International Journal of Advanced Research in Engineering and Related Sciences
Available online at: www.ijarers.com
VOL 1 ISSUE 4 (10-20)

Computational Efficiency Optimization in Python: Benchmarking High-Performance Frameworks for Data Science and Machine Learning Workflows

Priya Kapoor *[1]
[1] School of Computing, National University of Singapore, Singapore;

Abstract

In recent years, Python has become the go-to language for data science and machine learning app developers. It is important to note, however, that data scientists are not always programming experts. In spite of the fact that Python enables them to rapidly develop their algorithms, when they are operating at scale, the need for efficient computation becomes unavoidable. Thus, optimizing high-performance hardware, such as graphics processing units and multi-core processors, to achieve their maximum potential is no simple feat. It is possible to consider the current narrative survey to be a reference document for practitioners of the Python language, which will assist them in navigating the vast array of tools and approaches that are accessible for use with the Python programming language. The user scenarios that are the focus of our document are designed to cover the majority of the possible circumstances that users may encounter. Tool developers may also find this document useful; by reading it, they will be able to see where our work could be lacking in current tools and be more motivated to fill such gaps. We feel that this information may be of service to tool developers.

Keywords: Python, Data science, Machine Learning.

INTRODUCTION

The ability of any software application to respond to a query is the primary metric by which its performance is measured. This holds true regardless of whether the application is web-based, mobile-based, or desktop-based. Online Analytical Processing (OLAP) and Online Transactional Processing (OLTP) are two further processes that these programmes may go through as needed. While online transaction processing (OLTP) systems aim to process new data, online analytical processing (OLAP) apps aim to examine existing data.

OLTP applications are critical to the operation of the vast majority of e-commerce websites [1]. OLTP programmes are responsible for a variety of tasks, including inserting, deleting, querying, and updating data. The objective of online analytical processing, on the other hand, is to identify patterns through the collection of data, the implementation of intricate analytical estimations, and the provision of a variety of report views [2].

Data that has been retrieved from a variety of sources and data warehouses is what OLAP works with. Data querying and presentation are the tasks that are carried out by an OLAP application. OLAP is utilised by Decision Support Systems, Software Development Applications, and Business Intelligence Applications. Data science applications are primarily concerned with the extraction of meaningful information from structured and/or unstructured data, as well as the forecasting of trends [3].

Decision science (DS) is an interdisciplinary field that uses a variety of approaches to help businesses make sense of their large data sets and make strategic decisions [4].

The transformation of data into information, information into knowledge, and finally decision-making is accomplished by the use of data models, data analytics, computations, and learning respectively. Nevertheless, regardless of whether it is an OLTP or an OLAP style of application, it is expected that a DS programme will deliver meaningful information in a timely manner and with accuracy after processing the data. The performance of an application is therefore an essential nonfunctional criterion that must be met.

Performance optimisation is the process of improving any software application in such a way that the application can be operated quickly and efficiently while keeping the accuracy of the outputs of data analysis. Optimal performance can be achieved by paying equal attention to the front-end and back-end parts of the software application. Both internal and external aspects, including execution time, space management, and I/O operations, and hardware configuration, network bandwidth and latency, and data characteristics, can be optimised for performance. Despite the fact that hardware, memory models, and cache designs are external aspects that significantly affect software performance, development teams are often unable to directly control these factors [5].

The performance of a software is impacted by internal elements, which can be regulated by the software itself instead of external factors. In order to optimise code through software, application developers must evaluate algorithmic choices, remove redundant computations, implement algorithms with better time and space complexity, implement caching effectively, optimise database schema and queries, and explore and identify opportunities for parallelization. An optimised software will use less memory and space, run faster, and seek data more efficiently, all of which contribute to better overall performance. It is possible to improve the application's performance on these metrics by monitoring and analysing it [6]. As an example, if the memory demand increases, the programme execution time decreases, and vice versa; this is just one example of the inherent trade-offs between these features.

