

# Session 8: Chain Two Models Together

---

*AI + Research Level 2 — Supplementary Material*

**Concept:** MULTI-MODEL SYSTEMS AND ERROR CASCADES

**Space:** Image Story Pipeline

**Models:**

- `Salesforce/blip-image-captioning-base` (~1GB, image → caption)
- `distilbert-base-uncased-finetuned-sst-2-english` (caption → sentiment)

**Pre-built fallback:** Deploy at `profplate/image-story-pipeline` on HF before session

**Pre-warm:** Wake the Space at least 30 minutes before session. BLIP takes ~30-60 seconds to load on cold start from free CPU. Visit the Space URL and wait for it to load fully.

---

## Time Breakdown (2 hours)

---

### 0:00-0:05 — Show-and-Tell

Anyone try the between-session challenge? What bias pairs did you find? Quick share.

### 0:05-0:15 — Show the Finished Pipeline

Open the pre-built Space. Upload a clear, easy photo (a dog in a park, a sunset, a plate of food).

**Demo flow:**

1. Upload image. Wait for results.
2. Read aloud: "Caption: 'a dog sitting on the grass in a park.' Sentiment: POSITIVE."
3. "Two models just worked together. Model 1 looked at the picture and wrote a sentence. Model 2 read that sentence and told us the mood. Neither model knows the other exists."

Upload a second image — something more ambiguous (abstract art, a blurry photo).

1. The caption will likely be wrong or vague. Read it aloud.

2. "The caption is off. And look at the sentiment — it's analyzing a wrong description. That's the problem with chaining models: if the first one is wrong, everything after it is wrong too."

**Landing line:** "Today we're going to build this pipeline and then try to break it."

## 0:15-0:55 — Build It Live

This is the most complex build so far. Two models, image input, multi-step function.

### Build sequence:

1. Create new Space on HF: `image-story-pipeline`, Gradio SDK, Public, Free CPU.
2. Start `app.py`:

```
import gradio as gr
from transformers import pipeline, BlipProcessor, BlipForConditionalGeneration
from PIL import Image
```

**Say:** "More imports than usual. We need `BlipProcessor` and `BlipForConditionalGeneration` because the captioning model doesn't fit into the simple `pipeline()` shortcut. And `PIL` — Python Imaging Library — handles image files."

1. Load the models:

```
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
caption_model = BlipForConditionalGeneration.from_pretrained(
    "Salesforce/blip-image-captioning-base")
sentiment = pipeline("sentiment-analysis",
    model="distilbert-base-uncased-finetuned-sst-2-english")
```

**Say:** "Two models loading. The captioning model is about 1GB — that's why this Space takes a while to start. The sentiment model is the same one we've used before."

**Warn:** "This is going to take a while to build on free CPU. We'll keep coding while it loads."

1. Write the function:

```
def analyze_image(image):
    if image is None:
        return "Upload an image first!", "", ""
```

**Say:** "We check for no image first — someone might click Analyze without uploading anything."

```
# Step 1: Generate caption
inputs = processor(image, return_tensors="pt")
out = caption_model.generate(**inputs, max_length=50)
caption = processor.decode(out[0], skip_special_tokens=True)
```

**Say:** "Step 1: the image goes into BLIP. The processor converts the image into numbers the model understands. The model generates a caption. The processor converts the numbers back into words. Three lines, but a lot happening."

```
# Step 2: Analyze caption sentiment
result = sentiment(caption)[0]
sentiment_text = f"{result['label']} ({result['score']:.1%} confidence)"
```

**Say:** "Step 2: the caption — a string of text — goes into the sentiment model. Same pipeline call we've been doing since Session 4. The output of Model 1 becomes the input of Model 2."

```
# Step 3: Show the pipeline
pipeline_view = (
    f"IMAGE\n"
    f"  -> Model 1 (BLIP captioner): \"{caption}\"\\n"
    f"  -> Model 2 (sentiment): {result['label']} ({result['score']:.1%})"
)
return caption, sentiment_text, pipeline_view
```

