**Machine Learning-Based Prediction Models for NCNTFET and PCNFET in Low-Power VLSI Applications**

**1. Introduction**

The increasing demand for low-power and high-performance circuits in Very Large Scale Integration (VLSI) has led to the exploration of **Carbon Nano Tube Field-Effect Transistors (CNTFETs)** as an alternative to traditional MOSFETs. CNTFETs provide improved power efficiency and scaling capabilities in the sub-micron region. This project investigates **NCNTFET (N-type CNTFET) and PCNFET (P-type CNTFET)** behavior, using **Machine Learning (ML) models** to predict crucial parameters such as **DCNT, VTH, ID, and Power**, aiding in the design of memory elements for optimized power consumption.

This document details the implementation of **two ML-based predictive models**, developed using **Random Forest Regressor**, and explains the rationale for choosing this specific ML model, dataset preprocessing, training methodology, evaluation metrics, and the overall significance of ML-based predictions in this project.

**2. NCNTFET and PCNFET Models**

In this project, two separate ML models were designed to predict the behavior of **NCNTFETs and PCNFETs** based on user-inputted values of **N (Chiral Index), VGS (Gate-to-Source Voltage), VDD (Drain Voltage), and VDS (Source Voltage)**. These models provide dual predictions: **one based on mathematical formulas** and another using **trained ML models** for improved accuracy and optimization.

**3. Dataset and Data Preprocessing**

The dataset was generated using **Cadence Virtuoso simulations**, capturing key electrical parameters of CNTFETs. This dataset was preprocessed to remove missing values and standardize numerical features.

**Key Features Used:**

* **N (Chiral Index)**: Defines CNTFET structure.
* **VGS (Gate-Source Voltage)**: Controls transistor switching.
* **VDD (Drain Voltage)**: Determines power consumption.
* **VDS (Source Voltage)**: Affects output current.

**Target Variable:**

* **ID (Drain Current)**: Key output parameter for power analysis.

A **StandardScaler** was used to normalize the features for better model convergence.

**4. Machine Learning Model Selection**

**4.1 Choice of Model: Random Forest Regressor**

After evaluating multiple ML algorithms, **Random Forest Regressor** was selected due to its:

* **Ability to handle non-linearity**: CNTFET behavior is highly nonlinear.
* **Robustness to overfitting**: Multiple decision trees reduce variance.
* **High interpretability**: Feature importance can be extracted.
* **Scalability**: Works well with large datasets.

Alternative models like **Linear Regression** were ruled out due to poor performance in handling nonlinear relationships, while **Neural Networks** were considered but required excessive training time and hyperparameter tuning.

**4.2 Hyperparameter Tuning**

Hyperparameters were optimized using **RandomizedSearchCV**, tuning:

* n\_estimators: 50, 100, 200, 300
* max\_depth: None, 5, 10, 15, 20
* min\_samples\_split: 2, 5, 10, 15
* min\_samples\_leaf: 1, 2, 4
* max\_features: sqrt, log2, None

**5. Model Training and Evaluation**

**5.1 Training Procedure**

The dataset was split into **80% training and 20% testing**. The training phase involved fitting the **Random Forest model** with the optimal hyperparameters.

**5.2 Evaluation Metrics**

The model's accuracy was assessed using:

* **Mean Squared Error (MSE)**: Measures prediction error.
* **R-squared Score (R²)**: Evaluates model fit.

The final trained models were stored using **Joblib** for future inference.

**6. Prediction Mechanism**

The trained ML model was integrated into a **user-interactive program** that takes **N, VGS, VDD, and VDS** as inputs and predicts **ID and Power** using:

1. **Mathematical Formulas**: For direct analytical calculations.
2. **ML Predictions**: For refined estimates based on learned patterns.

**6.1 Formula-Based Calculation**

The drain current **(ID)** and power **(P)** are calculated using:

DCNT = 78.3 \* √(n² + m² + nm)

VTH = ± 0.43 / DCNT

mu\_Cox\_W\_L = 2.43e-3

Triode Region:

**ID** = mu\_Cox\_W\_L \* ((VGS - VTH) \* VDS - (VDS\*\*2 / 2))

Saturation Region:

**ID** = mu\_Cox\_W\_L \* (VGS - VTH)\*\*2 / 2

**Power** = VDD \* ID

**6.2 ML-Based Prediction**

The trained **Random Forest model** is used to predict **ID** based on the scaled input features, followed by power calculation: PML=VDD×IDMLP\_{ML} = VDD \times ID\_{ML}

**6.3 Comparison of Formula vs. ML Predictions**

The program prints the predicted **ID and Power** values from both approaches and computes their differences, showcasing the effectiveness of ML optimization.