LITERATURE REVIEW

Data Science (DS), scientific computing, data analytics, and Machine Learning (ML) are four rapidly expanding domains that make extensive use of Python. R, Fortran, and Matlab are just a few of the many data-centric and scientific computation computer languages that it replaces. It finds use in the field of application. Many libraries have been developed with DS and ML in mind, which is a major factor in DS's success. Matplotlib, Scikit-learn, SciPy, NumPy, and Pandas are a few examples of these libraries. Most high-performance Python libraries were developed using statically typed languages like C++, Fortran, and CUDA, rather than Python itself. This is because Diffusion and Machine Learning environments collect and process massive amounts of data.

The sluggish performance of the Python interpreter is the primary reason that libraries are built outside of Python. For this reason, libraries are developed outside of Python. [7] conducted an in-depth study on the overheads that are associated with the execution of Python programmes. First, it is slower than running code that has been compiled, which is something that is typically the case with interpreted languages. To be more specific, Python programmes, just as the majority of interpreted languages (for example, Java), are converted into bytecode before being executed by a virtual machine. Due to the fact that Python uses a dynamic object typing system, the programming language has an additional inherent inefficiency. According to [8], it is crucial to note that when a built-library function is used, the interpreter invokes a C function, which results in significant overheads. Thereafter, every instruction incurs the ongoing expense of setting up and cleaning a disposable execution environment. Python uses GIL, which is an acronym for "Global Interpreter Lock," by default. The GIL prevents multi-threading in addition to providing certain safety precautions for concurrent accesses. This occurs because each CPython process is limited to only running the interpreter thread. Assuring the object model is secure against concurrent access is one of the main goals of the GIL, which is to simplify the implementation. Within a single thread, CPython is responsible for executing code that is connected to the CPU. In addition, evaluations highlighted the fact that CPython demonstrates inadequate parallelism at the instruction level in this particular setting [9]. As a consequence of this, Python is not performing as well as other languages when it comes to efficiency. Because of this, there are a variety of tools that may be utilised to enhance the performance of programmes that are constructed using Python. This review article's goal is to provide a structured overview of Python's highperformance tool landscape. We may be able to connect with surveys that look at ways to speed up compilation for faster code execution, or surveys that think about ways to optimise communications within the framework of highperformance computing [10]. There is a strong relationship between the following related fields: input/output parallelization in HPC [11], software optimisation for embedded systems' energy efficiency [12], and GPU acceleration for specific machine learning tasks (like Frequent Itemset Mining [13]). But as far as we are aware, this is the first comprehensive list of Pythonspecific tools for code acceleration and high performance.

The fact that we organise our work in accordance with user situations is another one of the unique aspects of our approach. A report published on GitHub [14] states that the primary factor "behind Python's growth is a speedilyexpanding community of data science professionals and hobbyists." The Python community has grown substantially in the last decade, and this is all in reference to that. Reasons for this include the fact that languages like Python and its environment have simplified programming for the average user. Because of advancements in cloud

computing and scalable data processing technologies, which can handle massive amounts of data, and because deep learning is now within reach. These developments have made workflows that were previously insurmountable achievable within a reasonable length of time or less. A proliferation of helpful digital resources has been made possible by the easy, scalable, and accelerated computing capabilities that have been made available. These resources are contributing to the continued development of data science as its own unique area, which is attracting individuals from a wide variety of backgrounds and fields of study. Kaggle has become one of the most diversified of these communities since its first inception in 2010, and it was acquired by Google in 2017. It unites researchers and data scientists from over 194 countries, including amateurs with little expertise and industry heavyweights. Through the use of Kaggle, businesses are able to organise tournaments for difficult machine learning challenges that are currently being encountered in industry. These events allow members to collaborate and compete for awards. In many cases, the competitions end up producing public datasets that can be of assistance to further study and education. Additionally, Kaggle offers users the opportunity to share their knowledge and code in a social setting that encourages collaboration and makes available teaching materials. The data science community would do well to keep tabs on the tools used by top teams in Kaggle contests, as this provides concrete proof of the tools' utility. Since this information pertains to the tools, it is of special importance to the community. In order to organise a dataset's numerous fields, Pandas' data frame format employs columns. Furthermore, it allows for different data types for each column, unlike NumPy's ndarray container, where all objects have the same type. It uses close proximity of columns rather than the more common practice of storing record fields side by side, as in a commaseparated values (CSV) file. If the data is organised column by column, SIMD can be applied. In order to process rows of data, the CPU can pool memory accesses in this way. By doing so, the CPU is able to reduce the number of accesses to main memory while making efficient use of caching. The Apache Arrow cross-language development platform for in-memory data [15] standardises the columnar format, allowing data to be shared across libraries without the costs of copying and reformatting the data. Apache Parquet [16] is another available library that uses the columnar format. Libraries like Pandas and Apache Arrow were created with the goal of being used in memory, but Parquet was initially designed for data serialisation and disc storage. Modern workflows benefit greatly from the compatibility of Parquet with Arrow's columnar data structures for in-memory processing. This allows for the loading of data files from disc into Arrow, which is a highly efficient and effective procedure.

