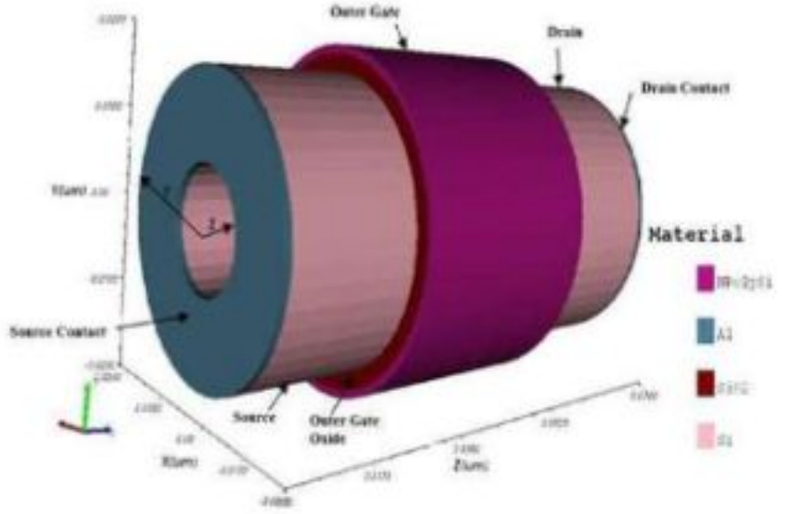


Fig. 1. 3D simulated structure of DGAA MOSFET.

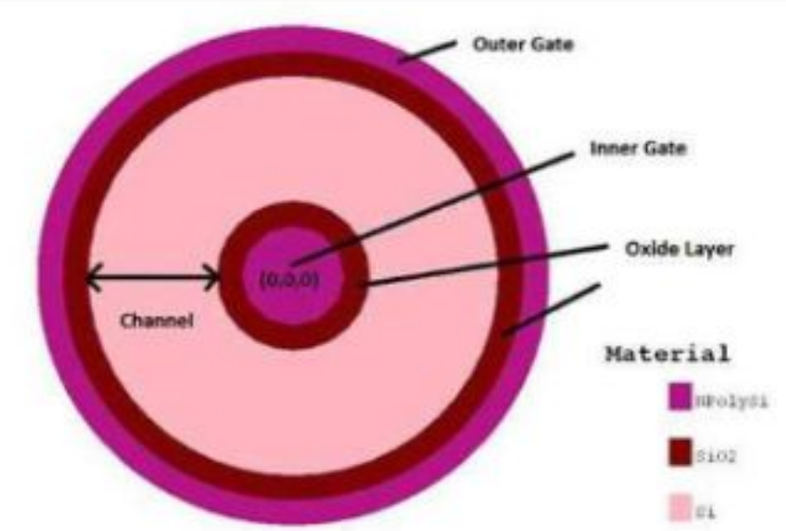
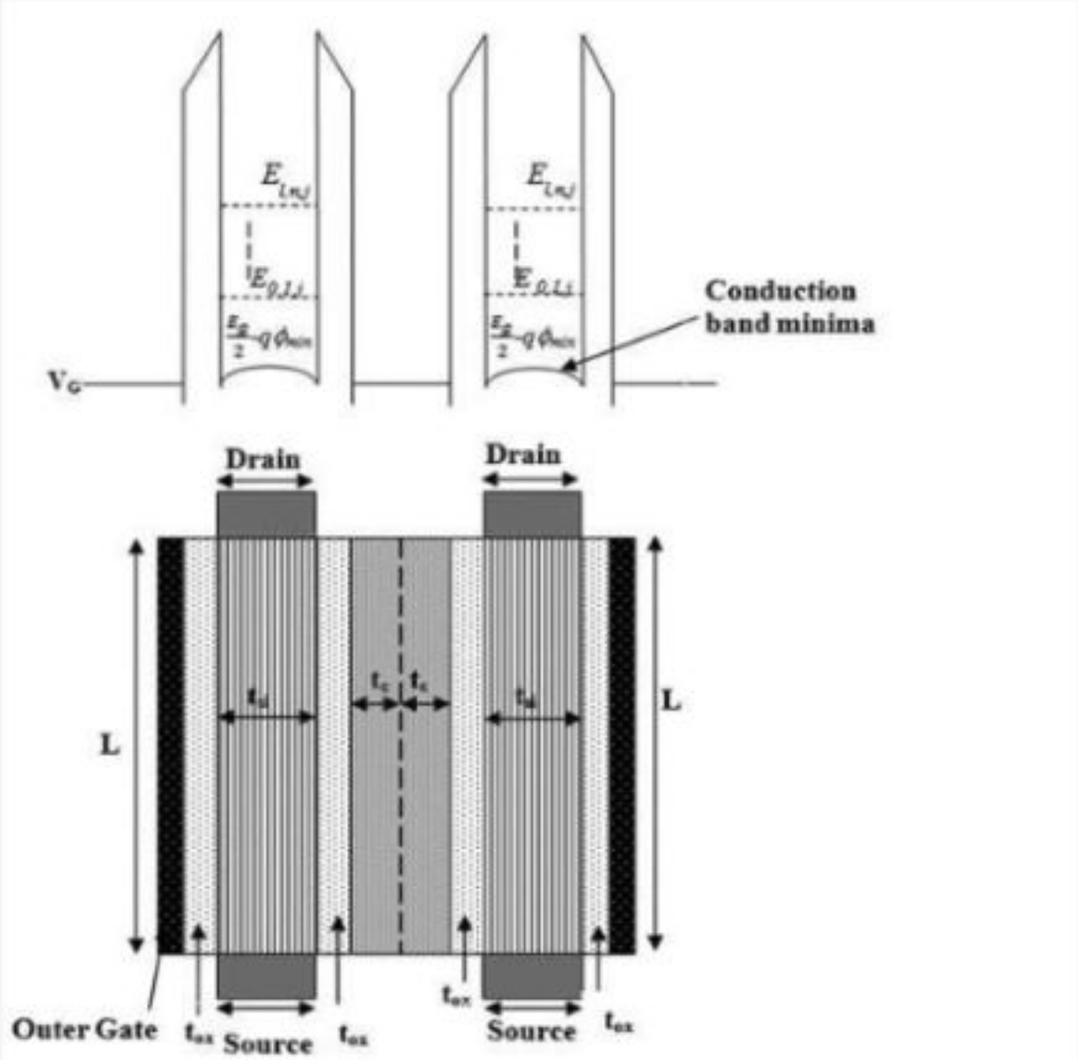


