**Phase 0: Foundation (Data Acquisition & Setup)**

Your project's "impact" score is set here.

1. **Data Acquisition:**
   * **Primary Target:** Go to data.gov.in or Kaggle and search for "Consumer Complaints," "Public Grievance," or "CPGRAMS" datasets.
   * **Excellent Example:** There is a "Consumer Complaints in India" dataset often available on Kaggle. This is perfect. It will have a Complaint Text column and a Product or Category column (e.g., "Banking Services," "E-Commerce," "Insurance"). This Category column is your target variable.
   * **Goal:** Get a .csv file with thousands of text samples and their corresponding categories.
2. **Environment Setup:**
   * Create your project folder.
   * Create a requirements.txt file. This is critical for your professor. It must include:
   * streamlit
   * scikit-learn
   * pandas
   * nltk
   * gensim
   * transformers
   * torch
   * textattack
   * lime
   * matplotlib
3. **Data Preprocessing (The notebook.ipynb):**
   * Create a Jupyter Notebook to do all your *one-time* analysis and model training.
   * **Load Data:** Read your .csv into a pandas DataFrame.
   * **Clean & Explore:**
     + Check for missing values.
     + Check the class distribution (your categories). Are some departments 10x more common? This is important.
     + Create a "classic" cleaning function: clean\_text(text) that does lowercasing, removes punctuation, and removes stopwords (use nltk). You will use this for Models 1, 2, and 3. (BERT/DistilBERT, Model 4, does *not* need this).
   * **Split Your Data:**
     + from sklearn.model\_selection import train\_test\_split
     + Split your data into train\_data and test\_data (e.g., 80/20 split). You will use this *same split* for all four models to ensure a fair comparison.

**Phase 1: Model Training (The Gauntlet)**

You will do this in your notebook. The goal is to train all 4 models and **save them to disk**. Your Streamlit app will *load* these files, not train them.

1. **Model 1: TF-IDF + Naive Bayes**
   * Create an sklearn Pipeline:
     + Step 1: TfidfVectorizer() (using your clean\_text function as the preprocessor).
     + Step 2: MultinomialNB()
   * Train it: model\_1.fit(train\_data['text'], train\_data['category'])
   * Save it: import joblib; joblib.dump(model\_1, 'model\_1\_nb.pkl')
2. **Model 2: TF-IDF + Logistic Regression**
   * Create an sklearn Pipeline:
     + Step 1: TfidfVectorizer() (same as above).
     + Step 2: LogisticRegression(max\_iter=1000)
   * Train it: model\_2.fit(train\_data['text'], train\_data['category'])
   * Save it: joblib.dump(model\_2, 'model\_2\_logreg.pkl')
3. **Model 3: Word2Vec + Logistic Regression**
   * This is a multi-step process:
   * **Train Word2Vec:** Use gensim to train a Word2Vec model on your train\_data['text'].
   * **Create a Vectorizer Function:** Write a function Word2VecVectorizer(text\_list) that takes a list of complaints, and for each complaint, averages the Word2Vec vectors of its words. This will output a numpy array.
   * **Train LogReg:** Train a *separate* LogisticRegression model on the output of your Word2VecVectorizer.
   * Save both the gensim.model and the LogisticRegression model.
4. **Model 4 (LSTM + W2V):** "The Sequential Thinker" (Understands word meaning *and* word order/sequence. It knows "not happy" is different from "happy not").
5. **Model 5: Fine-Tuned DistilBERT**
   * This is the most complex but most powerful model.
   * Use the transformers library with AutoTokenizer and AutoModelForSequenceClassification.
   * Tokenize your train\_data and test\_data.
   * Use the Trainer class from Hugging Face to fine-tune distilbert-base-uncased on your grievance data.
   * Save the fine-tuned model: trainer.save\_model('model\_4\_distilbert')

**Checkpoint:** You now have a folder with all your trained models. The hard ML part is done.

**Phase 2: The Streamlit App (The UI)**

Create your app.py file.

