Mitigating Unbounded Data Growth in Conversational Systems Using Tiered Retention and Compression

Mahwish Dadan
First Year Graduate Student,
M.Sc. Big Data Analytics,
St. Xavier's College (Empowered Autonomous Institute),
Mumbai, Maharashtra, India
mahwishdadan13@gmail.com

***Abstract*— The rapid growth of real-time systems and its user-generated data, especially chat logs, poses significant challenges in terms of storage, scalability, and system performance. This paper proposes a data management pipeline to manage unbounded data growth using a combination of deduplication, tiered retention, summarization, and compression. We used the Ubuntu Dialogue Corpus for our solution. The system classifies data into hot, warm, and cold storage tiers based on age, compressing older logs with Gzip and eliminating redundant entries to reduce footprint. Our results show that this approach not only lowers storage costs but also retains the accuracy for downstream analysis, making it practical for large-scale, real-world deployments.**

Keywords—Unbounded data, compression, algorithms, chat logs

I. Introduction

Over the past twenty years, the way we generate, share, and consume data has changed dramatically. Thanks to the explosion of digital devices, improvements in internet infrastructure, and the rise of cloud computing, the amount of data produced worldwide has reached levels that would have seemed unimaginable at the dawn of the digital age. The move from analogue to digital storage has made it possible to handle and process data on a much larger scale. This shift is closely tied to the emergence of the "Big Data" era, which has enabled the large-scale data analysis that powers many of today's technologies [1].

But it's not just better storage that's behind this data growth. The widespread adoption of connected devices, the rapid growth of the Internet of Things (IoT), and the increasing digital transformation across industries like healthcare, finance, education, entertainment, and scientific research also play a huge role. Everyday items, from wearable devices monitoring our health to industrial machines tracking performance, generate continuous streams of real-time data. Users themselves create vast amounts of multimedia content every day. Because of this, data can be either bounded or unbounded. Bounded data has a clear beginning and end, making it easier to handle in batches. On the other hand, unbounded data keeps flowing endlessly, requiring real-time processing through streaming systems.

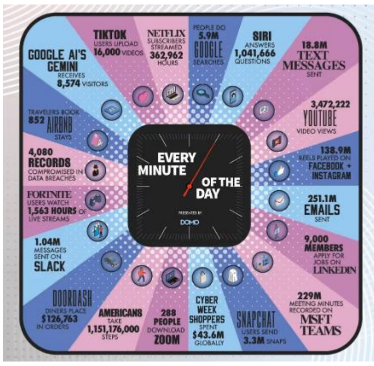

Fig. 1. Data generated in a minute on the Internet by Domo[2]

As a result, streaming platforms have become vital for real-time data processing and delivery. Services like Netflix and YouTube, as well as scientific projects such as CERN's Large Hadron Collider (LHC), produce and manage staggering volumes of information.. For example, Netflix alone accounted for over 51 exabytes of streaming traffic in 2021, while the LHC's distributed computing grid transmitted nearly 2 exabytes annually to support cutting-edge physics research [1]. According to Domo's 2024 infographic (Figure 1), the use of cloud storage continues to grow across a wide range of applications, reflecting the increasing reliance on cloud infrastructure. The total amount of data generated, captured, replicated, and consumed worldwide across digital platforms and systems is referred to as the Global DataSphere. The International Data Corporation (IDC) conducts a five-year forecast for the worldwide IDC Global DataSphere [3]. According to its 2020 report, IDC predicted that the Global Datasphere will grow from 33 Zettabytes (ZB) (1 ZB = 1021 Bytes or 270 Bytes) in 2018 to 175 ZB by 2025, as shown in Figure 2.