Fig. 2. Circular cross section view of DGAA MOSFET.



3. 2-D Cross sectional view of a double gate-all-around (DGAA) SFET with discrete energy levels in channel.

MATLAB CODE IMPLEMENTATION

Parameter Ranges

- Realistic Nanoscale Ranges:
- Channel length: 10–40 nm, oxide thickness: 1–3 nm, doping: 1e15–1e17 cm^{-3} for accurate simulation of real-world devices.
- Snapshot – Parameter Definitions:
- (Highlight variables like L_min, t_si_min, and other range-defining constants.)

Threshold Voltage Calculation

- Core Computation Logic:
- Solves a quadratic equation derived from Poisson-Schrödinger equations to compute V_{th}.
- Accounts for Key Effects:
- Includes flat-band voltage (V_{fb}), quantum confinement energy (E_{l,n,j}), and applied drain bias (V_{ds}).
- Snapshot – V_{th} Computation:
- (Show the for loop where A, B, C coefficients are calculated and used to find V_{th}.)

Visualization

- Data Exploration:
- Plots threshold voltage (V_{th}) against key variables: channel length, silicon thickness, and doping level.
- Snapshot – Scatter Plots:
- (Include example scatter plot showing V_{th} vs. channel length or other parameters.)

PARAMETER VARIATION IN MATLAB CODE

```
% Initialize parameter ranges (realistic values based on paper)
L_min = 10e-7;   % Minimum channel length (10 nm in cm)
L_max = 40e-7;  % Maximum channel length (40 nm in cm)

t_si_min = 5e-7; % Minimum channel thickness (5 nm in cm)
t_si_max = 6e-7;% Maximum channel thickness (6 nm in cm)

t_ox_min = 1e-7; % Minimum oxide thickness (1 nm in cm)
t_ox_max = 3e-7; % Maximum oxide thickness (3 nm in cm)

t_c_min = 2e-7;  % Minimum inner gate radius (2 nm in cm)
t_c_max = 3e-7;  % Maximum inner gate radius (3 nm in cm)

Na_min = 1e15;   % Minimum channel doping (cm^-3)
Na_max = 1e17;   % Maximum channel doping (cm^-3)

Nd = 1e20;       % Fixed source/drain doping (cm^-3)

Vds_min = 0.4;   % Minimum drain voltage (V)
Vds_max = 0.9;   % Maximum drain voltage (V)
```

