Interactive AI Demos - Lab

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

Lab Overview

Goal: Build THREE production-ready Streamlit apps to master interactive AI development.

Apps:

1. **Sentiment Analysis Dashboard** (~90 min)
2. **Image Classification App** (~90 min)
3. **YouTube Video Summarizer** (~120 min)

Skills You'll Learn:

- Streamlit fundamentals (widgets, layout, state)
- Integrating ML models
- Error handling and UX
- Deployment to Streamlit Cloud

Prerequisites: Python 3.8+, basic ML knowledge

Setup: Environment Preparation

Step 1: Create project directory

```
mkdir interactive-demos-lab
cd interactive-demos-lab
```

Step 2: Create virtual environment

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

Step 3: Install Streamlit

```
pip install streamlit
streamlit hello  # Test installation
```

You should see Streamlit's demo app at `http://localhost:8501`

Exercise 0: Hello Streamlit

Create `test_app.py`:

```
import streamlit as st

st.title("My First Streamlit App")

name = st.text_input("What's your name?")
if name:
    st.write(f"Hello, {name}!")

age = st.slider("How old are you?", 0, 100, 25)
st.write(f"You are {age} years old.")

if st.button("Click me!"):
    st.balloons()  # Fun celebration animation
```

Run:

```
streamlit run test_app.py
```

Exercise 0: Understanding State

Modify `test_app.py` to add a counter:

```
import streamlit as st

st.title("Counter Example")

# ❌ This WON'T work (resets on each run)
counter_bad = 0

if st.button("Increment (broken)"):
    counter_bad += 1

st.write(f"Broken counter: {counter_bad}")

# ✅ This WILL work (persists across reruns)
if 'counter' not in st.session_state:
    st.session_state.counter = 0

if st.button("Increment (working)"):
    st.session_state.counter += 1

st.write(f"Working counter: {st.session_state.counter}")
```

App 1: Sentiment Analysis Dashboard

Goal: Build a text sentiment analyzer with:

- Real-time prediction
- Confidence visualization
- History tracking

Final Features:

- Text area for user input
- Sentiment prediction (positive/negative/neutral)
- Confidence bar chart
- History of past predictions
- Clear history button

App 1 - Step 1: Install Dependencies

```
pip install transformers torch
```

Create `sentiment_app.py`:

```
import streamlit as st
from transformers import pipeline
import pandas as pd

st.set_page_config(
    page_title="Sentiment Analyzer",
    page_icon="😊",
    layout="wide"
)

st.title("😊 Sentiment Analysis Dashboard")
st.markdown("Analyze the sentiment of any text using AI")
```

Run: `streamlit run sentiment_app.py`

App 1 - Step 2: Load Model with Caching

Add to `sentiment_app.py`:

```
@st.cache_resource
def load_model():
    """Load model once and cache it"""
    return pipeline(
        "sentiment-analysis",
        model="distilbert-base-uncased-finetuned-sst-2-english"
    )

# Load model
with st.spinner("Loading AI model... (this may take a minute)"):
    classifier = load_model()

st.success("✅ Model loaded successfully!")
```

Why `@st.cache_resource`?

- Model loads **once** on first run
- Subsequent reruns reuse cached model

App 1 - Step 3: User Input

Add input section:

```
st.header("Enter Text to Analyze")

text = st.text_area(
    "Type or paste your text here:",
    height=150,
    placeholder="Example: I love this new feature! It's amazing."
)

# Provide example buttons
col1, col2, col3 = st.columns(3)

with col1:
    if st.button("Example: Positive"):
        st.session_state.example_text = "I absolutely love this product! Best purchase ever."
        st.rerun()

with col2:
    if st.button("Example: Negative"):
        st.session_state.example_text = "This is terrible. Worst experience of my life."
        st.rerun()

with col3:
    if st.button("Example: Neutral"):
        st.session_state.example_text = "The product arrived on time. It works as described."
        st.rerun()

# Use example text if set
if 'example_text' in st.session_state:
    text = st.session_state.example_text
    del st.session_state.example_text
```

App 1 - Step 4: Prediction Logic

```
if text:
    with st.spinner("Analyzing sentiment..."):
        result = classifier(text)[0]

    label = result['label']
    score = result['score']

    # Display result with color
    if label == "POSITIVE":
        st.success(f"✅ **Sentiment**: {label}")
    else:
        st.error(f"❌ **Sentiment**: {label}")

    # Confidence score
    st.metric("Confidence", f"{score:.2%}")

    # Visualization
    chart_data = pd.DataFrame({
        'Sentiment': [label],
        'Confidence': [score]
    })
    st.bar_chart(chart_data.set_index('Sentiment'))
else:
    st.info("👆 Enter some text above to analyze")
```

