Interactive AI Demos

Lab Session

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

Duration: 3 hours

Lab Objectives

1. **Streamlit Basics**: Build simple, interactive data apps
2. **Model Integration**: Connect ML models to frontend UI
3. **State Management**: Handle chat history and session variables
4. **UX Best Practices**: Loading states, error handling, feedback
5. **Deployment**: Deploy to Streamlit Cloud or Hugging Face Spaces

By the end: You'll have 3-4 working demos deployed publicly.

Prerequisites Check

Required installations:

```
# Core frameworks
pip install streamlit gradio pandas numpy matplotlib

# ML libraries
pip install transformers torch pillow

# Additional tools
pip install youtube-transcript-api python-dotenv
```

Test installation:

```
streamlit hello
# Should open demo app in browser
```

Part 1: Streamlit Fundamentals

Understanding the basics before building apps

Exercise 1.1: Hello Streamlit (15 min)

Goal: Understand Streamlit's execution model

Create `hello_app.py`:

```python
import streamlit as st
import time

st.title("My First Streamlit App")

# This prints every time the script runs!
print(f"Script ran at {time.time()}")

name = st.text_input("Enter your name:")

if name:
    st.write(f"Hello, {name}!")

# Add a button
if st.button("Click me"):
```

Exercise 1.1: Observations

What to notice:

1. Script runs top-to-bottom on every interaction
2. `print()` appears in terminal, not browser
3. Variables reset on each run
4. Check terminal - script runs multiple times!

Question: Click the button multiple times. What happens?

Answer: Balloons show each time because the entire script re-runs!

Exercise 1.2: State Management (20 min)

Goal: Persist data across re-runs

Create `counter_app.py` :

```python
import streamlit as st

st.title("Counter App")

# Initialize counter in session state
if 'count' not in st.session_state:
    st.session_state.count = 0

# Buttons to increment/decrement
col1, col2, col3 = st.columns(3)

with col1:
    if st.button("
➖
 Decrement"):
        st.session_state.count -= 1

with col2:
    st.metric("Count", st.session_state.count)

with col3:
    if st.button("
➕
 Increment"):
        st.session_state.count += 1
```

Exercise 1.2: Expected Behavior

Try this:

1. Click increment several times
2. Click decrement
3. Type in the text box above (from previous exercise if combined)
4. Counter should persist!

Key insight: `st.session_state` stores data between re-runs.

Common pattern: Initialize with `if 'key' not in st.session_state:`

Exercise 1.3: Caching (15 min)

Goal: Avoid expensive recomputations

Create `cache_demo.py`:

```python
import streamlit as st
import time
import pandas as pd

st.title("Caching Demo")

# Expensive function WITHOUT caching
def load_data_slow():
    time.sleep(3)  # Simulate slow operation
    return pd.DataFrame({
        'A': range(100),
        'B': range(100, 200)
    })

# Expensive function WITH caching
@st.cache_data
def load_data_fast():
    time.sleep(3)  # Still slow first time
```

Exercise 1.3: Caching (continued)

```python
# Let user choose
use_cache = st.checkbox("Use caching?", value=True)

if st.button("Load Data"):
    with st.spinner("Loading..."):
        if use_cache:
            data = load_data_fast()
        else:
            data = load_data_slow()

    st.success("Data loaded!")
    st.dataframe(data.head())

st.info("Try clicking 'Load Data' multiple times with caching ON vs OFF")
```

Observe: First load takes 3s. Subsequent loads instant with cache!

Part 1 Checkpoint

What you've learned:

- Streamlit's re-run model
- Using `st.session_state` for persistence
- Caching expensive operations with `@st.cache_data`
- Basic widgets: buttons, text inputs, metrics, columns

Next: Build real ML applications!

Part 2: ML Model Integration

Building practical AI demos

Exercise 2.1: Sentiment Analysis (45 min)

Goal: Complete sentiment analysis dashboard

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="
😊
")

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

# Load model with caching
@st.cache_resource
def load_model():
```

Exercise 2.1: Sentiment Analysis (continued)

```python
# Initialize history in session state
if 'history' not in st.session_state:
    st.session_state.history = []

# Input area
text = st.text_area(
    "Enter text to analyze:",
    placeholder="Type or paste text here...",
    height=150
)

# Analyze button
if st.button("Analyze Sentiment", type="primary"):
    if text.strip():
        with st.spinner("Analyzing..."):
            result = model(text)[0]

        # Display result
        label = result['label']
        score = result['score']

        col1, col2 = st.columns(2)
        with col1:
            st.metric("Sentiment", label)
        with col2:
            st.metric("Confidence", f"{score:.1%}")
```

Exercise 2.1: Sentiment Analysis (continued)

```
        # Color-coded result
        if label == "POSITIVE":
            st.success(f"
✅
 Positive sentiment ({score:.1%} confident)")
        else:
            st.error(f"
❌
 Negative sentiment ({score:.1%} confident)")

        # Add to history
        st.session_state.history.append({
            "text": text[:50] + "..." if len(text) > 50 else text,
            "sentiment": label,
            "confidence": f"{score:.1%}"
        })
    else:
        st.warning("Please enter some text first!")