VTH CALCULATION

```
k0_1_rc = 2.4048;
rc = t_c(i) + t_si(i)/2;
E_1_n_j = Eg/2 + (hbar^2)/(2*rc^2) * (1/mc + 1/mt) * (k0_1_rc)^2;
Qth = 1e12 * q;
a7 = E_1_n_j + kT * log(Qth / (q * sqrt(2*mz*kT/(pi*hbar^2)))) + ...
    q * Na(i) * a3 / (eps_si * alpha1);
a8 = (2*Vbi(i)^2 + 2*Vbi(i)*Vds(i)) * (cosh(sqrt(alpha1)*L(i)) - 1) - Vds(i)^2 + ...
    (2*Vbi(i) + Vds(i)) * q * Na(i) / (eps_si * alpha1) * (cosh(sqrt(alpha1)*L(i)) - 1);

A = a1 * a2^2 - a4^2;
B = a1^2 * a6 + 2 * a2 * a7;
C = a1^2 * a8 - a7^2;
discriminant = B^2 - 4*A*C;
if discriminant < 0
    Vth_sol = -B/(2*A);
else
    Vth_sol = (-B + sqrt(discriminant)) / (2*A);
end
Vth(i) = Vfb(i) + Vth_sol;
if Vth(i) < 0
    Vth(i) = abs(Vth(i)); % Take absolute value if negative
end
```

.CSV FILE(DATASET)

- 5000 DATASETS WHERE VARYING PARAMETERS ARE CHANNEL LENGTH,CHANNEL THICKNESS,OXIDE THICKNESS,INNER GATE RADIUS,CHANNEL DOPING,DRAIN VOLTAGE.

A	B	C	D	E	F	G
Channel_Length_nm	Channel_Thickness	Oxide_Thickness_nm	Inner_Gate_Radius_nm	Channel_Doping_cm3	Drain_Voltage_V	Threshold_Voltage_V
10.00034904	5.269119753	1.077944728	2.903206668	9.26E+16	0.473005184	0.315404888
10.00092157	5.497153816	2.745746572	2.715569635	2.50E+16	0.604827086	0.25401267
10.00404079	5.320181167	2.641350767	2.018730292	5.86E+16	0.855794694	0.281550374
10.00681115	5.966178025	2.705021273	2.70039358	5.08E+15	0.80354146	0.202367442
10.01793817	5.5375231	1.530298004	2.389527608	7.35E+16	0.56539391	0.307004635
10.01960172	5.277025449	1.99449096	2.110601751	7.06E+16	0.718155408	0.30177935
10.02830246	5.950611775	2.390658813	2.09000794	5.07E+16	0.521732591	0.278758626
10.03336283	5.950592013	1.559975099	2.296302045	9.89E+16	0.44261433	0.313879469
10.04060877	5.072376393	1.244170843	2.289049733	6.42E+15	0.692949485	0.245216435
10.04421466	5.41291119	2.935294185	2.018555539	3.44E+16	0.590217901	0.259623506
10.04695316	5.738150755	1.225214298	2.269636165	5.07E+16	0.808126356	0.298584217
10.05976406	5.313367694	2.33015488	2.052157231	4.92E+16	0.659655321	0.286709143
10.09654791	5.048464232	1.461880718	2.162858292	8.08E+16	0.495214732	0.311087454
10.11459768	5.178841069	1.069964977	2.361616341	7.14E+16	0.818859622	0.308519409
10.12562331	5.670974709	2.599658908	2.053465724	3.06E+16	0.574616145	0.264411864
10.13089809	5.325581133	1.128390972	2.461864459	9.86E+16	0.879520108	0.316977898
10.13896069	5.44588662	1.110435254	2.867993933	6.34E+15	0.647251982	0.245035347

ML MODELS IMPLEMENTED

1.XG BOOST

- Used XGBoost library for gradient boosting regression.
- Loaded dataset using pandas; target = Threshold_Voltage_V, features = all other columns.
- Performed 70/30 train-test split using train_test_split.
- Converted data into DMatrix format for optimized XGBoost training.
- Model used booster = gblinear and objective = reg:linear for linear regression.
- Trained the model for 10 boosting rounds.
- Made predictions using predict() on test data.
- Evaluated performance using R^2 Score and RMSE to measure prediction accuracy and error.

RESULTS

R2 SCORE=0.73003

RMSE SCORE=0.01181

ML MODELS IMPLEMENTED

2.RANDOM FOREST REGRESSOR

- Imported libraries: RandomForestRegressor from sklearn.ensemble for model training, pandas for data handling, train_test_split from sklearn.model_selection for data splitting, sklearn.metrics for evaluation (RMSE and R^2), and numpy for numerical operations.
- Loaded dataset from CSV using pandas; defined X as all features except the target and y as Threshold_Voltage_V.
- Split data into 80% training and 20% testing using train_test_split.
- Created a Random Forest Regressor with 2000 trees, max_leaf_nodes=1000, max_depth=100, and random_state=42 for reproducibility.
- Trained the model using .fit() on the training data.
- Made predictions on the test set using .predict().
- Evaluated performance using R^2 Score and RMSE to assess prediction accuracy and error.

