International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

Enhanced Earlier Deadline First Algorithm for Scheduling Updates in Streaming Data Warehouse

Dr.N.Sureshkumar, Logesh.S, A.Nandhakumar, V.Niranjan

Muthayammal Engineering College, Rasipuram, Tamil Nadu, India

Department of Electronics and Communication Engineering, Muthayammal College of Engineering, Rasipuram, Tamil Nadu, India

Publication History: Received: 25.02.2026; Revised: 20.03.2026; Accepted: 25.03. 2026; Published: 28.03.2026.

ABSTRACT: we discuss update scheduling in streaming data warehouses, which combine the features of traditional data warehouses and data stream systems. In our setting, external sources push append-only data streams into the warehouse with a wide range of interarrival times. While traditional data warehouses are typically refreshed during downtimes, streaming warehouses are updated as new data arrive. We model the streaming warehouse update problem as a scheduling problem, where jobs correspond to processes that load new data into tables, and whose objective is to minimize data staleness over time (at time t, if a table has been updated with information up to some earlier time r, its staleness is t minus r). We then propose a scheduling framework (using the round robin scheduling in one of the basic algorithms.) that handles the complications encountered by a stream warehouse: view hierarchies and priorities, data consistency, inability to preempt updates, heterogeneity of update jobs caused by different interarival times and data volumes among different sources, and transient overload. A novel feature of our framework is that scheduling decisions do not depend on properties of update jobs (such as deadlines), but rather on the effect of update jobs on data staleness. Finally, we present a suite of update scheduling algorithms and extensive simulation experiments to map out factors which affect their performance.

KEYWORDS: Earliest Deadline First, Scheduling Algorithms, Streaming Data Warehouse, Real-Time Data Processing, Task Prioritization, Update Scheduling, Deadline-Aware Systems

I. INTRODUCTION

Data mining is the extraction of hidden predictive information from large databases. Data are any facts, numbers, or text that can be processed by a computer. Operational or transactional data are such as, sales, cost, inventory, payroll, and accounting. Nonoperational data, such as industry sales, forecast data, and macro economic data. The patterns, associations, or relationships among all this data can provide information. Information can be converted into knowledge about historical patterns and future trends. Data warehousing is defined as a process of centralized data management and retrieval. Data extraction is the act or process of retrieving data out of data sources for further data processing or data storage.

Traditional data warehouses are updated during downtimes and store layers of complex materialized views over terabytes of historical data. On the other hand, Data Stream Management Systems (DSMS) support simple analyses on recently arrived data in real time. Streaming warehouses such as Data Depot combine the features of these two systems by maintaining a unified view of current and historical data. This enables a real-time decision support for business-critical applications that receive streams of append-only data from external sources.

Applications include Online stock trading, where recent transactions generated by multiple stock exchanges are compared against historical trends in nearly real time to identify profit opportunities Credit card or telephone fraud detection, where Streams of point-of-sale transactions or call details are collected in nearly real time and compared with past customer behavior;. Summaries to monitor network performance and detect network attacks The goal of a streaming warehouse is to propagate new data across all the relevant tables and views as quickly as possible. Once new data are loaded, the applications and triggers defined on the warehouse can take immediate action. This allows businesses to make decisions in nearly real time, which may lead to increased profits, improved customer satisfaction, and prevention of serious problems that could develop if no action was taken. Recent work on streaming warehouses has focused on speeding up the Extract-Transform-Load (ETL) process .There has also been work on supporting various warehouse maintenance policies, such as immediate (update views whenever the base data change), deferred (update views only when queried), and periodic . However, there has been a little work on choosing, of all the tables

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

that are now out-of-date due to the arrival of new data, which one should be updated next. This is exactly the problem we study in this paper.