**Say:** "Step 3 is just for us — a view of the whole chain so we can see what happened at each step."

1. Build the Blocks layout:

```

with gr.Blocks(title="Image Story Pipeline") as demo:
    gr.Markdown("# Image Story Pipeline\n...")
    with gr.Row():
        with gr.Column():
            image_input = gr.Image(type="pil", label="Upload an Image")
            btn = gr.Button("Analyze", variant="primary")
        with gr.Column():
            caption_output = gr.Textbox(label="Step 1: Caption (BLIP)")
            sentiment_output = gr.Textbox(label="Step 2: Sentiment of Caption")
            pipeline_output = gr.Textbox(label="Full Pipeline View", lines=4)

```

**Say:** "Blocks layout again — we used this last session. Image upload on the left, results stacked on the right. And `gr.Image(type='pil')` tells Gradio to give us a PIL image object, which is what BLIP expects."

```

btn.click(fn=analyze_image, inputs=image_input,
          outputs=[caption_output, sentiment_output, pipeline_output])

```

1. Create `requirements.txt`: transformers, torch, gradio, Pillow.

**Say:** "New this time: Pillow. That's the image library. Without it, the Space can't read image files."

1. Commit. This build will take longer than usual — BLIP is a big download.

**While the Space builds (~3-5 min):** Draw the pipeline on screen or describe it verbally:

```

IMAGE → [BLIP: image→text] → CAPTION → [DistilBERT: text→sentiment] → LABEL

```

Ask: "What could go wrong at each arrow?"

## 0:55-1:15 — Test with Images

Once the Space is live, test with different image types. Have students suggest images or use these:

**Suggested test images:**

Image type	Why it's interesting	Where to find
Happy scene (dog, birthday party)	Clear positive caption → positive sentiment. Establishes baseline.	Any stock photo site
Somber scene (rain, empty room)	Negative or neutral caption → how does sentiment respond?	Search "moody photography"
Ambiguous scene (person with neutral expression)	Caption might guess wrong emotion → sentiment follows the guess	Search "candid portrait"
Abstract art	BLIP wasn't trained on abstract art — caption will be off	Search "abstract painting"
Image with text (meme, sign)	BLIP may ignore text in image — caption misses the point	Any meme

### For each image, ask:

- "What did the captioner say? Is that accurate?"
- "Given that caption, does the sentiment make sense?"
- "Would the sentiment be different if the caption were more accurate?"

## 1:15-1:35 — Error Cascade Demo

Now deliberately break the pipeline. This is the core learning moment.

**Find an image where the captioner is wrong.** (Abstract art usually works.) Walk through it:

1. "What does this image actually show?" (Ask students)
2. "What did BLIP say?" (Read the caption)
3. "Is that caption right?" (No)
4. "Now look at the sentiment. It says POSITIVE. But is the image positive?" (Doesn't matter — the sentiment model never saw the image. It only read the caption.)

**Key teaching moment:** "The sentiment model did its job perfectly. It analyzed the text it was given. The problem is that text was wrong. The error started in Step 1 and cascaded to Step 2."

**Draw it:**

Wrong image description → Wrong sentiment

↑

Error starts here

↑

Error cascades here

**Ask:** "Where would you fix this? Step 1 or Step 2?"

**Extend the thinking:**

- "What if there were 5 models in the chain? Where would you look for the bug?"
- "Self-driving cars chain models: detect objects → predict paths → decide to brake. Where's the most dangerous place for an error?"
- "ChatGPT chains retrieval → reasoning → response. If it retrieves wrong info, does the reasoning fix it?"

