Session 3 — Break It on Purpose

Space Builder: Silly Phrase Finder (with Cleaning)

What You'll Build

An upgraded version of the Silly Phrase Finder that can handle messy input — emoji, ALL CAPS, extra spaces, abbreviations. The clean_text() function fixes the mess before the AI reads it.

Step 1 — Create a New Space

1. Go to huggingface.co/new-space (log in if needed)
2. In the "Space name" field, type: silly-phrase-finder-v2
3. Under "Select the Space SDK", choose Gradio
4. Under "Select the Space hardware", choose Free — CPU basic
5. Leave everything else as default
6. Click Create Space

Step 2 — Create the requirements.txt File

1. On your Space page, click the Files tab (near the top)
2. Click Add file → Create a new file
3. In the filename field at the top, type: requirements.txt
4. In the big text area below, copy and paste this exactly:

> *Your Space page will open. It's empty right now — we're about to add the code.*

5. Click Commit new file to main (the blue button at the bottom)

Step 3 — Create the app.py File

1. Click Add file → Create a new file (again)
2. In the filename field, type: app.py
3. In the big text area, copy and paste ALL of the code below:

```
transformers
torch
gradio
```

4. Click Commit new file to main

Step 4 — Wait for It to Build

1. Click the App tab (at the top of your Space page)
2. You'll see "Building" — this takes 2–5 minutes
3. When it's done, your Space will appear!

Step 5 — Try It Out!

1. Paste text with lots of emoji and ALL CAPS — does cleaning help?
2. Try the same messy text in your Session 1 Space (no cleaning) and this one — compare results
3. Can you find input that still breaks it even with cleaning?

Troubleshooting

```
import gradio as gr
from transformers import pipeline
import re

# Load the same zero-shot model from Session 1
classifier = pipeline("zero-shot-classification", model="valhalla/distilbart-mnli-12-3")

def clean_text(text):
    """
    Clean up messy input before sending it to the model.
    Each step fixes a specific kind of noise.
    """
    # 1. Strip leading/trailing whitespace
    text = text.strip()

    # 2. Collapse multiple spaces into one
    text = re.sub(r' {2,}', ' ', text)

    # 3. Limit repeated characters (e.g., "sooooo" -> "soo")
    text = re.sub(r'(.)\1{2,}', r'\1\1', text)

    # 4. Expand common abbreviations
    abbreviations = {
        "Rep.": "Representative",
        "Dr.": "Doctor",
        "Mr.": "Mister",
        "Mrs.": "Missus",
        "Ms.": "Ms",
        "Jr.": "Junior",
        "Sr.": "Senior",
        "Prof.": "Professor",
        "Gov.": "Governor",
        "Sen.": "Senator",
        "Gen.": "General",
        "St.": "Saint",
    }
    for abbr, full in abbreviations.items():
        text = text.replace(abbr, full)

    # 5. Remove emoji (covers most common emoji ranges)
    text = re.sub(
        r'[\U0001F600-\U0001F64F'
        r'\U0001F300-\U0001F5FF'
        r'\U0001F680-\U0001F6FF'
        r'\U0001F1E0-\U0001F1FF'
        r'\U00002702-\U000027B0'
        r'\U0000FE00-\U0000FE0F'
        r'\U0001F900-\U0001F9FF'
        r'\U0001FA00-\U0001FA6F'
        r'\U0001FA70-\U0001FAFF'
```

```
        r'\U00002600-\U000026FF]+',
        ' ', text
    )

    # 6. Normalize ALL CAPS to Title Case (if more than 3 words are all caps)
    words = text.split()
    caps_count = sum(1 for w in words if w.isupper() and len(w) > 1)
    if caps_count > 3:
        text = text.title()

    # Clean up any extra spaces introduced by cleaning
    text = re.sub(r' {2,}', ' ', text).strip()

    return text

def find_silliest(text):
    if not text or not text.strip():
        return "Paste some text above first!"

    # Clean the text before processing
    cleaned = clean_text(text)

    # Split on sentence-ending punctuation followed by whitespace
    sentences = [s.strip() for s in re.split(r'(?<=[.!?])\s+(?=[A-Z"])', cleaned) if len(s.stri
  p()) > 30]

    if len(sentences) == 0:
        return "I need at least a couple of sentences to compare. Paste a longer passage!"

    # Score every sentence for "silly" vs "serious" vs "ordinary"
    labels = ["silly and ridiculous", "serious and important", "ordinary and boring"]
    results = classifier(sentences, candidate_labels=labels)

    # Handle single sentence (returns dict instead of list)
    if isinstance(results, dict):
        results = [results]

    # Find the sentence with the highest "silly" score
    best_phrase = ""
    best_score = 0
    for sentence, result in zip(sentences, results):
        silly_idx = result["labels"].index("silly and ridiculous")
        score = result["scores"][silly_idx]
        if score > best_score:
            best_score = score
            best_phrase = sentence

    # Show both original and cleaned versions if they differ
    header = f'"{best_phrase}"\n\nSilliness score: {best_score:.0%}'
    if cleaned != text.strip():
```

```
        header += "\n\n---\n(Text was cleaned before analysis)"

    return header

demo = gr.Interface(
    fn=find_silliest,
    inputs=gr.Textbox(lines=10, placeholder="Paste a paragraph or two here — try messy text!"),
    outputs=gr.Textbox(label="The Silliest Phrase"),
    title="Silly Phrase Finder (with Cleaning)",
    description="Same as Session 1, but now with input cleaning! Try pasting messy text — emoji
  , ALL CAPS, abbreviations, extra spaces — and see how cleaning helps the model.",
    examples=[
        ["The quarterly budget report is due on Friday. My cat learned to open the refrigerator
   and now judges my food choices. Please remember to submit your timesheets. The printer on th
  e third floor is out of toner again."],
        ["OMG THIS IS SOOOOO AMAZING I CANT EVEN BELIEVE IT!!! Dr. Smith said the results were
  incredible.   Rep. Johnson     gave a speech about    pancakes."],
        ["lollll the penguin just vibed across the entire library floor. Meanwhile Mr. Rogers
  was teaching kids about kindness. THE WHOLE SCHOOL WAS WATCHING AND NOBODY COULD STOP LAUGHIN
  G."],
    ],
)

demo.launch()
```

```
Make sure you copy the ENTIRE code block — from the very first line to the very last. Missing e
  ven one line can cause errors.
```

```
If you see a red error: Click the Logs tab to read the error message. The most common fix is to
   double-check that requirements.txt has the right contents and that app.py was copied complet
  ely.
```

Problem	Fix
"Runtime error"	Check the Logs tab. Usually means a typo in app.py. Re-copy the code carefully.
Space stuck on "Building"	Wait up to 5 minutes. Free CPU Spaces can be slow. If it’s been more than 10 minutes, try deleting the Space and starting over.
"ModuleNotFoundError"	Your requirements.txt is missing a library. Make sure it matches exactly what’s shown above.
Space loads but nothing happens	Make sure the last line of app.py is demo.launch() with no extra spaces before it.