Immediate view maintenance may appear to be a reasonable solution for a streaming warehouse (deferred maintenance increases query response times, especially if high volumes of data arrive between queries, while periodic maintenance delays updates that arrive in the middle of the update period). That is, whenever new data arrive, we immediately update the corresponding "base" table T. After T has been updated, we trigger the updates of all the materialized views sourced from T, followed by all the views defined over those views, and so on. The problem with this approach is that new data may arrive on multiple streams, but there is no mechanism for limiting the number of tables that can be updated simultaneously. Running too many parallel updates can degrade performance due to memory and CPU-cache thrashing (multiple memories intensive ETL processes are likely to exhaust virtual memory), disk-arm thrashing, context switching, etc. This motivates the need for a scheduler that limits the number of concurrent updates and determines which job (i.e., table) to schedule (i.e., update) next.

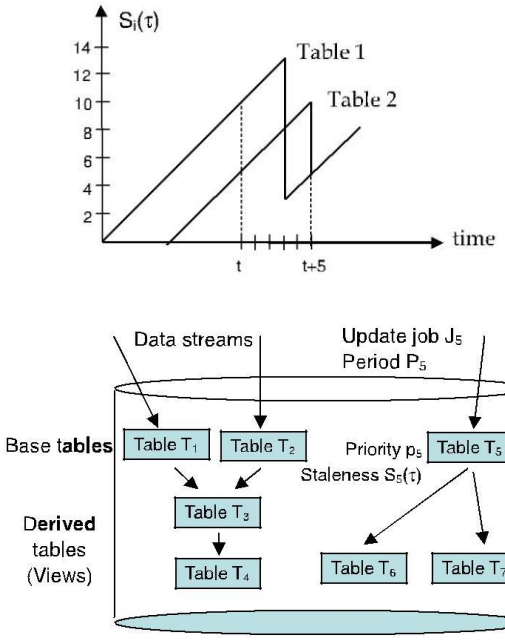

Fig.1. Staleness of two tables scheduled according to Max Benefit.

concurrency and for separating long jobs from short jobs (with the number of tracks being the limit on the number of concurrent jobs). For simplicity, we assume that the same type of basic scheduling algorithm is used for each track.

TABLE.1.A streaming data warehouse

Symbol	Meaning
P_s	Period of stream s
J_i	Update job for table i
$E_i(n)$	Execution time of J_i on data produced in a time interval of length n
R_i	Release time of J_i
p_i	Priority of table i
$F_i(\tau)$	Freshness of table i at time τ
$S_i(\tau)$	Staleness of table i at time τ
ΔF_I	Freshness delta of table i
α_i	Time to initialize the ETL process for table i
β_i	Data arrival rate to table i

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

II. SCHEDULING ALGORITHMS

This section presents our scheduling framework. The idea is to partition the update jobs by their expected processing times, and to partition the available computing resources into tracks. A track logically represents a fraction of the computing resources required by our complex jobs, including CPU, memory, and disk I/Os. When an update job is released, it is placed in the queue corresponding to its assigned partition (track), where scheduling decisions are made by a local scheduler running a basic algorithm (however, the algorithm that we will present in Section 3.2.3 generalizes this assumption). We assume that each job is executed on exactly one track, so that tracks become a mechanism for limiting At this point, one may ask why we do not precisely measure resource utilization and adjust the level of parallelism on-the-fly. The answer is that it is difficult to determine performance bottlenecks in a complex server, and performance may deteriorate even if resources are far from fully utilized. The difficulty of cleanly correlating resource use with performance leads us to schedule in terms of abstract tracks instead of carefully calibrated CPU and disk usage.

Below, we first discuss basic algorithms, followed by job partitioning strategies, and techniques for dealing with view hierarchies and transient overload.