RESULTS

R2 SCORE=0.99608

RMSE SCORE=0.00143

RANDOM FOREST REGRESSOR CODE

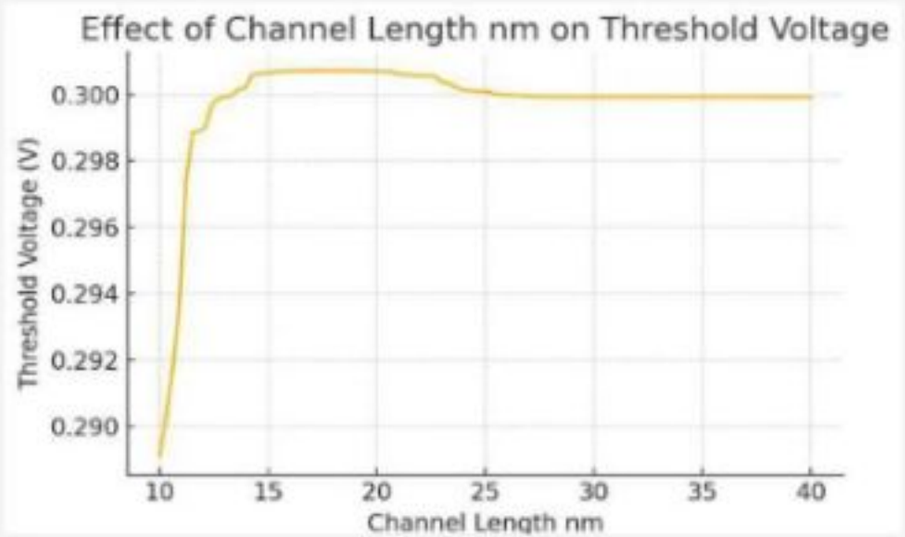
```python
from sklearn.ensemble import RandomForestRegressor as Rf
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE, r2_score
import numpy as np

# Load data
df = pd.read_csv("dgaa_mosfet_threshold_voltage_dataset.csv")

# Prepare data
X = df.drop("Threshold_Voltage_V", axis=1)
y = df["Threshold_Voltage_V"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)

# Train model
model = Rf(n_estimators=2000, random_state=42, max_leaf_nodes=1000, max_depth=100)
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(MSE(y_test, y_pred))
```

