Annals of Emerging Technologies in Computing (AETiC)
Vol. 7, No. 3, 2023

Research Article

GreenPy: Evaluating Application-Level Energy Efficiency in Python for Green Computing

Nurzihan Fatema Reya, Abtahi Ahmed, Tashfia Zaman and Md. Motaharul Islam*

United International University, Dhaka, Bangladesh
nreya201085@bscse.uiu.ac.bd; aahmed202247@bscse.uiu.ac.bd; tzaman201141@bscse.uiu.ac.bd;
motaharul@cse.uiu.ac.bd
*Correspondence: motaharul@cse.uiu.ac.bd

Received: 5th May 2023; Accepted: 27th June 2023; Published: 1st July 2023

Abstract: The increased use of software applications has resulted in a surge in energy demand, particularly in data centers and IT infrastructures. As global energy consumption is projected to surpass supply by 2030, the need to optimize energy consumption in programming has become imperative. Our study explores the energy efficiency of various coding patterns and techniques in Python, with the objective of guiding programmers to a more informed and energy-conscious coding practices. The research investigates the energy consumption of a comprehensive range of topics, including data initialization, access patterns, structures, string formatting, sorting algorithms, dynamic programming and performance comparisons between NumPy and Pandas, and personal computers versus cloud computing. The major findings of our research include the advantages of using efficient data structures, the benefits of dynamic programming in certain scenarios that saves up to 0.128J of energy, and the energy efficiency of NumPy over Pandas for numerical calculations. Additionally, the study also shows that assignment operator, sequential read, sequential write and string concatenation are 2.2 times, 1.05 times, 1.3 times and 1.01 times more energy-efficient choices, respectively, compared to their alternatives for data initialization, data access patterns, and string formatting. Our findings offer guidance for developers to optimize code for energy efficiency and inspire sustainable software development practices, contributing to a greener computing industry.

Keywords: ***Algorithmic Efficiency; Cloud; Comparison; Energy Consumption; Performance Analysis; Python***

1. Introduction

In the modern world, we are all inextricably surrounded by digital technology. Programming is on its way to becoming a day-to-day activity soon. Currently, where there is increasing demand for energy-efficient computing systems, we must address the energy consumption issues of computers, especially in the programming and developer sector. As of 2023, around 62 percent of the global population has access to the internet, where many use computers. If a single computer is turned on round the clock, it would release 341kg of carbon dioxide in a year [1]. The field of bringing energy efficiency in computer engineering has been growing recently, though it has been tough to pinpoint exact areas where efficiency can be improved due to the complex structure of computers that we have today. Researchers have sought to identify the problems [2, 3], such as distinct programming languages, to find out how they perform against one another in terms of energy efficiency [4-6]. The utilization of Cloud computing has brought about a significant change in the management and use of data centers in recent years [7]. Nevertheless, cloud data centers' energy consumption leads to elevated operating expenses and carbon dioxide (CO_2) discharge to the atmosphere [8].

In recent years, there have been numerous studies [11-23] that focused on energy efficiency in software development. These studies have investigated various aspects of programming languages, tools, and

Nurzihan Fatema Reya, Abtahi Ahmed, Tashfia Zaman and Md. Motaharul Islam, "GreenPy: Evaluating Application-Level Energy Efficiency in Python for Green Computing", *Annals of Emerging Technologies in Computing (AETiC)*, Print ISSN: 2516-0281, Online ISSN: 2516-029X, pp. 92-110, Vol. 7, No. 3, 1st July 2023, Published by International Association for Educators and Researchers (IAER), DOI: 10.33166/AETiC.2023.03.005, Available: http://aetic.theiaer.org/archive/v7/v7n3/p5.html.

techniques, as well as hardware and software components to optimize energy consumption. However, despite these efforts, no study has specifically focused on energy efficiency monitoring in Python, which is a popular and widely used programming language. The use of Python is prevalent in diverse aspects of computing including web development, data science, machine learning, and scientific computing [9]. According to a recent survey conducted by Stack Overflow, Python has become the most popular programming language, surpassing languages like Java, C, and C++. It is the 4th most used language in GitHub, and the 5th largest Stack Overflow community [10]. However, like any other programming language, Python has its own energy consumption issues that need to be addressed to improve energy efficiency in computing.

Various studies have been conducted in recent years to explore energy efficiency in software development. In [11], researchers investigated the energy efficiency of programs run in 27 different programming languages, revealing key misconceptions and correlations between memory usage, execution times, and energy consumption. EnSights, a tool developed in [12], aimed to increase the efficiency of Android Studio IDE for mobile applications using matrix methods and coefficient and correlation techniques. In [13], a comprehensive review has been done of energy-aware software engineering research, summarizing techniques and approaches for reducing energy consumption, including optimization of algorithms, data structures, and employing dynamic voltage and frequency scaling. However, these techniques have primarily focused on mobile applications, leaving a gap in energy efficiency research for other software applications. In a related study [14], researchers explored application-level energy management in Java, identifying strategies that resulted in significant improvements in energy consumption. Meanwhile, in [15], the energy consumption of two key blockchain algorithms, Merkel Tree (MT) root calculation and Proof of Work (PoW), was measured in Python using pyRAPL. Our paper aims to address the gap in the existing literature by focusing on Python, a popular and widely used programming language. The major contributions of this paper are summarized below:

1. We have identified specific programming practices that can be used to optimize energy consumption in Python, such as minimizing unnecessary computations and utilizing efficient data access and structure patterns.
2. We have developed a set of energy-efficient best practices and guidelines for Python programming, helping developers create more environmentally friendly and sustainable applications.
3. We have provided valuable insights into the energy consumption patterns and trends of Python, which can inform future research in this field and guide developers toward more energy-efficient coding practices.
4. We have addressed the energy consumption issues of Python and best practices through a series of case studies, providing quantitative evidence of the potential energy savings achieved by adopting our proposed methods.

2. Literature Review

In [11], the researchers have studied the energy efficiency of programs run in 27 different programming languages, using a performance-oriented source, Computer Language Benchmarks Game (CLBG)[1] and an educational source, Rosetta. They have brought to light some key misconceptions and correlations, as to if and how memory usage and fast execution times correlate with energy consumption.

In the previous year, a tool named 'EnSights' was proposed to make efficient mobile applications with increasing the efficiency of Android Studio IDE. They used matrix methods to make energy-efficient code and applied their 'EnSights' method to the previous code and observed the comparison of previous matrices of the code and currently prepared matrices with the coefficient and correlation techniques [12].

A related study, Eder *et al.* [13] has stated that energy consumption should be considered as a design constraint in addition to traditional constraints such as functionality, reliability, and maintainability. It is a literature review and survey of the existing research on energy-aware software engineering and provides an overview of the various techniques and approaches that can be implemented to decrease energy

[1] https://benchmarksgame-team.pages.debian.net/benchmarksgame/

consumption in software systems, such as optimizing algorithms and data structures, reducing computation and communication overheads, and employing dynamic voltage and frequency scaling. Eder *et al.* have made a well-defined framework to assess and mark energy consumptions of processes at different levels of a computer, such as the hardware as well as instruction set architecture, intermediate languages and source code. However, they have not based their work on a single programming language, or how a certain language could be improved.

Liu *et al.* [14] has contributed some work in the application-level energy management in Java language, where different strategies have been shown to bring noticeable results in energy consumption. It proposed a data-oriented approach to characterize the energy consumption of software applications. They used a combination of static and dynamic analysis techniques to extract energy-related metrics from the source code. The key areas of work include data access patterns, data representation strategies, data organization, data precision choices, and data I/O configurations. A unified attempt had also been made to manage these application-level strategies with the hardware to meet a sweet spot between energy efficiency and fast execution time. The results showed that their approach was effective in identifying energy inefficiencies in software applications.

Escobar *et al.* [15] used pyRAPL to measure the energy consumption of two algorithms - the Merkel Tree (MT) root calculation and the Proof of Work (PoW) algorithm - which are key in the blockchain. They implemented both the original and the optimized algorithm and found significant results. The optimized version was implemented after the use of the Energy Complexity Model and hashing, which helped reengineer the algorithms to get reduced energy complexity.

Garus[2] worked to build an extension that measures energy consumption among the different components of a computer both on the server side and the notebook side when programmers program in Jupyter Notebooks. They have conducted both internal (Running Average Power Limit, RAPL) and external measurements (MCP) which gave an insight into how hardware also plays a big role in energy consumption. However, the number of programs tested was small. More programs could have been tested in order to provide a more rigid conclusion about the viability of programming in different ways with Python.

As Abdulsalam *et al.* [16] and Fog [17] mentioned, there aren't a lot of easy-to-use and available tools to measure energy consumption, and programmers often rely on upgrades to hardware so that their programs consume less energy. The requirements of optimizing the software for speed or size conflict with the importance of structured and object-oriented programming, modularity, reusability, and systematization of the software development process.