A. Basic Algorithms

The basic scheduling algorithms prioritize jobs to be executed on individual tracks, and will serve as building blocks of our multitrack solutions For example, the Earliest-Deadline-First (EDF) algorithm orders released jobs by proximity to their deadlines. EDF is known to be an optimal hard real-time scheduling algorithm for a single track (w.r.t. maximizing the number of jobs that meet their deadlines), if the jobs are preemptible [7]. Since our jobs are prioritized, using EDF directly does not result in the best performance. Instead we use one of the following basic algorithms. Prioritized EDF (EDF-P) orders jobs by their priorities, breaking ties by deadlines. Our model does not directly have deadlines, but they may be estimated as follows: For each job J_i , we define its release time r_i as the last time T_i's freshness delta changed from zero to nonzero (i.e., the last arrival of new data in case of base tables, or, for derived tables, the last movement of the trailing edge point of its source tables). Then, we estimate the deadline of J_i to be r_i + P_i (recall that the period of a derived table is the maximum of the periods of its descendants).Max Benefit Recall that the goal of the scheduler is to minimize the weighted staleness. In this context, the benefit of executing a job J_i may be defined as $p_{i\Delta} F_i$, i.e., its priority-weighted freshness delta (decrease in staleness). Similarly, the marginal benefit of executing J_i is its benefit per unit of execution time: $p_{i\Delta} F_i / E_i$ ($_{\Delta}F_i$) Þ. A natural online greedy heuristic is to order the jobs by the marginal benefit of executing them. We will refer to this heuristic as Max Benefit. Since marginal benefit does not depend on the period, we can use Max Benefit for periodic and aperiodic update jobs.

For example, suppose that jobs J_1 and J_2, corresponding to Tables 1 and 2, respectively, are released at time , with p_1 = p_2 = 1, E_1= 3, E_2 = 2, $_{\Delta}F_1$ = 10, and $_{\Delta}F_2$ = 5. Max Benefit schedules J_1 at time because its delta freshness, and therefore its marginal benefit, is higher. Fig. 3 plots the weighted staleness of these two tables between time r and r+5, assuming that J_1 runs first. Table 1 begins with a weighted staleness (and delta freshness) of 10 at time. Three time units later, J1 is done and staleness drops by 10 (from 13 down to 3). The weighted staleness of Table 2 is five at time t and 8 at time r+ 3 when J_2 begins. At time r+ 5, J_2 is completed and the staleness of Table 2 drops from ten to five. Between times r and r+ 5, the area under Table 1's staleness curve works out
To 42.5 and the area under Table 2's staleness curve is 37.5, for a total weighted staleness of 80. We leave it as an exercise for the reader to verify that if J_2 were to run first, the total staleness in this time interval would be 85.

Since our jobs are assumed to be (approximately) periodic, one may argue that Max Benefit ignores useful information about the release times of future jobs. Recall the above example and suppose that J_3 is expected to be released at

time r+1, with p_3 = 10, E_3 = 3, and $_{\Delta}F_3$ = P_3 =50. It is better to schedule J_2 at time and then schedule J_3 when J_2 finishes at time r+2, rather than scheduling J_1 at time t and making J_3 wait two time units. To see this, note that J_3 accrues weighted staleness at a rate of 10 * 50 =500 per unit time, while J_1 accrues weighted staleness at a rate of 10 per unit time. Hence, making J_1 wait for several time units is better than delaying J_3 by one time unit. Motivated by this observation, we have tested the following extension to the Max Benefit algorithm:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

Max Benefit with Lookahead chooses the next job to execute as follows:

1. J_i = released job with the highest marginal benefit.
2. For each J_k whose expected release time r_k is within E_i of the current time and whose marginal benefit is higher than that of J_i
   a. For each set S of released jobs J_m such that $r_k <= \sum(E_m) < E_j$
   $B[S] = (\sum_m (p_{m\Delta} F_m) + p_{k\Delta} F_k) / (\sum_m (E_m) + E_k)$
3. If $\max_S B[S]$ > marginal benefit of J_i
   a. Schedule the job with highest marginal benefit from set $S^* = \text{argmax}_S B[S]$
4. Else
   a. Schedule J_i.

In the above example, J_1 has the highest marginal benefit, but a job with a higher marginal benefit (J_3) will be released before J_1 is completed. The Look ahead algorithm needs to find alternate sequences of jobs whose total running time is between 1 and 2, and compute their B[S] values; intuitively, B[S] represents the marginal benefit of running all the tasks in S followed by J_3. There is one such sequence: {J_2, J_3}, with B[S]= 505/5 = 101, which is higher than the marginal benefit of J_1 of 10/3. Thus, it schedules J_2 instead of J_1.

