A Review on Apache Spark

Aryan Mukesh Rajpurohit
School of Computer Scienece & Engineering,Lovely Professional University,
Jalandhar, India
rajpurohitaryan20@gmail.com

Rishu Raj Kumar
School of Computer Scienece & Engineering,Lovely Professional University,
Jalandhar, India

Ratnesh Kumar
School of Computer Scienece & Engineering,Lovely Professional University,
Jalandhar, India

Puneet Kumar
School of Computer Scienece & Engineering,Lovely Professional University,
Jalandhar, India
puneetlpu1@gmail.com

***Abstract*— Apache Spark is an open-source, distributed computing system designed for large-scale data processing. It is designed to be flexible, fast, and easy to use, making it an ideal choice for big data analytics and machine learning tasks. Spark provides a range of high-level APIs for batch processing, stream processing, and machine learning, as well as lower-level APIs for custom data processing tasks. One of the main features of Spark is its ability to handle data processing tasks in memory, which can significantly improve performance over traditional disk-based systems. Spark also provides fault tolerance through its use of lineage information, which enables the recovery of lost data in case of node failures. Spark supports multiple programming languages, including Scala, Python, R, Java, and Julia, making it easy for users to work with their preferred language. It also supports integration with popular big data technologies such as Cassandra, Hadoop, and Kafka, allowing users to easily process data from a different source. Spark provides a range of deployment options, including standalone, yarn, Mesos, and Kubernetes, making it easy to deploy on a variety of clusters and cloud platforms. It also provides a variety of tools for monitoring and debugging Spark jobs, including a web-based interface for tracking job progress and diagnosing errors. Spark is a powerful and flexible platform for large-scale data processing and analytics. Its ease of use, flexibility, and performance make it a popular choice in big data processing tasks, machine learning, and real-time stream processing.**

Keywords: spark, cluster computing, Rdd, Dataframe, Hadoop.

INTRODUCTION

Spark is an open-source big data processing framework developed by Apache Software Foundation. It was created to handle analytics and large-scale data processing in distributed computing environments. Spark has become increasingly popular due to its ability to process data much faster than traditional batch processing systems. The core of Spark is based on Resilient Distributed Datasets (RDDs), which is fault-tolerant collections of data that can be processed in parallel. RDDs are immutable and distributed across a cluster of machines. Spark uses DAG (Directed Acyclic Graph) to keep track of the transformations and actions that are applied to the RDDs. Transformations are operations that create further an RDD from an existing one, while actions are used to return a result or write data to an external storage system. Spark supports a variety of programming languages, including Python, Java, Scala, and R, and provides multiple APIs for data processing, machine learning, graph processing, and stream processing. Spark includes several components as Spark ML-lib, Spark Core, Spark Streaming, Spark SQL, and Spark Graph-X, each designed for specific tasks. Spark Core is the foundation of the Spark platform and provides distributed task scheduling, fault tolerance, and memory management. Spark SQL is used for processing structured data and allows users to perform SQL queries on data stored in Spark [1]. Spark Streaming is used for data processing in real time and can be used to build real-time dashboards, detect anomalies, and perform sentiment analysis. Spark ML-lib provides a scalable machine-learning library that can be used for regression, classification, clustering, and collaborative filtering. Spark Graph-X is used for graph processing and allows users to build and process large-scale graphs. One of the main features of Spark is its ability to cache data in memory. This allows frequently accessed data to be stored in memory and accessed much faster than if it were stored on disk. Spark also provides fault tolerance through a mechanism called lineage. If in any case partition of an RDD shows an error due to a node failure, Spark can use the lineage information to recreate the partition from other nodes in the cluster. Spark includes several components, such as Spark SQL, Spark Core, Spark ML-lib, Spark Streaming, Spark SQL, and Spark Graph-X, each designed for specific tasks. Spark Core is the foundation of the Spark platform and provides distributed task scheduling, fault tolerance, and memory management. Spark SQL is used in processing structured data and allows users to perform SQL queries on data stored in Spark. Spark Streaming is used in real-time data processing and can be used to build real-time dashboards, detect anomalies, and perform sentiment analysis. Spark ML-lib provides a scalable machine-learning library that are used for clustering, classification, regression, and collaborative filtering. Spark Graph-X is used for graph processing and allows users to build and process large-scale graphs [2].

