📅 WEEK 16 — Mini Project + LLM Mindset (End-to-End NLP App)

🎯 Goal of Week 16 (In Simple Words)

Take the fine-tuned model from Week 15 and turn it into a **real working NLP application** with a simple UI (Gradio), plus evaluation + README.

✅ At the end, you will have a full "portfolio" project:

- Model
- Inference pipeline
- UI
- Testing
- Error analysis
- Report / README

✅ STEP 0 — Choose Your Project (Pick One)

🅰 Option A: Sentiment Analysis Chatbot ✅ (Recommended)

Input:

User types message/review

Output:

- Sentiment label (Positive/Negative/Uncertain)
- Confidence score
- Explanation (simple rule-based explanation)

✅ Best because:

- IMDB dataset is easy
- Labels are clean
- Works well with fine-tuned DistilBERT
- Great for beginners & submissions

🅱 Option B: Fake News Detector (Advanced)

Input:

News headline or article

Output:

- Fake / Real probability

Harder because:

- Fake news datasets can be noisy
- “Real vs fake” can be subjective
- Need more cleaning + bias discussion

📌 I will write the steps in a way that works for both, but I’ll assume **Option A** for example code (you can swap dataset/model later).

✅ STEP 1 — Project Folder Setup (Very Professional)

Create a clean project folder like this:

```
week16_nlp_project/
│
├── app/
│   ├── app.py              # Gradio UI
│   └── inference.py        # prediction function
│
├── model/
│   └── fine_tuned_model/    # saved model + tokenizer (from week 15)
│
├── data/
│   ├── sample_inputs.txt     # real testing examples
│   └── optional_dataset.csv  # if you use custom data
│
├── results/
│   ├── predictions.csv
│   ├── confusion_matrix.png
│   └── error_analysis.md
│
├── notebooks/
│   ├── evaluation.ipynb
│   └── testing.ipynb
│
├── README.md
└── requirements.txt
```

✅ Why this structure?

- Looks like real ML project
- Easy to explain in interview
- Easy to maintain

✅ STEP 2 — Load Fine-Tuned Model (From Week 15)

Definition (Beginner)

Inference means:

> Using a trained model to make predictions (no training now).

Code: Load model + tokenizer

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

MODEL_DIR = "./model/fine_tuned_model"

tokenizer = AutoTokenizer.from_pretrained(MODEL_DIR)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_DIR)
model.eval()
```

Explanation (Line by line)

- from_pretrained(MODEL_DIR) loads your saved files
- model.eval() makes predictions stable (no dropout)

✅ STEP 3 — Build Inference Function (Core of the App)

What should inference function do?

Input text → output:

- label
- confidence
- probability scores (optional)
- explanation

Code (Very Clean)

```
import torch

id2label = model.config.id2label

def predict(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)

    with torch.no_grad():
        outputs = model(**inputs)

    logits = outputs.logits
    probs = torch.softmax(logits, dim=1)[0]

    pred_id = torch.argmax(probs).item()
    label = id2label[pred_id]
    confidence = probs[pred_id].item()

    return label, confidence, probs.tolist()
```

Step-by-step explanation

- Tokenizer makes tensors
- Model outputs logits
- Softmax converts logits → probabilities
- Argmax chooses best label
- Confidence is probability of chosen label

✅ STEP 4 — Add “Uncertain” Feature (Real ML Product Behavior)

Definition

A **threshold** is:

> minimum confidence needed to accept prediction.

If confidence is low → show “Uncertain”.

Code

```
def predict_with_threshold(text, threshold=0.80):
    label, confidence, probs = predict(text)

    if confidence < threshold:
        return "UNCERTAIN", confidence, probs

    return label, confidence, probs
```

✅ This makes your app more realistic.

✅ STEP 5 — Add Explanation (Simple but Very Important)

Transformers are black-box.

But for beginner projects, you can give:

- rule-based explanation
- highlight simple keywords

Simple explanation example (Sentiment)

- If positive words like “amazing, loved, great” → positive
- If negative words like “worst, boring, hate” → negative

Code: Rule-based explanation