The Look ahead algorithm employs a number of heuristics to prune the number of alternate job sets. First, it only considers sequences that never leave the system idle (line 2a: $r_k <= \sum_m (E_m)$), since it is potentially dangerous to avoid job invocation based on unreliable information about future job releases. Second, it only considers future jobs scheduled to arrive before the current job with highest marginal benefit is expected to complete. In our experiments, the scheduling overhead of the Look ahead algorithm as compared to standard Max Benefit was negligible. However, the performance gain of the Look ahead algorithm was very minor as it almost always chose the same job to execute as Max Benefit.

Job Partitioning

If a job set is heterogeneous with respect to the periods and execution times (long execution times versus short periods), scheduler performance is likely to benefit if some fraction of the processing resources are guaranteed to short jobs (corresponding to tables that are updated often, which generally have higher priority). The traditional method for ensuring resource allocation is to partition the job set and to schedule each partition separately [7] (and to repartition the set whenever new tables or sources are added or existing ones removed, or whenever the parameters of existing jobs change significantly). However, recent results indicate that global scheduling (i.e., using a single track to schedule one or more jobs at a time) provides better performance, especially in a soft real-time setting, where job lateness needs to be minimized [5], . In this section, we investigate two methods for ensuring resources for short jobs while still providing a degree of global scheduling: EDF-Partitioned and Proportional.

View Hierarchies

Materialized view hierarchies can make the proper prioritization of jobs difficult. For example if a high-priority view is sourced from a low priority view, then it cannot be updated until the source view is which might take a long time since the source view has low priority. Therefore, source views need to inherit the priority of their dependent views. Let Ip_i be the inherited priority of table T_i. We explore three ways of inheriting priority:

Sum: Ip_i is the sum of the priorities of its dependent views (including itself).

Max: Ip_i is the maximum of the priorities of its dependent views (including itself).

Max-plus: Ip_i is K times the maximum of the priorities of its dependent views (including itself), for some K > 1:0. A large value of K increases the priority of base tables relative to derived tables, especially those derived tables which have a long chain of ancestors.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

Dealing with Transient Overload

During transient overload, low-priority jobs are deferred in favor of high priority jobs. When the period of transient overload is over, the low-priority jobs will be scheduled for execution. Since they have been delayed for a long time, they will have accumulated a large freshness delta and therefore a large execution time and therefore might block the execution of high-priority jobs. A solution to this problem is to "chop up" the execution of the jobs that have accumulated a long freshness delta to a maximum of c times their period, for some c ≥ 1:0. This technique introduces a degree of perceptibility into long jobs, reducing the chances of priority inversion (low-priority jobs blocking high-priority jobs) [7].

A reasonable rule-of-thumb for choosing c is to use a small number greater than one. Large values of c result in little chopping, while setting c =1 forces the scheduler to act as though there is no overload and every individual chunk of data needs to be loaded separately.

III. EXPERIMENTS

Setting

We ran two types of experiments. In the first type, the simulation always executes in a slowdown period, and we adjust S to vary the total utilization U_{tot}. In the second type, the simulation alternates between normal and slow periods, and we vary α_i and β_i to vary U_{tot}. As we vary job execution times, the table staleness will vary. In order to obtain comparable numbers, we report the relative lateness, which is the average weighted staleness reported for an experiment divided by the average weighted staleness reported by an experiment with the same parameters, but with no contention for resources (i.e., every job is executed when released).

Effect of Priorities

Intuitively, a prioritized scheduler performs better than a non prioritized scheduler when jobs have varying priorities. To measure the benefit, we ran the experiment shown in Fig. 3. We used two classes of jobs which are identical except that the first class has a priority of one and the second has a priority of 10. The prioritized basic algorithm (in this case, Max-Benefit) incurs a far lower lateness than the non prioritized basic algorithms, EDF, and Random.