The selected applications on which the experiment was conducted by Abdulsalam *et al.* [16] were Fast Fourier Transform (FFT), Linked List insertion/deletion (LL), and Quick Sort (QS). Quick Sort was the only application where Abdussalam *et al.* used Python. They compared C, C++ (using array), C++ using vector, Java, and Python. From the perspective of the compiler, the C and C++ implementations had optimization techniques applied to them, which led to better results. Additionally, they studied the impact of different implementation choices such as alloc, new, vector, and array on program energy efficiency. In all, based on their conclusion, C performed best when no optimization was used, whereas C++ came out on top with optimization techniques applied. Java performed badly mostly due to the running of the Java Virtual Machine, though in Java, for large data, linked list implementation performed better. Abdulsalam *et al.*, however, did not optimize the Python programs or apply different ways to implement the Python programs.

Fog [17] also considers C++ to be the fastest executable language due to the availability of good compilers and optimized function libraries. C++ also includes the low-level C language as a subset, giving access to low-level improvements. The majority of C++ compilers produce assembly language output, which can be used to assess how well a piece of code was optimized. However, Fog [17] also concluded that the best-performing programming language would be one with a mix of a compiled language, which would help in code optimization, and a high-level framework to ensure portability and ease of development.

Pereira *et al.* [18] measure the energy consumption of 10 programs in 27 programming languages, using The Computer Language Benchmarks Game. They also measure the execution time and memory usage of

[2] https://marcelgarus.dev/jupyter-energy

these programs and analyze the correlations between the use of memory, energy, and execution time. While these studies have made significant contributions to the field of energy efficiency in software development, they have some limitations. Even though Pereira *et al.* [18] have made a significant contribution in identifying which programming language to use based on some benchmark programs, they have not focused on how a specific language, such as Python, could be used in a different manner so that energy can be saved.

Bree *et al.* [19] has conducted an assessment on the impact of two popular design-level refactoring on energy consumption in the Java programming language. Specifically, they focused on the refactoring techniques of replacing Inheritance with Delegation and vice versa. The researchers assessed the energy consumption by running code snippets for both refactoring and measuring average power consumption and energy consumption. The study revealed that Inheritance proved to be more efficient than Delegation. It exhibited a 77% reduction in runtime and a 4% decrease in average power consumption when compared to Delegation. However, a significant limitation of the study was the experiments were conducted in an Interpreted mode, which does not accurately reflect real-life scenarios where Just-in-Time (JIT) enabled compilers are commonly utilized.

Pereira *et al.* [20] proposed novel and language-independent methodology called SPELL (SPectrum-based Energy Leak Localization) to identify energy inefficient sections in software. This technique, implemented within a dedicated tool, enables the simultaneous detection of abnormalities in energy consumption during program execution and the identification of faults in program execution. By adapting Spectrum-based Fault Localization (SFL) techniques, the researchers establish a correlation between energy consumption and the software's source code. They attribute varying degrees of responsibility for energy consumption to different components of the underlying system. To put their SPELL concept into practice, they have developed an analysis tool in Java and utilized Intel's RAPL for precise energy measurement. Moreover, an empirical study was conducted involving programmers to assess the effectiveness of the technique. Remarkably, programmers who followed SPELL recommendations achieved an average energy optimization of 43%, as demonstrated by the study's findings.

Rahaman *et al.* [21] have made a similar approach to how we have implemented our methodology. They used tools to measure certain parameters, and equations to calculate energy consumption. They implemented several programs and reshaped them after applying code transformations to optimize them. One crucial concept of theirs was that they integrated the requirement for energy efficient software in the SDLC model by including a new step called Energy Efficiency Analysis. After testing the transformed programs, they found that their model performed better than the agile method after implementing EE Analysis. However, the types of programs and the kind of transformation they used were not clear in the paper. The choice of programming language (Java, in this case) was also not justified, and there was discrepancy between the chosen language and the sample programs.

Mancebo *et al.* [22] developed a methodology where they tested a full software to evaluate and identify the areas of high energy consumption. They based their evaluation on maintainability of a software, which included number of lines used, comments and repetitive code, among other criteria. The comparison was made among different versions of a single software after certain tests were conducted on them, based on which they concluded, for every criterion, whether or not energy consumption changed while following a pattern. However, it is obvious that the results are not generalizable, as different software will have different changes in their versions, leading to different patterns, and also because the programming languages used in the software itself could be multiple. In Table 1, the summary of the literature review is given.

Table 1. Summary of the Literature Review

Authors	Study Description	Limitations	Method Adopted
Alvi *et al.* [12]	Proposed efficient applications with less wastage of battery. Matrix methods have been used to make energy-efficient code.	A single case study of mobile applications developed on the Android platform. No work to evaluate effectiveness in various contexts and compare performance.	The tool identifies the energy inefficiency to lower the software's energy consumption by up to 22% using code restructuring, loop unrolling, and caching.
Liu *et al.* [14]	A data-oriented approach to characterize the energy consumption of software. They used a combination	Only focused on Java language. Due to the measurement of energy consumption in fragments of code, the proposed	Combination of static and dynamic analysis techniques to observe energy-related metrics.

	of static and dynamic analysis techniques.	good habits of programmers are too general.	
Escobar *et al.* [15]	Applied algorithm re-engineering techniques to improve energy efficiency of blockchain without compromising system quality and security.	Only applicable for blockchain and not much focus on programming languages.	Used the Energy Complexity Model along with re-engineering hash techniques on Merkle Tree and Proof of Work - two key elements of blockchain to increase efficiency.
Eder *et al.* [13]	Highlight various techniques and approaches to reduce energy consumption in software systems.	Focuses on academic research rather than practical applications of energy-aware software engineering in the industry.	A comprehensive analysis of the current state of research on energy-aware software engineering.
Pereira *et al.* [11]	Compare 27 programming languages with respect to their efficiency and energy consumption and establish their rankings.	The various ways to produce the same output within a specific programming language were absent.	Well-defined algorithms were tested using Intel's RAPL and validated using the repository from Rosetta Code.
Garus[3]	Awareness of energy consumption in the Jupyter Notebooks. Analyze and compare the energy consumption of architecture.	Access to Wi-Fi could not be abandoned, which incurred the use of more energy.	Run benchmarks and record the energy consumption using multiple sources like the RAPL from software, and Microchip Power Monitor from the hardware.
Abdulsalam *et al.* [16]	Study the energy impact of the languages C, C++, Java, and Python based on the different implementations and optimizations.	The number of applications or algorithms tested is too small and has little focus on Python.	Intel Power Governor library measures the energy consumption of CPU and DRAM power when implementing a few algorithms.
Pereira *et al.* [18]	Uses benchmark problems of CBLG on programming languages, and measures the energy consumption using Intel's RAPL, which collects and analyzes resultant data on execution time and memory usage.	Experiments are conducted on a single machine and operating system. The results of the study may not be generalized to all types of programs.	Benchmark suite of programs provides guidance on which programming languages tend to be more energy efficient. For energy-constrained environments, there is a trade-off between execution time and energy consumption.
Bree *et al.* [19]	Assessed the energy consumption of two popular refactoring approaches - replacing Inheritance with Delegation and vice-versa.	Experiment conducted in interpreted mode, which does not reflect the real-life scenario.	Investigated the energy measurement of by using Watts Up Pro tool on a few modified programs and analysed the result.
Pereira *et al.* [20]	Developed an energy leak localization tool for source code, which is language and approach independent. Located energy leak by SPELL matrix construction.	Considered only energy consumption of CPU. Experiment conducted on same set of programs; the results may differ for other programs.	Associated energy consumption of different software components with a percentage of responsibility, that pinpoints where improvement is needed.
Rahaman *et al.* [21]	Integrated a new step in the SDLC relating to Energy Efficiency and investigated the improvements.	Discrepancy existed between the sample programs and lack of clarity on the code transformations.	Applied code transformations and compared the energy consumptions between two versions of several programs before and after the modified SDLC models.
Mancebo *et al.* [22]	Developed a framework that evaluates a software application on certain aspects in terms of energy efficiency.	Tested only a single application and multiple languages may have been used, so results may not be generalizable.	Tested different versions of a single application and analyzed the patterns relating to energy and maintainability of the application.

Table 2. Gap Analysis

Authors	Analyzation of Specific Language	Optimization	Energy Measurement	Diverse Programs
Alvi *et al.* [12]	No	No	Yes	No
Liu *et al.* [14]	Yes	Yes	Yes	Yes
Eder *et al.* [13]	No	Yes	No	No
Pereira *et al.* [11]	No	No	Yes	Yes
Garus[3]	Yes	No	Yes	No
Abdulsalam *et al.* [16]	No	Yes	Yes	No
Pereira *et al.* [18]	No	No	Yes	No
Bree *et al.* [19]	Yes	Yes	Yes	No
Pereira *et al.* [20]	No	Yes	Yes	No

[3] https://marcelgarus.dev/jupyter-energy

Rahaman *et al.* [21]	Yes	Yes	Yes	No
Mancebo *et al.* [22]	No	No	Yes	Yes

The main gap between other papers and our paper is the comparison of different ways to program within Python. What we observed mostly in the literature review is the findings on which language is more energy efficient, or just how energy can be measured in software programs. A major gap we found in the literature review is the lack of focus on Python and the measurement of energy consumptions among the different programs within Python. In our paper, we have chosen certain topics, similar to [14], [15] and [21], and measured the energy consumption among the different programs within each topic, after which we suggested which program is more energy efficient. The gap analyses based on the literature review is presented in Table 2.