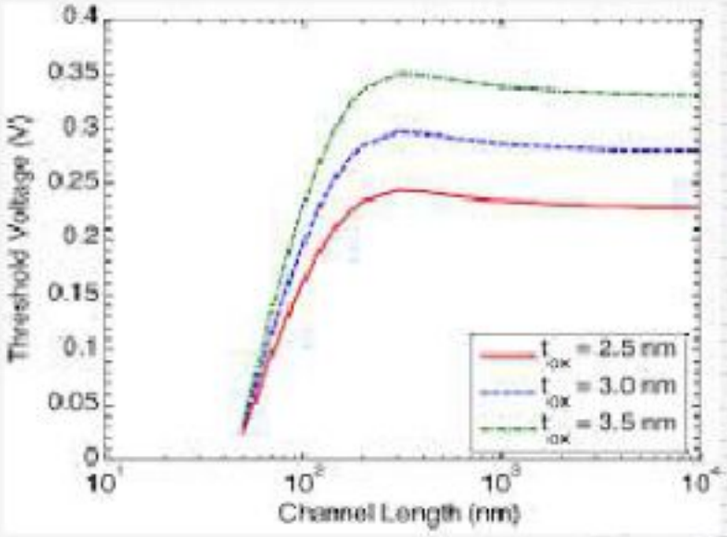
R2 SCOREAND RMSE CALCULATION

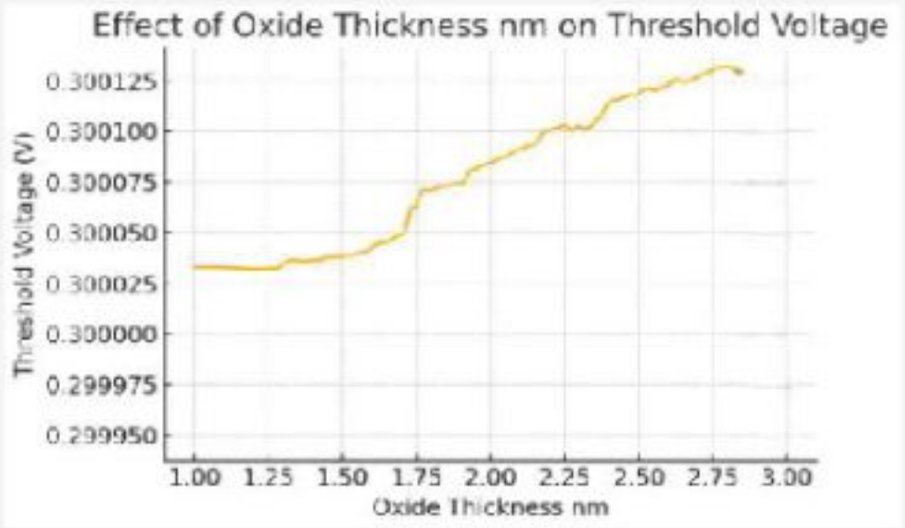
```python
def predict_threshold_voltage():
    print("\nEnter MOSFET parameters to predict threshold voltage:")

    # Get user input for each feature
    channel_length = float(input("Channel Length (nm): "))
    channel_thickness = float(input("Channel Thickness (nm): "))
    oxide_thickness = float(input("Oxide Thickness (nm): "))
    inner_gate_radius = float(input("Inner Gate Radius (nm): "))
    channel_doping = float(input("Channel Doping (cm^-3): "))
    drain_voltage = float(input("Drain Voltage (V): "))

    # Create input array
    input_data = np.array([[
        channel_length,
        channel_thickness,
        oxide_thickness,
        inner_gate_radius,
        channel_doping,
        drain_voltage
    ]])

    # Make prediction
    threshold_voltage = model.predict(input_data)
```