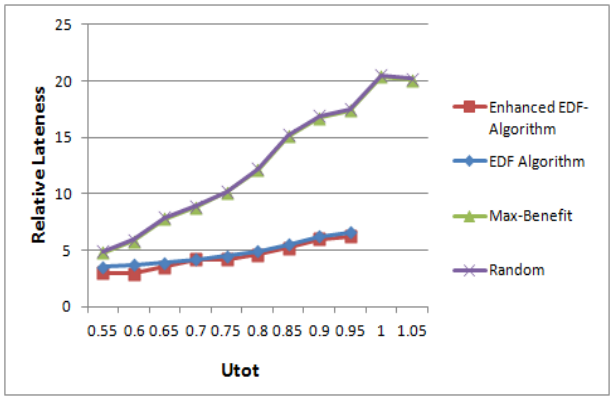

Fig.3 Effects of prioritized jobs.

View Hierarchies

What is the effect of job hierarchies? One problem with this class of experiments is the wide range of view dependency graphs that can arise in practice. We ran experiments with a variety of graphs and received results similar to those shown in Fig. 8. In this particular experiment, all hierarchies are a single chain with a depth of three. Base tables have priorities of 0.001, the next level (i.e., all the derived tables sourced from base tables) has priority of one, and the next level has a priority of either 1, 10, or 100 (with identical numbers of each). We ran the Max Benefit algorithm and tested what happens if no inheritance occurs, versus if the inheritance mechanism is Max, Sum, or Max-plus. Performance is poor if there is no inheritance because 1) base tables are starved, and 2) the scheduler cannot determine which base tables feed high versus low priority derived tables. Among the algorithms that use priority inheritance, their relative performance is often very similar (as in Fig. 8), but, overall, Max performs the best while Sum performs the worst. This is because Max achieves the right balance between prioritizing a base table high enough for it to run in a timely manner, but not so high as to starve the actual high-priority tables.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

Update Chopping

Finally, we evaluate the benefits of update chopping. Following slowdown periods, low-priority jobs may have accumulated a large freshness delta and can block high-priority jobs for a long time. We proposed update chopping to avoid this kind of blocking by adding a degree of preemptibility to the jobs. We report a representative set of results on a job set whose dependency graph consists of single-chain hierarchies with a depth of two (i.e., each base table is used to define one derived table). The base table jobs all have a priority of one and the derived tables have a priority of 10. We configured update chopping to limit the amount of data loaded at once to three times the job period.

IV. RELATED WORK

One important difference between a DSMS and a data stream warehouse is that the former only has a limited working memory and does not store any part of the stream permanently. Another difference is that a DSMS may drop a fraction of the incoming elements during overload, whereas a streaming data warehouse may defer some update jobs, but must eventually execute them. Scheduling in DSMS has been discussed in [2], [8] but all of these are concerned with scheduling individual operators inside query plans.

V. CONCLUSION

In this paper, we motivated, formalized, and solved the problem of no preemptively scheduling updates in a real-time streaming warehouse. We proposed the notion of average staleness as a scheduling metric and presented scheduling algorithms designed to handle the complex environment of a streaming data warehouse. We then proposed a scheduling framework that assigns jobs to processing tracks and uses basic algorithms to schedule jobs within a track. The main feature of our framework is the ability to reserve resources for short jobs that often correspond to important frequently refreshed tables, while avoiding the inefficiencies associated with partitioned scheduling techniques.

We have implemented some of the proposed algorithms in the Data Depot streaming warehouse, which is currently used for several very large warehousing projects within AT&T. As future work, we plan to extend our framework with new basic algorithms. We also plan to fine-tune the Proportional algorithm in our experiments, even the aggressive version with “all” allocation still exhibits signs of multiple operating domains, and therefore can likely be improved upon (however, it is the first algorithm of its class that we are aware of). Another interesting problem for future work involves choosing the right scheduling “granularity” when it is more efficient to update multiple tables together, We intend to explore the tradeoffs between update efficiency and minimizing staleness in this context.