**6.4 NCNFET Code**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

import joblib

# 1. Load Data

try:

    from google.colab import drive

    drive.mount('/content/drive')

    data\_path = '/content/drive/My Drive/Major Project/NCNFET.csv'

    df = pd.read\_csv(data\_path)

except ImportError:

    data\_path = 'NCNFET.csv'

    df = pd.read\_csv(data\_path)

    print("Running locally, ensure 'NCNFET.csv' is in the same directory.")

# 2. Data Preprocessing

df.dropna(inplace=True)

numerical\_features = ['N', 'M', 'VGS', 'VDD']

scaler = StandardScaler()

X = df[numerical\_features]

y = df['ID'].values.ravel()  # Convert y to 1D array

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_train\_scaled\_df = pd.DataFrame(X\_train\_scaled, columns=numerical\_features, index=X\_train.index)

X\_test\_scaled = scaler.transform(X\_test)

X\_test\_scaled\_df = pd.DataFrame(X\_test\_scaled, columns=numerical\_features, index=X\_test.index)

# 3. Model Training with Hyperparameter Tuning (Using RandomizedSearchCV)

param\_grid = {

    'n\_estimators': [50, 100, 200, 300],

    'max\_depth': [None, 5, 10, 15, 20],

    'min\_samples\_split': [2, 5, 10, 15],

    'min\_samples\_leaf': [1, 2, 4],

    'max\_features': ['sqrt', 'log2', None]

}

model = RandomForestRegressor(random\_state=42)

n\_iter = 20

random\_search = RandomizedSearchCV(estimator=model, param\_distributions=param\_grid,

                                    n\_iter=n\_iter, cv=5, scoring='neg\_mean\_squared\_error',

                                    n\_jobs=-1, random\_state=42)

try:

    random\_search.fit(X\_train\_scaled\_df, y\_train)

except KeyboardInterrupt:

    print("Random search interrupted. Saving intermediate results...")

    joblib.dump(random\_search, 'random\_search\_checkpoint.pkl')

    print("Checkpoint saved.")

else:

    best\_model = random\_search.best\_estimator\_

    joblib.dump(best\_model, 'ncnfet\_model.pkl')

    print("Random search completed successfully. Best model saved.")

# 4. Model Evaluation

y\_pred = best\_model.predict(X\_test\_scaled\_df)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse:.2e}")

print(f"R-squared: {r2:.4f}")

# 5. User Input, Calculations, and Comparison

def calculate\_from\_formulas(N, M, VGS, VDD, VDS):

    DCNT = 78.3 \* np.sqrt(N\*\*2 + M\*\*2 + N \* M)

    if DCNT < 1e-6:

        print("Warning: DCNT is very small. Check input N.")

        VTH = 0.0

    else:

        VTH = 0.43 / DCNT

    mu\_Cox\_W\_L = 2.43e-3

    if VDS < (VGS - VTH):

        ID = mu\_Cox\_W\_L \* [(VGS - VTH) \* VDS - (VDS\*\*2 / 2)]

    else:

        ID = (mu\_Cox\_W\_L / 2) \* (VGS - VTH)\*\*2

    Power = VDD \* ID

    return DCNT, VTH, ID, Power

def predict\_from\_model(input\_data):

    scaled\_input = scaler.transform(input\_data[numerical\_features])

    scaled\_input\_df = pd.DataFrame(scaled\_input, columns=numerical\_features, index=input\_data.index)

    predicted\_id = best\_model.predict(scaled\_input\_df)

    return predicted\_id

while True:

    try:

        N = float(input("Enter the value for N: "))

        VGS = float(input("Enter the value for VGS: "))

        VDD = float(input("Enter the value for VDD: "))

        VDS = float(input("Enter the value for VDS: "))  # VDS must be equal to VDD

        user\_input = pd.DataFrame({'N': [N], 'M': [0], 'VGS': [VGS], 'VDD': [VDD]})

        DCNT\_formula, VTH\_formula, ID\_formula, Power\_formula = calculate\_from\_formulas(N, 0, VGS, VDD, VDS)

        predicted\_id = predict\_from\_model(user\_input)

        Power\_ml = VDD \* predicted\_id[0]

        print("\n--- Results ---")

        print("Formula Calculations:")

        DCNT\_formatted = f"{DCNT\_formula \* 1e-3:.6f}"  # Multiply by 10^3 for 10^-3 representation

        VTH\_formatted = f"{VTH\_formula \* 1e3:.6f}"  # Multiply by 10^-3 for 10^3 representation