## 1:35-1:50 — Name the Concept: MULTI-MODEL SYSTEMS AND ERROR CASCADES

**Key points to name:**

- **Multi-model system / pipeline** — connecting the output of one model to the input of another
- **Error cascade** — when an error in an early step causes errors in every later step
- **Garbage in, garbage out** — at every step, not just the first one
- **Debugging pipelines** — you have to check each step independently to find where things went wrong

**Say:** "Almost every AI product you use is actually multiple models working together. Siri chains speech-to-text → language understanding → response generation. Each step depends on the one before it. When it misunderstands you, the error cascades through every step after."

Show the BLIP model card:

- <https://huggingface.co/Salesforce/blip-image-captioning-base>
- Point out: trained on COCO dataset (Common Objects in Context) — everyday photos
- "What kinds of images are NOT in 'common objects in context'?"
- Abstract art, medical images, satellite photos, memes with text — all outside BLIP's training domain

## 1:50-1:55 — Notebook Time

Share the Colab link in the Zoom chat.

**Walk through together:**

1. Run the setup cell (loads BLIP and sentiment model — this takes a minute, be patient)
2. Run the image upload cell together — show them the file picker
3. "Click Choose Files, pick a photo, and watch the pipeline run"

**Notebook skill being introduced:** Working with images in Colab — uploading files, displaying them, and feeding them to models.

**Say:** "The notebook lets you upload your own images and test the pipeline. Try to find an image that breaks it — is the caption wrong, or the sentiment, or both? Fill in the table in the notebook."

## 1:55-2:00 — Between-Session Suggestion

Share the between-session challenge (see BETWEEN-SESSION.md).

**Say:** "Find an image that breaks the pipeline. Figure out which step failed — the caption or the sentiment. Next week, we're going to think about the people who actually use these tools. We'll redesign a Space for a real audience."

---

## Memory and Performance Notes

---

**Model sizes (approximate):**

- BLIP captioning: ~1GB (weights) + ~500MB (processor)
- DistilBERT sentiment: ~250MB
- Total: ~1.75GB — fits within free CPU tier (16GB RAM) with room to spare

**Cold start time:** First load after Space sleeps takes 30-60 seconds. Pre-warm by visiting the Space URL before session.

**If memory is tight (Space crashes):**

- Remove print statements (minor savings)
- If still crashing, consider using the HF Inference API instead of loading BLIP locally:

```
python from huggingface_hub import InferenceClient client = InferenceClient() caption = client.image_to_text("Salesforce/blip-image-captioning-base", image)
```

This uses HF's servers instead of loading the model into the Space's memory.

**Image processing time:** Each image takes 5-15 seconds on free CPU. Warn students: "It's thinking. Free CPU is slower than the GPUs these models usually run on."

## What Could Go Wrong

Problem	Fix
Space takes 5+ minutes to build	BLIP is large. Fill time with pipeline diagrams and discussion. "While we wait, let's think about what could go wrong."
Space crashes on load (OOM)	Check that only two models are loaded. If still crashing, use Inference API fallback (see Memory Notes).
Image upload doesn't work	Check <code>Pillow</code> is in requirements.txt. Try a different image format (JPG usually safest).
Caption is wrong	That's a feature! "The captioner made an error. Now watch what happens to the sentiment."
Caption is always the same generic text	Model might be receiving a thumbnail/placeholder. Make sure <code>type="pil"</code> is set on <code>gr.Image</code> .
Students can't find good test images	Have 4-5 images ready on your desktop. Stock photo sites, memes, abstract art.
Sentiment is always POSITIVE	The captioner tends to generate neutral-to-positive descriptions. Try images with dark, dramatic, or sad content.

## Key Vocabulary (introduce casually, don't drill)

- **Pipeline / multi-model system** — connecting models so the output of one feeds the input of the next
- **Error cascade** — when one model's mistake causes every model after it to also be wrong
- **Caption** — a text description of an image, generated by a vision model
- **BLIP** — Bootstrapping Language-Image Pre-training, the model that generates captions
- **PIL / Pillow** — Python library for working with image files
- **Cold start** — the delay when a model loads for the first time