REFERENCES

1. B. Adelberg, H. Garcia-Molina, and B. Kao, “Applying Update Streams in a Soft Real-Time Database System,” Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 245-256, 1995.
2. B. Babcock, S. Babu, M. Datar, and R. Motwani, “Chain: Operator Scheduling for Memory Minimization in Data Stream Systems,” Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 253-264, 2003.
3. S. Babu, U. Srivastava, and J. Widom, “Exploiting K-constraints to Reduce Memory Overhead in Continuous Queries over Data Streams,” ACM Trans. Database Systems, vol. 29, no. 3, pp. 545-580, 2004.
4. S. Baruah, “The Non-preemptive Scheduling of Periodic Tasks upon Multiprocessors,” Real Time Systems, vol. 32, nos. 1/2, pp. 9-20, 2006.
5. C.Nagarajan and M.Madheswaran - ‘Stability Analysis of Series Parallel Resonant Converter with Fuzzy Logic Controller Using State Space Techniques’- Taylor &Francis, Electric Power Components and Systems, Vol.39 (8), pp.780-793, May 2011. DOI: 10.1080/15325008.2010.541746
6. C.Nagarajan and M.Madheswaran - ‘Experimental verification and stability state space analysis of CLL-T Series Parallel Resonant Converter’ - Journal of Electrical Engineering, Vol.63 (6), pp.365-372, Dec.2012. DOI: 10.2478/v10187-012-0054-2
7. C.Nagarajan and M.Madheswaran - ‘Performance Analysis of LCL-T Resonant Converter with Fuzzy/PID Using State Space Analysis’- Springer, Electrical Engineering, Vol.93 (3), pp.167-178, September 2011. DOI 10.1007/s00202-011-0203-9
8. S.Tamilselvi, R.Prakash, C.Nagarajan,“Solar System Integrated Smart Grid Utilizing Hybrid Coot-Genetic Algorithm Optimized ANN Controller” Iranian Journal Of Science And Technology-Transactions Of Electrical Engineering, DOI10.1007/s40998-025-00917-z,2025
9. S.Tamilselvi, R.Prakash, C.Nagarajan,“ Adaptive sliding mode control of multilevel grid-connected inverters using reinforcement learning for enhanced LVRT performance” Electric Power Systems Research 253 (2026)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