App 1 - Step 5: History Tracking

Initialize session state:

```
if 'history' not in st.session_state:
    st.session_state.history = []
```

Store predictions:

```
if text:
    with st.spinner("Analyzing sentiment..."):
        result = classifier(text)[0]

    label = result['label']
    score = result['score']

    # Add to history
    st.session_state.history.append({
        'text': text[:50] + '...' if len(text) > 50 else text,
        'sentiment': label,
        'confidence': score
    })
```

App 1 - Step 6: Display History

Add history section:

```
st.header("📊 Prediction History")

if st.session_state.history:
    # Convert to DataFrame
    df = pd.DataFrame(st.session_state.history)

    # Display table
    st.dataframe(
        df,
        use_container_width=True,
        hide_index=True
    )

    # Clear history button
    if st.button("🗑️ Clear History"):
        st.session_state.history = []
        st.rerun()
else:
    st.info("No predictions yet. Analyze some text to build history!")
```

App 1 - Step 7: Sidebar Configuration

Add sidebar with options:

```
with st.sidebar:
    st.header("⚙️ Settings")

    show_raw = st.checkbox("Show raw model output", value=False)

    st.markdown("——")
    st.markdown("### About")
    st.markdown("""
    This app uses **DistilBERT** fine-tuned on SST-2 dataset.

    **Model**: `distilbert-base-uncased-finetuned-sst-2-english`

    **Capabilities**:
    - Binary sentiment (positive/negative)
    - English text only
    - Max 512 tokens
    """)

# Show raw output if enabled
if text and show_raw:
    st.json(result)
```

App 1 - Testing Checklist

Test your app:

- [ ] Model loads without errors
- [ ] Text input works
- [ ] Example buttons populate text area
- [ ] Predictions are accurate
- [ ] History updates correctly
- [ ] Clear history works
- [ ] Sidebar settings toggle raw output
- [ ] App doesn't crash on empty input

Try edge cases:

- Very long text (>512 tokens)
- Non-English text

App 2: Image Classification

Goal: Build an image classifier with:

- File upload (drag & drop)
- Multiple model selection
- Top-K predictions visualization
- Batch processing

Models (choose one or support multiple):

- ResNet50 (ImageNet)
- MobileNetV2 (lightweight)
- EfficientNet

App 2 - Step 1: Setup

Install dependencies:

```
pip install torch torchvision pillow
```

Create `image_classifier_app.py`:

```
import streamlit as st
from PIL import Image
import torch
from torchvision import models, transforms
import json

st.set_page_config(
    page_title="Image Classifier",
    page_icon="🖼️",
    layout="wide"
)

st.title("🖼️ Image Classification App")
st.markdown("Upload images to classify them using state-of-the-art models")
```

App 2 - Step 2: Load Model and Labels

```python
@st.cache_resource
def load_model(model_name):
    """Load pretrained model"""
    if model_name == "ResNet50":
        model = models.resnet50(pretrained=True)
    elif model_name == "MobileNetV2":
        model = models.mobilenet_v2(pretrained=True)

    model.eval()  # Set to evaluation mode
    return model

@st.cache_data
def load_labels():
    """Load ImageNet class labels"""
    url = "https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json"
    import urllib.request
    with urllib.request.urlopen(url) as f:
        return json.load(f)

labels = load_labels()
```

App 2 - Step 3: Preprocessing Pipeline

```
def preprocess_image(image):
    """Preprocess image for model"""
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])

    # Convert to RGB if needed
    if image.mode ≠ 'RGB':
        image = image.convert('RGB')

    return transform(image).unsqueeze(0)  # Add batch dimension
```

App 2 - Step 4: Sidebar Model Selection

```
with st.sidebar:
    st.header("⚙️ Model Settings")

    model_name = st.selectbox(
        "Choose Model",
        ["ResNet50", "MobileNetV2"],
        help="ResNet50 is more accurate, MobileNetV2 is faster"
    )

    top_k = st.slider(
        "Number of predictions",
        min_value=1,
        max_value=10,
        value=5,
        help="Show top K predictions"
    )

    confidence_threshold = st.slider(
        "Confidence threshold",
        min_value=0.0,
        max_value=1.0,
        value=0.1,
        step=0.05,
        help="Only show predictions above this confidence"
    )

# Load selected model
model = load_model(model_name)
```