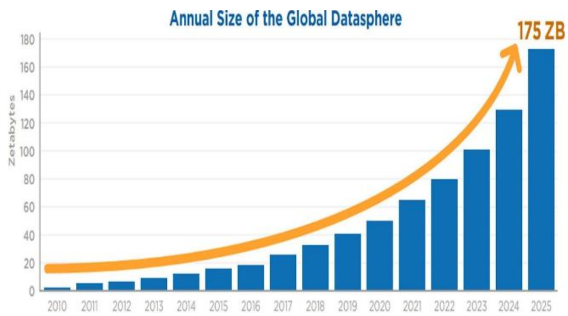

Fig. 2. Annual Size of Global Datasphere according to IDC 2020 Report [3]

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

In recent years, research on streaming systems has largely centred around improving real-time performance, scalability, and reducing latency, all while keeping costs in check. But as the Internet of Things (IoT) and Big Data technologies continue to evolve, one area that hasn't received as much attention is space complexity. As data volumes grow, efficient use of memory and storage becomes increasingly important—especially in environments where resources are limited, like edge devices with constrained bandwidth, memory, and processing power. In such cases, managing space complexity isn't just a nice-to-have; it's essential to ensuring long-term system sustainability. In this paper, we aim to study the existing compression techniques and propose a real-time data management pipeline that integrates lightweight filtering, compression, deduplication, and tiered data retention to manage space complexity in streaming systems. Our goal is to reduce storage overhead while maintaining analytical readiness for unbounded data streams.

II. Literature Review

In this section, I provide an overview of the existing techniques used to manage space complexity. These approaches help control data growth by either limiting the data stored or minimizing its size through various transformation methods. They can be categorized into policy-based strategies, such as data retention and deletion, and system-level techniques, including log rotation, deduplication, compression, and real-time summarization.

A. Data Retention and Deletion Policies

One way of the most common way to manage unbounded data growth is retention-driven and on-demand deletion mechanisms. These aren't just good practices, they are required by legal and regulatory requirements aimed at protecting user privacy. For example, the European Union's General Data Protection Regulation (GDPR) introduced the "right to be forgotten," which gives individuals the ability to request permanent deletion of their personal data, unless there are legal grounds to retain it. Similarly, U.S.-based laws like the California Consumer Privacy Act (CCPA) and the Virginia Consumer Data Protection Act (VCDPA) require companies to delete user data within specific timeframes—usually within 30 to 45 days. Failing to comply can result in heavy penalties [4].

These regulations don't apply to only active databases but also extend to backups and archived data, which creates technical challenges. For instance, systems built on immutable storage formats often don't allow for easy, in-place deletion. In such cases, data has to be rewritten or reorganized, which can be both time-consuming and computationally expensive. Even in systems that do support deletion, it's common to rely on "logical deletes," where data is simply marked as deleted but still physically exists in storage. This can lead to inefficient storage use, bloated indexes, and potential privacy risks if that data remains accessible [4]. In high-volume systems, these inefficiencies can scale quickly. In high-volume systems, these problems can add up fast. That's why it's important to design deletion-aware architectures that can enforce retention policies efficiently. At the same time, technical strategies like compression, deduplication, and real-time summarization can be used to reduce storage pressure before deletion even becomes necessary.

B. Data Compression

Data compression is widely used techniques for managing storage and bandwidth usage in systems that handle large volumes of data. By reducing the size of data before storage or transmission, compression helps to optimize resource usage without necessarily losing important information. The basic flow of a data compression system is shown in Figure 3.

In a typical compression process, the algorithm processes the input data in two main stages. First, it scans the data to identify recurring patterns or statistical properties. These insights are then used to encode the data more efficiently during the second stage. The compressed output, often accompanied by necessary metadata, can then be stored or transmitted. On the receiving end, a decompression algorithm reconstructs the original data using this metadata and the compressed file. In this way, data compression algorithms help reduce the bandwidth usage of a network and the storage requirements. However, it is important to note that the process itself consumes computational resources. This becomes especially relevant in low-resource environments such as embedded systems or edge devices, where processing power and memory are limited. Hence, there arises a need for a good compression algorithms that strike a balance between the compression ratio, speed, and system resource usage [5].

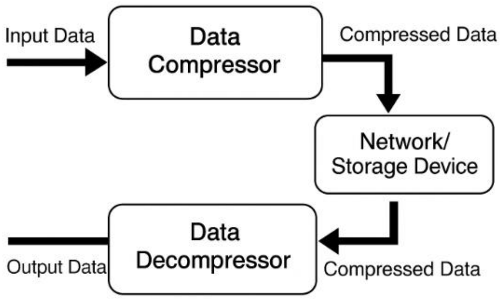

Fig. 3. *Components of a Data Compression System*

Compression algorithms generally fall into two main categories: lossless and lossy. Lossless compression algorithms reconstruct the original message exactly as it is from the compressed message with no loss of information. This type of algorithm is preferred for text files or critical data where precision is required and can otherwise lead to bad decisions, serious consequences, or failures. Two common types of lossless compression techniques include: those based on statistical modelling, such as Huffman coding, that assigns shorter codes to frequently occurring symbols, and dictionary-based methods like the Lempel-Ziv (LZ) family of algorithms, which replace repeated patterns with reference to earlier occurrences [6]. Some widely used compression tools and algorithms include GZIP, LZ4, ZSTD, ZIP, PNG, etc.