Spark can be used for a variety of use cases, including real-time data streaming, data processing, machine learning, and graph processing. It is widely used in industries such as healthcare, finance, retail, and telecommunications. Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied [3].

RESLIENT DISTRIBUTED DATASETS (RDDs)

Resilient Distributed Datasets (RDDs) is a fundamental data structure in Apache Spark, which is open-source distributed computing framework. RDDs provide an efficient way to represent and process large datasets across a cluster of computers. RDDs are immutable, distributed collections of objects that can be processed in parallel [4]. They are fault-tolerant and can automatically recover from node failures in a cluster. RDDs can be created by loading data from a file system or by transforming existing RDDs. Once an RDD is created, it can be processed using two types of operations: actions and transformations. Transformations are operations that create a new RDD from an existing one without changing the original RDD. Examples of transformations include flatmap, map, join, and filter. Actions, on the other hand, return a value or write data to an external storage system. Examples of actions include reduce, count and save. RDDs provide a simple programming interface for distributed computing and can be used to implement a wide range of applications, including machine learning, graph processing, and stream processing [5].

DATAFRAMES

DataFrame is a distributed collection of data organized into columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python/Scala. Spark's DataFrame API provides a programming interface for working with structured data. DataFrames can be constructed from a wide set of data sources, including structured data files, Hive tables, external databases, and streaming data [6]. A data frame is built on top of RDDs (Resilient Distributed Datasets), which provide fault-tolerant, distributed storage and processing of data. However, unlike RDDs, DataFrames provide a more structured and optimized API for manipulating structured data. One of the main features of DataFrames in Spark is their ability to optimize the execution plan of operations on the data. When a user issues a query or performs an operation on a data frame, Spark's Catalyst optimizer will analyze the query and generate an optimized execution plan based on the available data and operations. This can significantly improve the performance of data processing and analytics. DataFrames in Spark support a variety of data sources and file formats, including JSON, Parquet, Avro, ORC, CSV, and JDBC. DataFrames can also be integrated with external data sources, such as Apache Hbase, Apache Cassandra, and Elasticsearch. DataFrames in Spark can be manipulated using a variety of APIs and programming languages, including Python, Scala, java and R [7]. The DataFrame API provides a rich set of functions for filtering, aggregating, transforming, and joining data. Some of the commonly used functions include select(), filter(), groupBy(), join(), distinct(), orderBy(), and agg(). DataFrames also provide support for user-defined functions (UDFs) and can be used with Spark's machine learning library, MLlib. In addition to DataFrames, Spark also provides a similar concept called Dataset. A Dataset is an extension of DataFrame API and is strongly typed. It is available in java and scala but not in Python and R [8].

Overall, DataFrames in Spark provide a powerful and flexible API for working with structured data. They provide optimization and performance improvements over traditional RDD-based data processing, and their support for a wide set of data sources and programming languages make them a popular choice in big data processing and analytics [9].

EXAMPLE

Here, we demonstrate a few Spark programme examples. As Scala allows type inference, you will see that we omit variable types.Suppose that we wish to count the words in a large file stored locally. This can be done as:

```
val text = sc.textFile("mytextfile.txt")
val counts = text.flatMap(a =>a.split(" "))
val mapping = counts.map(b => (b,1))
val reducing = mapping.reduceByKey(_+_)
val result = reducing.count()
```

We created a distributed dataset called text that represent that local file as a collection of lines. Then using flatmap and split we have splited the words after each space, then maping each word with a key as 1 after doing these operation we use reduceByKey to reduce the pair based on key and count the value associated with it. Finally calling an action to evaluate and print the output of the program [10].