3. Methodology

Although Python is a very inefficient language, it has become extremely popular among general programmers and researchers due to the extensive libraries available. Therefore, we will investigate the energy consumption of Python programs in PyCharm, an IDE used to run Python programs. To achieve our goal, the following methodology will be followed:

Step 1: Selection a set of Python programs representative of typical workloads that users might run on these platforms (i.e., read/write a pattern, Quick Sort, etc.). Comparisons between various data structures as well as Python functions will be made.

Step 2: Installation of Intel's Power Gadget, which is a power usage monitoring tool enabled for Intel Core i5 processors.

Step 3: Running each program with the energy consumption measured using Power Gadget. Each experiment will be repeated multiple times to obtain statistically significant results.

Step 4: Calculate each program's cumulative energy consumption and average power to identify significant differences. We will try to formulate the relationship between time duration and energy consumption.

Step 5: Analysis of the results to identify the factors that contribute to the differences in energy consumption. These factors might include the hardware configuration, the software environment, and the workload characteristics.

Step 6: Proposal of good programming practices that can help optimize programs' energy consumption. Examples of such practices include using efficient algorithms, choosing the best functions, and libraries for computations, etc.

Step 7: Discussion of the results' implications for users concerned about energy consumption.

The System Architecture in Figure 1 visualizes our methodology.

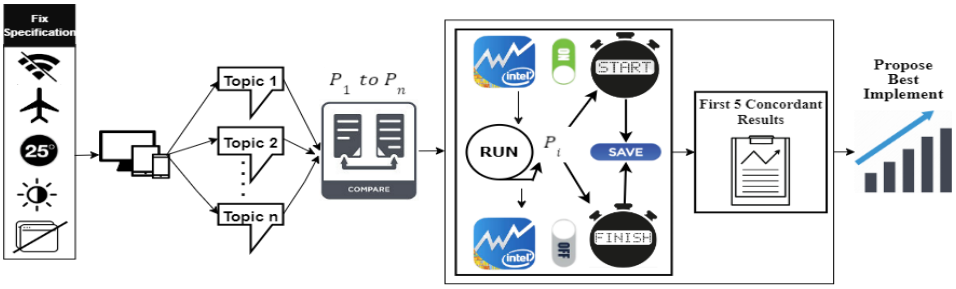

Figure 1. System Architecture

3.1. Energy Measurement Tool

The choice of a tool to measure energy consumption was an important one as our study depends on comparing energy consumption. Intel's Running Average Power Limit (RAPL) [10,14], seems to be the research standard. However, based on the literature review, researchers have used a variety of tools, such as data collector frameworks [1], Intel's Power Governor [16], EnSights [12], or other versions of RAPL such as jRAPL [14] and pyRAPL [15].

Due to the complexity of using RAPL, we have turned to Intel's Power Gadget, which is simple yet equally competent at measuring energy consumption in terms of the parameters we are looking for. It is a

software-based tool for tracking power usage that is compatible with Intel Core i5 processors, which is capable of measuring processor energy, package power, CPU utilization, GPU utilization, and DRAM power among other parameters. We require only the processor energy and power; therefore, Power Gadget is a sufficient tool for our study.

3.2. Experiment Environment

The laptop used for the experiments in our research was equipped with an Intel Core i5 Processor, a total of 4 cores, and 8 threads with an 8 MB Intel Smart Cache. The processor graphics on the laptop is Intel Iris Xe Graphics. The laptop has 8GB of DDR4 memory running at a speed of 3200MHz, consisting of $2x4$ GB modules. The maximum memory size supported is 64GB.

3.3. Algorithm

Algorithm 1 outlines the process for analyzing the energy consumption of different programs (P) running on various topics (T) for a given number of iterations (n). The algorithm works by iterating through each topic in T, then for each program in P, and subsequently running the program n times. During each run, it measures the start and end times and the power consumption. It then calculates the average power consumption and total energy consumption for each program run. After all iterations are completed for a given program, the algorithm stores the total energy consumption and average power consumption results. It then represents and analyzes the results, aiming to identify the factors that contribute to the energy consumption differences between different programs.

The Flowchart based on the algorithm is given below in Figure 2.

Algorithm 1. Energy Consumption Analysis of Python Programs

Input: Topics T, Programs P, n.
Output:

```
1. Procedure: CalcEnergy(T, P, n)
2.   for a topic in T do
3.     for a program in P do
4.       for i = 1 to n do
5.                 start_measurement()
6.                 start_time[i] = datetime.now()
7.                 execute_program(program)
8.                 end_time[i] = datetime.now()
9.                 stop_measurement()
10.                energy_file = store_energy_consumption()
11.                average_power_consumption[i] = calculate_average_power(energy_file, start_time,end_time)
12.                total_energy_consumption[i] = sum_energy_consumption(energy_file, start_time,end_time)
13.      end for
14.      program_results[program] = {total_energy_consumption, average_power_consumption}
15.    end for
16.    represent_and_analyze(program_results)
17.    propose_programming_practices()
18.  end for
19.  discuss_results_implications()
```

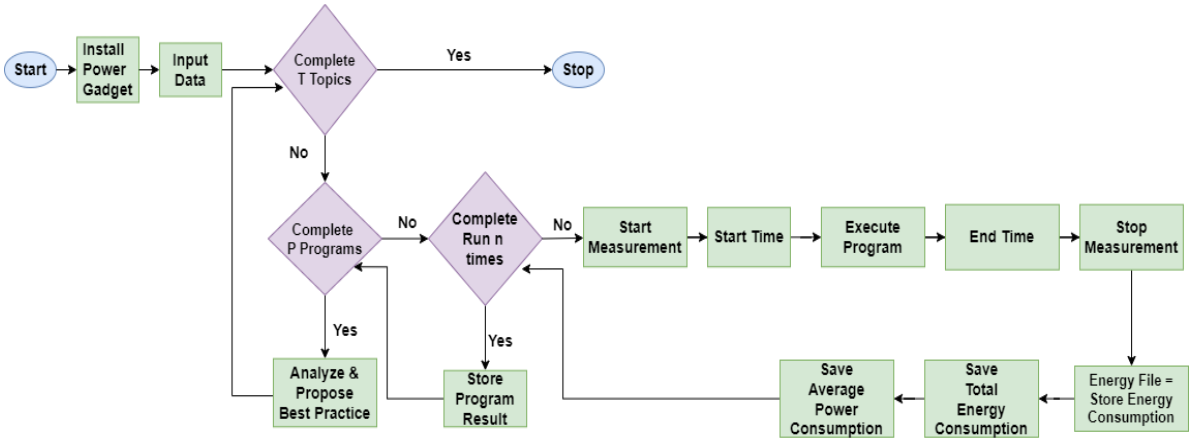

Figure 2. Flowchart

4. Implementation, Results, and Comparative Analysis

This section shows the results of the various implementations of the same program and provides a comparative analysis between those implementations.

4.1. Assignment Operator Vs Append()

A list is a heterogeneous built-in data structure in Python. It follows indexing, so it is ordered. To add elements to a list, we look at two ways- the Assignment operator ($\boldsymbol{list}[\boldsymbol{i}] = \boldsymbol{x}$) and the $\boldsymbol{list.append}()$ method. The assignment operator simply puts the desired value to that list index, while the append() method adds an item to the end of the list.

To compare the two implementations, we initialized two lists of size 20 million with all zeroes. We then measured the energy consumption for the time duration of the program ran. The code snippets for List in Python are given below in Figure 3a and Figure 3b.

```
alist=[0 for i in range(20000000)]
```

Figure 3a. Assignment Operator

```
alist=[]
for i in range(20000000):
  alist.append(0)
```

Figure 3b. List Append Method

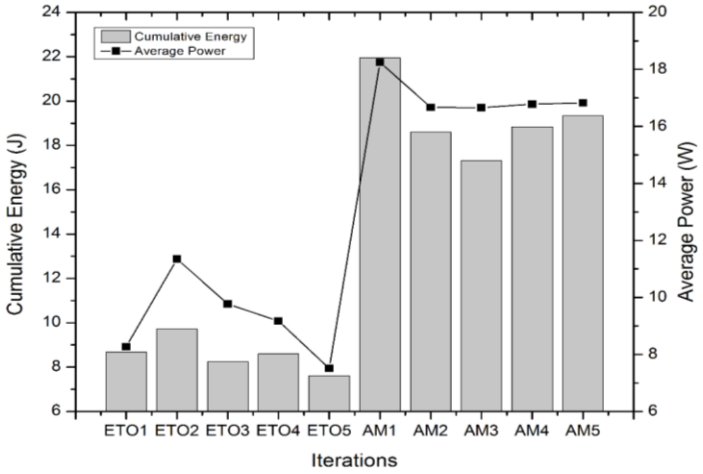

Figure 4. Assignment Operator Vs Append Method Graph

Figure 4 shows the graph for the energy consumption of the two implementations. We measured the cumulative processor energy and the average processor power. From the graph, we can clearly observe that the energy consumption is less (2.2 times more energy efficient) when the assignment operator (with prefix ETO) is used, and higher for the append() method (with prefix AM). This is expected, as the call to the append() method pushes an instance of a frame to its call stack and pops it when the function returns, which costs more memory and time, whereas the assignment operator has no such overheads.