- GZIP is based on 'DEFLATE', which is a lossless data compression algorithm that uses a combination of LZ77 (a dictionary-based algorithm) and Huffman coding (an entropy-based algorithm). It supports both static and dynamic Huffman encoding based on the nature of the data: static for real-time compression and dynamic for non-real-time compression. It is widely used for different web servers such as Apache, but it hasn't quite achieved high performance for real-time data [7].

- LZ4 is a high-speed, lossless compression algorithm introduced in 2011. It is derived from LZ77 but uses optimized encoding to reduce processing time, making it ideal for real-time or low-latency systems [6].
- Brotli is a lossless, general-purpose compression algorithm developed by Google in 2013, designed for high-density web data compression. Due to its performance, it is now commonly used in web browsers like Chrome and Firefox, as well as in HTTP servers like Apache and IIS [8].
- Other notable algorithms include Snappy, a lightweight compression method developed by Google researchers, known for its fast decompression and an average compression ratio, and ZSTD, developed by Facebook researchers, which offers faster compression and decompression, though with a slightly lower compression ratio [9].

In contrast, the Lossy algorithm can only reconstruct an approximation of the original message. It removes unnecessary data permanently, so the original data cannot be completely regenerated. For instance, image and audio files usually contain some random noise that has high information. Other times, there are certain features in images or sound files that may be undetectable to humans (such as very high or very low frequencies, similar color information, etc). Dropping such information can significantly reduce file size with no significant reduction in quality. Lossy compression algorithms typically achieve up to twice the compression ratio of lossless methods on media files like images or audio, with minimal or undetectable quality loss [10]. Some widely used lossy compression algorithms include JPEG, MP3, AAC, H.264, HEVC, Opus, WebP, and the VP9 codec. which is used by platforms like Netflix for streaming video.

C. Log rotation/Pruning

Many modern systems, such as operating systems, application servers, and cloud platforms, generate a massive volume of log data every day. These logs capture detailed information about how the system runs and often need to be kept for a long time, sometimes up to a year, to help identify problems, monitor security, and analyse system behaviour [11]. Log rotation and pruning is often combined with compression to manage this growing volume of data.

Log rotation is the process of archiving logs on a server while retaining only the recent logs on the client, typically from the last few days or weeks. This approach not only saves disk space but makes searching through logs easier [12]. However, archiving logs can take significant storage space, resulting in increased operational costs. To address this issue, compression algorithms are used. However, traditional compression tools such as gzip are not specifically designed for system logs. Unlike general text documents, log files follow a fixed structure with a repeatable sequence of tokens such as date, time, IP addresses, etc., along with separators [13]. This redundancy specific to log files can be utilized for efficient log compression. Logzip [11] is one such log compression approach that utilizes log structure through an iterative clustering technique, reducing operational costs for log storage. It enables easy integration with other data compression tools and generates a semi-structured representation that can be used for future log mining tasks.

Log retention is the practice of storing logs according to rules set by organizations or regulations [14]. Log pruning follows these rules by deleting old logs once they are no longer needed. Different regulatory frameworks have different minimum retention periods based on the log types; for example, HIPAA typically requires data to be kept for up to 6–7 years. Depending on how critical a system is, the retention settings can vary, some systems need logs kept longer, rotated more often, and analysed more frequently. High-impact systems might rotate logs every few hours and send them off for analysis every few minutes, which can create very large amounts of log data and duplicates over time.

D. Data Deduplication

In large systems, data duplication happens frequently because logs are repeated, entries overlap, and multiple backups or archives are created. While backups are important for recovery and fault tolerance, they can waste a lot of storage space by keeping identical copies. Data deduplication helps solve this by finding and removing redundant data chunks. It compares new data to existing data (often using hashes) to spot duplicates, then replaces the repeated data with pointers to a single copy. This saves storage space without losing any important information or breaking backups [15].