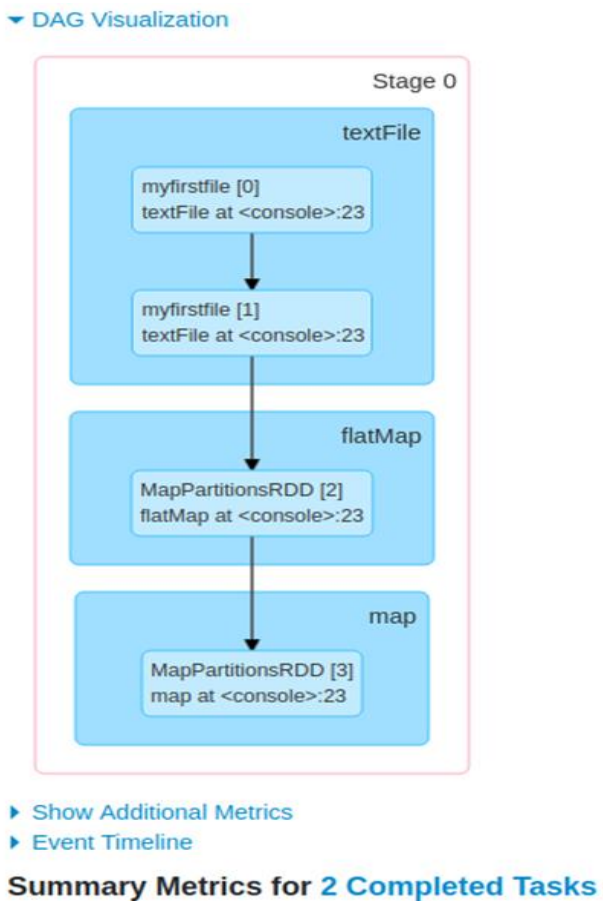

Fig. 1. This figure shows step by step transformation of RDD.

Cluster Computing Frameworks

Apache Spark is a distributed computing framework that is designed to run on a cluster of computers. Spark supports various cluster computing frameworks, which makes it easy to manage resources and deploy Spark applications on different infrastructure [11]. The following are some of the popular cluster computing frameworks that Spark supports:

Standalone: The default cluster manager in Spark called Standalone cluster manager. Standalone mode is the simplest way to deploy Spark, as it does not require any additional cluster management software. In standalone mode, Spark is run as a standalone application on a single machine or on a cluster of machines [12]. Spark's built-in cluster manager is used to manage the resources on the cluster and coordinate the execution of Spark applications. The standalone mode of deployment is ideal for small to medium-sized clusters or for running Spark on a single machine. It is also a good option for organizations that do not have access to other cluster management software or want to keep their deployment simple. To deploy Spark in standalone mode, you need to install Spark on each machine in your cluster and configure each machine's environment variables. You will also need to start the Spark master and worker processes, which are responsible for managing the resources and executing Spark applications on the cluster. Once you have set up your cluster, you can submit Spark applications to the cluster using the spark-submit command [13]. The Spark applications are then executed on the cluster, and the results are returned to the user. Standalone mode provides several benefits, including ease of deployment, low latency, and efficient use of resources. However, it also has some limitations, such as limited fault tolerance and scalability. Standalone mode is not recommended for large-scale deployments or for organizations that require high levels of fault tolerance or resource utilization. Standalone mode is a simple and efficient way to deploy Spark on small to medium-sized clusters or on a single machine. While it has its limitations, standalone mode is a good option for organizations that want to keep their deployment simple and do not require high levels of fault tolerance or scalability [14].

MapReduce YARN: Hadoop YARN stands for, Yet Another Resource Negotiator is a cluster management technology used in Apache Hadoop. It allows Spark to run on Hadoop clusters alongside other Hadoop applications. YARN provides resource management and job scheduling, making it easy to deploy Spark applications on a Hadoop cluster. YARN also provides features such as security and fault-tolerance, which make it suitable for production environments [15].

Kubernetes: Kubernetes is an open-source container orchestration system used for deploying, scaling, and managing containerized applications. Spark can be deployed on Kubernetes using the Kubernetes scheduler, which allows Spark to run in a containerized environment. Kubernetes provides features such as load balancing, scaling, and fault-tolerance, which make

it easy to manage Spark applications in a distributed environment. Kubernetes can be used with various cloud providers, making it easy to deploy Spark on cloud-based infrastructure [16].

Amazon EMR: Amazon EMR (Elastic MapReduce) is a managed Hadoop framework offered by Amazon Web Services (AWS). EMR allows users to run Spark applications on a managed Hadoop cluster without having to manage the cluster infrastructure. EMR provides features such as auto-scaling, monitoring, and security, which make it easy to deploy Spark applications on AWS. EMR also integrates with other AWS services, making it easy to ingest and process data from various sources [17].

