Model Optimization and Tuning Phase

Date	17th July 2024
Team ID	SWTID1720025517
Project Title	CodeXchange: An Ai-Powered Code Translator Tool Using Palm's Chat-Bison-001
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters
PaLm chat-bison-001	**Hyperparam1: Prompt Design** • **Description:** The prompts were carefully designed to guide the model in generating more accurate and relevant responses. `You are a highly skilled software developer with expertise in multiple programming languages. Translate the following {source_lang} code to {target_lang}. Ensure that the functionality remains the same and that you follow best practices and idioms of the {target_lang} language. I want you to give only the code no need any other explainations.\n\n {source_code}` • **Screenshot:** C++ code to Java code.

C++

Source Code

```
// Factorial of n = 1*2*3*...*n

#include <iostream>
using namespace std;

int factorial(int);

int main() {
  int n, result;

  cout << "Enter a non-negative number: ";
  cin >> n;

  result = factorial(n);
  cout << "Factorial of " << n << " = " << result;
  return 0;
}

int factorial(int n) {
  if (n > 1) {
    return n * factorial(n - 1);
  } else {
    return 1;
  }
}
```

Java

Sure, here is the Java code that corresponds to the C++ code you provided:

```
import java.util.Scanner;

public class Factorial {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a non-negative number: ");
        int n = sc.nextInt();
        System.out.println("Factorial of " + n + " = " + factorial(n));
    }

    public static int factorial(int n) {
        if (n == 0) {
            return 1;
        } else {
            return n * factorial(n - 1);
        }
    }
}
```

Transalation done successful...

- **Screenshot:** Java Code to Python.

Select Source Language

Java

Source Code

```
// Factorial of n = 1*2*3*...*n

import java.util.Scanner;

public class Factorial {

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter a non-negative number: ");
    int n = scanner.nextInt();

    int result = factorial(n);
    System.out.println("Factorial of " + n + " = " + result);
  }

  public static int factorial(int n) {
    if (n > 1) {
      return n * factorial(n - 1);
    } else {
      return 1;
    }
  }
}
```

Select Target Language

Python

```
# Factorial of n = 1*2*3*...*n

def factorial(n):
    if n > 1:
        return n * factorial(n - 1)
    else:
        return 1


def main():
    print("Enter a non-negative number: ")
    n = int(input())
    print("Factorial of {} = {}".format(n, factorial(n)))


if __name__ == "__main__":
    main()
```

Transalation done successful...

- **Screenshot:** Python code to C++.

Python

Source Code

```
# Factorial of n = 1*2*3*...*n

def factorial(n):
  if n > 1:
    return n * factorial(n - 1)
  else:
    return 1


def main():
  print("Enter a non-negative number: ")
  n = int(input())
  print("Factorial of {} = {}".format(n, factorial(n)))


if _name_ == "_main_":
  main()
```

C++

Here is the translation of the Python code to C++:

```
#include <iostream>

using namespace std;

int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}

int main() {
    cout << "Enter a non-negative number: ";
    int n;
    cin >> n;
    cout << "Factorial of " << n << " = " << factorial(n) << endl;
    return 0;
}
```

Transalation done successful...

Translate

5.2 Final Model Selection Justification (2 Marks):

Final Model	Reasoning
PaLm chat-bison-001	The PaLM Chat Bison model with the designed prompts and a **temperature setting of 0** was chosen as the final optimized model because it consistently produced accurate and relevant responses. The prompt design effectively guided the model to focus on key information, and the temperature setting balanced creativity and determinism, ensuring both precision and variability in the output.