Temperature Based Fan Speed Controller System

BY: Pawan Kumar – 2021UCS1692

Paryas – 2021UCS1676

Introduction:

The temperature-based fan speed controller project utilizing TinyML (Tiny Machine Learning) on Arduino is an innovative application that combines embedded systems and machine learning capabilities for efficient thermal management. This project employs a small, low-power microcontroller like Arduino, enhanced with TinyML, which enables on-device machine learning inference. The system is designed to monitor ambient temperature using a temperature sensor and dynamically adjust the speed of a fan accordingly. The TinyML model, trained to recognize temperature patterns, runs directly on the Arduino board, eliminating the need for a constant connection to external servers. This enhances responsiveness, reduces latency, and optimizes energy consumption.

Hardware Requirements:

1. Arduino Uno R3
2. Temperature Sensor [TMP36]
3. Fan (DC Motor)
4. H-bridge Motor Driver
5. 9V Battery

Software Requirements:

1. Arduino IDE(Online Simulator) – Tinkercad.com
2. TensorFlow Lite for Arduino Library
3. Python with TensorFlow and scilit-learn for model training

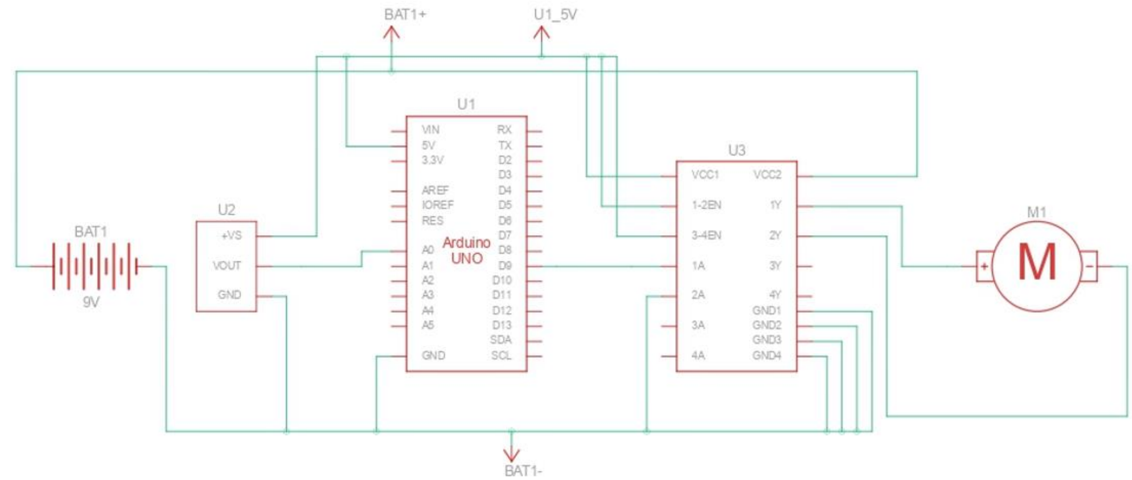
Steps:

1. Collect Data:

Collect temperature and corresponding fan speed data. Use the Arduino to log temperature and fan speed values.

2. Connect Hardware:

Connect the temperature sensor to the Arduino and power the cooling fan through a Temperature Sensor or relay controlled by one of the Arduino pins.



3. Install Libraries:

Install the necessary libraries in the Arduino IDE. You can find the TensorFlow Lite for Arduino library here.

4. Train Machine Learning Model:

Use Python with TensorFlow and scikit-learn to train a machine learning model that predicts fan speed based on temperature. Save the trained model as a TensorFlow Lite model.

5.Model Selection:

Linear regression is chosen as the modeling technique due to its simplicity and ability to capture linear relationships.

6.Data Splitting:

The dataset is split into training and testing sets to assess the model's generalization performance.

7.Model Training:

The scikit-learn LinearRegression class is employed to train the model on the training data.