Apache Mesos: Apache Mesos is a distributed system kernel that offers resource management and scheduling capabilities for data centre-scale computing. Mesos is often used in conjunction with Apache Spark to manage cluster resources and run Spark applications at scale. In a typical Mesos and Spark setup, Mesos serves as the cluster manager responsible for allocating resources, such as CPU, memory, and network bandwidth, to Spark tasks running on the cluster. The Mesos master node is responsible for resource allocation and scheduling, while one or more Mesos slave nodes execute tasks. Apache Spark can be run in two modes on Mesos: cluster mode and client mode. In cluster mode, the Mesos master launches Spark executors on Mesos slave nodes. On the other hand, in client mode, the Spark driver program runs outside the Mesos cluster and communicates with the Mesos master to launch Spark executors on Mesos slaves.

Using Mesos to manage resources and scheduling for Spark applications can provide many advantages, including improved resource utilization, dynamic resource allocation, and fault tolerance. Mesos can also be utilized to run other distributed applications, allowing for efficient use of resources and greater flexibility in a data center environment.

Overall, Spark provides support for various cluster computing frameworks, which allows users to choose the best framework based on their requirements and infrastructure. The choice of cluster computing framework depends on various factors such as scalability, resource allocation, fault-tolerance, and security. By supporting multiple cluster computing frameworks, Spark provides users with the flexibility to deploy Spark applications on different infrastructure and manage resources efficiently.

LANGUAGE INTEGRATION

Apache Spark supports several programming languages, making it a versatile platform for data processing and analytics. The following are the programming languages supported by Spark:

Scala: Scala is a general-purpose programming language that run on the Java Virtual Machine (JVM). Scala is the primary language for developing Spark applications. Spark cam also be implemented in Scala, which makes it easy to integrate Spark with Scala code. Scala is a statically language that provides functional programming features, making it suitable for building distributed and concurrent applications.

Java: Java is a popular programming language that runs on the JVM. Spark provides APIs for Java, which makes it easy to develop Spark applications in Java. Java is a statically typed language that provides object-oriented features, making it suitable for building complex applications

Python: It is a popular high-level programming language used for data analysis and scientific computing. Spark provides APIs for Python, which makes it easy to develop Spark applications in Python. Python is an interpreted language that provides dynamic typing and functional programming features, making it suitable for building data-driven applications.

R: It is a programming language used for statistical computing and graphics. Spark provides APIs for R, which makes it easy to develop Spark applications in R. R provides built-in functions for data manipulation and analysis, making it suitable for building data-intensive applications.

SQL: SQL stands for Structured Query Language is a standard language used for managing and querying relational databases. Spark provides a SQL interface called Spark SQL, which allows users to query data stored in Spark using SQL. Spark SQL supports various data sources, making it easy to integrate with other data processing systems.

Julia: Julia is a high-performance programming language used for numerical and scientific computing. Spark provides APIs for Julia, which makes it easy to develop Spark applications in Julia. Julia provides dynamic typing and high-level abstractions, making it suitable for building data-intensive applications.

Table 1: Difference between Apache Hadoop and Apache Spark

Feature	***Apache Hadoop***	***Apache Spark***
Data processing paradigm	Batch processing	Batch, real-time, and stream processing
Programming languages	Java, Python, C++, Ruby, and others	Scala, Java, Python, R, and SQL