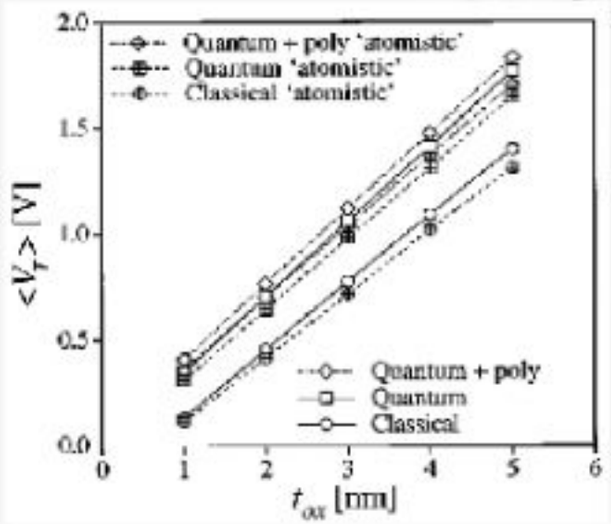
OUTPUT PREDICTION

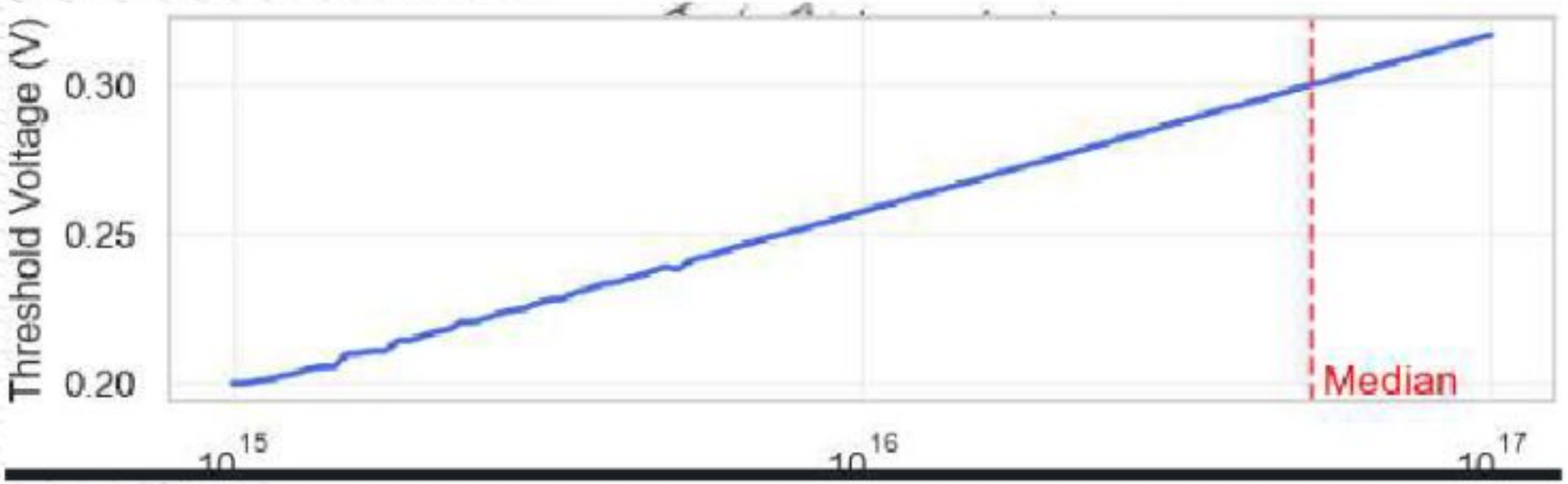
COMPARISION BETWEEN THE PREDICTED VALUE AND AND THE EXPECTED

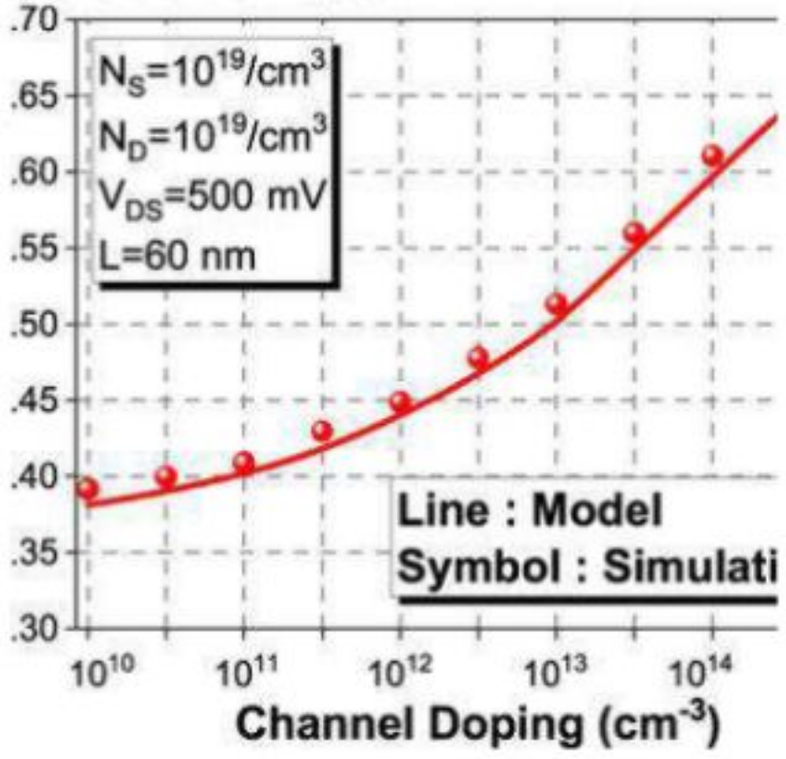


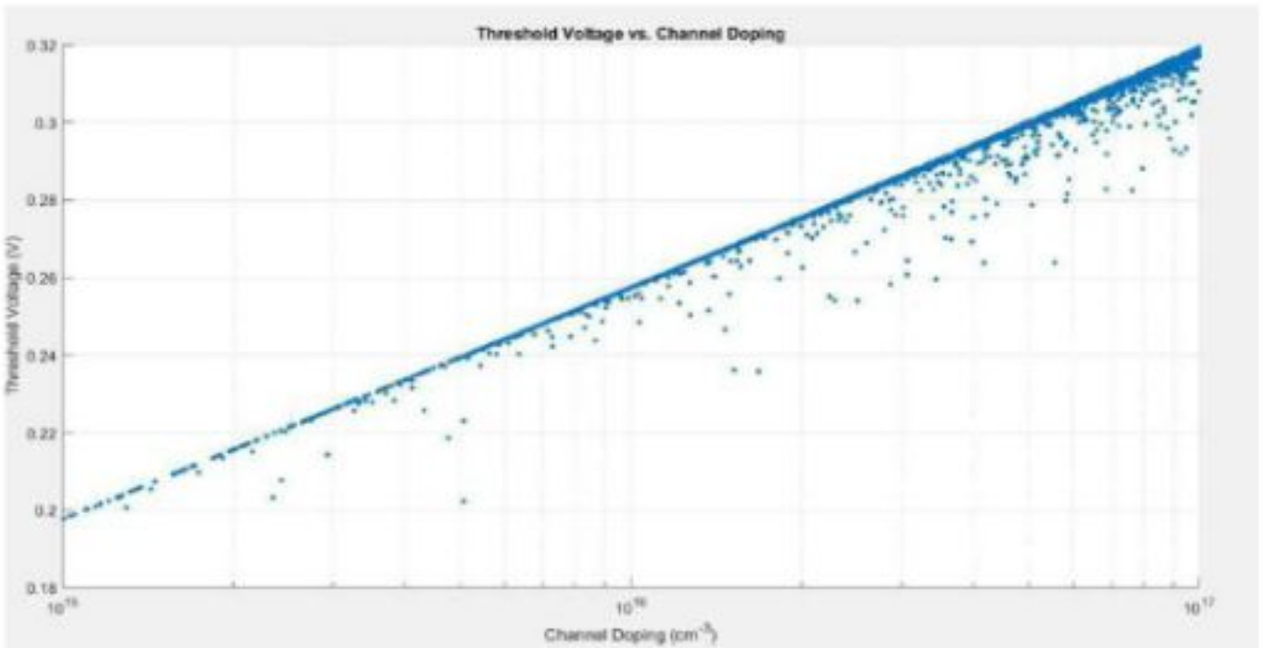