```
positive_words = ["love", "amazing", "great", "awesome", "fantastic", "best"]
negative_words = ["hate", "worst", "boring", "bad", "terrible", "waste"]

def simple_explanation(text):
    text_lower = text.lower()
    pos_hits = [w for w in positive_words if w in text_lower]
    neg_hits = [w for w in negative_words if w in text_lower]

    if pos_hits and not neg_hits:
        return f"Detected positive words: {pos_hits}"
    if neg_hits and not pos_hits:
        return f"Detected negative words: {neg_hits}"
    if pos_hits and neg_hits:
        return f"Mixed words found. Positive: {pos_hits}, Negative: {neg_hits}"
    return "No strong keywords detected. Model used full context to decide."
```

✅ In report, mention:

- this explanation is heuristic
- not always perfect

✅ STEP 6 — Handle Long Text (Important for Fake News especially)

Transformers have max length (512 tokens).

Long text gets truncated.

Solution: Chunking

Split text → run prediction on each chunk → average.

Code: Chunking

```
def chunk_text(text, chunk_size=300):
    return [text[i:i+chunk_size] for i in range(0, len(text), chunk_size)]

def predict_long_text(text):
    chunks = chunk_text(text, 300)
    all_probs = []
    for c in chunks:
        _, _, probs = predict(c)
        all_probs.append(probs)

    avg_probs = np.mean(all_probs, axis=0)
    pred_id = int(np.argmax(avg_probs))
    label = id2label[pred_id]
    confidence = float(avg_probs[pred_id])

    return label, confidence, avg_probs.tolist()
```

✅ Needed for:

- news articles
- long reviews
- paragraphs

✅ STEP 7 — Build Gradio UI (Very Simple & Powerful)

Definition

Gradio is a Python tool that creates UI for ML models quickly.

Install

```
pip install gradio
```

Gradio App Code (app.py)

```
import gradio as gr

def app_predict(text):
    label, confidence, probs = predict_with_threshold(text, threshold=0.80)
    explanation = simple_explanation(text)

    return {
        "Label": label,
        "Confidence": round(confidence, 4),
        "Probabilities": probs,
        "Explanation": explanation
    }

demo = gr.Interface(
    fn=app_predict,
    inputs=gr.Textbox(lines=4, placeholder="Type your text here..."),
    outputs="json",
    title="Sentiment Analysis Chatbot (Fine-tuned DistilBERT)",
    description="Enter text → get sentiment + confidence + simple explanation."
)

demo.launch()
```

Explanation

- input = text box
- output = JSON result
- deploys locally in browser

✅ STEP 8 — Testing with Real Examples (Must Do)

Create a file data/sample_inputs.txt and test:

- normal positive
- normal negative
- mixed sentiment
- sarcasm
- long text

Example test set

1. “This movie was fantastic and emotional.”
2. “Worst movie ever, total waste of time.”
3. “Good acting but terrible story.”
4. “Yeah great… I slept through it.” (sarcasm)

✅ Save outputs to results/predictions.csv.

✅ STEP 9 — Error Analysis + Bias Discussion (Very Important)

What is Error Analysis?

Find wrong predictions and explain why.

Common reasons:

- sarcasm
- mixed sentiment
- long text
- domain mismatch

Bias Discussion (Simple)

- Model trained on movie reviews → may fail on product reviews
- Language style differences
- Non-English or Hinglish might be misclassified

✅ Write 8–10 lines about this in results/error_analysis.md.

✅ STEP 10 — Write README (Final Submission)

README should include:

1. Project title
2. What it does
3. Model used
4. Dataset used
5. How to run (install + commands)
6. Example inputs/outputs
7. Limitations (bias + errors)
8. Future improvements

✅ STEP 11 — Final Checklist (Submission Ready ✅)

✅ Fine-tuned model saved

✅ Inference function works

✅ Threshold logic added

✅ Explanation added

✅ Long text handling added

✅ Gradio UI running

✅ Testing examples done

✅ Error analysis written

✅ README written

✅ requirements.txt created

📌 WEEK 16 DAILY PLAN (So You Know What To Do Each Day)

Day 1: Setup project folder + load model

Day 2: Build inference + threshold

Day 3: Add explanation logic

Day 4: Long text chunking

Day 5: Build Gradio UI

Day 6: Testing + CSV results

Day 7: Error analysis + README + final packaging