E. Real-time summarization

Devices with limited resources, like wireless sensors or edge devices, often can't send or store large amounts of data. Much of the data they collect can be repetitive. For example, if a sensor reports the same value over time, instead of sending every reading, it can send just the average value, how many times it was recorded, and a measure of variation (which might be zero if all readings are identical). This kind of summarization helps reduce storage and communication needs while still keeping the main insights intact without significant loss of information [16]. Various aggregation techniques, such as statistical summaries and window-based aggregation, are widely used in modern data processing systems.. Typical statistical summaries include the mean, median, standard deviation, minimum, and maximum, among others.

In Sliding window, data is grouped and summarized over a moving time window. Traditional methods works well for single-item inserts and deletes, even for slightly out-of-order data. However, they struggle with real-world streams that come both in bursts and highly out-of-order. A recent study [17] introduces has introduced new algorithms that handle bulk updates and out-of-order data more efficiently, improving both performance and accuracy in real-time data streams.

III. Proposed Approach

To tackle the problem of unchecked data growth in chat-based systems, we built a layered, real-time data management pipeline aimed at reducing storage use while maintaining data integrity and analytical value. The system processes a live stream of messages by using deduplication, tiered retention, compression, and summarization.

1. Deduplication: The system first removes redundant messages as they are received. In many chat systems, users often send repeated or identical messages, which unnecessarily increase the size of log files. A simple cache detects and eliminates duplicates on the spot.

2. Tiered Data Retention: he system sorts data into three storage layers - hot, warm, and cold, each reflecting a different stage of the data lifecycle. Recent logs are kept in the hot tier for active use and then gradually moved to warm and cold tiers as they get older. This model draws inspiration from industry practices in log management, where recent data is prioritized for access while older data is archived.

3. Compression for Cold Data: When messages reach the cold tier, they are compressed using the Gzip algorithm. Gzip strikes a good balance between speed and compression efficiency, making it suitable for archiving logs without heavy processing demands.

4. Summarization and Visualization: To support monitoring and analysis, the system periodically generates summaries like message volume per hour which are saved as CSV files and visualized using simple plots and charts.

Overall, this approach shows how various lightweight techniques, when combined in an organized pipeline, can effectively control storage costs in systems that manage continuous, high-volume text data.

IV. Methodology & Implementation

The implementation was done in Python using real-world data from the Ubuntu Dialogue Corpus, a large collection of multi-turn chat dialogues related to technical support. The system consists of five main components, each focusing on a specific part of the data handling process:

A. Data Ingestion

The dataset is stored across three CSV files. These files are read line by line to mimic real-time message streaming. Each message includes metadata such as timestamp, sender, receiver, and text content. As the system processes messages, each message is stored as an individual JSON file in the 'hot_chat_logs' folder.

B. Deduplication:

To prevent unnecessary storage of duplicate content, the system compares each new message against a cache of recent entries. If the same user has sent the same message before, it is ignored. This step helps cut down on storage use and prepares the data for better compression.

C. Tiered Retention

Messages are automatically moved through a storage hierarchy based on age:

1. Hot Tier: Active logs, stored for a short duration.
2. Warm tier: Logs that are older but not yet archived.
3. Cold tier: Logs ready for compression and long-term storage.

A timestamp-based policy is used to decide when files will be moved between tiers, ensuring that storage space is not overwhelmed by old logs.

D. Compression

Once files enter the cold tier, they are compressed using Gzip. This step significantly cuts their size, often by 80 to 90% without losing any content. Compressed files are saved in a 'compressed_logs' directory for archival access.

E. Summarization & Visualization

The system tracks how many messages are processed over time and creates hourly summaries. These statistics are stored in CSV format and visualized using line charts to show patterns such as usage spikes or quiet periods. This provides a lightweight monitoring layer without the needing to scan raw logs.

Components such as the deduplication policy, compression method, or retention schedule can be adjusted to match the needs of different deployments. Overall, this implementation offers a practical solution to the managing large volumes of real-time chat data efficiently.

