ax.add_patch(patches.Rectangle((0.1, 0.5), 0.3, 0.4, color='skyblue', label="Generator"))
ax.text(0.25, 0.65, "Generator", ha='center')
# Discriminator
ax.add_patch(patches.Rectangle((0.6, 0.5), 0.3, 0.4, color='salmon'))
ax.text(0.75, 0.65, "Discriminator", ha='center')
# Arrows
ax.annotate("", xy=(0.4, 0.7), xytext=(0.6, 0.7), arrowprops=dict(arrowstyle="->"))
ax.text(0.5, 0.75, "Synthetic Image", ha='center')

ax.annotate("", xy=(0.9, 0.7), xytext=(0.6, 0.7), arrowprops=dict(arrowstyle="->"))
ax.text(0.75, 0.85, "Adversarial Feedback", ha='center')

ax.axis('off')
plt.title("GAN Architecture Flow")
plt.show()
```

🧠 2. Vision Transformer Flow

📈 With `plotly.graph_objects`

```
import plotly.graph_objects as go

fig = go.Figure()

steps = ["Image", "Patchify", "Embedding", "Self-Attention", "Classification"]
for i, step in enumerate(steps):
    fig.add_shape(type="rect", x0=i, x1=i+0.8, y0=0, y1=0.5,
                  line=dict(color="black"), fillcolor="lightgreen")
    fig.add_annotation(x=i+0.4, y=0.25, text=step, showarrow=False)

# Arrows
for i in range(len(steps)-1):
    fig.add_annotation(x=i+0.8, y=0.25, ax=i+0.9, ay=0.25,
                       arrowhead=2, arrowsize=1.5)

fig.update_layout(height=200, width=800, showlegend=False,
                  title="Vision Transformer Flow", plot_bgcolor="white",
                  margin=dict(l=20, r=20, t=40, b=20))
fig.show()
```

🔗 3. Multimodal Fusion Diagram

📌 With `graphviz`

```
from graphviz import Digraph

dot = Digraph()

dot.node('I', 'Image → CNN')
dot.node('Q', 'Question → BERT')
dot.node('F', 'Fusion Layer')
dot.node('A', 'Answer Prediction')

dot.edge('I', 'F')
dot.edge('Q', 'F')
dot.edge('F', 'A')

dot.render('multimodal_fusion', format='png', cleanup=True)
```

⚖️ 4. Fairness Audit Flow

📉 With `graphviz`

```
flow = Digraph()

flow.node("D", "Image Dataset")

flow.node("B", "Bias Detection")

flow.node("M", "Metrics Report")

flow.node("S", "Mitigation Strategy")

flow.edge("D", "B")

flow.edge("B", "M")

flow.edge("M", "S")

flow.render("fairness_flow", format="png", cleanup=True)
```

📐 5. Paper Composition Map

📋 With `matplotlib` Text Annotations

```
fig, ax = plt.subplots(figsize=(10, 2))

sections = ["Title", "Abstract", "Introduction", "Methodology", "Experiments", "Conclusion"]

for i, section in enumerate(sections):

    ax.text(i, 0.5, section, fontsize=12, ha='center', bbox=dict(boxstyle="round", facecolor="lightyellow"))

    if i < len(sections) - 1:

        ax.annotate("", xy=(i+0.5, 0.5), xytext=(i+0.9, 0.5), arrowprops=dict(arrowstyle="->"))

ax.axis('off')

plt.title("Research Paper Composition Flow")

plt.show()
```

🧩 Full-Module Widgets

🔮 VQA Answer Selector Widget

```
from ipywidgets import interact, Dropdown
import matplotlib.pyplot as plt

questions = Dropdown(
    options=["What is on the table?", "How many dogs?", "Is it daytime?"],
    description='Question:',
)
def predict_answer(q):
    img = plt.imread("sample.jpg")
    plt.imshow(img)
    plt.title(f"Predicted Answer: {q} → 'Bottle'")
    plt.axis('off')
    plt.show()
interact(predict_answer, q=questions)
```

⚖️ Bias Threshold Slider

```
from ipywidgets import FloatSlider
import matplotlib.pyplot as plt
import numpy as np
def audit(threshold):
    disparity = max(0, 1 - threshold)
    plt.bar(['Privileged', 'Unprivileged'], [1.0, 1.0 - disparity], color=['green', 'red'])
    plt.title(f"Fairness Audit @ Bias Threshold: {threshold}")
    plt.ylabel("Acceptance Rate")
    plt.show()
interact(audit, threshold=FloatSlider(min=0, max=1, step=0.1, value=0.5))
```

🎨 Segmentation Overlay Widget