REAL-WORLD APPLICATIONS OF PYTHON IN DATA SCIENCE AND MACHINE LEARNING

Predictive Analytics and Forecasting

Predictive analytics and forecasting make heavy use of Python; this area analyses historical data to make predictions about future patterns. The management of supply chains, marketing, and finance are three areas that can benefit tremendously from this. Because of the utilisation of machine learning models, organisations are able to make wellinformed decisions and efficiently distribute their resources.

Image and Speech Recognition

As a result of the development of deep learning, Python has emerged as a formidable tool for applications involving speech and picture recognition. Computers are able to recognise objects in photos and transcribe spoken words into text thanks to the training of sophisticated neural networks, which is made possible by libraries such as TensorFlow and PyTorch.

Healthcare and Biotechnology

Python plays an important part in the fields of healthcare and biotechnology, particularly in the areas of analysing medical data, forecasting disease outbreaks, and producing personalised treatment. Models that are based on machine learning are utilised in order to analyse patient data, recognise patterns, and provide assistance in the process of diagnosis and treatment planning. **Fraud Detection and Cybersecurity**

Python is currently being utilised extensively in the field of cybersecurity for a variety of activities, including the identification of anomalies and fraud situations. With the use of machine learning algorithms, it is possible to examine huge amounts of data in order to recognise unexpected patterns that may be indicative of fraudulent activity or security breaches.

Future of Python in Data Science and Machine Learning

There are still obstacles to overcome, despite Python's dominance in data science and ML. A significant obstacle is the interpretability of deep neural network and other machine learning models.

It gets more and more challenging to comprehend how these models make decisions as they grow in complexity. Improving these models' interpretability is a hot topic among researchers and practitioners. When it comes to making decisions using machine learning, there are also ethical considerations to consider. Predictions that are influenced by biases in the training data can have an effect on both individuals and communities. The data science community must work together to create honest and open algorithms that can solve these ethical problems. The prospects for Python's continued use in data science and ML are bright. The capabilities of these technologies are projected to be further enhanced by ongoing advances in the Python ecosystem, as well as by advancements in hardware and algorithms. Integrating Python with emerging technologies such as the Internet of Things (IoT) and edge computing also opens up new avenues for innovation.

Advantages of Using Python for Machine Learning and Data Science Rich Ecosystem of Libraries

The large ecosystem of Python libraries is a key factor in the language's supremacy in data science and ML. Data analysis, visualisation, and manipulation are greatly supported by libraries such as NumPy, Pandas, and Matplotlib. For machine learning tasks, Scikit-learn's basic and efficient tools are great, but when it comes to deep learning, TensorFlow and PyTorch are the ones to beat. With these robust frameworks at their disposal, developers may cut corners and put more effort into finding solutions, rather than wasting time trying to recreate the wheel.

Ease of Learning and Readability

For those just starting out in data science and machine learning, Python is a great choice because to its easy-to-read syntax and succinct, straightforward expressions. Thanks to its easy-to-understand design, not only is the learning curve shortened, but teamwork is also improved. Because data science projects are inherently iterative, the simple syntax facilitates rapid development and experimentation.