4.2. Sequentially Vs Random Read and Write

We compared the energy consumption between sequential read and write versus random read and write. To conduct the reading part of the experiment, we initialized a list of size 20 million all with 0s, and then printed the list in a loop with sequential increments in indices versus random indices. For writing, the same process was followed, except that we initialized the list with 1s, then overwrote the indices with 0s.

Figures 5a, 5b, 5c, and 5d show the sequential versus random access code segments for sequential and random read and write respectively, and Figure 6a and Figure 6b show the graphs for read and write respectively.

From the read graph in Figure 6a, it is observed that random read (prefix RR) takes more energy and less power compared to sequential read (prefix SR). On average, the random read takes 1.05 times more energy. From our measurements, or by applying basic physics to the reader, it is observed that random reading takes more time to finish the program. This is clearly because of the cache locality. Accessing the list sequentially can keep a lot of future index addresses in the cache which would make more cache hits and reduce the time taken. Random access has a much greater chance to incur cache misses.

```
alist = [0 for i in range(20000000)]

for i in range(20000000):
  x=random.randrange(0,20000000,1)
  print(alist[i])
```

Figure 5a: Sequential Read

```
alist = [0 for i in range(20000000)]

for i in range(20000000):
  x=random.randrange(0,20000000,1)
  print(alist[x])
```

Figure 5b: Random Read

```
for i in range(20000000):
  x=random.randrange(0,20000000,1)
  alist[i]=0
```

Figure 5a. Sequential Read

```
for i in range(20000000):
  x=random.randrange(0,20000000,1)
  alist[x]=0
```

Figure 5b. Random Read

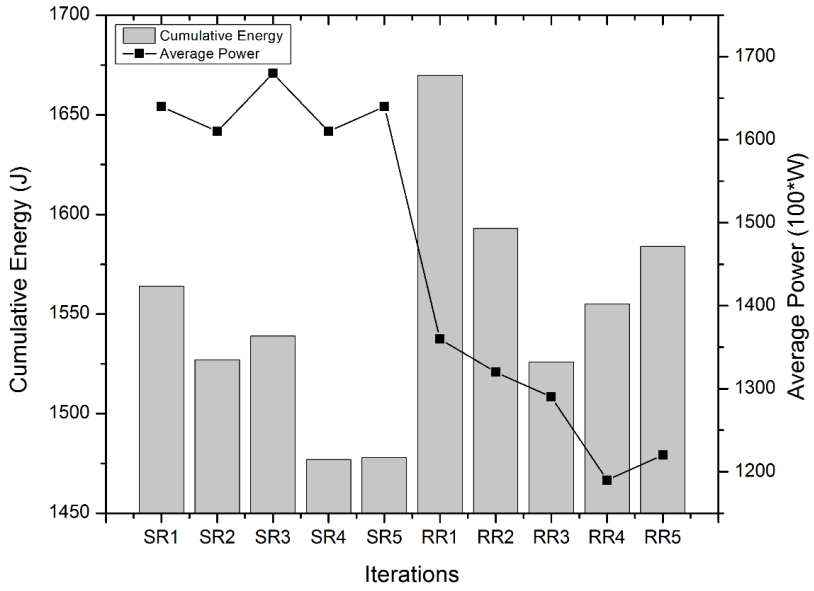

Figure 6a. Sequential Vs Random Read

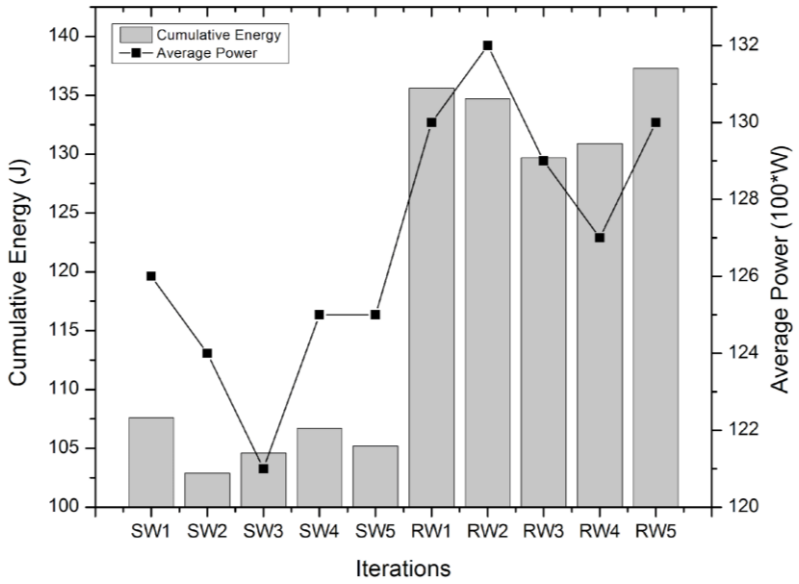

Figure 6b. Sequential Vs Random Write

For sequential write (prefix SW) and random write (prefix RW), the graphs are like read graphs. The energy consumption of random writing is higher (around 1.3 times) and takes slightly more power than sequential writing, the reason being the same as before, which is cache locality. By our calculations, we have seen that random writing takes more time as well than sequential writing.

4.3. List Vs NumPy Array Vs Dictionary

A list in Python is a heterogeneous data structure. It is one-dimensional, and the data inside can be modified. It is quite a flexible data structure of Python that can handle simple tasks very well, however, it is unable to perform some complicated operations, such as element-wise operations.

NumPy Array is an extension in Python that allows for scientific computing in Python. NumPy Array is also a homogeneous data structure and is mostly used for numerical calculations. It can be multidimensional, although then the order of the matrix must be consistent. It has a wide variety of features and methods that can be used to easily implement complex algorithms, such as dot product and element-wise operations.

Dictionary in Python is another heterogeneous and changeable data structure. It, however, does not necessarily follow indexing, but follow the key-value pair to store values.

To compare the three data structures, we have implemented algorithms to create two of each data structures, initialize them with all 1s and all 2s respectively, and store the addition of the values in another the variable of the same type. All the data structures were accessed sequentially. Figure 7a, Figure 7b, and Figure 7c below show the data structure comparison code segments for the list, NumPy array, and dictionary respectively, and Figure 8 shows the graph.

```
alist = [1 for i in range(20000000)]
blist = [2 for i in range(20000000)]
clist = [alist[i]+blist[i] for i in range(20000000)]

print(clist)
```

Figure 7a. List

```
anumpyarr= np.array([1 for i in range(20000000)])
bnumpyarr= np.array([2 for i in range(20000000)])
cnumpyarr= np.array([anumpyarr[i]+bnumpyarr[i] for i in range(20000000)])

print(cnumpyarr)
```

Figure 7b. NumPy Array

```
adict = {i:1 for i in range(20000000)}
bdict = {i:2 for i in range(20000000)}
cdict = {i:adict[i]+bdict[i] for i in range(20000000)}

print(cdict)
```

Figure 7c. Dictionary

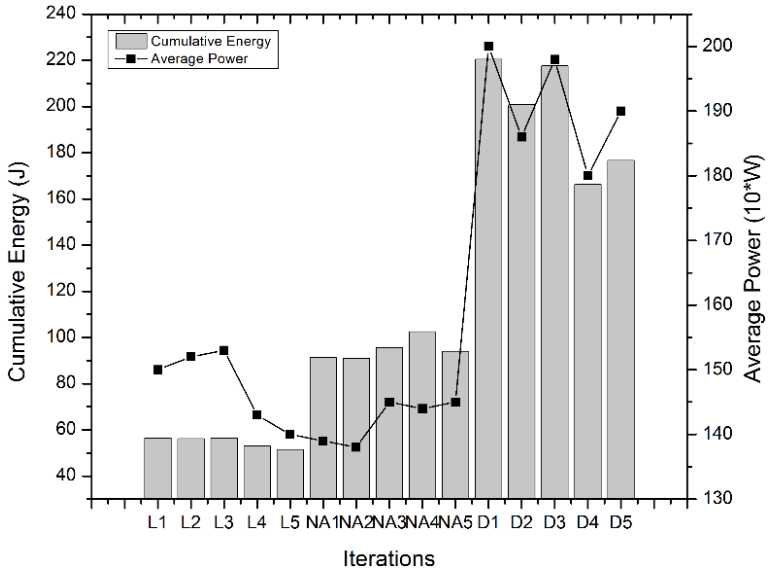

Figure 8. List Vs NumPy Vs Dictionary Graph

From the Figure 8, it is observed that List performs the best with the lowest energy consumption (around 1.73 times better). The average CPU power is similar for both list and NumPy array, which is

surprising since the NumPy array is very efficient in terms of memory and time complexity. The discrepancy in energy consumption for the NumPy array could be the cause of the NumPy array method that creates the array in the first place, as we know function calls are stacked and cost energy and time. Without the methods, the NumPy array is expected to be faster and more energy efficient. More research is needed to verify this.