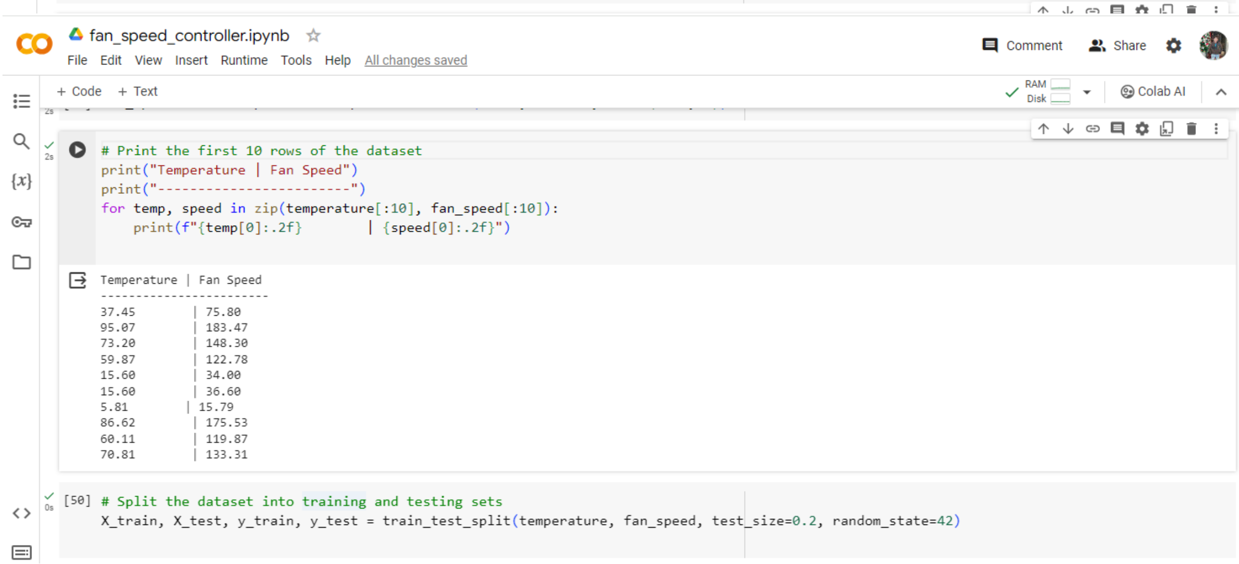
The fit method is used to find the optimal parameters (slope and intercept) that best fit the data.

In a linear regression model,the parameter w_0 (intercept) and w_1 (slope) represent the weights associated with the input features.For a simple linear regression with one feature,the model can expressed as:

$$\textbf{fan\_speed} = w_0 + w_1 \times \textbf{temperature}$$

8.Integration of TinyML: Enabling Machine Learning on Resource-Constrained Devices

Tiny Machine Learning (TinyML) represents a cutting-edge approach to deploying machine learning models on devices with limited computational resources, such as microcontrollers and edge devices. This integration is a multi-step process, beginning with model training and culminating in the deployment of a compact and efficient model directly onto the target device.



fan_speed_controller.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text RAM Disk Colab AI

```
[29] # Standardize features (optional but can be helpful)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Train a linear regression model using Keras
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(1,)),
  LinearRegression: model
  sklearn.linear_model._base.LinearRegression instance
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train_scaled, y_train, epochs=100, verbose=0)
```

```
<keras.src.callbacks.History at 0x7c53cae901c0>
```

```
[31] # Convert the Keras model to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

```
[32] # Save the TensorFlow Lite model
open('fan_speed_model.tflite', 'wb').write(tflite_model)
```

```
1080
```

fan_speed_controller.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text RAM Disk Colab AI

1080

```
[33] model = LinearRegression()
model.fit(temperature, fan_speed)
```

```
▾ LinearRegression
LinearRegression()
```

```
w0 = model.intercept_
w1 = model.coef_[0]

print(f'Intercept (w0): {w0}')
print(f'Slope (w1): {w1}')
```

```
Intercept (w0): [0.87390129]
Slope (w1): [1.99226074]
```

```
[35] y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
accuracy = 1 - (mse / np.var(y_test))
print(f"Accuracy on the test set: {accuracy:.2%}")
```