112428, doi.org/10.1016/j.epsr.2025.112428

10. S.Thirunavukkarasu, C. Nagarajan, 2024, "Performance Investigation on OCF and SCF study in BLDC machine using FTANN Controller," Journal of Electrical Engineering And Technology, Volume 20, pages 2675–2688, (2025), doi.org/10.1007/s42835-024-02126-w
11. C. Nagarajan, M.Madheswaran and D.Ramasubramanian- 'Development of DSP based Robust Control Method for General Resonant Converter Topologies using Transfer Function Model'- Acta Electrotechnica et Informatica Journal , Vol.13 (2), pp.18-31,April-June.2013, DOI: 10.2478/aeei-2013-0025.
12. C.Nagarajan and M.Madheswaran - 'DSP Based Fuzzy Controller for Series Parallel Resonant converter'- Springer, Frontiers of Electrical and Electronic Engineering, Vol. 7(4), pp. 438-446, Dec.12. DOI 10.1007/s11460-012-0212-0.
13. C.Nagarajan and M.Madheswaran - 'Experimental Study and steady state stability analysis of CLL-T Series Parallel Resonant Converter with Fuzzy controller using State Space Analysis'- Iranian Journal of Electrical & Electronic Engineering, Vol.8 (3), pp.259-267, September 2012.
14. C.Nagarajan and M.Madheswaran, "Analysis and Simulation of LCL Series Resonant Full Bridge Converter Using PWM Technique with Load Independent Operation" has been presented in ICTES'08, a IEEE / IET International Conference organized by M.G.R.University, Chennai.Vol.no.1, pp.190-195, Dec.2007
15. Suganthi Mullainathan, Ramesh Natarajan, "An SPSS and CNN modelling based quality assessment using ceramic materials and membrane filtration techniques", Revista Materia (Rio J.) Vol. 30, 2025, DOI: https://doi.org/10.1590/1517-7076-RMAT-2024-0721
16. M Suganthi, N Ramesh, "Treatment of water using natural zeolite as membrane filter", Journal of Environmental Protection and Ecology, Volume 23, Issue 2, pp: 520-530,2022
17. S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," Algor-ithmica, vol. 15, pp. 600-625, 1996.
18. M.H. Bateni, L. Golab, M.T. Hajiaghayi, and H. Karloff, "Scheduling to Minimize Staleness and Stretch in Real-time Data Warehouses," Proc. 21st Ann. Symp. Parallelism in Algorithms and Architectures (SPAA), pp. 29-38, 2009.
19. A. Burns, "Scheduling Hard Real-Time Systems: A Review," Software Eng. J., vol. 6, no. 3, pp. 116-128, 1991.
20. D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker, "Operator Scheduling in a Data Stream Man-ager," Proc. 29th Int'l Conf. Very Large Data Bases (VLDB), pp. 838-849, 2003.
21. Kiran, A., Rubini, P., & Kumar, S. S. (2025). Comprehensive review of privacy, utility and fairness offered by synthetic data. IEEE Access.
22. Gopinathan, V. R. (2024). Real-Time Financial Risk Intelligence Using Secure-by-Design AI in SAP-Enabled Cloud Digital Banking. International Journal of Computer Technology and Electronics Communication, 7(6), 9837-9845.
23. Udayakumar, R., Elankavi, R., Vimal, R., & Sugumar, R. (2023). Improved Particle Swarm Optimization with Deep Learning-Based Municipal Solid Waste Management in Smart Cities. Environmental & Social Management Journal, 17(4).
24. Anand, L. (2023). An Intelligent AI and ML–Driven Cloud Security Framework for Financial Workflows and Wastewater Analytics. International Journal of Humanities and Information Technology, 5(02), 87-94.
25. Soundappan, S. J. (2020). Big Data Analytics in Healthcare: Applications for Pandemic Forecasting. International Journal of Advanced Research in Computer Science & Technology, 3(1), 2248-2253.
26. Rajasekar, M. (2024). Real-Time Predictive DevOps Intelligence for Risk-Aware Digital Business Processes in Cloud and SAP Ecosystems. International Journal of Advanced Research in Computer Science & Technology, 7(4), 10713-10718.
27. Poornima, G., & Anand, L. (2024, May). Novel AI Multimodal Approach for Combating Against Pulmonary Carcinoma. In 2024 5th International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE.
28. Prabha, P. S., & Rengarajan, A. (2025). Adaptive Cloud Resource Allocation Using Attention-Driven Deep Reinforcement Learning. Engineering, Technology & Applied Science Research, 15(6), 29334-29340.
29. Jagadeesh, S., & Sugumar, R. (2017). A Comparative study on Artificial Bee Colony with modified ABC algorithm. European Journal of Applied Sciences, 9(5), 243-248.
30. Varma, K. K., & Anand, L. (2025, March). Deep Learning Driven Proactive Auto Scaler for High-Quality Cloud Services. In International Conference on Computing and Communication Systems for Industrial Applications (pp. 329-338). Singapore: Springer Nature Singapore.
31. Kumar, S. A., & Anand, L. (2025). A Novel EEG-Based Deep Learning Framework for Enhancing Communication in Locked-In Syndrome Using P300 Speller and Attention Mechanisms. KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS, 19(11), 3841-3855.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802312