It is clear from the Figure 8 that dictionary performed the worst in terms of sequential access and arithmetic operations. The same function calls to create the dictionaries can be blamed for the poor performance, but since a dictionary does not follow the typical indexing, the search for the key and retrieving the value must also have contributed to the time duration of the program.

We can conclude that the list is very light-weight when it comes to creation and simple operations, however, the NumPy array is expected to be more efficient and capable of solving complex calculations much faster, whereas the dictionary also has its uses in specific applications where key-value pair is necessary but fails to perform well in simple creation and arithmetic operations.

4.4. Different Ways to Print a String with Variables

In this section, we have compared the four different ways to print a string with a variable. The f-string method is the most used [10] implementation to print strings with variables, however, the format() method and ',' methods are also commonly used. f-string uses curly braces to print individual variables. format() does the same thing but breaks down the print statement into two distinct parts- string and variable(s). However, format() is soon losing popularity amid the arrival of f-strings in Python 3.6. The concatenation (+) method and comma (',') methods are similar in structure as they can mix strings and variables in the same line, with the difference being that the comma method automatically adds a space before and after the variable. The Figure 9a, 9b, 9c, and 9d show the code segments. To compare the implementations, we have simply created a string variable and printed it 10 million times. Figures 9a, 9b, 9c, and 9d show the print string with variable code segments, and Figure 10 shows the graph.

```
for i in range(10000000):
  print(f"Hello {flist1} World!")
```

Figure 9a. f-String

```
for i in range(10000000):
  print("Hello {} World!".format(flist2))
```

Figure 9b. format()

```
for i in range(10000000):
  print("Hello "+flist3+" World!")
```

Figure 9c. String Concatenate

```
for i in range(10000000):
  print("Hello ",flist4," World!")
```

Figure 9d. Comma Method

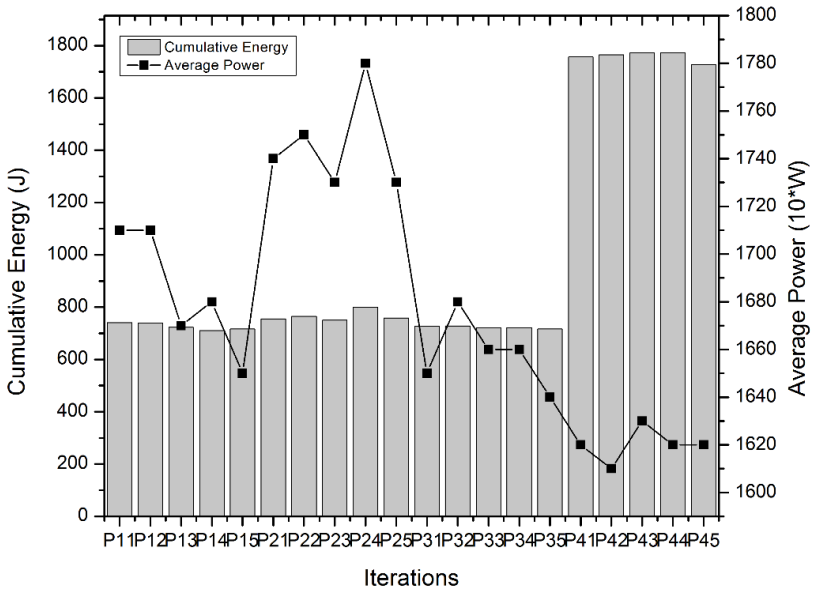

Figure 10. Different Prints Graph

From the Figure 10, it is observed that f-string (prefix P1), format() (prefix P2), and concatenation (prefix P3) all consume quite similar energy and power. format() costs a bit more energy, which may be due to the function call on the stack. String concatenation consumes the lowest energy (1.01 times more energy efficient than the next best implementation, f-strings), though the change is negligible. The most surprising result came from the comma (',') implementation (prefix P4). It took the most time to run and hence the most energy. One reason could be that this implementation itself adds spaces before and after the variables, which takes time.

From this experiment, we can conclude that string concatenation is the most energy-efficient, although it adds a layer of work to be done by the programmers in terms of adding spaces manually.

4.5. Quick Vs Merge Sort

In this section, we have compared Quick sort and Merge sort. These are two popular sorting algorithms that use the divide-and-conquer approach. Quick Sort has an average time complexity of $\boldsymbol{O(nlogn)}$ and a worst-case of $\boldsymbol{O(n^2)}$, while Merge Sort has a worst-case time complexity of $\boldsymbol{O(nlogn)}$. We performed these two sorting algorithms on a file containing 10 million integers.

The standard algorithms for the Quick Sort and Merge Sort algorithms have been used. Figure 11 shows the graph.

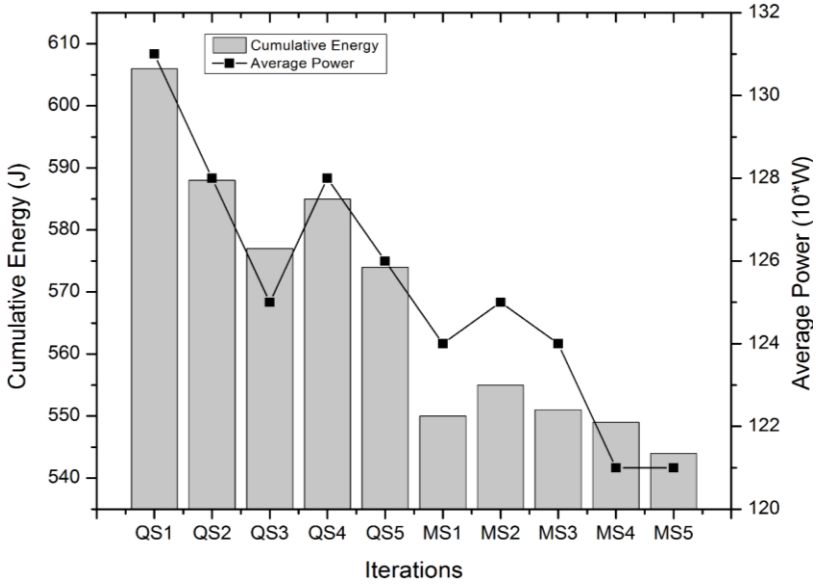

Figure 11. Quick Sort Vs Merge Sort

As we can see from the Figure 11, both sorting algorithms performed similarly in terms of energy consumption and power. This is expected since the average time complexity for both algorithms are $\boldsymbol{O(nlogn)}$. However, Merge Sort took slightly less energy. This could be because we know that Quick Sort doesn't perform well on large datasets, especially when the worst case applies. This very well could be our case, which helped Merge Sort perform slightly better (0.96 times less energy) than Quick Sort, even though Merge Sort had a poorer cache locality than Quick Sort. When it comes to energy consumption analysis, both quick sort and merge sort can be evaluated based on the number of operations they perform. The number of operations, in turn, is related to the amount of energy consumed by the algorithms. The worst-case scenario is when the pivot chosen is either the array's smallest or largest element, creating an array of size n-1 and an array of size 0 respectively. In this case, quick sort becomes inefficient and can result in higher energy consumption. On the other hand, merge sort always has a time complexity of $\boldsymbol{O(nlogn)}$ regardless of the input, making it a more predictable algorithm in terms of energy consumption.

Additionally, because it maintains the relative order of like elements in the input array, it is a stable sorting algorithm.

4.6. Fibonacci Using Loop Vs Dynamic Programming

Based on the experiment's findings, it can be concluded that dynamic programming using tabulation is a better approach in terms of energy consumption, as it consumes less energy compared to using a loop. On average, dynamic programming using tabulation consumed 0.24J of energy, while using a loop consumed 0.368J of energy. However, using loop is faster than dynamic programming using tabulation, as it takes only 0.0085 seconds to execute the program, while dynamic programming using tabulation takes 0.017 seconds. However, looped implementation always has four operations every iteration, compared to the single addition and assignment in tabulation, which takes more energy, even though tabulation uses more space. If energy consumption is a crucial factor, then dynamic programming using tabulation is recommended. On the other hand, if time efficiency is more important, then dynamic programming using a loop can be a better choice. We have also implemented Fibonacci using plain recursion and memorization and found that they required much more time and energy. It is obvious since they incur a lot of function calls building up in the stack. Additionally, we found that these two implementations were unable to calculate values of N that were higher, further limiting their usability and efficiency. Therefore, we have chosen not to include the findings of these two approaches in our analysis. Figures 12a and 12b show the Fibonacci code segments and Figure 13 shows the graph.

```
def Fib(n):
  f[0] = 0
  f[1] = 1

  for i in range(2, n+1):
    f[i] = f[i-1] + f[i-2]
  return f[n]

n = 20000
print(Fib(n))
```

Figure 12a. Tabulation

```
def Fib(n):
    a = 0
    b = 1
    if n == 1:
        return a
    elif n == 2:
        return b
    else:
        for i in range(3,n+1):
            c = a + b
            a = b
            b = c
        return b

n = 20000
print(Fib(n))
```

