WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Leveraging Online GPT Transformers for Sustainable IT Solutions

Varun K A[1], Dr. Kamalraj R[2]

[1]Department of CS&IT, JAIN(Deemed-To-Be-University) Bangalore, India rjvarun43@gmail.com
[2]Department of CS&IT, JAIN(Deemed-To-Be-University) Bangalore, India r.kamalraj@jainuniversity.ac.in
DOI: https://doi.org/10.55248/gengpi.5.0324.0723

A B S T R A C T

Software applications have assimilated into our daily life in a time when technical progress is relentlessly pursued. These programs operate our smartphones, fuel our enterprises, and support vital industrial and commercial systems. The energy consumption of these software programs has nonetheless become a serious worry as the globe gets more linked, both in terms of its effects on the economy and the environment. The information technology (IT) industry contributes significantly to the world's energy consumption, and as our reliance on software applications increases, there is an urgent need for more energy-efficient solutions. Software energy consumption affects more than just information technology; it also has significant effects on the environment, and the long-term viability of the digital age. The accompanying energy expenditures are a significant source of worry from a financial and environmental standpoint as software continues to develop and expand its role in daily life.

Keywords: Sustainability, Online GPT, Time complexity, Energy efficiency, Program, IT industry.

1. Introduction

The purpose of this study is to investigate methods and recommended procedures for increasing software programmes' energy efficiency. We strive to identify critical areas where energy waste can be reduced, ranging from reviewing code-level improvements to considering the more comprehensive system architecture and deployment tactics. By doing this, we hope to open the door for a more improved suggestion.

In this paper we are going to discuss about the programs suggestion given by online GPT and discussing what result it suggests whether it gives iterative program or recursion program. This study aims to address this problem by examining and by calculating the time and space complexity of those programs and finding a way how online GPT can improve itself by suggesting less time complexity so energy efficiency can be reduced in software applications.

Software applications use a lot of computational resources, which uses a lot of energy because of their numerous functionalities and capacities. The need for energy-efficient solutions is more urgent than ever as our reliance on software applications increases. This study aims to address this problem by examining the crucial relationship between temporal complexity and energy efficiency in software systems. In the context of software, temporal complexity refers to how well a programme uses computational resources, notably in terms of the execution time. One fundamental premise is that we can considerably improve software applications' energy efficiency by lowering their time complexity, which will result in cheaper operational costs and a smaller environmental impact.

Reducing energy consumption is very important nowadays because of the raise in environmental issues, pollutions and more energy consumptions in IT industry is more and it can be reduced by using less time complexity programs in software applications which will decrease run time of applications and uses less energy this can also reduce cost. The overall objective of this research paper is to improve how quickly a program runs. Programs with lower time complexity run faster, which can lead to significant energy savings. Also, programs with lower time complexity are often easier to understand and maintain, which can reduce the overall cost of software development and maintenance [2].

2. Sustainability in IT industry

The evolution of the modern world is being led by the information technology (IT) sector. It is a crucial component of our daily life since it drives our productivity, commerce, entertainment, and communication. The ideas of sustainability and environmental responsibility have taken front stage as the IT industry keeps growing at an unrelenting rate. Due to the industry's tremendous growth, significant concerns have been expressed about how it will affect society and the environment in the long run. The fast-paced innovation and technological development that define the information technology sector also result in significant energy and resource consumption. The energy use and environmental impact of IT operations are a growing concern due to the size of the data centers, the availability of cloud computing services, and the abundance of personal and business devices. These difficulties are made worse by the growing demand for digital services, which has increased energy use, electronic trash creation, and carbon emissions.

As a result of the importance of these issues, this study examines the intricate and dynamic sustainability landscape within the IT sector. It tries to answer the crucial query: How can the IT industry develop to become more sustainable, ethically conscious, and ecologically responsible? We will explore a variety of sustainability-related topics in addition to energy efficiency. It includes ethical supply chain considerations, responsible resource management, the minimization of electronic waste, and the wider societal effects of IT practices. This study is motivated by the idea that the IT sector has a special ability to both reduce its environmental effect and contribute to global change for the better. We will examine several dimensions of sustainability in the IT business using a multidisciplinary approach. This entails analyzing the relevance of the problem, figuring out the difficulties it presents, and coming up with creative fixes and best practices. We will also think about how changing to more sustainable IT practices would affect society and the economy.