```
Accuracy on the test set: 99.36%
```

fan_speed_controller.ipynb

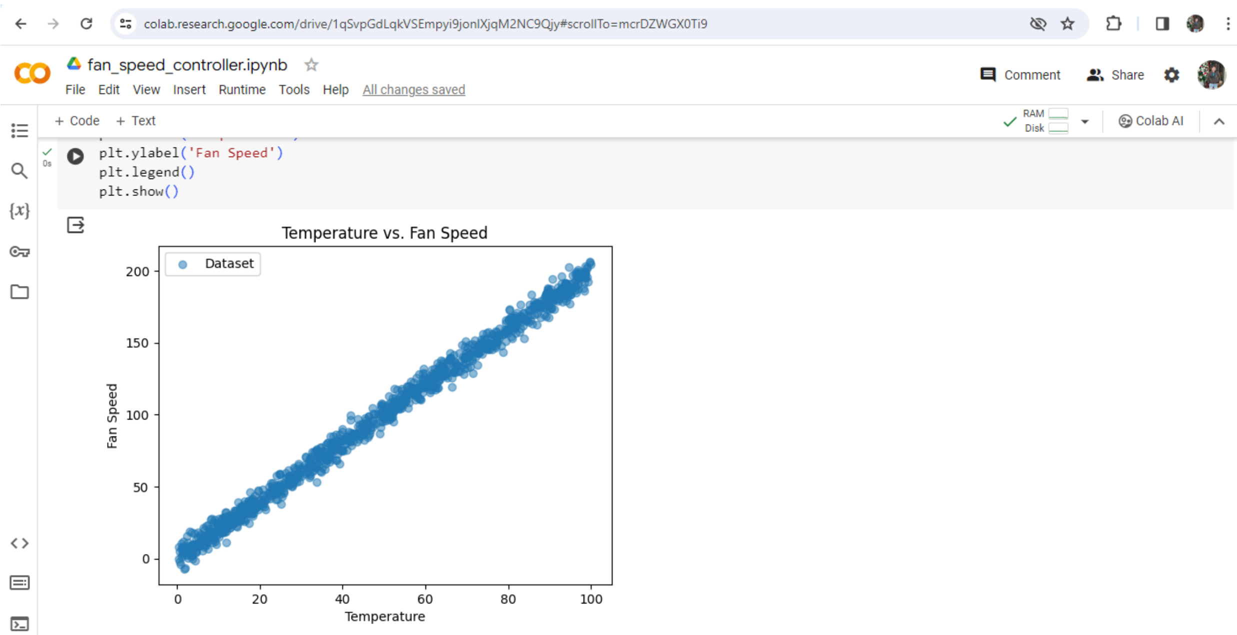
File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text RAM Disk Colab AI

```
# Print regression metrics
mse = mean_squared_error(y_test, y_pred)
r2 = model.score(X_test, y_test)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