32. Poornima, G., & Anand, L. (2025). Medical image fusion model using CT and MRI images based on dual scale weighted fusion based residual attention network with encoder-decoder architecture. Biomedical Signal Processing and Control, 108, 107932.
33. Archana, R., & Anand, L. (2025). Residual u-net with Self-Attention based deep convolutional adaptive capsule network for liver cancer segmentation and classification. Biomedical Signal Processing and Control, 105, 107665.Kumar, S. A., & Anand, L. (2025). A Novel EEG-Based Deep Learning Framework for Enhancing Communication in Locked-In Syndrome Using P300 Speller and Attention Mechanisms. KSII Transactions on Internet and Information Systems, 19(11), 3841-3855.
34. Rengarajan, A. (2025). Cloud-Based AI-Driven Threat Detection Framework for Smart Grid Cybersecurity. International Journal of Future Innovative Science and Technology, 8(6), 16065.
35. Murugeshwari, B., Sudharson, K., Panimalar, S. P., Shanmugapriya, M., & Abinaya, M. (2020). SAFE–Secure Authentication in Federated Environment using CEG Key code.
36. Raj A. A., & Sugumar, R. (2023). Early Detection of COVID-19 with Impact on Cardiovascular Complications using CNN Utilising Pre-Processed Chest X-Ray Images. 2023 International Conference on Applied Intelligence and Sustainable Computing (ICAISC), IEEE.
37. Jagadeesh, S., & Sugumar, R. (2017). A Comparative study on Artificial Bee Colony with modified ABC algorithm. European Journal of Applied Sciences, 9(5), 243-248.
38. Selvi, G. V., Anbarasan, A. B., Murthy, B. A., & Prabavathy, S. (2023). An Application Oriented Integrated Unequal Clustering Algorithm for Wireless Sensor Network. In Underwater Vehicle Control and Communication Systems Based on Machine Learning Techniques (pp. 140-154). CRC Press.
39. Sruthi, R. S., Ananya, S., & Murugeshwari, B. (2010). Web Based Virtual Control System Laboratory and On-Line Temperature Control of Electrophoresis Equipment using LabVIEW. International Journal of Computer Applications, 975, 8887.
40. Vimal Raja, G. (2021). Mining Customer Sentiments from Financial Feedback and Reviews using Data Mining Algorithms. International Journal of Innovative Research in Computer and Communication Engineering, 9(12), 14705-14710.
41. MATHEW, A. R. (2025). Neurosecurity and Brain-Computer Interfaces.
42. Soundappan, S. J. (2024). AI-Driven Customer Intelligence in Enterprise Lakehouse Systems Sentiment Mining Governance-Aware Analytics and Real-Time Data Synchronization. International Journal of Advanced Engineering Science and Information Technology (IJAESIT), 7(5), 14905.
43. Mathew, A. (2025). Human–AI Collaboration in Security Operations: Measuring Alert Trust, Automation Bias, and Analyst Upskilling in AI-Augmented SOC Environments. International Journal of Computer Technology and Electronics Communication, 8(5), 11375-11380.
44. Soundappan, S. J. (2022). AI-Based Fault Detection and Isolation for Reliability in Modern Power Systems. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 5(4), 7106-7110.
45. Poornima, G., & Anand, L. (2024, April). Effective Machine Learning Methods for the Detection of Pulmonary Carcinoma. In 2024 Ninth International Conference on Science Technology Engineering and Mathematics (ICONSTEM) (pp. 1-7). IEEE.Garg, V. K., Soundappan, S. J., & Kaur, E. M. (2020). Enhancement in intrusion detection system for WLAN using genetic algorithms. South Asian Research Journal of Engineering and Technology, 2(6), 62–64.
46. Rengarajan, A., Jayakumar, C., & Sugumar, R. (2012). Optimization Of Recent Attacks Using Internet Protocol. National Journal of System and Information Technology, 5(1), 8.
47. Mathew, A. (2024). AI TRiSM: Trust, Risk, and Security Management in Cybersecurity. Cybersecurity, 4(3), 84-90.
48. Mathew, A. (2025). Deep seek vs. ChatGPT: A deep dive into AI Language mastery. Int J Multidisciplinary Res, 7(1), 1-5.