```
from ipywidgets import FloatSlider
import numpy as np
import matplotlib.pyplot as plt


def overlay(alpha=0.4):
    image = plt.imread("dashcam.jpg")
    mask = np.load("lane_mask.npy")
    plt.imshow(image)
    plt.imshow(mask, cmap='Reds', alpha=alpha)
    plt.title("Lane Segmentation Overlay")
    plt.axis('off')
    plt.show()
interact(overlay, alpha=FloatSlider(min=0.0, max=1.0, step=0.05, value=0.4))
```

🎛️ Interactive Widgets for Simulation

Using `ipywidgets`, we can simulate model behavior and user input controls directly in notebooks.

🔧 GAN: Adjust Noise Vector Dimensionality

```
from ipywidgets import interact
import torch
import matplotlib.pyplot as plt

def generate_image(noise_dim=100):
    noise = torch.randn(1, noise_dim, 1, 1)
    # Placeholder generator logic
    img = noise.squeeze().detach().numpy()[:28, :28]  # Fake output
    plt.imshow(img, cmap='gray')
    plt.title(f"GAN Output (dim={noise_dim})")
    plt.axis('off')
    plt.show()

interact(generate_image, noise_dim=(10, 200, 10))
```

🧠 Transformer Attention Head Selector

```
from ipywidgets import Dropdown
import matplotlib.pyplot as plt
import numpy as np

attention_heads = Dropdown(
    options=['Head 1', 'Head 2', 'Head 3'],
    value='Head 1',
    description='Attention Head:',
)

def plot_attention(head):
    attn_map = np.random.rand(8, 8)  # Fake 8x8 patch attention
    plt.imshow(attn_map, cmap='viridis')
    plt.title(f"Attention Map: {head}")
    plt.colorbar()
    plt.show()

attention_heads.observe(lambda change: plot_attention(change['new']), names='value')
plot_attention(attention_heads.value)
```

⚙️ 1. Real-Time Widget ↔ Pretrained Model Link

🎮 Example: VQA Widget with Inference Logic

```
import torch
from transformers import BlipProcessor, BlipForQuestionAnswering
from PIL import Image
from ipywidgets import interact, Dropdown
# Load pretrained BLIP VQA model
processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")
model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base")
def vqa_inference(image_path, question):
    image = Image.open(image_path).convert("RGB")
    inputs = processor(image, question, return_tensors="pt")
    out = model.generate(**inputs)
    answer = processor.decode(out[0], skip_special_tokens=True)
    return answer
# Interactive widget
def ask(question):
    answer = vqa_inference("vqa_sample.jpg", question)
    print(f"Answer: {answer}")
interact(ask, question=Dropdown(options=[
    "What is the person doing?",
    "Is the dog sitting?",
    "What color is the backpack?"
], description="Ask:"))
```

☝️ This links directly to a pretrained model with instant answers. You can extend this to other models (YOLOv8, CLIP, segmentation networks) using similar logic.

🚀 2. FastAPI Backend for Deployment

Create a RESTful API that serves predictions to frontend widgets or external apps.

📦 Install Dependencies

pip install fastapi uvicorn pillow transformers torch torchvision

📁 File Structure

cv-api/

│

├── app.py

├── models/

│ ├── vqa_model.py

│ └── segmentation_model.py

├── static/

│ └── sample.jpg

🔧 VQA Endpoint (app.py)

```
from fastapi import FastAPI, UploadFile, Form
from models.vqa_model import answer_question
app = FastAPI()
@app.post("/vqa")
async def vqa(image: UploadFile, question: str = Form(...)):
    img_bytes = await image.read()
    result = answer_question(img_bytes, question)
    return {"answer": result}
```

🔍 Model Logic (models/vqa_model.py)

```
from transformers import BlipProcessor, BlipForQuestionAnswering
from PIL import Image
import io
```

```
processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")
model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base")


def answer_question(image_bytes, question):
    image = Image.open(io.BytesIO(image_bytes)).convert("RGB")
    inputs = processor(image, question, return_tensors="pt")
    out = model.generate(**inputs)
    return processor.decode(out[0], skip_special_tokens=True)
```

🌐 Deployment & Demo Connection

- Run locally:

  ```
  uvicorn app:app --reload
  ```

- To connect a widget frontend:
- Use JavaScript (`fetch()` or `axios`)
- Or send requests via Python notebooks:

```
import requests

files = {'image': open('vqa_sample.jpg', 'rb')}
data = {'question': 'What is in the hand?'}
res = requests.post("http://localhost:8000/vqa", files=files, data=data)
print(res.json())
```

🧩 Next Modules You Can Link This Way

Task	Model	Endpoint
Object Detection	YOLOv8	`/detect`
Segmentation	U-Net / DINO-SAM	`/segment`
Image Classification	CLIP / ResNet	`/classify`
Bias Audit	Custom fairness report	`/audit`

🚀 Building: CV Lab Overview

A full-stack application that lets users interact with cutting-edge CV models like:

- 🧠 VQA (Visual Question Answering)
- 🎯 Object Detection (YOLOv8)
- 🎨 Image Segmentation (U-Net, SAM)
- 📊 Bias Auditing Tools

All accessed via:

- 💻 Web UI (React + Tailwind)
- 🧠 Backend APIs (FastAPI)
- ☁️ Serverless Deployment (Render, Vercel, or Azure Functions)
- 🔍 Real-time visualization (Plotly, OpenCV.js, TensorFlow.js)

🧱 Frontend Scaffold (React + Tailwind)

📁 Directory Structure

```
cv-lab-frontend/
│
├── public/
│   └── images/
├── src/
│   ├── components/
│   │   ├── VQADemo.jsx
│   │   ├── DetectionViewer.jsx
│   │   ├── SegmentationOverlay.jsx
│   ├── pages/
│   │   └── Dashboard.jsx
│   ├── App.jsx
│   └── index.js
├── tailwind.config.js
├── package.json
```

🧩 **Sample Component (VQADemo.jsx)**

```
import { useState } from "react";
import axios from "axios";
export default function VQADemo() {
  const [question, setQuestion] = useState("");
  const [answer, setAnswer] = useState("");
  const askQuestion = async () => {
    const formData = new FormData();
    formData.append("image", document.querySelector("#vqa-image").files[0]);
    formData.append("question", question);
    const res = await axios.post("https://api.example.com/vqa", formData);
    setAnswer(res.data.answer);
  };
  return (
    <div className="p-6">
      <input id="vqa-image" type="file" />
      <input
        type="text"
        value={question}
        onChange={(e) => setQuestion(e.target.value)}
        placeholder="Ask something..."
        className="border p-2"
      />
      <button onClick={askQuestion} className="bg-blue-500 text-white p-2">Ask</button>
      <p className="mt-4">Answer: {answer}</p>
    </div>
  );
}
```

🔧 Backend APIs (FastAPI + PyTorch/HuggingFace)

🌐 Live Endpoint Example

```
@app.post("/vqa")
async def vqa(image: UploadFile, question: str = Form(...)):
    image_bytes = await image.read()
    result = answer_question(image_bytes, question)
    return {"answer": result}
```

✨ Extend with `/detect`, `/segment`, `/audit`, etc.

☁️ Deployment Guide: Serverless API Hosting

☑ Options

- **Render**: Easy FastAPI deploy via GitHub
- **Vercel (frontend)** + **Supabase functions (backend)**
- **Azure Functions**: For scalability + identity management
- **HuggingFace Spaces**: For model hosting + UI together

📊 Visualization Integration

🧠 In-browser Visuals

- `Plotly.js` for charts (bias metrics, loss curves)
- `OpenCV.js` for segmentation mask blending
- `TensorFlow.js` for real-time demos (e.g. webcam object detection)

🧩 Future Extensions

Feature	Tech Stack
Webcam object detection	TensorFlow.js + React hooks
U-Net mask overlay	Canvas + OpenCV.js
Bias audit report visualizer	Plotly bar charts & heatmaps
VQA image gallery	Cloud upload + MongoDB GridFS

🧠 Final Touches: Dashboard Design

Create a landing dashboard with cards for each demo:

```
const models = ["VQA", "Segmentation", "Detection", "Fairness Audit"];
models.map((name) => (
  <div className="bg-white p-4 rounded shadow">
    <h2>{name}</h2><button>Launch Demo</button>
  </div>));
```

A complete full-stack scaffold for your **serverless Computer Vision Lab**, including:

- ☑ Web frontend (React + Tailwind)
- ☑ Backend API (FastAPI) with pre-trained model endpoints
- ☑ Deployment guides (for Render, Vercel, or HuggingFace Spaces)
- ☑ Real-time inference integration
- ☑ File structure and starter code snippets

🧱 Project Directory Structure