2.1 Time complexity optimization:

The IT sector has played a crucial part in creating the modern world since it advances quickly in technological terms. However, this rapid advancement in technology also brings with it the urgent need to solve the escalating economic and environmental issues related to resource use and energy use. Because it uses a lot of energy, the IT industry has a big impact on the environment and the world's energy consumption. Sustainability has become a top priority due to the rise in energy consumption and carbon emissions caused by the rising usage of digital services, cloud computing, and data centers. This research provides a fresh strategy as we stand at a critical juncture: utilizing temporal complexity optimization within software programmers as a key remedy for minimizing the environment.

The concept of time complexity in software engineering describes how effectively algorithms and programmers use computer resources. We aim to reduce energy consumption, operational costs, and the carbon footprint of IT operations by optimizing time complexity. The basic idea behind the hypothesis is that we can improve the sustainability of the industry by making sure that software applications function more effectively and consume less resources. The goal is to look into the many facets of time complexity optimization as a long-term fix for the IT sector. It tries to investigate numerous approaches and methods for reaching this goal, such as algorithmic advancements, simplified data structures, and the use of parallel processing techniques. We strive to improve the overall software performance by reducing computational overhead.

3. Suggestion by online GPT

Online GPT will be asked with certain programs and taking into considerations that what type of program will be suggested whether loop or recursive program based on the program time complexity will be calculated and based on that study will be done whether online GPT's will help to transform the sustainability in IT.

Some of the example questions asked to online GPT are as follows:

3.1 The first question asked to online GPT is to suggest factorial program in java and did not mention any technique purposefully to find out which program it suggests and it suggested recursion technique which consumes more time to run a program and the pseudocode for that program is as follows:

```
function factorial(n)
  if n < 0
    return "Factorial is undefined for negative numbers"
  else if n == 0
    return 1
  else
    result = 1
    for i from 1 to n
      result = result * i
    return result
  end if
end function
```

The method for computing a non-negative integer's factorial using a recursive function is described in the accompanying pseudocode. It starts by defining the function calculate Factorial, whose first parameter is an integer n. This function determines whether n is equal to 0 or 1, and if it is, it returns 1 because the factorial of 0 and 1 is 1. Otherwise, it returns 0. The calculate Factorial function is called repeatedly if n is more than 1 with the parameter n - 1 and the result is multiplied by n.

Time estimation typically involves an action that can be carried out. The analysis correlates the amount of time needed to complete a step. Some people, for instance, count adding two numbers as one step. In some circumstances, the same supposition might not be considered. For instance, the time required

for a single addition cannot be thought of as a constant because it fluctuates with the number when there are potentially very big numbers involved in the computation. Run-time analysis is a theoretical presumption that calculates how much an algorithm's run time increases in relation to its input values. A program's execution time can range from a few seconds to many minutes, depending on how the algorithm is implemented [1].

Considering the execution time, which is called the time complexity of an Algorithm, it only considers the program step more specifically called as implementation steps.

In general, the steps other than declarative can be considered as implementation steps. Therefore, our concern is to count the number of implementation steps in the function.

So, counting the total implementation steps and the number of times they occur,

Step 1: n+1 step 2: n step 3: n step 4: 1

The time complexity of the function is given as f(n)=(n+1) +n+n+1=3n+2.

3.2 Followed by first program we mentioned to give factorial program using loop and it gave loop program it has been proved that loop runs faster than recursion because loops have built-in support in CPUs, whereas recursion is implemented using the generally slower function call / return mechanism and the pseudocode of the loop suggested is as follows.

```
def factorial(n):
  if n < 0:
    print("n must be a non-negative integer")
  elif n == 0:
    return 1
  else:
    return n * factorial (n - 1)
```

The first line of the pseudocode sets the initial values of the variable's "number" and "factorial" to 0 and 1, respectively. The user is then prompted to enter a non-negative integer. The program notifies the user that factorial is undefinable for negative integers if the entered number is negative. If the input, however, is not negative, it then uses a "while" loop. It begins the loop by setting the variable 'i' to 1 and keeps running the loop as long as 'i' is less than or equal to 'number.' The 'factorial' is changed at each loop iteration by multiplying it by 'i,' and 'i' is then increased by 1 after that. The user is shown both the original number and the calculated factorial after the loop is finished

Considering the execution time, which is called the time complexity of an Algorithm, it only considers the program step more specifically called as implementation steps.