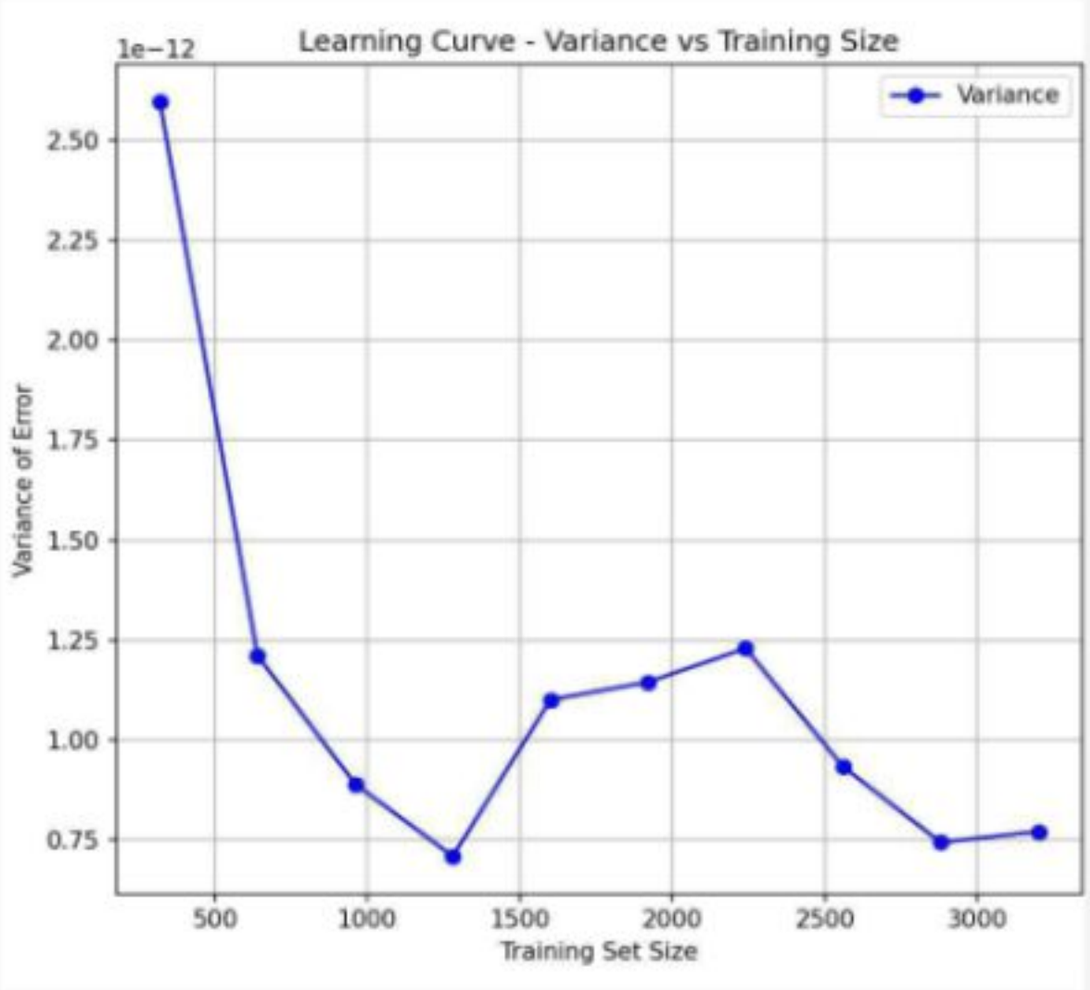



Threshold Voltage (V)
0.30
0.25
0.20
Median
10^15
10^16
10^17


.70
.65
.60
.55
.50
.45
.40
.35
.30
N_S=10^19/cm^3
N_D=10^19/cm^3
V_DS=500 mV
L=60 nm
Line : Model
Symbol : Simulati
10^10
10^11
10^12
10^13
10^14
Channel Doping (cm^-3)