Data storage	Hadoop Distributed File System (HDFS),	Hadoop Distributed File System (HDFS), Apache Cassandra, Apache HBase, and others
Data processing engine	MapReduce, YARN, and Hadoop Distributed File..	Spark Core, Spark SQL, Spark Streaming, Spark MLlib, and Spark GraphX
Performance	Slower due to disk I/O	Faster due to in-memory processing
Ease of use	Requires more setup and configuration	Easier to use with a simpler setup process
Community support	Large community and ecosystem	Rapidly growing community and ecosystem
Fault tolerance	High fault tolerance	High fault tolerance
APIs for data processing	MapReduce, Pig, Hive, HBase, Sqoop, Flume, Oozie	Spark SQL, Spark Streaming, Spark MLlib, and others
Cluster Management System	Apache Ambari, Cloudera Manager, and others	Apache Mesos, Hadoop YARN, and Kubernete
Machine Learning Libraries	Mahout, MLlib	MLlib, GraphFrames, TensorFlow on Spark, and others
Graph Processing Libraries	Giraph, Hama	GraphX, GraphFrames, Neptune, and others
Real-time Data Processing	Kafka, Storm, and Samza	Spark Streaming, Flink, and Storm
Data Visualization Tools	Apache Zeppelin, Hue	Apache Zeppelin, Jupyter Notebook, and others
Stream Processing Frameworks	Apache Nifi, Apache Storm, and Kafka Streams	Apache Flink, Apache Beam, and Kafka Streams

Apache Hadoop and Apache Spark are both highly capable platforms, and the features they offer cater to different data processing needs. In addition to the previously mentioned features, some other important features that differentiate these platforms include APIs for data processing, cluster management systems, machine learning libraries, graph processing libraries, real-time data processing, data visualization tools, and stream processing frameworks.

Apache Hadoop offers APIs such as MapReduce, Pig, Hive, HBase, Sqoop, Flume, and Oozie. It has cluster management systems such as Apache Ambari and Cloudera Manager, and machine learning libraries such as Mahout and MLlib. Hadoop has graph processing libraries like Giraph and Hama, and real-time data processing frameworks such as Kafka, Storm, and Samza. It also has data visualization tools like Apache Zeppelin and Hue.

Apache Spark offers APIs like Spark SQL, Spark Streaming, Spark MLlib, and others. It has cluster management systems like Apache Mesos, Hadoop YARN, and Kubernetes, and machine learning libraries like MLlib, GraphFrames, and TensorFlow on Spark. Spark has graph processing libraries like GraphX, GraphFrames, and Neptune and real-time data processing frameworks such as Spark Streaming, Flink, and Storm. It also has data visualization tools like Apache Zeppelin, Jupyter Notebook, and others.

Overall, the choice between Apache Hadoop and Apache Spark will depend on the specific needs of the project, and considering all the features that both platforms offer will help in making an informed decision.

LINEAGE

Lineage in Spark refers to the history of transformations and actions performed on a particular RDD (Resilient Distributed Dataset). It is a record of the lineage of a RDD, which allows the system to recover lost data or perform optimizations by re-executing lost or incomplete operations. In simple terms, lineage in Spark is a directed acyclic graph (DAG) of RDD dependencies that is generated when an RDD is created and transformed. Each node in the lineage represents a RDD and the edges represent the transformations that were performed on the RDD to create the next RDD in the lineage. The lineage graph is crucial to Spark's fault tolerance mechanism. Whenever a node in the lineage is lost due to node failure, the system can use the lineage graph to recover the lost data by re-executing the transformations that created the lost RDD. This is possible because Spark's transformations are deterministic, meaning that they will always produce the same output given the same input. Moreover, lineage also enables Spark to perform optimizations by identifying unnecessary computations and eliminating them. For instance, if two RDDs have the same lineage up to a certain point, Spark can reuse the previous RDD instead of recomputing it, thus improving performance and reducing computation time. Lineage also helps Spark to maintain consistency and correctness of the data in distributed systems. As Spark stores data in RDDs and processes it in a distributed manner, it's essential to ensure that the data is consistent and accurate across all nodes. The lineage graph ensures that every node has the same data and the same view of the data, by re-executing transformations on lost or inconsistent data. In conclusion, lineage is a vital feature of Spark that enables it to provide fault tolerance, optimization, and consistency in distributed systems. It allows Spark to recover lost data, eliminate unnecessary computations, and maintain the consistency and correctness of data across nodes. The lineage graph provides a powerful mechanism to recover and re-execute transformations on lost data, making Spark a reliable and efficient distributed processing framework.