Community and Documentation Support

The dynamic and engaged Python community is a key factor in the language's steady evolution. It is easy for people to get aid when they need it because there are so many online resources, forums, and community-driven projects. Python libraries are known for their rich documentation, which greatly aids in both installation and troubleshooting. With the help of this solid infrastructure, data scientists and developers can confidently take on difficult tasks.

Integration Capabilities

As a language and platform that works well with many others, Python connects the many parts of a pipeline for data science and machine learning. It can be easily integrated with high-performance modules because to its interoperability with languages such as C and Java. The efficient handling of enormous datasets is further enabled by the integration of Python with big data processing frameworks such as Apache Spark.

Scope of Python in Data Science and Machine Learning Industry Adoption and Job Opportunities

The extensive adoption of Python across industries is a reflection of its dominance in data science and machine learning. Businesses in many industries use Python to mine their data for insights, including healthcare, marketing, and finance. Consequently, there is a growing need for experts who can teach data science and machine learning using Python. Aspiring data scientists and machine learning engineers can greatly benefit from studying Python because it opens up a multitude of work prospects and career routes. **Innovation in Artificial Intelligence**

Libraries in Python, such as TensorFlow and PyTorch, enable state-of-the-art research and applications in the field of artificial intelligence (AI). Natural language processing, computer vision, and reinforcement learning are three areas where Python has been crucial in pushing innovation as AI has progressed. Python provides the means to transform abstract concepts into workable solutions, and the potential for expanding the frontiers of AI is enormous.

Education and Research

Python has become the language of choice in research and educational organisations due to its accessibility and adaptability. Teaching, experimentation, and model development are some of the many uses for Python among students, researchers, and faculty. Data science and machine learning research is driven by the dynamic environment created by Python's open-source nature, which promotes cooperation and information exchange. **Final Verdict**

It is a tribute to Python's flexibility and community support that the language has evolved from a general-purpose language to a data science and machine learning behemoth.

Data scientists and machine learning practitioners around the world like it because of its extensive library ecosystem, user-friendliness, and versatility. Python is a reliable partner as we explore the dynamic world of technology; it propels innovation and shapes the way data-driven decisions are made in the future. Whether you're just starting out and want to learn the ropes or are a seasoned pro looking to push the envelope, Python has everything you need to succeed in the ever-changing and fascinating world of data science and machine learning. If you are interested in learning Python for data science and machine learning, Gyansetu offers courses that are both thorough and organised. Learners are prepared to tackle the challenges of data science and machine learning with the help of expert-led training, practical projects, and an emphasis on practical applications at Gyansetu. Discover the limitless possibilities of Python in these ever-changing and game-changing domains by enrolling in one of Gyansetu's Python courses.

METHODOLOGY

Python, a high-level programming language, is presently used in many data science and machine learning-related applications. A dynamic type system, extensive libraries, memory management, and built-in data structures are just a few of the many features offered by the Python programming language. A large number of operating systems are compatible with its interpreters. Application developers can improve software programmes' performance by taking use of its many features.

The following methodology is utilised throughout this paper.

- The characteristics such as generators, vectorization, parallelism, and caching are discovered and applied in DS applications that are written in Python in order to investigate the enhancement in performance. One of the most prominent websites, kaggle.com, provided the dataset that was used to sample student performance. A random dataset is also utilised in addition to this particular dataset.
- A comparison is made between code created using the usual method and code that has unique characteristics such as generator, vectorization, concurrency, and parallelism in order to evaluate the impact of the features that have been found.
- Both kinds of code are run on the same data in order to compare the outcomes in terms of the amount of time and space that are required for the execution.
- Other aspects, such as I/O operations and caching, which have the potential to improve the performance of an application are also discussed and their benefits are carefully considered.