1. **Load Assets:**
   * At the top of your app, load all 4 models *once* using st.cache\_resource (this is crucial so it doesn't reload them constantly).
   * model\_1 = joblib.load('model\_1\_nb.pkl')
   * model\_4\_tokenizer = AutoTokenizer.from\_pretrained('model\_4\_distilbert')
   * model\_4 = AutoModelForSequenceClassification.from\_pretrained('model\_4\_distilbert')
2. **Build the UI:**
   * st.title("🔬 The Model Interrogator: NLP Grievance-Routing Stress Test")
   * user\_input = st.text\_area("Enter a citizen complaint:", "The power in my area is out, but the garbage hasn't been collected in three days either.")
   * attack\_type = st.selectbox("Select Attack:", ["WordSwap (Synonym)", "CharacterDelete (Typos)"])
   * st.button("Run Interrogation!")
3. **The "Interrogation" Logic (The 'Aww-Factor' Core):**
   * When the button is clicked:
     1. **Get Original Predictions:**
        + Run model.predict\_proba(user\_input) for Models 1, 2, 3.
        + Run Model 4 (BERT) on the user\_input.
        + Get the top predicted category (e.g., "Electricity") and the confidence score (e.g., 0.85) for all four.
     2. **Run the Attack:**
        + Use textattack to generate the attacked text.
        + from textattack.augmentation import WordSwapEmbedding, CharDelete
        + if attack\_type == ...: augmenter = WordSwapEmbedding()
        + attacked\_text = augmenter.augment(user\_input)[0]
     3. **Get Attacked Predictions:**
        + Run all 4 models again on the new attacked\_text.
        + Get the new predicted category and confidence scores.
4. **Display the 2x2 Grid (The 'Aww-Factor' Visual):**
   * col1, col2 = st.columns(2)
   * col3, col4 = st.columns(2)
   * **In col1 (Naive Bayes):**
     1. st.header("Model 1: Naive Bayes")
     2. st.subheader(f"Original: {orig\_pred\_1} ({orig\_conf\_1\*100:.0f}%)")
     3. st.progress(orig\_conf\_1)
     4. st.subheader(f"Attacked: {atk\_pred\_1} ({atk\_conf\_1\*100:.0f}%)")
     5. st.progress(atk\_conf\_1) (Use st.markdown to color this red if the pred changed!)
   * **Repeat for col2, col3, col4** for the other models.
   * **The "Humor" / "Wow" Moment:** The professor will see "Electricity" (85%) flip to "Waste Management" (70%) on Model 1, while Model 4 (BERT) stays at "Electricity" (95%) and its confidence barely drops.

**Phase 3: Adding LIME & The README**

1. **LIME Integration (The Final "Aww-Factor"):**
   * from lime.lime\_text import LimeTextExplainer
   * You need to write a special "predictor" function for each of your sklearn models that LIME can understand.
   * Inside your button logic, you will also generate the LIME explanation for the *original* input.
   * In each column (e.g., col1), add:
     + st.write("🔎 Why it made this choice (LIME):")
     + explainer = LimeTextExplainer(...)
     + explanation = explainer.explain\_instance(user\_input, model\_1.predict\_proba, ...)
     + st.components.v1.html(explanation.as\_html())
   * This is the masterstroke. The professor will now see *why* Naive Bayes was fooled (e.g., it only looked at "garbage") while BERT looked at the whole phrase.
2. **The README.md (Your Final Report):**
   * **Title:** The Model Interrogator: A Comparative Study of NLP Robustness for Public Grievance Classification
   * **GIF:** Record that GIF of your app working (as you planned) and put it right at the top.
   * **Dataset:** Link to data.gov.in or Kaggle. Explain why this "Grievance Routing" problem is a high-impact, real-world task.
   * **The Model Gauntlet:** Explain your 4 models.
   * **📈 Robustness Analysis:**
     + Create a separate script (evaluate.py) that runs textattack on 200 samples from your test\_data.
     + Count how many times the prediction *category* flipped for each model.
     + Plot this **one bar chart** in your README: **"Attack Success Rate (How often the model was fooled)"**
   * **Qualitative Analysis:** Add 2-3 screenshots of your app (the 2x2 grid with LIME) showing a clear example of a "brittle" model failing and a "robust" model succeeding.
   * **How to Run:** pip install..., streamlit run...