        DCNT\_formatted = DCNT\_formatted.rstrip('0').rstrip('.') if '.' in DCNT\_formatted else DCNT\_formatted

        VTH\_formatted = VTH\_formatted.rstrip('0').rstrip('.') if '.' in VTH\_formatted else VTH\_formatted

        print(f"• DCNT (x10^-3): {DCNT\_formatted}") #Added units

        print(f"• VTH (x10^3): {VTH\_formatted}") #Added units

        print(f"• ID: {ID\_formula:.2e}")

        print(f"• Power: {Power\_formula:.2e}")

        print("\nML Predictions:")

        print(f"• ID: {predicted\_id[0]:.2e}")

        print(f"• Power: {Power\_ml:.2e}")

        # Comparison

        print("\n--- Comparison ---")

        print(f"ID Difference: {abs(ID\_formula - predicted\_id[0]):.2e}")

        print(f"Power Difference: {abs(Power\_formula - Power\_ml):.2e}")

    except ValueError:

        print("Invalid input. Please enter numbers only.")

    another\_prediction = input("Do you want to make another prediction? (yes/no): ").lower()

    if another\_prediction != 'yes':

        break

**Note: PCNTFET code is almost the same with only minor changes in the formula. Prediction part is the same.**

**7. Importance in Real-Life Circuit Design**

The robustness and accuracy of this ML-based prediction system make it highly relevant in modern circuit design, particularly for **low-power applications**. Key advantages include:

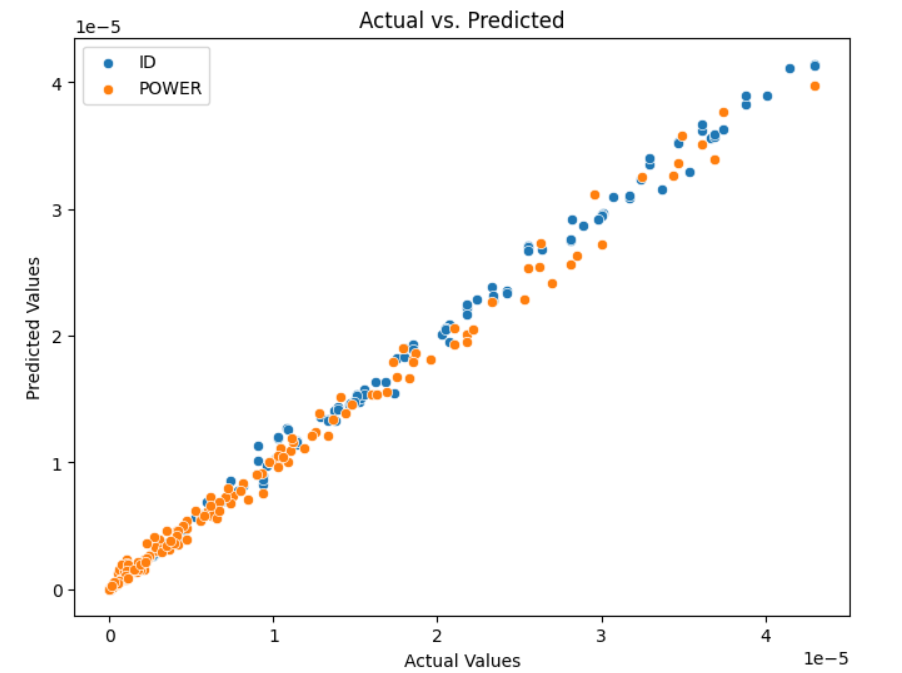
* **Precision in circuit behavior prediction**: Reduces trial and error in hardware design.
* **Enhances energy efficiency**: ML-based predictions help in minimizing power consumption.
* **Adaptability**: Can be extended to various semiconductor technologies beyond CNTFETs.
* **Faster prototyping**: Reduces reliance on costly and time-consuming SPICE simulations.
* **Reliable and scalable**: Random Forest ensures stable and generalizable predictions.

**8. Conclusion**

This project successfully implements **ML-based predictive models** for **NCNTFET and PCNFET** design, offering a powerful tool for optimizing memory elements in **low-power VLSI applications**. By leveraging **Random Forest Regression**, we achieve accurate **ID and Power predictions**, enabling faster and more efficient circuit design for next-generation **nanotechnology-based computing systems**.

The **synergy between ML and traditional mathematical models** in this project underscores its importance in advancing **CNTFET-based VLSI designs**, making it a **highly impactful research endeavor** in the field of **low-power electronics**.

**9. Output**

****