RESULT AND DISCUSSION Use of Generators

Using Generator, it is possible to iterate over large datasets without having to create complete sequences in memory. Using Generators makes memory use better, which is very helpful when dealing with big datasets. Due to its lack of memory storage for complete intermediate values, it only returns values one at a time (9). The key advantages of utilizing Generator are its concise syntax, lazy evaluation, and little memory use. Delaying evaluation of expression until its value is required is the goal of lazy evaluation, which aims to avoid repeated evaluation. By reading a file line by line using Generator, memory utilization is significantly reduced compared to loading the complete file into memory. An alternative to the return keyword for producing a succession of values, Python's Generator function makes use of the yield keyword. Using both the traditional approach and the Generator, a 10,000-element Fibonacci series is generated in Python to demonstrate that Generator is more memory efficient.

Afterwards, we scrutinize the memory consumption of both programs. This is an example of traditional Python code in the lines that follow.

```
import cProfile
from memory_profiler import profile
#run using python -m memory_profiler <filename.py>
def fibonacci_generator(n):
    fibonacci_series = [0, 1]
    while len(fibonacci_series) < n:
        next_number = fibonacci_series[-1] + fibonacci_series[-2]
        fibonacci_series.append(next_number)
    return fibonacci_series
@profile
def main():
    ret=fibonacci_generator(10000)
if __name__ == '__main__':
    main()
```

The Python code that utilises Generator is displayed in the lines that follow.

```
import cProfile
from memory_profiler import profile
#run using python -m memory_profiler <filename.py>
def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b
@profile
def main():
    fib_gen = fibonacci_generator()
    for i in range(10000):
        next(fib_gen)
if __name__ == '__main__':
    main()
```

The utilization of memory is depicted in Figures 1 and 2, by means of the Generator and Conventional techniques, respectively.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     9     47.7 MiB     47.7 MiB           1   @profile
    10                                         def main():
    11     47.7 MiB      0.0 MiB           1       fib_gen = fibonacci_generator()
    12     47.7 MiB      0.0 MiB       10001       for i in range(10000):
    13     47.7 MiB      0.0 MiB       10000           next(fib_gen)
```

Fig 1: Use of Generator

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    10     47.6 MiB     47.6 MiB           1   @profile
    11                                         def main():
    12     52.8 MiB      5.2 MiB           1       ret=fibonacci_generator(10000)
```

Fig 2: Conventional Method

It has been determined that when a conventional approach is utilised, it requires 5.2 MiB (Mebibyte) of additional memory, however when Generator is utilised, it requires 0.0 MiB of memory. It is important to note that 1 MiB is equivalent to 1.048576 MB. It is possible that this outcome will be slightly different when it is carried out on various machines. **Use of Vectorization**

Vectorization is an effective method for improving the efficiency and speed of scientific computations and data manipulation procedures. Without the need for explicit loops, it allows users to apply conventional mathematical functions on full data arrays or matrices simultaneously. It shortens the time needed to complete the tasks. In Python, it takes a long time to apply a loop over an array. If you want to run operations on an entire array quickly in Python, you should use standard mathematical functions instead of loops. An example of such a library is NumPy. When doing vectorization, the Outer, dot, and multiply functions could be useful. Here is a Python programme that shows how efficient the outer product utilising vectorization is compared to the classical method: it takes two vectors, n x 1 and 1 x m, and returns a matrix of size n x m. The identical collection of vectors is used to evaluate the execution time of both codes. See Figure 3 for a screenshot of the partial code for the first method and Figure 4 for the second. Using Vectorization Technique

```
#using vectorization
vector1 = array.array('i') # i is for signed int.
for i in range(100):
    vector1.append(i);
vector2 = array.array('i')

for i in range(100, 300):
    vector2.append(i)

StartTime = time.process_time()
outer_product = numpy.outer(vector1, vector2)
EndTime = time.process_time()

print("outer_product = "+ str(outer_product));
print("Execution time = "+str(1000*(EndTime - StartTime ))+"ms")
```

Fig 3: Using Vectorization Technique Using Classical Method (with loops)

```
vector1 = array.array('i') # i is for signed int.
for i in range(100):
    vector1.append(i);
vector2 = array.array('i')

for i in range(100, 300):
    vector2.append(i)

# classic Method
StartTime = time.process_time()
outer_product = numpy.zeros((100, 200))

for i in range(len(vector1)):
    for j in range(len(vector2)):
        outer_product[i][j]= vector1[i]*vector2[j]

EndTime = time.process_time()

print("outer_product = "+ str(outer_product));
print("Execution time = "+str(1000*(EndTime - StartTime ))+"ms")
```

Fig 4: Using Classical Method (with loops)

Figure 5 displays the code outputs of the two methods.