In general, the steps other than declarative can be considered as implementation steps. Therefore, our concern is to count the number of implementation steps in the function.

So, counting the total implementation steps and the number of times they occur,

Step 1: n step 2:1 step 3: n-1

The time complexity for the above factorial function is f(n)=(n)+1+(n-1) =2n.

Calculating time complexity of recursion method:

Analyzing the quantity of fundamental operations (such as comparisons, assignments, etc.) carried out as a function of input size is necessary to determine the temporal complexity of a recursive technique. Understanding how frequently the recursive function is called and how work is distributed among recursive calls is essential to this research. A recurrence relation can typically be used to express the temporal complexity of a recursive technique. You divide the issue into smaller issues and quantify the temporal complexity in terms of the quantity and size of those issues. The closed-form solution that captures the entire time complexity is then obtained by solving the recurrence relation [1].

For the program suggested by online GPT the number of times the calculate Factorial function is used and how the input size decreases with each recursive iteration, for instance, in the situation of computing the factorial of a number using recursion. The formula for the temporal complexity is T(n) = T(n-1) + O (1), where T(n) is the amount of time needed to calculate the factorial of n. You discover that when you solve this recurrence relation, the time complexity is O(n), which denotes a linear time complexity in terms of the input size n.

4. Calculating time complexity of loop program

We normally count the number of fundamental operations that are carried out as a function of the input size to determine the temporal complexity of a loop approach. In "Big O" notation, the time complexity is stated as the upper constraint on the growth rate of the algorithm's execution time.

Find the loop or loops in the code first. The main causes of temporal complexity are frequently these loops. To ascertain how many activities are carried out during each iteration, examine the loop's source code. Calculations, comparisons, assignments, and function calls are a few examples of relevant variables. Next, based on the input size, decide how many iterations the loop(s) will run. Understanding how the input is related to the loop's termination condition may be necessary for this.

To determine the overall time complexity, add the number of operations each iteration to the total number of iterations. If you want an upper bound on the algorithm's growth rate, express this in terms of "Big O" notation, such as O(n) for linear time, O(n^2) for quadratic time, or O (log n) for logarithmic time. Eliminate constants and lower-order terms from the formula to make it simpler. This explains the relationship between the running time of the algorithm and the size of the input.

Table 1: Time complexity of the programs suggested by online GPT:

Number of runs (in milliseconds)	**Recursion method**	**Loop method**
Run 1	10ms	10ms
Run 2	12ms	12ms
Run 3	11ms	8ms
Run 4	12ms	8ms

The above table 1 results are obtained under the windows operating Environment and using the java language considering the variable number as 16. Same can be observed for different values of number [4].

5. Recursion or loop

Whenever a program is asked to online GPT it suggests recursion technique instead of loop method by this method what we suggest online GPT is to provide loop method whenever a program is asked why because a stack frame will be created by the programme in the background whenever a function is called. The stack frame is where variables for called functions are formed and the scope of variables for calling functions is retained. It is quite instructive to step through the execution of the set up and tear down of these stack frames to invoke variable scoping if you have taken a class on assembly language and have access to an assembly level debugger.

On the call stack of a process, these stack frames are produced. On Unix, the call stack's memory allocation is constrained and shared with the heap. The global and malloc'ed variables are kept on the heap. Every time a call is performed in recursion, a stack frame is constructed. The amount of free memory that is still available between the top of the call stack and the top of the heap (which grow in opposite directions) determines how many recursive calls you can perform. However, there are restrictions on the number of recursive calls you can make, and debugging this will be difficult if you don't understand the environment in which you are working because it can crash in unexpected circumstances.