Figure 12b. Loop

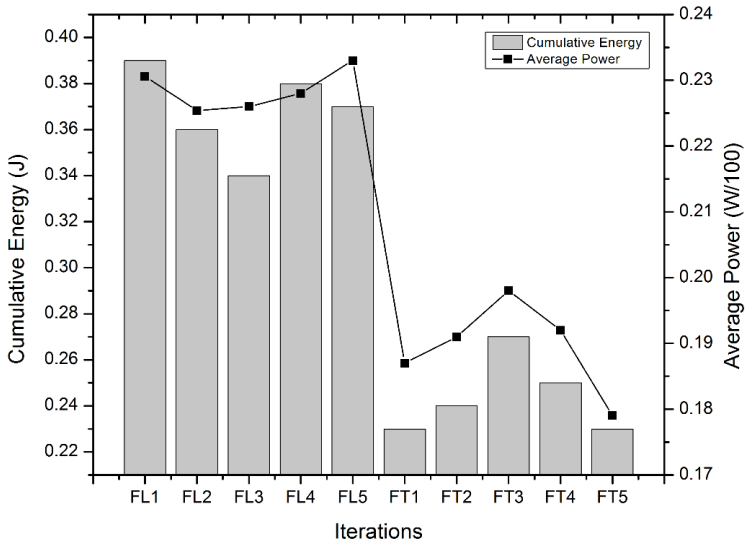

Figure 13. Loop Vs Tabulation Graph

4.7. NumPy Vs Panda

We have conducted a comparative analysis of the performance of two different Python libraries, NumPy and Pandas, which are commonly used for data analysis, machine learning, or scientific computing.

This experiment involves benchmarking the same set of tasks across the two libraries and comparing their runtime and memory usage. Such an analysis could provide valuable insights for researchers and practitioners in these fields, helping them to make informed decisions about which library to use for a given task, and how to optimize their code for maximum performance.

4.7.1. Mean and Standard Deviation Calculation

After analyzing the performance of NumPy and Pandas for calculating mean and standard deviation on a dataset of 10 million random numbers, we found that NumPy outperforms Pandas in terms of computation time. On average, NumPy takes 0.062 seconds to compute the mean and standard deviation, while Pandas takes 0.14 seconds. In terms of energy consumption, NumPy is also more efficient than Pandas. This is because NumPy is optimized for numerical calculations and is written in C, while Pandas is built on top of NumPy and uses more memory and CPU resources for data manipulation. Therefore, for large datasets and numerical calculations, it is recommended to use NumPy instead of Pandas to achieve better performance and energy efficiency. However, if the dataset requires complex data manipulation and analysis, Pandas may still be a better choice due to its more powerful data processing capabilities.

Figures 14a and 14b show the mean standard deviation code segments and Figure 15 shows the graph.

```
start_time_np = datetime.now()
mean_np = np.mean(data)
std_np = np.std(data)
end_time_np = datetime.now()
```

Figure 14a. NumPy Mean Standard Deviation

```
start_time_pandas = datetime.now()
mean_pd = df.mean()[0]
std_pd = df.std()[0]
end_time_pandas = datetime.now()
```

Figure 14b. Pandas Mean Standard Deviation

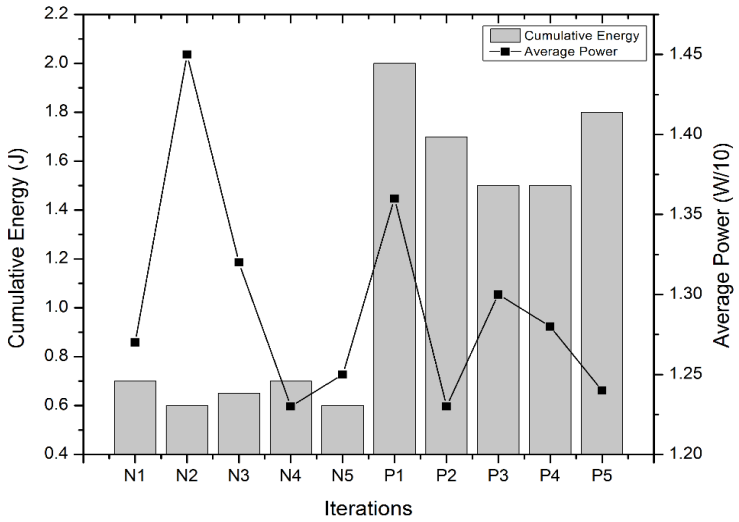

Figure 15. NumPy Vs Pandas Mean Standard Deviation Graph

4.7.2. Data Cleaning and Manipulation

In this experiment, we have used two libraries, NumPy and Pandas for cleaning, i.e., detecting and removing duplicates, filling missing values in a large dataset, containing 10 million rows and 5 columns. In the context of cleaning and manipulating large datasets, both Pandas and NumPy are commonly used libraries in Python. From the graph, it is observed that NumPy performs better than Pandas in the context of energy consumption for this program. In terms of performance, the analysis shows that NumPy outperforms Pandas for this task. For cleaning and manipulating a dataset of 10000000 rows and 5 columns, NumPy takes an average of 7.62 seconds while Pandas takes 12.42 seconds. However, for a smaller amount of data, for example, 10000 rows and 5 columns, Pandas performs better with a runtime of 0.0046 seconds compared to NumPy's 0.0052 seconds. The difference in performance is due to the fact that NumPy is optimized for numerical computations and is implemented in C, whereas Pandas is built on top of NumPy and is implemented in Python. Additionally, NumPy uses vectorization to perform operations on entire arrays rather than looping over individual elements, which makes it more efficient for large datasets.

Figures 16a and 16b show the data cleaning and manipulation code segments and Figure 17 shows the graph.

```
np_df = np.random.rand(10000000, 5)
np_df[np.isnan(np_df)] = 0
start_time = datetime.now()
np_df = np.unique(np_df, axis=0)
np_df = np.insert(np_df, 0, [1, 2, 3, 4, 5], axis=0)
np_df = np_df.astype('float64')
end_time = datetime.now()
```

Figure 16a. NumPy Data Cleaning and Manipulation

```
df = pd.DataFrame(np.random.rand(10000000, 5))
df.iloc[100:150, 0] = np.nan
df.iloc[5000:5500, 1] = np.nan
df.iloc[7500:7550, 2] = np.nan
df.iloc[10000:10100, 3] = np.nan
df = pd.concat([df, df.iloc[5000:5500]], axis=0)

start_time = datetime.now()
df.drop_duplicates(inplace=True)
df.fillna(method='ffill', inplace=True)
df.columns = ['A', 'B', 'C', 'D', 'E']
end_time = datetime.now()
```

Figure 16b. Pandas Data Cleaning and Manipulation

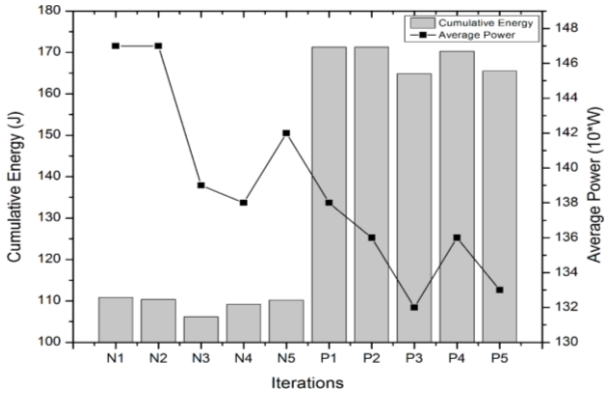

Figure 17. NumPy Vs Pandas Data Cleaning and Manipulation

4.7.3. Sorting Dataset

Based on the results, both Pandas and NumPy perform similarly, 1.48 seconds in terms of sorting a large dataset. Pandas uses the *sort_values*() method to sort the Data Frame based on a specified column, while NumPy uses the *argsort*() method to obtain the indices that would sort the array along a specified axis, and then uses those indices to sort the array. In terms of technical aspects, NumPy's *argsort*() method is faster compared to Pandas' *sort_values*() method due to its use of an optimized C implementation. Additionally, NumPy's sorting algorithm is more memory-efficient since it doesn't create a copy of the original array, whereas Pandas creates a copy of the Data Frame to perform the sorting operation. This can be especially important for very large datasets where memory usage is a concern. Regarding energy consumption, NumPy consumes less energy compared to Pandas since it uses a more memory-efficient algorithm.