In Figure 6, we can see the outer operator mathematically represented as vector1 with 3x1 dimensions and vector2 with 1x3 dimensions. Other processes that make use of vectorization also have their execution times reduced compared to classical approaches.

```
outer_product = [[    0.     0.     0. ...     0.     0.     0.]
 [  100.   101.   102. ...   297.   298.   299.]
 [  200.   202.   204. ...   594.   596.   598.]
 ...
 [ 9700.  9797.  9894. ... 28809. 28906. 29003.]
 [ 9800.  9898.  9996. ... 29106. 29204. 29302.]
 [ 9900.  9999. 10098. ... 29403. 29502. 29601.]]
Execution time = 15.625ms
outer_product = [[    0     0     0 ...     0     0     0]
 [  100   101   102 ...   297   298   299]
 [  200   202   204 ...   594   596   598]
 ...
 [ 9700  9797  9894 ... 28809 28906 29003]
 [ 9800  9898  9996 ... 29106 29204 29302]
 [ 9900  9999 10098 ... 29403 29502 29601]]
Execution time = 0.0ms
```

Fig 5: Vectorization Dimensions

Vector 1= $\begin{bmatrix}1\\2\\3\end{bmatrix}$ Vector2 = [1 2 3], then Outer Product = $\begin{bmatrix}1 & 2 & 3\\2 & 4 & 6\\3 & 6 & 9\end{bmatrix}$

Where dimension of Vector1 is 3x1 and that of Vector2 is 1x3. The dimension of result outer product is 3x3.

Fig 6: Vectorization Dimensions of 3x1

Use of Profiling Tools

Profiling tools are designed to pinpoint the time and space bottlenecks, which are important to consider while optimising (10). A number of profiling libraries are available in Python, including cProfile, Profile, Line_profiler, memory_profiler, and Py-Spy. You can find out how many times a function is called and how long it takes to execute it via the cProfile library. The analysis of the execution time is aided by this. Finding and fixing code bottlenecks is the next step after analysis. Table 1 summarises the results of five separate executions of each programme to record its execution time. The above-mentioned task is found to be executed more quickly using the parallelism technique compared to the standard execution method.

Table 1: Execution Time

Round	Execution Time in ms (With Parallelism)	Execution Time in ms (With Normal Execution)
1	15.625	46.875
2	0.000	62.500
3	15.625	62.500
4	15.625	31.250
5	31.250	46.875
Average	15.625	50.000

The data's format and source, including whether it's a database or a flat file, also affect the code's performance. The format of the data is important since data science applications depend significantly on big datasets. Data is kept in a specific format in databases, while in flat files it is just plain text. In a database, you can get to the data immediately, but with a flat file, you have to read it in sequence. When the amount of data is too big to fit into a flat file, databases are utilised. Database management is also made easy with SQL queries. Consequently, there are a number of considerations to make when deciding between a database and a flat file. Table 2 lists only a few criteria.

Table 2: Difference between a Flat File and a Database

Sr. No.	Factor	Flat File	Database
1	Scalability	Limited	High
2	Structure of data	Plain text	Formatted
3	Accessibility	Sequential	Direct through SQL (more efficient)
4	Size	Limited	Large
5	Security	Poor	High
6	Organization	Single/multiple files without any relationship	Relationship exists among different tables

Table 3: Record of Execution Time

Round	With Normal Procedure	With Parallelism	Variance and then SD
1	187.50	46.87	125.00
2	187.50	62.50	125.00
3	171.87	62.50	156.25
4	171.87	93.70	125.00
5	156.25	78.12	125.00
Average	175.00	68.74	131.25

Table 3 shows that the parallelism-using programme has the best execution time compared to the other two scenarios. It also demonstrates that the two associated metrics, variance and standard deviation, are best calculated sequentially. It is more efficient to compute standard deviation from variance rather than both separately, which is why this approach is preferable in terms of execution time.