However, we can overcome operating system restrictions and extend the lifespan of your programme by switching from recursion to a loop. Your programme may occasionally run more quickly if you omit the call frame setup and teardown involved with function calls.

Reason behind the problem:

Factorial calculations can be performed either recursively or loop-based (iteratively), depending on a variety of variables. Recursive solutions are frequently recommended because, especially for lower factorial calculations, they might result in simpler, more legible code. Since recursion closely resembles the mathematical concept of factorial, it is occasionally an obvious choice. Although, an iterative approach is typically more effective, especially when working with big input data. Recursion can be less memory-efficient because it depends on the call stack, and if the input is particularly huge, this could result in a stack overflow problem. Contrarily, loops frequently use less memory and provide greater performance for large inputs.

Modern interpreters and compilers can sometimes make tail-recursive functions—where the recursive call comes last in the function—just as efficient as loops. Although not always supported, this optimization may not result in a noticeable performance improvement. Also, because they don't rely on the call stack, loops offer better performance characteristics that are more predictable and controllable, which might be essential when working with huge inputs. Recursion or a loop should be chosen based on the needs of the programme as a whole, taking into account trade-offs such memory utilization, performance demands, and code simplicity.

6. Methodology

Online GPT might not be familiar with every programming language or might only have a basic understanding of some languages, which could limit its capacity to produce code in those languages. Its training data don't cover a variety of programming languages and their syntax from a training standpoint. As a result, Online GPT's knowledge of programming languages can be restricted to the patterns and connections it has discovered while studying the programming language data it was trained on. Programming languages are also continuously changing, with new releases and modifications being made all the time. Keeping up with these changes can be difficult, and it might not be possible to retrain Online GPT with modernized language syntax and functionality [3].

Explicit memory is absent from the Online GPT architecture. Online GPT lacks an explicit memory system like the one found in human brains, which enables long-term storage and retrieval of information. This means that it can only rely on the information that is given to it during each turn of the conversation and cannot save information from one conversation round to the next. Additionally, focusing Online GPT's fine-tuning on certain conversational data can aid in enhancing its capacity to produce responses in the context of a particular task or domain. Fine-tuning can, however, also lead to restrictions, such as a lack of generalization to novel conversational flows and a diminished capacity to manage more long-term memories. It is challenging for Online GPT to produce logical and consistent responses in the context of a conversation because the data used to train it is mostly unstructured and frequently lacks obvious conversational flow and context.[3]

7. Conclusion

We can conclude that Adopting software programmes with less time complexity in the IT sector is essential for developing sustainability as well as improving performance. These initiatives provide several advantages, including lower energy usage, which is consistent with the industry's emphasis on energy-efficient computing. They also maximize resource usage, minimizing the need for new technology and lessening the impact on the environment. Their capacity to scale allows for the expansion of the user base without the need for significant hardware changes, and the quick performance that results boosts user happiness and productivity. Additionally, less maintenance is needed for efficient software, which lowers operational costs and the accompanying environmental impact. This tactical decision supports environmental stewardship, the IT sector's dedication to minimizing ecological effect, and the advancement of a greener and more sustainable future.

In conclusion, Online GPT is an effective tool that may be used for program suggestion and more. In this regard, it serves as the ideal tool for collaborative intelligence and can or ought to be employed in conjunction with human developers. It is possible to explore novel ideas and advance software development by integrating online GPT into research and innovation strategies, which will increase the quantity and quality of work.

References

Sulov, V. (2016). Iteration vs recursion in introduction to programming classes: An empirical study. Cybernetics and Information Technologies, 16(4), 63-72.

Phalake, V. S., Joshi, S. D., Rade, K. A., Phalke, V. S., & Molawade, M. (2023). Optimization for achieving sustainability in low code development platform. International Journal on Interactive Design and Manufacturing (IJIDeM), 1-8.

Sadik, A., Ceravola, A., Joublin, F., & Patra, J. (2023). Analysis of ChatGPT on Source Code. arXiv preprint arXiv:2306.00597.

B. Swathi (2016). A Comparative Study and Analysis on the Performance of the Algorithm. International Journal of Computer Science and Mobile Computing, Vol.5 Issue.1, January- 2016, pg. 91-95.