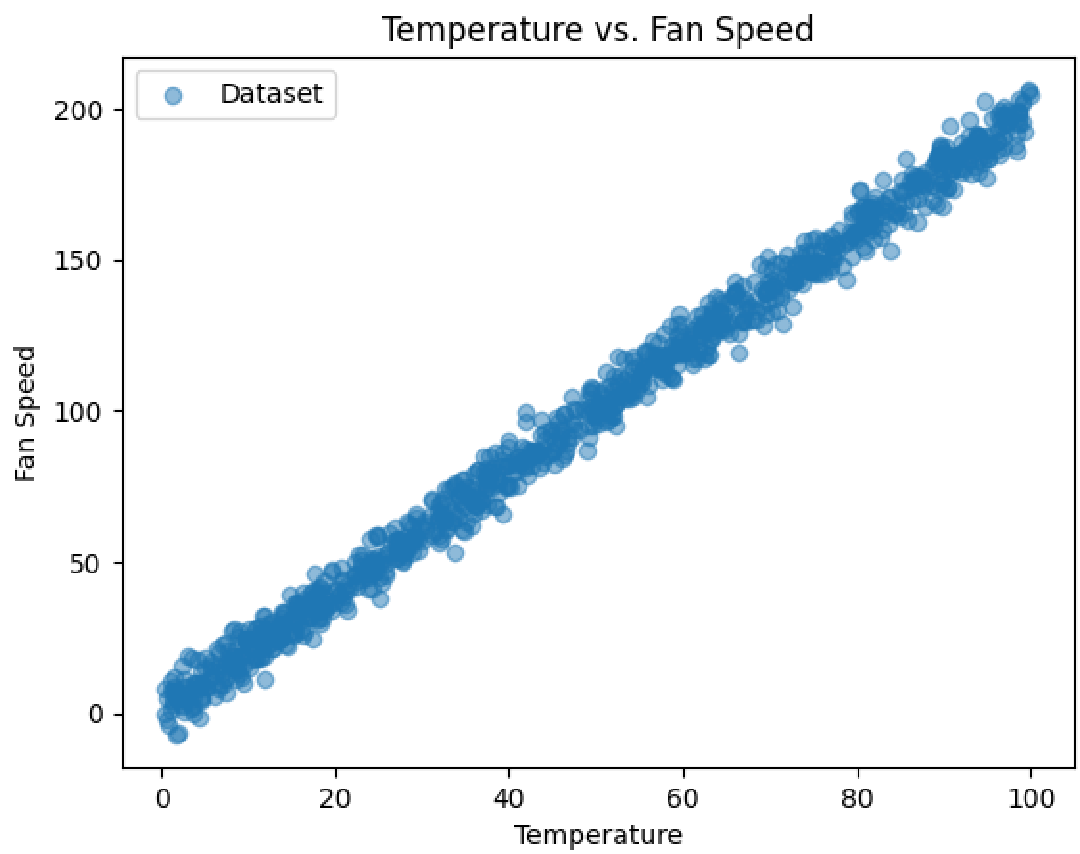
```
Mean Squared Error (MSE): 21.42
R-squared (R2): 0.99
```



Visualizations:

Scatter Plot:

The Scatter plot overlays the linear regression model on top of the actual data points. The red line represents the linear regression line, showing the model's attempt to capture the underlying relationship between temperature and fan speed.



Arduino Code:

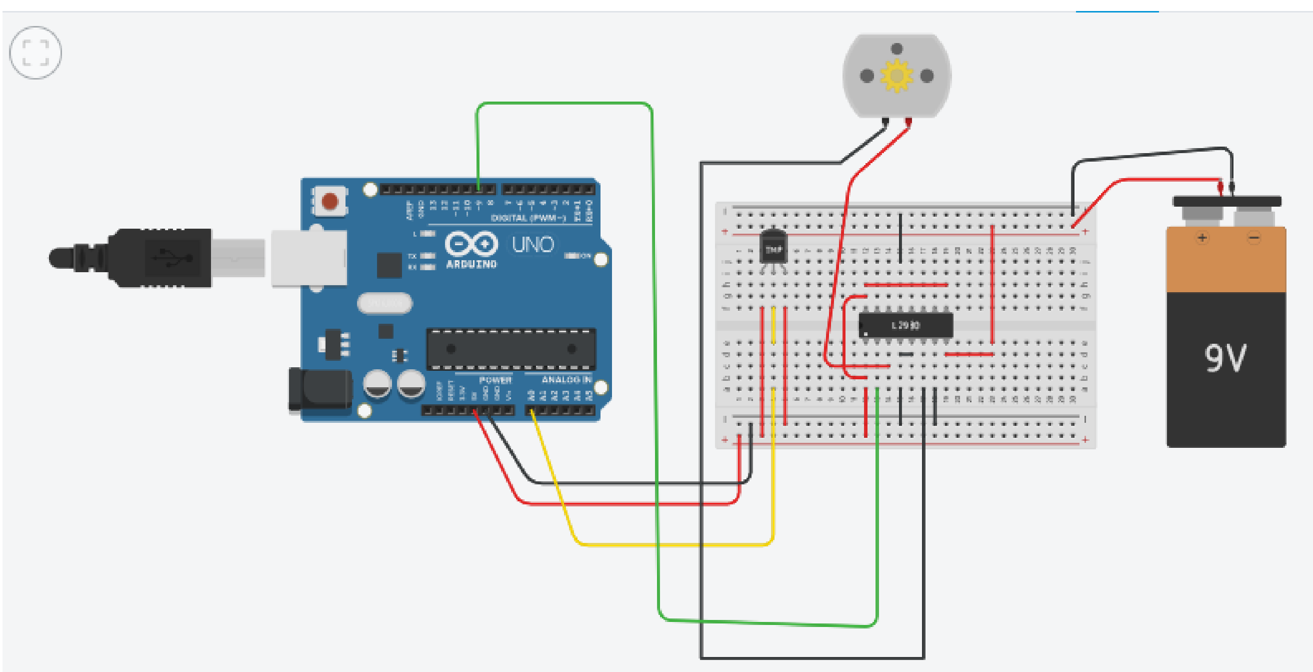
```
const int fanPin = 9;
const int tempPin = A0;
float w0 = 0.87390129;
float w1 = 1.99226074;
void setup()
{
  pinMode(fanPin, OUTPUT);
  Serial.begin(9600);
  while (!Serial);
}
int calculateFanSpeed(float temperature, float w0, float w1)
{
  int fanSpeed = constrain(int(w0 + w1 * temperature), 0, 255);

  return fanSpeed;
}
void loop()
{
  int reading = analogRead(tempPin);
  float voltage = reading *(5.0/1024.0);
  float temp = (voltage-0.5)*100;

  // Use linear regression coefficients to determine fan speed
  int fanSpeed = calculateFanSpeed(temp, w0, w1);

  Serial.println("------------------------------------------------");
  Serial.println("               Temperature Control System");
```

Text

1 (Arduino Uno R3)

```
  int fanSpeed = constrain(int(w0 + w1 * temperature), 0, 255);

  return fanSpeed;
}
void loop()
{
  int reading = analogRead(tempPin);
  float voltage = reading *(5.0/1024.0);
  float temp = (voltage-0.5)*100;

  // Use linear regression coefficients to determine fan speed
  int fanSpeed = calculateFanSpeed(temp, w0, w1);

  Serial.println("------------------------------------------------");
  Serial.println("               Temperature Control System");
  Serial.println("------------------------------------------------");
  Serial.print("Current Temperature: ");
  Serial.print(temp);
  Serial.write(0xB0);
  Serial.println("C");
  Serial.print("Estimated Fan Speed: ");
  Serial.println(fanSpeed);

  analogWrite(fanPin, fanSpeed);

  delay(1000);

}
```

Circuit Diagram:



Limitations:

- **Limited Processing Power and Memory:** Arduino boards, especially the ones commonly used for TinyML applications, may have limited processing power and memory. This can restrict the complexity of the machine learning model you can deploy on the device

- **Model Size:** TinyML models are designed to be small and lightweight to run on microcontrollers. As a result, you might be limited in terms of the size and complexity of the machine learning model you can use. Larger models may not fit into the available memory.

- **Training Data and Accuracy:** Training a machine learning model requires sufficient and representative data. Gathering a diverse dataset for your specific temperature-based fan speed control application might be challenging. The accuracy of your model heavily depends on the quality and quantity of training data.

Conclusion:

In conclusion, the development of a temperature-based fan speed controller using TinyML on Arduino has proven to be a successful and practical application of machine learning in the realm of embedded systems. This project aimed to create an intelligent and energy-efficient solution for maintaining optimal temperatures in various environments.