Figures 18a and 18b show the data cleaning and manipulation code segments and Figure 19 shows the graph.

```
np.random.seed(0)
data = np.random.randint(0, 1000, size=(10000000, 10))

start_time = datetime.now()
data_sorted = data[data[:, 0].argsort()]
end_time = datetime.now()
```

Figure 18a. NumPy Sorting Dataset

```
np.random.seed(0)
data = np.random.randint(0, 1000, size=(10000000, 10))
df = pd.DataFrame(data, columns=['col'+str(i) for i in range(10)])
start_time = datetime.now()
df.sort_values(by='col0', inplace=True)
end_time = datetime.now()
```

Figure 18b. Pandas Sorting Dataset

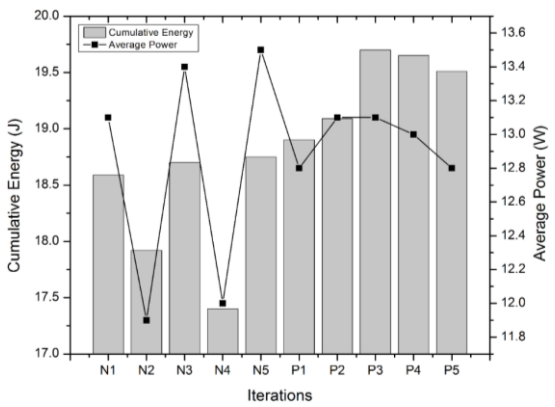

Figure 19. NumPy Vs Pandas Sorting Dataset

4.8. Personal Computer Vs Cloud Computing

In this section, we compared the energy consumption and time required for running logistic regression on a personal computer using PyCharm and on Google Colaboratory, a cloud computing platform. The dataset used was the iris dataset which was split into training, validation, and test sets. The logistic regression algorithm was implemented using Python libraries such as NumPy and random. The training and validation sets were used to optimize the weights of the logistic regression model. The accuracy of the model was then evaluated using the test set.

The results showed that PyCharm took 0.0112 minutes and 0.603 watt-minute energy consumption, while Google Colaboratory took 0.0184 minutes and 1.09 watt-minute energy consumption on average. The energy consumption was calculated by using Green algorithms tool[4] [23]. In Table 3, the result of the energy consumption is shown.

Table 3. Personal Device Vs Cloud Energy Comparison

#	PyCharm Time (Minutes)	Energy (Watt-minute)	Colaboratory Time (Minutes)	Energy (Watt-minute)
1	0.013	7.02E-01	0.018	1.07E+00
2	0.012	6.48E-01	0.020	1.19E+00
3	0.011	5.92E-01	0.017	1.01E+00
4	0.010	5.38E-01	0.021	1.25E+00
5	0.014	7.56E-01	0.016	9.54E-01

In terms of energy consumption and time, our findings show that PyCharm on a personal computer had a lower energy consumption and required less time compared to Google Colaboratory, represented in Figure 20. On average, PyCharm (represented by PC) takes only 0.012 seconds and consumes 0.647 Watt/minute, while Google Colaboratory (represented by GC) takes 0.0184 seconds and consumes 1.09 Watt/minute. This means that Google Colaboratory takes 1.6 times longer than PyCharm and consumes 1.8 times more energy than PyCharm. It is important to note that the free version of Google Colaboratory was used, which may have contributed to higher energy consumption and longer running time. However, in the context of global energy consumption in programming and software development, cloud computing has the potential to be more energy-efficient compared to personal computers. This is because cloud providers can leverage economies of scale and use more energy-efficient hardware, such as specialized processors and servers, compared to individual personal computers. Additionally, cloud providers can also optimize their data centers for energy efficiency, such as using renewable energy sources and improving cooling systems. Along with that, the energy consumption of cloud computing also depends on factors such as the size of the workload, the type of application, and the location of the data center. It is important to note that this study only compared the energy consumption and time for logistic regression and may not be representative of all programming and software development tasks.

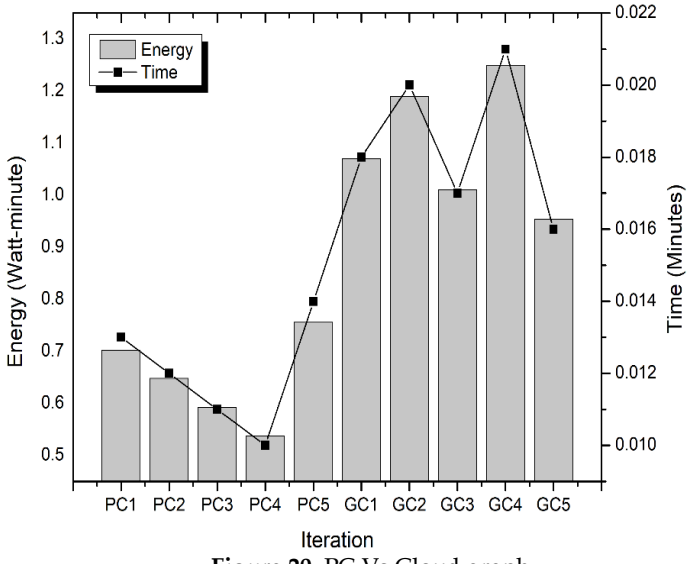

Figure 20. PC Vs Cloud graph

[4] http://calculator.green-algorithms.org/

Based on the results obtained from the experiments conducted on seven different topics related to energy consumption in Python, we can conclude that the methodology used was effective in achieving the research objectives. The experiments were conducted using various Python functions, data initialization strategies, data access patterns, various data structures, string formatting, and Python libraries for data analysis and visualization. We also conducted an experiment to compare energy consumption between cloud computing and personal computer usage, in order to determine which approach is more efficient. The results obtained provided valuable insights into energy consumption patterns and trends and can be used to guide future research in this field and better coding practices. Overall, the methodology used in this study was robust and reliable and can be replicated and adapted for further studies on energy consumption in Python or other related fields.

5. Limitations and Future Work

Like any typical experiment, ours also has some limitations. The first comes in the use of our tool to measure energy consumption. Power Gadget is a lightweight tool. However, it didn't give us the DRAM Power in our device which would have helped us make a more reliable comparison. In our experiment, we used only one device to conduct the measurements. Multiple devices should be used to calculate the average and arrive at a conclusion. The limitations of our work also contain focusing on the energy consumption of Python programs, which might not be representative of the other programming languages. The lack of space-complexity calculation in our formula may also affect our results. Additionally, we have only considered a limited set of programs and workloads, which may not generalize to all the software applications.

Researchers can bring space complexity into the formula for future work and see how it affects energy efficiency. We also found that programming in the cloud uses fewer resources on the programmer's end, however, how much energy it consumes on the server side of the cloud can be a research area, which would properly conclude the better habit. Our first aim in the future would be to implement all the experiments using a renowned energy measurement tool such as RAPL. Along with that, some sort of hardware tool involvement in our experiment would add to our reliability and transparency. Our work initially involved measuring the energy discrepancies between different operating systems. Due to the processor-dependent property of Power Gadget, we were unable to perform the experiments on another operating system such as MacOS or Linux, as common devices supporting those operating systems do not have Intel microprocessors. We aim to continue our experiment to measure the impact of different operating systems on the same algorithms in terms of energy consumption. While working on the list vs NumPy array vs dictionary experiment, we explored a lot more features that could be compared and used to optimize algorithms. We aim to further delve into the depth of these data structures with possibly more of them to find out which data structure and its features would perform best in which scenarios. We may also explore other languages that have been identified in our literature review, such as C, C++ and Java. Finally, we would also like to identify some sort of relationship between time complexity, space complexity, and energy consumption to better propose smart programming habits and ranking of certain algorithms.

6. Conclusion

The demand for energy-efficient computing systems has never been more significant than now, with the increasing use of software applications in various sectors. In this paper, we have proposed a comparative study of the energy consumption of different implementations of the same problem in the Python programming language, using the Power Gadget tool. Our study has revealed that efficient programming practices can significantly decrease the energy use of software programs. We have proposed good programming practices that can be adopted by programmers to reduce energy loss in software applications. The novelty of our research work lies in the fact that we have compared the energy consumption of different aspects of the Python programming language. Our study has also contributed to the field of energy-efficient development by proposing energy-efficient coding practices. Our study has also highlighted the potential benefits of cloud computing in reducing energy consumption, especially in personal devices. In conclusion, our study has demonstrated the importance of energy-efficient software development in the context of the increasing demand for energy and the growing use of programming. The

proposed techniques can help programmers and general users to optimize energy consumption and contribute to a more efficient approach to programming.

References

[1] Paolo Ciancarini, Shokhista Ergasheva, Zamira Kholmatova, Artem Kruglov, Giancarlo Succi *et al.*, "Analysis of energy consumption of software development process entities", *Electronics*, EISSN: 2079-9292, p. 1, Vol. 9, No. 10, 14th October 2020, Published by Multidisciplinary Digital Publishing Institute (MDPI), DOI:10.3390/electronics9101678, Available: https://www.mdpi.com/2079-9292/9/10/1678.

[2] Luis Cruz and Rui Abreu, "Performance-based guidelines for energy efficient mobile applications", in *Proceedings of the IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 22-23 May 2017, Buenos Aires, Argentina, E-ISBN: 978-1-5386-2669-6, Print on Demand (PoD) ISBN: 978-1-5386-2670-2, DOI: 10.1109/MOBILESoft.2017.19, pp. 46-57, Published by Institute of Electrical and Electronics Engineers (IEEE), Available: https://ieeexplore.ieee.org/abstract/document/7972717.

[3] Fangwei Ding, Feng Xia, Wei Zhang, Xuhai Zhao and Chengchuan Ma, "Monitoring Energy Consumption of Smartphones", in *Proceedings of the International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, 19-22 October 2011, Dalian, China, Print ISBN: 978-1-4577-1976-9, DOI: 10.1109/iThings/CPSCom.2011.122, pp. 610–613, Published by Institute of Electrical and Electronics Engineers (IEEE), Available: https://ieeexplore.ieee.org/abstract/document/6142190.