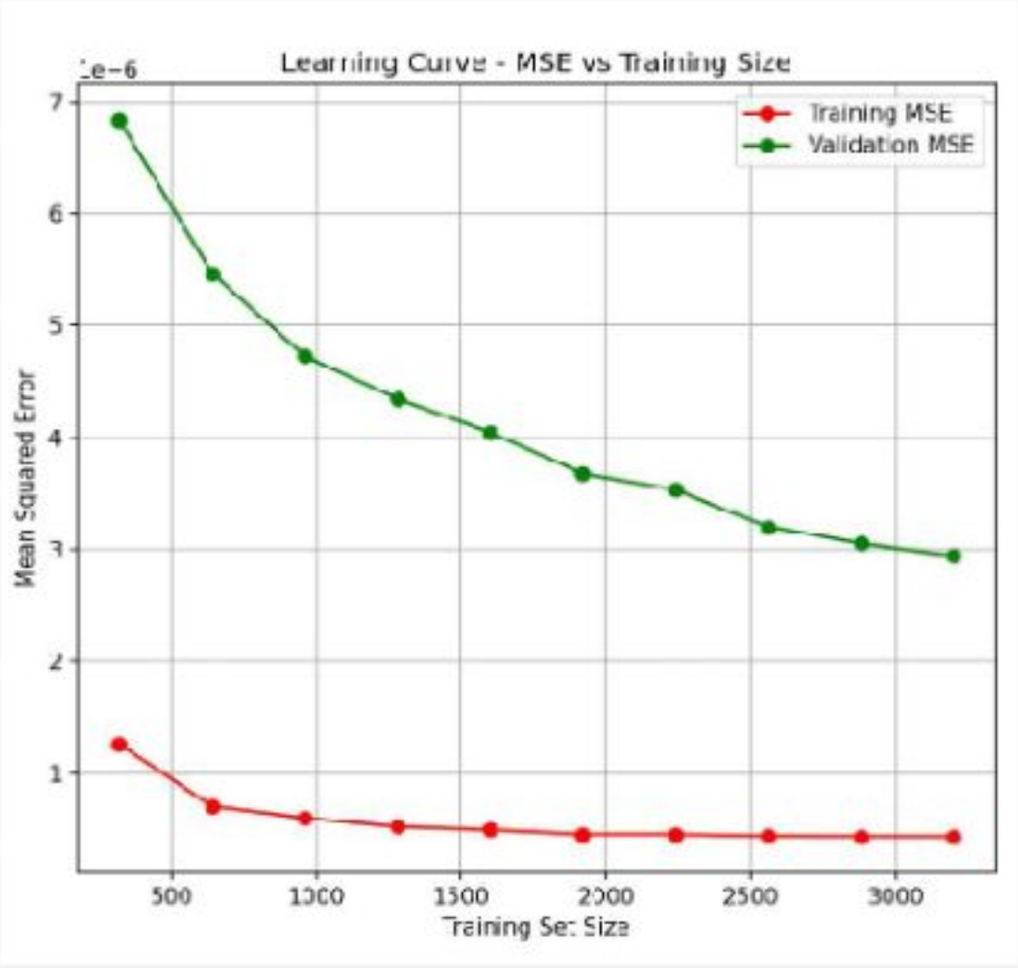

Threshold Voltage vs. Channel Doping
Threshold Voltage (V)
Channel Doping (cm^-3)

Error Analysis



- Adding more training data improves model performance, especially on validation data.
- The model benefits from having more examples to learn from.
- The learning curve indicates the model hasn't reached a hard limit yet.
- This suggests that adding even more data could further enhance performance.

cont..



1.Model is learning well:

- As training size increases, both training and validation MSE decrease.
- This indicates that the model's performance is improving.

1.No overfitting:

- There's a gap between training and validation MSE, but it's not increasing.
- In fact, the gap is decreasing, which is a positive sign.

1.More data helps:

- Validation error continues to decrease as more data is added.
- This shows the model hasn't plateaued yet in terms of performance gains from additional training data.

SENSITITVITY ANALYSIS