# Show history
if st.session_state.history:
    st.subheader("Analysis History")
    df = pd.DataFrame(st.session_state.history)
    st.dataframe(df, use_container_width=True)

    if st.button("Clear History"):
        st.session_state.history = []
        st.rerun()
```

Exercise 2.1: Test Cases

Try analyzing these:

1. "I absolutely love this product! Best purchase ever!"
2. "This is terrible. Worst experience of my life."
3. "The weather is okay, I guess."
4. "Not bad, could be better though."

Expected behavior:

- First two should be clearly positive/negative
- Last two might show lower confidence
- All should appear in history table

Exercise 2.2: Image Classification (60 min)

Goal: Build an image classifier with visual feedback

Create `image_classifier.py` :

```
import streamlit as st
from PIL import Image
from transformers import pipeline
import io

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

st.title("
🖼️
 Image Classification")
st.write("Upload an image to classify it using AI")

# Load model
@st.cache_resource
def load_classifier():
```

Exercise 2.2: Image Classification (continued)

```python
# File uploader
uploaded_file = st.file_uploader(
    "Choose an image...",
    type=['png', 'jpg', 'jpeg'],
    help="Upload a PNG or JPG file"
)

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

    col1, col2 = st.columns([1, 1])

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

    with col2:
        st.subheader("Classification Results")

        # Classify button
        if st.button("Classify Image", type="primary"):
            with st.spinner("Analyzing image..."):
                # Run classification
                predictions = classifier(image)

            # Display top 3 predictions
            st.write("**Top 3 Predictions:**")
            for i, pred in enumerate(predictions[:3], 1):
                label = pred['label']
                score = pred['score']

                # Progress bar for confidence
                st.write(f"{i}. **{label}**")
                st.progress(score)
                st.caption(f"Confidence: {score:.1%}")
```

Exercise 2.2: Image Classification (continued)

```
            # Show all predictions in expandable section
            with st.expander("See all predictions"):
                import pandas as pd
                df = pd.DataFrame(predictions)
                df['score'] = df['score'].apply(lambda x: f"{x:.1%}")
                st.dataframe(df, use_container_width=True)

else:
    # Helpful instructions when no image uploaded
    st.info("
👆
 Upload an image to get started")

    # Example images option
    if st.button("Try example image"):
        st.image("https://images.unsplash.com/photo-1574158622682-e40e69881006",
                caption="Example: Cat image",
                use_column_width=True)
        st.info("Download this image and upload it to test!")
```

Exercise 2.2: Enhancements

Add these features:

1. **Image info display**:

```python
# After loading image
st.sidebar.subheader("Image Info")
st.sidebar.write(f"Format: {image.format}")
st.sidebar.write(f"Size: {image.size}")
st.sidebar.write(f"Mode: {image.mode}")
```

2. **Preprocessing options**:

```python
if st.sidebar.checkbox("Resize image"):
    size = st.sidebar.slider("Size", 100, 800, 400)
    image = image.resize((size, size))
```

Exercise 2.3: Text Generation App (45 min)

Goal: Interactive text generation with streaming

Create `text_gen_app.py`:

```
import streamlit as st
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM
import torch

st.title("
✍️
 AI Text Generator")

# Load model
@st.cache_resource
def load_generator():
    model_name = "gpt2"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name)
    return tokenizer, model

tokenizer, model = load_generator()

# Input
```

Exercise 2.3: Text Generation (continued)

```python
# Settings in sidebar
st.sidebar.subheader("Generation Settings")
max_length = st.sidebar.slider("Max length", 50, 500, 100)
temperature = st.sidebar.slider("Temperature", 0.1, 2.0, 1.0, 0.1)
top_k = st.sidebar.slider("Top-k", 0, 100, 50)

# Generate button
if st.button("Generate Text", type="primary"):
    if prompt.strip():
        with st.spinner("Generating..."):
            # Encode input
            inputs = tokenizer.encode(prompt, return_tensors="pt")

            # Generate
            outputs = model.generate(
                inputs,
                max_length=max_length,
                temperature=temperature,
                top_k=top_k,
                do_sample=True,
                pad_token_id=tokenizer.eos_token_id
            )

            # Decode
            generated_text = tokenizer.decode(outputs[0],
                                             skip_special_tokens=True)