[4] Pawel Dymora and Andrzej Paszkiewicz, "Performance Analysis of Selected Programming Languages in the Context of Supporting Decision-Making Processes for Industry 4.0", *Applied Sciences*, EISSN 2076-3417, p. 8521, Vol. 10, No. 23, 28th November 2020, Published by Multidisciplinary Digital Publishing Institute (MDPI), DOI:10.3390/app10238521, Available: https://www.mdpi.com/2076-3417/10/23/8521.

[5] Marco Couto, Rui Pereira, Francisco Ribeiro, Rui Rua and Jo˜ao Saraiva, "Towards a Green Ranking for Programming Languages", in *Proceedings of the 21st Brazilian Symposium on Programming Languages*, 21st September 2017, CE, Fortaleza, Brazil, ISBN: 9781450353892, pp. 1–8, Published by Association for Computing Machinery (ACM), DOI: 10.1145/3125374.3125382, Available: https://dl.acm.org/doi/abs/10.1145/3125374.3125382.

[6] Shashikala Mahadevappa and Silvia Figueira, "Energy-Efficient Programming Languages for Mobile Applications", in *Proceedings of the IEEE Global Humanitarian Technology Conference (GHTC)*, 19-23 October 2021, Seattle, WA, USA, E-ISBN: 978-1-6654-3372-3, Print on Demand (PoD) ISBN: 978-1-6654-3373-0, Print on Demand (PoD) ISSN: 2377-6919, pp. 33–38, Published by Institute of Electrical and Electronics Engineers (IEEE), DOI: 10.1109/GHTC53159.2021.9612479, Available: https://ieeexplore.ieee.org/abstract/document/9612479.

[7] Ayob Sether, "Cloud computing benefits", *Social Science Research Network (SSRN)*, 2016, DOI: 10.2139/ssrn.2781593, Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2781593.

[8] Anton Beloglazov and Rajkumar Buyya, "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers", in Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, Australia, 17-20 May 2010, Electronic ISBN: 978-1-4244-6988-8, Print ISBN: 978-1-4244-6987-1, Published by IEEE, DOI: 10.1109/CCGRID.2010.45, Available: https://ieeexplore.ieee.org/document/5493430.

[9] Ekaba Bisong, "Python", *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Published by Apress, Berkeley, CA, Online ISBN: 978-1-4842-4470-8, Print ISBN: 978-1-4842-4469-2, pp. 71–89, 28th September 2019, DOI: 10.1007/978-1-4842-4470-8_9, Available: https://link.springer.com/chapter/10.1007/978-1-4842-4470-8_9.

[10] K. R. Srinath, "Python–the fastest growing programming language", *International Research Journal of Engineering and Technology*, E-ISSN: 2395-0056, Print ISSN: 2395-0072, Vol. 4, No. 12, pp: 354-357, December 2017, Published by Fast Track Publication, Available: https://www.irjet.net/archives/V4/i12/IRJET-V4I1266.pdf.

[11] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, J´acome Cunha *et al.*, "Ranking programming languages by energy efficiency", *Science of Computer Programming*, ISSN 0167-6423, Vol. 205, No. 10, 1st May 2021, DOI: 10.1016/j.scico.2021.102609, Available: https://www.sciencedirect.com/science/article/abs/pii/S0167642321000022.

[12] Hamza M Alvi, Hareem Sahar, Abdul A Bangash and Mirza O Beg, "Ensights: A tool for energy aware software development", in *Proceedings of the 13th International Conference on Emerging Technologies (ICET)*, 27-28 December 2017, Islamabad, Pakistan, E-ISBN: 978-1-5386-2260-5, USB ISBN: 978-1-5386-2259-9, Print on Demand (PoD) ISBN: 978-1-5386-2261-2, pp. 1–6, DOI: 10.1109/ICET.2017.8281713, Published by Institute of Electrical and Electronics Engineers (IEEE), Available: https://ieeexplore.ieee.org/abstract/document/8281713.

[13] Kerstin Eder, John P. Gallagher, G. Fagas, L. Gammaitoni and D. J. Paul, "Energy-aware software engineering", *ICT-energy concepts for energy efficiency and sustainability*, ISBN: 978-953-51-3011-6, Electronic ISBN: 978-953-51-3012-3, pp. 103–127, 2017, DOI: 10.5772/65985, Published by InTechOpen, Available: https://library.oapen.org/handle/20.500.12657/49211.

[14] Kenan Liu, Gustavo Pinto and Yu David Liu, "Data-oriented characterization of application-level energy optimization", in *Proceedings of the Fundamental Approaches to Software Engineering: 18th International Conference*

(FASE), London, UK, 11-18 April 2015, pp. 316–331, Print ISBN: 978-3-662-46674-2, Online ISBN: 978-3-662-46675-9, DOI: 10.1007/978-3-662-46675-9_21, Available: https://link.springer.com/chapter/10.1007/978-3-662-46675-9_21.

[15] Cesar Castellon Escobar, Swapnoneel Roy, O. Patrick Kreidl, Ayan Dutta and Ladislau B¨oloni, "Toward a Green Blockchain: Engineering Merkle Tree and Proof of Work for Energy Optimization", *IEEE Transactions on Network and Service Management*, Electronic ISSN: 1932-4537, Vol. 19, No. 4, pp: 3847-3857, 4th November 2022, DOI: 10.1109/TNSM.2022.3219494, Published by Institute of Electrical and Electronics Engineers (IEEE), Available: https://ieeexplore.ieee.org/abstract/document/9939185.

[16] Sarah Abdulsalam, Donna Lakomski, Qijun Gu, Tongdan Jin and Ziliang Zong, "Program energy efficiency: The impact of language, compiler and implementation choices", in *Proceedings of the International Green Computing Conference*, 03-05 November 2014, Dallas, TX, USA, Electronic ISBN: 978-1-4799-6177-1, DOI: 10.1109/IGCC.2014.7039169, pp. 1-6, Published by Institute of Electrical and Electronics Engineers (IEEE), Available: https://ieeexplore.ieee.org/abstract/document/7039169.

[17] Agner Fog, "Optimizing software in c++", 2016, Available: http://www.cajunbot.com/wiki/images/3/3e/Optimizing_software_in_cplusplus.pdf.

[18] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, J´acome Cunha *et al.*, "Energy efficiency across programming languages: how do energy, time, and memory relate", in *Proceedings of the 10th ACM SIGPLAN international conference on software language engineering*, 23-24 October 2017, BC, Vancouver, Canada, ISBN: 9781450355254, DOI: 10.1145/3136014.3136031, pp. 256–267, Published by Association for Computing Machinery (ACM), Available: https://dl.acm.org/doi/abs/10.1145/3136014.3136031.

[19] Déaglán Connolly Bree and Mel Ó Cinnéide, "Inheritance versus Delegation: which is more energy efficient?", in *Proceedings of the 42nd International Conference on Software Engineering*, 23 – 29 May 2020, Seoul, Republic of Korea, ISBN: 9781450379632, pp. 323—329, Published by Association for Computing Machinery (ACM), DOI: 10.1145/3387940.3392192, Available: https://dl.acm.org/doi/abs/10.1145/3387940.3392192.

[20] Rui Pereira, Tiago Carção, Marco Couto, Jácome Cunha and João Paulo Fernandes, "SPELLing out energy leaks: Aiding developers locate energy inefficient code", *Journal of Systems and Software*, Online ISSN: 1873-1228, Print ISSN: 0164-1212, Vol. 161, p. 110463, March 2020, DOI: 10.1016/j.jss.2019.110463, Published by Elsevier, Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121219302377.

[21] Muhammad Aminur Rahaman, Md. Solaiman Mia, Mahbubur Rahman and Md. Maskawath Latif, "An Energy Efficient Model of Software Development Life Cycle for Mobile Application", in *Proceedings of the 4th International Conference on Sustainable Technologies for Industry 4.0 (STI)*, 17-18 December 2022, Dhaka, Bangladesh, Electronic ISBN: 978-1-6654-9045-0, Print on Demand (PoD) ISBN: 978-1-6654-9046-7, pp. 1-6, Published by Institute of Electrical and Electronics Engineers (IEEE), DOI: 10.1109/STI56238.2022.10103246, Available: https://ieeexplore.ieee.org/abstract/document/10103246.

[22] Javier Mancebo, Coral Calero and Félix García, "Does maintainability relate to the energy consumption of software? A case study", *Software Quality Journal*, Electronic ISSN: 1573-1367, Print ISSN: 0963-9314, Vol. 29, No. 1, pp: 101-127, 6th January 2020, DOI: 10.1007/s11219-020-09536-9, Published by Springer, Available: https://link.springer.com/article/10.1007/s11219-020-09536-9.

[23] Loïc Lannelongue, Jason Grealey and Michael Inouye, "Green algorithms: quantifying the carbon footprint of computation", *Advanced Science*, Vol. 8, No. 12, 2nd May 2021, DOI: 10.1002/advs.202100707, Published by Wiley Online Library, Available: https://onlinelibrary.wiley.com/doi/full/10.1002/advs.202100707.