Name- Adhitya Kondeti

Student id- 8997046

Question asked by the professor:

Where the linear regression is ?

Our **Linear Regression** is **trained in src/train.py / the notebook**, saved as **co2_predictor.joblib**, and **loaded by the app** to make instant energy and CO_2 predictions from **Layers, FLOPs, Hours, and Complexity**—that's what powers the dashboard and the optimization tips."

What is Linear Regression? 🧠

Linear regression is a straightforward machine learning algorithm that predicts a numeric outcome. It works by finding the best straight-line relationship between a set of inputs (called **features**) and the output you want to predict (the **target**).

In our project:

- **Features**: Layers, FLOPs_in_TFLOPs, Training_Hours, and the NLP-derived Complexity score.
- **Targets**: Energy_kWh and CO_2_kg.

The model learns a simple equation. For a single target, it looks like this:

$\hat{y} = w1 \cdot Layers + w2 \cdot FLOPs + w3 \cdot Hours + w4 \cdot Complexity + b$
Each weight, or coefficient (w_i), tells you exactly how much the prediction changes when that specific feature increases by one unit, assuming all other features stay the same. This makes linear regression highly **interpretable**—which is perfect for explaining *why* a model predicts a certain energy or CO_2 cost.

Why Linear Regression is Useful in This Project

1. **Transparency**: It's easy to explain. We can clearly show stakeholders how moving a slider in the UI (like Layers or FLOPs) directly impacts the predicted energy and CO_2 emissions.
2. **Speed**: Making a prediction is incredibly fast—it's just one quick mathematical calculation (a dot product). This is essential for the real-time, responsive feel of our Streamlit dashboard.
3. **Strong Baseline**: It provides a solid, understandable baseline model. We can use its performance metrics to justify switching to more complex models (like a Random Forest) if they offer significantly better accuracy.

The Model We Used ⚙️

We implemented a two-tier modeling approach:

- **Baseline Model**: We started with LinearRegression from scikit-learn. Its main advantages are speed and explainability, making it ideal for demonstrating the core concept.
- **Upgraded Model**: The current artifact saved in our repository (rf_model.pkl) is a RandomForestRegressor. We chose this because the relationships between our features and the energy/CO_2 targets can be non-linear. A Random Forest can capture these complex patterns and interactions (e.g., the combined effect of FLOPs × Hours) for higher accuracy.

Our system is **model-agnostic**, meaning we can easily swap the model being used behind the API without changing the user interface or the underlying service.

If your professor asks, “Which model is active now?” you can say:

"We started with Linear Regression as a transparent baseline. After benchmarking, we found that a Random Forest Regressor provided a better fit for the non-linear patterns in our data. Therefore, our currently saved model artifact is the Random Forest (rf_model.pkl). The dashboard and APIs remain unchanged; only the regressor class is different."

How We Use the Model in Our Pipeline 🛠️

1. **Feature Building**: The process starts when the user interacts with the UI. The values for Layers, FLOPs, and Hours are taken from the UI sliders, while the Complexity score is calculated on the fly from our NLP text analysis pipeline.
2. **Training (Offline)**:
    - We split our dataset into training and testing sets (e.g., an 80/20 split) or use K-fold cross-validation for more robust evaluation.
    - To handle features with different scales (like Layers vs. FLOPs), we use a Pipeline that includes a StandardScaler. This prevents features with larger values from unfairly dominating the model.
    - Since we have two targets (Energy_kWh and CO_2_kg), we use scikit-learn's MultiOutputRegressor to train a model that can predict both simultaneously.

3. **Evaluation**: We measure the model's performance using standard regression metrics: **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, and **R-squared (R^2)**. These tell us how accurate our predictions are and how well the model explains the variance in the data. We compare the performance of the Linear Regression and Random Forest models to choose the one that generalizes best without overfitting.
4. **Serialization**: Once trained, the final model object is saved to a file (e.g., `models/rf_model.pkl`) using `joblib`. We also version the model and its associated feature schema.
5. **Inference (Online)**:
    - The Streamlit UI sends the four feature values to a `/predict` API endpoint.
    - The web service loads the saved model once at startup. When a request comes in, it runs `model.predict()` on the features, which takes only milliseconds.
    - The predicted `energy_kWh` and `co2_kg` are sent back to the UI to be displayed on the charts.

Connecting Regression to Anomaly Detection & Optimization

The same 4-D feature vector `[layers, flops, hours, complexity]` that we use for prediction is also sent to an **Isolation Forest** model. This model is trained to flag inefficient configurations—for example, a job that uses an unusually high amount of energy for its given complexity.

When a configuration is flagged as an anomaly, our system can automatically generate rewritten, more efficient prompts. We then feed these new prompts (with their lower complexity scores) back into the regression model to re-predict the energy and CO_2. The difference between the original and new predictions is the **estimated savings**, which we display to the user.

Tiny Code Snippets You Can Mention

You can mention these pseudo-Python snippets to illustrate the process.

Training a multi-output linear model:

Python

```
# Training (baseline linear, multi-output)
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.multioutput import MultiOutputRegressor

# Define the features (X) and targets (Y)
X = df[["Layers", "FLOPs_in_TFLOPs", "Training_Hours", "Complexity"]]
Y = df[["Energy_kWh", "CO2_kg"]]

# Create a pipeline that scales data and then fits the regressor
pipe = Pipeline([
  ("scale", StandardScaler()),
  ("reg", MultiOutputRegressor(LinearRegression()))
])
```

```
# Train the entire pipeline
pipe.fit(X_train, Y_train)
```

Saving, loading, and predicting:

Python

```
import joblib

# Save the trained model
joblib.dump(pipe, "models/linear_multioutput.pkl")

# Load a model (e.g., the Random Forest version)
model = joblib.load("models/rf_model.pkl")

# Make a prediction with new data
features = [[layers, flops_tflops, hours, complexity]]
pred_energy, pred_co2 = model.predict(features)[0]
```

Assumptions & Pitfalls (and Our Solutions)

- **Multicollinearity (Linear Model Pitfall)**: Features like FLOPs, Layers, and Hours might be correlated with each other. We mitigate this by scaling features and can use techniques like Ridge Regression if it becomes an issue.
- **Overfitting (Random Forest Pitfall)**: A Random Forest can memorize the training data too well. We prevent this by using cross-validation and tuning hyperparameters like the number and depth of the trees.
- **Data Shift**: The model's performance could degrade if the real-world data it sees over time is different from its training data. We plan for this with scheduled retraining and drift monitoring.

One-Liners for Q&A

- **What is linear regression here?**
  - A fast, interpretable baseline model that maps our four key features to energy and CO_2 predictions.
- **Why not only use a linear model?**
  - We start with a linear model for its simplicity and explainability. We upgrade to a Random Forest when its ability to capture non-linear patterns gives us better accuracy.
- **How do you evaluate your model?**
  - With MSE, MAE, and R-squared using cross-validation. We choose the model that generalizes best to unseen data.
- **Where is the model used?**
  - It's served via a /predict API that backs our Streamlit dashboard. Its outputs drive the charts and the anomaly detection checks.
- **How does this actually reduce energy?**
  - By giving users an instant preview of the predicted energy/CO_2 cost, it allows them to make informed decisions. It also proactively suggests lower-complexity prompts that our model predicts will consume less energy.