CONCLUSION

Use of the generator, vectorization, profiling, concurrency and parallelism, I/O activities, caching, and sequencing of operations are just a few of the numerous elements that affect the performance of any Python software application, particularly a data science application. Depending on the needs, there is always a trade-off between executing quickly

and efficiently using memory. The results showed that the traditional method of programming necessitated 5.2 MiB more space. No extra room was needed because generators were used on the same data. Similarly, this paper's results corroborated the idea that vectorization reduced execution time compared to conventional code. Additionally, as compared to the program's normal execution, the parallelism technique demonstrated a 68% improvement in execution time. Locating slow spots in the code was made easier with the use of the profiling functionality. People have noticed that this function really helps with performance. Additional elements that were discovered to be helpful in enhancing efficiency include managing I/O activities, utilising caching, and having the option to choose between flat file data and database. Considering that hardware and other external factors also significantly impact application performance. This aspect can be investigated at a later date. Additional aspects needed to optimise Data Science applications will also be investigated in the future.

REFERENCES

[1]. El-Sayed N, Sun Z, Sun K, Mayerhofer R. OLTP in Real Life: A Large-scale Study of Database Behavior in Modern Online Retail. In: 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS); vol. 1. IEEE. 2021. Available from: https://doi.org/10.1109/MASCOTS53633.2021.9614295.

[2]. Vatresia A, Johar A, Utama FP, Iryani S. Automated Data Integration of Biodiversity with OLAP and OLTP. Journal of Information System (e-journal). 2020;7(2):80–89. Available from: https://journal.unika.ac.id/index.php/sisforma/article/view/2817/pdf.

[3]. Kandula YS, Madireddy BB, Sourab BR. Analysis of Hardware Impact on Software Performance. International Research Journal of Engineering and Technology. 2021;8(12):1326–1332. Available from: https://www.irjet.net/archives/V8/i12/IRJET-V8I12223.pdf.

[4]. Russel R. Open University Learning Analytics Dataset: Course, Student and Assessment Data. . Available from: https://www.kaggle.com/datasets/rocki37/ open-university-learning-analytics-dataset/.

[5]. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. CoRR abs/1603.04467 (2016). http://arxiv.org/abs/1603.04467

[6]. Developers, P. How fast is PyPy? 2020. Available online: https://speed.pypy.org (accessed on 1 February 2020).

[7]. Team, G. The State of the Octoverse 2020. Available online: https://octoverse.github.com (accessed on 25 March 2020).

[8]. Preston-Werner, T. Semantic Versioning 2.0.0. 2013. Semantic Versioning. Available online: https://semver.org/ (accessed on 26 January 2020).

[9]. Authors, N. NumPy Receives First Ever Funding, Thanks to Moore Foundation. 2017. Available online: https://numfocus.org/blog/numpy-receives-first-ever-funding-thanks-to-moore-foundation (accessed on 1 February 2020).

[10]. Fedotov, A.; Litvinov, V.; Melik-Adamyan, A. Speeding up Numerical Calculations in Python. 2016. Available online: http://russianscdays.org/files/pdf16/26.pdf (accessed on 1 February 2020).

[11]. eam, O. OpenBLAS: An Optimized BLAS Library. 2020. Available online: https://www.openblas.net (accessed on 1 February 2020).

[12]. Team, I. Python Accelerated (Using Intel® MKL). 2020. Available online: https://software.intel.com/enus/blogs/python-optimized (accessed on 1 February 2020).

[13]. Diefendorff, K.; Dubey, P.K.; Hochsprung, R.; Scale, H. Altivec extension to PowerPC accelerates media processing. IEEE Micro 2000, 20, 85–95. [Google Scholar] [CrossRef]
[14]. Pedregosa, F.; Michel, V.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Vanderplas, J.; Cournapeau, D.; Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. 2011, 12, 2825– 2830.
[15]. Team, A.A. Apache Arrow—A Cross-Language Development Platform for In-memory Data. 2020. Available online: https://arrow.apache.org/ (accessed on 1 February 2020).