V. Evaluation And Discussion

The tiered data management pipeline introduced in this work was evaluated mainly on its ability to control data growth while maintaining accessibility and efficient storage. The effectiveness of the tiered data management approach can be seen in several key areas:

A. Space Savings and Efficiency

The system demonstrated substantial reductions in storage usage by combining deduplication, tiered retention, and compression. Removing duplicate messages during ingestion stopped unnecessary data duplication early, directly lowering the volume of logs stored in the "hot" tier. As data aged, moving logs through the hot, warm, and cold tiers allowed for better use of storage by gradually offloading less frequently accessed data.

Compression in the cold tier reduced the disk space, with Gzip achieving compression ratios between 80% and 90%. This tiered approach balances strikes a balance between quick access to recent data and efficient archiving of older records.

B. Performance and Scalability

The design emphasizes simplicity and modularity, allowing smooth operation on a real-world dataset without excessive resource use. However, the unlimited deduplication cache poses a potential scalability risk, as its memory use increases over time. Implementing limits or expiration policies for this cache is crucial for long-term deployment.

Retention policies based on fixed time intervals for moving data between tiers were effective but somewhat rigid. In real systems, adjusting these thresholds based on storage capacity, system load, or user access patterns could enhance performance and resource use.

C. Data Accessibility and Usability

Keeping recent logs readily available in the hot tier supports immediate analysis and troubleshooting, while compressed archives still allow retrieval of historical data when needed. Generating lightweight hourly summaries provided meaningful insight into message volumes without scanning entire datasets, though at the cost of losing granular detail

The implementation of a lightweight filtering mechanism to remove exact duplicates within each chunk slightly boosted overall compression performance. While the number of redundant messages was relatively low due to the random nature of the synthetic dataset, this step is still beneficial for real-world datasets where repetition is common, such as system logs or telemetry data. Furthermore, the filtering stage acted as an important preprocessing step that reduced the input volume sent to the compression engine, thereby decreasing compression time and output size. However, the improvements were not significant in this case due to the limited redundancy in the generated input.

VI. LIMITATIONS

Some limitations that were identified during the course of the study are as follows:

- The compression step introduces CPU overhead, which could limit throughput in high-velocity data streams.
- Deduplication only considers exact duplicates, which may miss near-duplicates or semantically similar messages, leaving room for further optimization.
- Summarization reduces the volume of stored data but sacrifices detail, which may not suffice for all analytical needs.

VII. FUTURE DIRECTIONS

While the current tiered system provides a solid foundation for managing growing chat data, there are several ways in which the current solution can extend the applicability and effectiveness of the proposed solution:

- Adaptive Strategies: Instead of fixed time intervals, data movement across tiers could be based on storage capacity, message frequency, or user behaviour.
- Improve Deduplication process: Introducing approximate or fuzzy matching could help detect near-duplicate messages and reduce storage further.
- Compression-Aware Storage Formats: Columnar file formats like Parquet or ORC can further reduce space usage [18]. These formats are designed for structured data and provide built-in compression and encoding optimizations suitable for large-scale analytics workloads.
- Advanced Compression Techniques: While gzip works well, other algorithms could offer better speed or compression ratios depending on the use case. A hybrid approach that combines compression with summarization or lossy techniques could provide even greater space savings for less critical data.
- Hybrid Compression Pipelines: combining summarization (e.g., statistical aggregation) with traditional compression could reduce space even further for non-critical logs.
- Integration with Log Management Systems: Integrating this system with existing logging platforms could enhance usability, enabling alerts, dashboards, and deeper analysis. This could include adding automated alerts when storage limits are close, or more detailed analytics built on the summaries generated by the system.

With these improvements, the tiered data management approach can become more flexible, efficient, and scalable, better suited to handle the growing volumes of conversational data found in modern applications

VIII. CONCLUSION

This project demonstrated a practical way to manage data growth in chat systems by using a layered storage model. By combining deduplication, tiered retention, compression, and summarization, it reduced storage needs while allowing access to important data. Recent logs are available for immediate use, while older logs are compressed for storage. Hourly summaries offer a lightweight monitoring option without the need for full log scans. Although the system is simple, it works well with real-world data and provides a solid base for future improvements. In summary, the project illustrates how simple, modular techniques can create a scalable and efficient storage pipeline for ongoing conversational data streams.

REFERENCES

[1] L. Clissa, M. Lassnig, and L. Rinaldi, “How big is Big Data? A comprehensive survey of data production, storage, and streaming in science and industry,” Front. Big Data, vol. 6, Oct. 2023, doi: 10.3389/fdata.2023.1271639.