        # Display result
        st.subheader("Generated Text")
        st.write(generated_text)

        # Copy button
        st.code(generated_text)
    else:
        st.warning("Please enter a prompt!")
```

Part 2 Checkpoint

What you've built:

- Sentiment analysis with history tracking
- Image classification with visual results
- Text generation with configurable parameters

Key patterns learned:

- Model loading with `@st.cache_resource`
- File uploads and image handling
- Progress indicators and spinners
- Sidebar for settings
- Column layouts

Part 3: Advanced Features

Building production-ready demos

Exercise 3.1: Chat Interface (45 min)

Goal: Build a conversational AI interface

Create `chat_app.py`:

```
import streamlit as st
import os
from dotenv import load_dotenv

load_dotenv()

st.title("
💬
 AI Chat Assistant")

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

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

# Chat input
```

Exercise 3.1: Chat Interface (continued)

```
    # Generate response (simplified - use real LLM in practice)
    with st.chat_message("assistant"):
        with st.spinner("Thinking..."):
            # Placeholder for actual LLM call
            # In real app: response = call_llm_api(prompt)
            response = f"You said: '{prompt}'. This is a demo response!"

        st.markdown(response)

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

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

Exercise 3.2: Error Handling (20 min)

Goal: Gracefully handle errors

Add to any previous app:

```
try:
    # Model prediction
    result = model(input_data)

except Exception as e:
    st.error(f"
❌
 An error occurred: {str(e)}")

    with st.expander("Error Details"):
        import traceback
        st.code(traceback.format_exc())

    st.info("
💡
 Try:")
    st.write("- Checking your input format")
    st.write("- Using a smaller input")
```

Exercise 3.3: User Feedback Collection (20 min)

Goal: Collect feedback on model outputs

Add after model prediction:

```
# Display prediction
st.write(prediction)

# Feedback section
st.divider()
col1, col2, col3 = st.columns([3, 1, 1])

with col1:
    st.write("Was this prediction helpful?")

with col2:
    if st.button("
👍
 Yes"):
        # Log positive feedback
        st.session_state.setdefault('feedback', []).append({
            "input": input_text,
            "output": prediction,
            "rating": "positive"
        })
        st.success("Thanks for the feedback!")

with col3:
    if st.button("
👎
 No"):
```

Part 3 Checkpoint

Advanced features covered:

- Chat interfaces with message history
- Comprehensive error handling
- User feedback collection
- State persistence patterns

Part 4: Deployment

Sharing your apps with the world

Exercise 4.1: Prepare for Deployment (15 min)

Create `requirements.txt`:

```
# In your project directory
pip freeze > requirements.txt
```

Clean it up (only include what you need):

```
streamlit==1.28.0
transformers==4.35.0
torch==2.1.0
pillow==10.1.0
pandas==2.1.0
```

Create `.gitignore`:

```
.env
__pycache__/
```

Exercise 4.2: Deploy to Streamlit Cloud (20 min)

Steps:

1. **Push to GitHub**:

```
git init
git add .
git commit -m "Initial commit"
git remote add origin YOUR_REPO_URL
git push -u origin main
```

2. **Deploy**:
   - Go to https://share.streamlit.io
   - Click "New app"
   - Select your repository
   - Choose branch and main file

Exercise 4.3: Deploy to Hugging Face Spaces (20 min)

Steps:

1. **Create Space**:
   - Go to https://huggingface.co/new-space
   - Choose name and SDK (Streamlit)
   - Set visibility (Public/Private)
2. **Add files**:
   - Upload `app.py`
   - Upload `requirements.txt`
   - Add `README.md` with metadata:

Part 4 Checkpoint

Deployment skills gained:

- Creating proper `requirements.txt`
- Using `.gitignore` for secrets
- Deploying to Streamlit Cloud
- Deploying to Hugging Face Spaces

You now have live, shareable demos!

Bonus Challenges

If you finish early, try these:

1. **Multi-page app**: Create separate pages for different models
2. **Database integration**: Save predictions to SQLite
3. **Custom CSS**: Add custom styling with `st.markdown`
4. **Analytics**: Add Google Analytics tracking
5. **Authentication**: Add password protection

Lab Wrap-up

What you've accomplished:
✅
Built 3-4 interactive ML demos
✅
Learned state management and caching
✅
Implemented error handling and UX best practices
✅
Deployed apps to the cloud

Deliverables:

1. GitHub repository with all apps
2. At least 2 deployed public URLs
3. Screenshot of each app working

Resources

Documentation:

- Streamlit: https://docs.streamlit.io
- HF Spaces: https://huggingface.co/docs/hub/spaces

Inspiration:

- https://streamlit.io/gallery
- https://huggingface.co/spaces

Troubleshooting:

- Streamlit community: https://discuss.streamlit.io
- Teaching assistants during lab hours