App 2 - Step 5: File Upload

```
st.header("📤 Upload Image")

uploaded_file = st.file_uploader(
    "Choose an image...",
    type=['jpg', 'jpeg', 'png'],
    help="Drag and drop or click to upload"
)

if uploaded_file is not None:
    # Display uploaded image
    image = Image.open(uploaded_file)

    col1, col2 = st.columns(2)

    with col1:
        st.subheader("📷 Uploaded Image")
        st.image(image, use_column_width=True)

        # Show image info
        st.info(f"""
        **Filename**: {uploaded_file.name}
        **Size**: {image.size}
        **Mode**: {image.mode}
        **Format**: {image.format}
        """)
```

App 2 - Step 6: Classification

```
with col2:
    st.subheader("🔍 Predictions")

    with st.spinner("Classifying ... "):
        # Preprocess
        input_tensor = preprocess_image(image)

        # Inference
        with torch.no_grad():
            output = model(input_tensor)

        # Get probabilities
        probabilities = torch.nn.functional.softmax(output[0], dim=0)

        # Get top K predictions
        top_probs, top_indices = torch.topk(probabilities, top_k)

        # Display results
        for i, (prob, idx) in enumerate(zip(top_probs, top_indices)):
            prob_value = prob.item()

            if prob_value >= confidence_threshold:
                label = labels[idx.item()]

                st.metric(
                    label=f"#{i+1}: {label}",
                    value=f"{prob_value:.2%}"
                )
                st.progress(prob_value)
```

App 2 - Step 7: Batch Processing

Add batch upload capability:

```
st.header("📦 Batch Processing")

batch_files = st.file_uploader(
    "Upload multiple images",
    type=['jpg', 'jpeg', 'png'],
    accept_multiple_files=True
)

if batch_files:
    st.write(f"Processing {len(batch_files)} images...")

    results = []

    for uploaded_file in batch_files:
        image = Image.open(uploaded_file)
        input_tensor = preprocess_image(image)

        with torch.no_grad():
            output = model(input_tensor)

        probabilities = torch.nn.functional.softmax(output[0], dim=0)
        top_prob, top_idx = torch.max(probabilities, 0)

        results.append({
            'filename': uploaded_file.name,
            'prediction': labels[top_idx.item()],
            'confidence': f"{top_prob.item():.2%}"
        })
```

App 2 - Step 8: Display Batch Results

```
# Display results table
import pandas as pd
df = pd.DataFrame(results)
st.dataframe(df, use_container_width=True)

# Download results
csv = df.to_csv(index=False)
st.download_button(
    label="📥 Download Results (CSV)",
    data=csv,
    file_name="classification_results.csv",
    mime="text/csv"
)

# Display thumbnails
st.subheader("Thumbnails")
cols = st.columns(5)
for idx, uploaded_file in enumerate(batch_files):
    with cols[idx % 5]:
        image = Image.open(uploaded_file)
        st.image(image, caption=results[idx]['prediction'], use_column_width=True)
```

App 2 - Error Handling

Add robust error handling:

```
def classify_image_safe(image, model, labels):
    """Classify image with error handling"""
    try:
        input_tensor = preprocess_image(image)

        with torch.no_grad():
            output = model(input_tensor)

        probabilities = torch.nn.functional.softmax(output[0], dim=0)
        return probabilities

    except Exception as e:
        st.error(f"Error during classification: {str(e)}")
        return None

# Use in classification code
if uploaded_file:
    image = Image.open(uploaded_file)

    probabilities = classify_image_safe(image, model, labels)

    if probabilities is not None:
        # Display results
        pass
```

App 2 - Testing

Test scenarios:

- [ ] Single image upload works
- [ ] Batch upload processes all images
- [ ] Model switching works without errors
- [ ] Top-K slider updates predictions
- [ ] Confidence threshold filters correctly
- [ ] CSV download contains correct data
- [ ] Thumbnails display properly

Edge cases:

- Grayscale images
- RGBA images (with alpha channel)
- Very large images (>10MB)

App 3: YouTube Video Summarizer

Goal: Most comprehensive app with:

- YouTube URL input
- Transcript extraction
- AI-powered summarization (using Gemini API)
- Chat interface for follow-up questions
- Deployment to cloud

Technologies:

- `youtube-transcript-api` for transcripts
- Gemini API for summarization
- Streamlit chat components

App 3 - Step 1: Setup

Install dependencies:

```
pip install youtube-transcript-api google-generativeai
```

Get Gemini API key:

1. Go to https://makersuite.google.com/app/apikey
2. Create API key
3. Save it for later

Create `youtube_summarizer_app.py`:

```
import streamlit as st
from youtube_transcript_api import YouTubeTranscriptApi
import google.generativeai as genai
import re

st.set_page_config(
```

App 3 - Step 2: API Key Configuration

```
with st.sidebar:
    st.header("🔑 API Configuration")

    api_key = st.text_input(
        "Enter Gemini API Key",
        type="password",
        help="Get your key from https://makersuite.google.com/app/apikey"
    )

    if api_key:
        genai.configure(api_key=api_key)
        st.success("✅ API key configured")
    else:
        st.warning("⚠️ Enter API key to continue")

    st.markdown("——")
    st.markdown("""
    ### How to use:
    1. Enter Gemini API key
    2. Paste YouTube URL
    3. Get instant summary
    4. Ask follow-up questions
    """)
```

App 3 - Step 3: Extract Video ID

```
def extract_video_id(url):
    """Extract video ID from various YouTube URL formats"""
    patterns = [
        r'(?:youtube\.com\/watch\?v=|youtu\.be\/)([^&\n?#]+)',
        r'youtube\.com\/embed\/([^&\n?#]+)',
        r'youtube\.com\/v\/([^&\n?#]+)'
    ]

    for pattern in patterns:
        match = re.search(pattern, url)
        if match:
            return match.group(1)

    return None

def get_video_info(video_id):
    """Get video thumbnail and title"""
    thumbnail = f"https://img.youtube.com/vi/{video_id}/maxresdefault.jpg"
    return thumbnail
```

App 3 - Step 4: Fetch Transcript

```
def fetch_transcript(video_id):
    """Fetch video transcript"""
    try:
        transcript_list = YouTubeTranscriptApi.get_transcript(video_id)

        # Combine all text
        transcript = " ".join([entry['text'] for entry in transcript_list])

        return transcript, None

    except Exception as e:
        error_msg = str(e)

        if "No transcript" in error_msg:
            return None, "❌ No transcript available for this video"
        elif "disabled" in error_msg:
            return None, "❌ Subtitles are disabled for this video"
        else:
            return None, f"❌ Error: {error_msg}"
```

App 3 - Step 5: Summarize with Gemini

```
@st.cache_data
def summarize_transcript(_model, transcript, summary_type="concise"):
    """Generate summary using Gemini"""

    prompts = {
        "concise": f"""Provide a concise summary (3-5 sentences) of this video transcript:

{transcript}

Summary:""",

        "detailed": f"""Provide a detailed summary with:
- Main topic
- Key points (bulleted)
- Important insights
- Conclusion

Transcript:
{transcript}

Summary:""",

        "bullet": f"""Create a bullet-point summary of the main ideas from this transcript:

{transcript}

Bullet points:"""
    }

    response = _model.generate_content(prompts[summary_type])
    return response.text
```

App 3 - Step 6: Main Interface

```
if not api_key:
    st.info("👈 Enter your Gemini API key in the sidebar to get started")
    st.stop()

# Initialize model
model = genai.GenerativeModel('gemini-pro')

st.header("📝 Enter YouTube URL")

url = st.text_input(
    "YouTube URL:",
    placeholder="https://www.youtube.com/watch?v=..."
)

# Summary type
col1, col2, col3 = st.columns(3)
with col1:
    summary_type = st.radio(
        "Summary Type",
        ["concise", "detailed", "bullet"],
        horizontal=True
    )

if url:
    video_id = extract_video_id(url)

    if not video_id:
        st.error("❌ Invalid YouTube URL")
        st.stop()
```

App 3 - Step 7: Display Video & Summary

```
# Display video thumbnail
col1, col2 = st.columns([1, 2])

with col1:
    st.image(
        get_video_info(video_id),
        caption="Video Thumbnail",
        use_column_width=True
    )

with col2:
    st.video(url)

# Fetch transcript
with st.spinner("Fetching transcript ... "):
    transcript, error = fetch_transcript(video_id)

if error:
    st.error(error)
    st.stop()

# Show transcript stats
st.info(f"""
**Transcript Stats:**
- Characters: {len(transcript):,}
- Words: {len(transcript.split()):,}
""")
```

App 3 - Step 8: Generate Summary

```
# Generate summary
st.header("📋 Summary")

with st.spinner("Generating AI summary..."):
    summary = summarize_transcript(model, transcript, summary_type)

st.markdown(summary)

# Download buttons
col1, col2 = st.columns(2)

with col1:
    st.download_button(
        "📥 Download Summary",
        data=summary,
        file_name=f"summary_{video_id}.txt",
        mime="text/plain"
    )

with col2:
    st.download_button(
        "📥 Download Transcript",
        data=transcript,
        file_name=f"transcript_{video_id}.txt",
        mime="text/plain"
    )
```