[2] “Data Never Sleeps 12.0.” Accessed: Sept. 21, 2025. [Online]. Available: https://www.domo.com/learn/infographic/data-never-sleeps-12

[3] “Worldwide IDC Global DataSphere Forecast, 2025-2029.” Accessed: Sept. 21, 2025. [Online]. Available: https://www.giiresearch.com/report/id1726448-worldwide-idc-global-datasphere-forecast.html

[4] M. Athanassoulis, S. Sarkar, T. I. Papon, Z. Zhu, and D. Staratzis, “Building Deletion-Compliant Data Systems”.

[5] A. Gupta, A. Bansal, and V. Khanduja, “Modern lossless compression techniques: Review, comparison and analysis,” in 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore: IEEE, Feb. 2017, pp. 1–8. doi: 10.1109/ICECCT.2017.8117850.

[6] W. Liu, F. Mei, C. Wang, M. O’Neill, and E. E. Swartzlander, “Data Compression Device Based on Modified LZ4 Algorithm,” IEEE Trans. Consumer Electron., vol. 64, no. 1, pp. 110–117, Feb. 2018, doi: 10.1109/TCE.2018.2810480.

[7] A. Shah and M. Sethi, “The Improvised GZIP, A Technique for Real Time Lossless Data Compression,” EAI Endorsed Transactions on Context-aware Systems and Applications, vol. 6, no. 17, p. 160599, June 2019, doi: 10.4108/eai.1-10-2019.160599.

[8] J. Alakuijala et al., “Brotli: A General-Purpose Data Compressor,” ACM Trans. Inf. Syst., vol. 37, no. 1, pp. 1–30, Jan. 2019, doi: 10.1145/3231935.

[9] C. Zhu, B. Han, and G. Li, “PAC: A monitoring framework for performance analysis of compression algorithms in Spark,” Future Generation Computer Systems, vol. 157, pp. 237–249, Aug. 2024, doi: 10.1016/j.future.2024.02.009.

[10] M. Hosseini, “A Survey of Data Compression Algorithms and their Applications,” Feb. 20, 2025, arXiv: arXiv:2506.10000. doi: 10.48550/arXiv.2506.10000.

[11] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, “Logzip: Extracting Hidden Structures via Iterative Clustering for Log Compression,” Sept. 24, 2019, arXiv: arXiv:1910.00409. doi: 10.48550/arXiv.1910.00409.

[12] I. Journal, “IRJET-IMPORTANCE OF CENTRALIZED LOG SERVER AND LOG ANALYZER SOFTWARE FOR AN ORGANIZATION”, Accessed: Sept. 21, 2025. [Online]. Available: https://www.academia.edu/14385368/IRJET_IMPORTANCE_OF_CENTRALIZED_LOG_SERVER_AND_LOG_ANALYZER_SOFTWARE_FOR_AN_ORGANIZATION

[13] P. Skibiński and J. Swacha, “Fast and efficient log file compression”.

[14] A. K. Kent (NIST) and A. M. Souppaya (NIST), “SP 800-92, Guide to Computer Security Log Management.” Accessed: Sept. 21, 2025. [Online]. Available: https://csrc.nist.rip/publications/detail/sp/800-92/final
[15] “(PDF) Data deduplication techniques.” Accessed: Sept. 21, 2025. [Online]. Available: https://www.academia.edu/2091478/Data_deduplication_techniques
[16] T. He, L. Gu, L. Luo, T. Yan, J. A. Stankovic, and S. H. Son, “An Overview of Data Aggregation Architecture for Real-Time Tracking with Sensor Networks”.
[17] K. Tangwongsan, M. Hirzel, and S. Schneider, “Out-of-Order Sliding-Window Aggregation with Efficient Bulk Evictions and Insertions (Extended Version),” Proc. VLDB Endow., vol. 16, no. 11, pp. 3227–3239, July 2023, doi: 10.14778/3611479.3611521.
[18] “Data formats in analytical DBMSs: performance trade-offs and future directions | The VLDB Journal.” Accessed: Sept. 21, 2025. [Online]. Available: https://link.springer.com/article/10.1007/s00778-025-00911-1?utm_source=chatgpt.com