```
# genovate_backend.py

import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load or simulate data (replace with real data or CSV if needed)
def load_data():
    np.random.seed(42)
    num_samples = 200
    mutations = np.random.choice(['PKD1', 'PKD2', 'PKHD1'], num_samples)
    organs = np.random.choice(['Kidney', 'Liver'], num_samples)
    methods = np.random.choice(['LNP', 'Electroporation'], num_samples)

    efficiency = np.where(methods == 'LNP',
                          np.random.normal(0.72, 0.05, num_samples),
                          np.random.normal(0.85, 0.04, num_samples))

    off_target = np.where(methods == 'LNP',
                          np.random.normal(0.07, 0.02, num_samples),
                          np.random.normal(0.12, 0.03, num_samples))

    cell_viability = np.where(methods == 'LNP',
                              np.random.normal(0.92, 0.03, num_samples),
                              np.random.normal(0.75, 0.05, num_samples))

    cost = np.where(methods == 'LNP',
                    np.random.randint(1, 3, num_samples),
                    np.random.randint(3, 5, num_samples))

    data = pd.DataFrame({
        "Mutation": mutations,
        "TargetOrgan": organs,
        "DeliveryMethod": methods,
        "Efficiency": np.clip(efficiency, 0, 1),
        "OffTargetRisk": np.clip(off_target, 0, 1),
        "CellViability": np.clip(cell_viability, 0, 1),
        "Cost": cost
    })
    return data

# Prepare and train model
def train_model(data):
    le_mut = LabelEncoder()
    le_org = LabelEncoder()
    le_method = LabelEncoder()

    data['Mutation_enc'] = le_mut.fit_transform(data['Mutation'])
    data['Organ_enc'] = le_org.fit_transform(data['TargetOrgan'])
    data['Method_enc'] = le_method.fit_transform(data['DeliveryMethod'])

    X = data[['Mutation_enc', 'Organ_enc', 'Efficiency', 'OffTargetRisk', 'CellViability', 'Cost']]
    y = data['Method_enc']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    return model, le_mut, le_org, le_method

# Prediction function
def predict_method(model, le_mut, le_org, le_method, mutation, organ, eff, off, viability, cost):
    features = np.array([[le_mut.transform([mutation])[0],
                          le_org.transform([organ])[0],
                          eff, off, viability, cost]])
    pred = model.predict(features)[0]
    return le_method.inverse_transform([pred])[0]

# PAM Finder (for integration)
def find_pam_sites(dna_sequence, pam="NGG"):
    pam_sites = []
```

```
for i in range(len(dna_sequence) - len(pam) + 1):
    window = dna_sequence[i:i+len(pam)]
    match = all(b == p or p == 'N' for b, p in zip(window, pam))
    if match:
        pam_sites.append((i, window))
return pam_sites
```