App 3 - Step 9: Chat Interface

```
st.header("💬 Chat with Video")

# Initialize chat history
if 'messages' not in st.session_state:
    st.session_state.messages = []

# Store transcript in session for chat context
if 'current_transcript' not in st.session_state:
    st.session_state.current_transcript = transcript

# Display chat history
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# Chat input
if prompt := st.chat_input("Ask a question about the video..."):
    # Add user message
    st.session_state.messages.append({
        "role": "user",
        "content": prompt
    })

    with st.chat_message("user"):
        st.markdown(prompt)
```

App 3 - Step 10: Chat Response

```
        # Generate assistant response
        with st.chat_message("assistant"):
            with st.spinner("Thinking ... "):
                # Create context-aware prompt
                chat_prompt = f"""Based on this video transcript:

{st.session_state.current_transcript}

User question: {prompt}

Answer:"""

                response = model.generate_content(chat_prompt)
                answer = response.text

                st.markdown(answer)

        # Add assistant message
        st.session_state.messages.append({
            "role": "assistant",
            "content": answer
        })

    # Clear chat button
    if st.button("🗑️ Clear Chat"):
        st.session_state.messages = []
        st.rerun()
```

Deployment: Preparing for Production

Step 1: Create `requirements.txt`:

```
streamlit==1.29.0
youtube-transcript-api==0.6.1
google-generativeai==0.3.1
transformers==4.36.0
torch==2.1.0
torchvision==0.16.0
pillow==10.1.0
```

Step 2: Create `.gitignore`:

```
venv/
__pycache__/
*.pyc
.env
.streamlit/secrets.toml
*.pt
*.pth
```

Deployment: Streamlit Cloud

Step 1: Push to GitHub:

```
git init
git add .
git commit -m "Initial commit - YouTube Summarizer"
git branch -M main
git remote add origin https://github.com/YOUR_USERNAME/youtube-summarizer.git
git push -u origin main
```

Step 2: Deploy on Streamlit Cloud:

1. Go to https://share.streamlit.io/
2. Sign in with GitHub
3. Click "New app"
4. Select your repository
5. Choose branch: `main`
6. Main file: `youtube_summarizer_app.py`

Deployment: Secrets Management

In Streamlit Cloud dashboard:

1. Click on your app
2. Go to "Settings" → "Secrets"
3. Add:

```
GEMINI_API_KEY = "your-actual-api-key-here"
```

Update your app to use secrets:

```
# In sidebar, add:
if "GEMINI_API_KEY" in st.secrets:
    api_key = st.secrets["GEMINI_API_KEY"]
    st.success("✅ Using stored API key")
else:
    api_key = st.text_input("Enter Gemini API Key", type="password")
```

Now users don't need to enter API key every time!

Lab Submission

What to submit:

1. **GitHub repository** with all three apps
2. **Deployed links**:
   - Sentiment Analysis: `your-sentiment.streamlit.app`
   - Image Classifier: `your-classifier.streamlit.app`
   - YouTube Summarizer: `your-summarizer.streamlit.app`
3. **README.md** with:
   - App descriptions
   - Installation instructions
   - Usage examples
   - Screenshots

Bonus points:

- Multi-page app combining all three

Grading Rubric

Category	Points	Criteria
Functionality	40	All features work correctly
Code Quality	20	Clean, commented, follows best practices
UX/UI	20	Intuitive, responsive, error handling
Deployment	10	Successfully deployed, accessible
Documentation	10	Clear README, comments
Bonus	+10	Multi-page, custom features, innovation

Total: 100 points (+10 bonus)

Resources & Further Learning

Streamlit Documentation:

- Gallery: https://streamlit.io/gallery
- API Reference: https://docs.streamlit.io/
- Community: https://discuss.streamlit.io/

Model Hubs:

- Hugging Face: https://huggingface.co/models
- TorchVision: https://pytorch.org/vision/stable/models.html

Deployment Platforms:

- Streamlit Cloud: https://streamlit.io/cloud
- Hugging Face Spaces: https://huggingface.co/spaces
- Render: https://render.com/

Inspiration:

Questions?

Need help?

- Post on course discussion board
- Office hours: [Time/Location]
- Email: [instructor email]

Show off your apps!

- Share on Twitter with #CS203IITGn
- Post in course Slack/Discord

Happy Building! 🚀