CONCLUSION

Apache Spark is a popular big data processing framework that has gained widespread adoption in recent years. It is an open-source, distributed computing system designed to handle large-scale data processing workloads across clusters of commodity hardware. In this essay, I will provide an overview of Apache Spark and its key features, benefits, and drawbacks. Apache Spark was originally developed in 2009 at UC Berkeley's AMPLab, with the goal of providing a fast and flexible big data processing framework that could handle a wide range of workloads. Since then, it has become one of the most widely used big data processing frameworks, thanks to its ability to handle complex data processing tasks and its integration with other big data tools like Hadoop and Cassandra. One of the key benefits of Apache Spark is its ability to handle large-scale data processing tasks efficiently. Unlike traditional batch processing frameworks, Spark can process data in real-time, making it an ideal solution for streaming data processing tasks. Additionally, Spark provides a unified programming model, which enables developers to write applications in multiple languages like Java, Scala, and Python. Another key benefit of Apache Spark is its ability to handle a wide range of data processing tasks, including batch processing, interactive querying, machine learning, and graph processing. This makes it a versatile tool for data scientists and developers, who can use it to develop a wide range of applications and algorithms.

However, like any tool, Apache Spark has its drawbacks. One of the biggest challenges with Spark is its complexity. While Spark provides a unified programming model, it requires a deep understanding of distributed systems and parallel processing to use effectively. Additionally, Spark requires significant computational resources, which can make it difficult to deploy on smaller clusters or single machines. Despite these challenges, Apache Spark remains one of the most widely used big data processing frameworks, thanks to its performance, versatility, and community support. In recent years, Spark has continued to evolve and improve, with new features like structured streaming and machine learning pipelines making it even more useful for data scientists and developers. Overall, Apache Spark is a powerful and flexible big data processing framework that provides developers with a wide range of tools for handling complex data processing tasks. While it has its challenges, Spark's benefits and versatility make it an essential tool for any organization working with big data.

REFERENCES

[1] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the Netflix prize. In AAIM '08, pages 337-348, Berlin, Heidelberg, 2008. Springer-Verlag

[2] B. Hindman, A. Konwinski, M. Zaharia, and I. Stoica. A common substrate for cluster computing. In Workshop on Hot Topics in Cloud Computing (HotCloud) 2009, 2009

[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI, 2004.

[4] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica University of California, Berkeley. Spark: Cluster Computing with Working Sets

[5] R. Power and J. Li. Piccolo: Building fast, distributed programs with partitioned tables. In Proc. OSDI 2010.

[6] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In SIGMOD '07, pages 1029-1040. ACM, 2007

[7] Encyclopedia of Big Data Technologies, Springer Science and Business Media LLC, 2019 Publication

[8] Subhashini Chellappan, Dharanitharan Ganesan. "Practical Apache Spark", Springer Science and Business Media LLC, 2018.

[9] Hien Luu. "Beginning Apache Spark 2" , Springer Science and Business Media LLC, 2018

[10] Tao Li, Xueyu Li, Xu Zhang. "The Design and Implementation of Vector Autoregressive Model and Structural Vector Autoregressive Model Based on Spark" , 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM), 2017 Publication

[11] Dan Chen, Hai Jin, Long Zheng, Yu Huang et al. "A General Offloading Approach for NearDRAM Processing-In-Memory Architectures" , 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022

[12] Matei Zaharia. "An Architecture for Fast and General Data Processing on Large Clusters", Association for Computing Machinery (ACM), 2016 Publication

[13] 2016.Sharma, A., & Singh, D. (2022, October). Intelligent Systems for Diagnosis of Chronic Kidney Disease–A Review. In 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 1-5). IEEE.

[14] Sharma, A., & Singh, D. (2022, November). A Statistical Review on Machine Learning Based Medical Diagnostic Systems for Chronic Kidney Disease. In 2022 3rd International Conference on Computation, Automation and Knowledge Management (ICCAKM) (pp. 1-5). IEEE.

[15] Singh, J., & Singh, D. (2022, October). A Comprehensive Review on Sign Language Recognition Using Machine Learning. In 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 1-6). IEEE

[16] Singh, D., Prashar, D., Singla, J., Khan, A. A., Al-Sarem, M., & Kurdi, N. A. (2022). Intelligent Medical Diagnostic system for hepatitis B. Computers, Materials & Continua, 73(3), 6047-6068.

[17] Singh, D., Verma, S., & Singla, J. (2020, June). A comprehensive review of intelligent medical diagnostic systems. In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184) (pp. 977-981). IEEE.