```
cv-lab/
├── backend/
│   ├── app.py
│   ├── models/
│   │   ├── vqa_model.py
│   │   ├── detection_model.py
│   │   └── segmentation_model.py
│   ├── requirements.txt
│   └── static/
│       └── sample.jpg
│
├── frontend/
│   ├── public/
│   ├── src/
│   │   ├── components/
│   │   │   ├── VQADemo.jsx
│   │   │   ├── ObjectDetection.jsx
│   │   │   └── SegmentationMask.jsx
│   │   ├── pages/Dashboard.jsx
│   │   ├── App.jsx
│   │   └── index.js
│   ├── tailwind.config.js
```

```
│   └── package.json
```

🧠 **Backend API Scaffold (FastAPI)**

🔧 `app.py`

```
from fastapi import FastAPI, UploadFile, Form
from models.vqa_model import answer_question
from models.detection_model import detect_objects
from models.segmentation_model import segment_image
app = FastAPI()
@app.post("/vqa")
async def vqa_endpoint(image: UploadFile, question: str = Form(...)):
    img_bytes = await image.read()
    result = answer_question(img_bytes, question)
    return {"answer": result}
@app.post("/detect")
async def detection_endpoint(image: UploadFile):
    img_bytes = await image.read()
    result = detect_objects(img_bytes)
    return {"boxes": result}


@app.post("/segment")
async def segmentation_endpoint(image: UploadFile):
    img_bytes = await image.read()
    mask = segment_image(img_bytes)
    return {"segmentation_mask": mask.tolist()}
```

🔍 **`vqa_model.py`**

```
from transformers import BlipProcessor, BlipForQuestionAnswering
from PIL import Image
```

```
import io
processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")
model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base")
def answer_question(img_bytes, question):
    image = Image.open(io.BytesIO(img_bytes)).convert("RGB")
    inputs = processor(image, question, return_tensors="pt")
    out = model.generate(**inputs)
    return processor.decode(out[0], skip_special_tokens=True)
```

📦 Add similar modules for YOLO detection and U-Net segmentation in `detection_model.py` and `segmentation_model.py`.

💻 **Frontend UI Scaffold (React + Tailwind)**

📦 Setup

```
npx create-react-app frontend
cd frontend
npm install axios tailwindcss
npx tailwindcss init
```

Configure `tailwind.config.js` and add Tailwind directives to `src/index.css`.

🧩 **`VQADemo.jsx`**

```
import { useState } from "react";
import axios from "axios";
export default function VQADemo() {
  const [question, setQuestion] = useState("");
  const [answer, setAnswer] = useState("");
  async function handleSubmit() {
    const imageInput = document.getElementById("vqa-img");
    const formData = new FormData();
    formData.append("image", imageInput.files[0]);
```

```
    formData.append("question", question);
    const res = await axios.post("https://your-backend-url/vqa", formData);
    setAnswer(res.data.answer);
  }
  return (
    <div className="p-6 bg-white rounded shadow">
      <input id="vqa-img" type="file" />
      <input
        type="text"
        value={question}
        onChange={(e) => setQuestion(e.target.value)}
        placeholder="Ask your question..."
        className="border p-2 m-2"
      />
      <button onClick={handleSubmit} className="bg-blue-600 text-white px-4 py-2">
        Submit
      </button>
      <p className="mt-4">Answer: {answer}</p>
    </div>
  );
}
```

🔁 Replicate this pattern for other modules (`ObjectDetection.jsx`, `SegmentationMask.jsx`) using their respective endpoints.

🚀 Deployment Guide

🧠 FastAPI Server

- Push `backend/` to GitHub
- Deploy to Render: Add a `start` command `uvicorn app:app --host 0.0.0.0 --port 10000`
- Set runtime: Python 3.10+, allow file uploads
- Add environment variables if needed

💻 React Frontend

- Push `frontend/` to GitHub
- Deploy with Vercel or Netlify
- Point API endpoints to the deployed backend

☑️ Optional: Bundle frontend with HuggingFace Spaces for a single public-facing demo.

🎛️ Add Live Visualization Libraries

- Use OpenCV.js for bounding box overlay
- Use TensorFlow.js for webcam-based demos
- Use Plotly.js for bias scores and training metrics
- Add canvas layers for segmentation masks