ViSemZ: High-performance Visual Semantics Compression for AI-Driven Science

Anonymous Author(s)

Abstract

Scientific images are essential in many experimental sciences, but the large volumes of data present significant challenges. Effective image compression must be fast, achieve high compression ratios, and preserve important domain-specific features. Existing compressors, such as JPEG or SZ, can distort critical textures at high compression ratios. AI-based compressors, on the other hand, offer excellent image quality and high compression ratios but are significantly slower than traditional methods. To address this discrepancy, we designed ViSemZ, a high-performance AI-based compressor that preserves visual semantics. Our method enhances AI compression by incorporating sparse encoding with varied-length integer truncation, optimized lossless encoding using bitshuffle and decoupled lookback prefix-sum, and pipelining for efficient data streaming and asynchronous processing. Evaluations on general and scientific datasets demonstrate that, under similar compression ratios, ViSemZ preserves the best image quality while achieving a 1.9× speedup (vs. nvJPEG) compared to non-AI-based compressors, and performs almost on par with AI-based compressors while delivering a 9.6× overall compression speedup. This effectively bridges the performance gap between traditional and AI-based compression methods.

1 Introduction

Modern scientific applications generate enormous volumes of image data for post-analysis across various domains such as microscopy [2, 66], astronomy [5], medical imaging [36], climate science [65], and earth observation [11]. This large volume of data creates a significant bottleneck for high-performance computing (HPC) systems. To address this issue, efficient data reduction techniques are essential. Although lossless compression methods preserve image quality, their compression ratio is extremely limited, typically around 2:1 [12]. Conversely, lossy compression methods offer significantly higher compression ratio by introducing controllable errors to the data, making them the preferred choice for big data compression in HPC [9, 20, 25, 28, 29, 52, 57–60].

1.1 Motivation for High Quality Compressors

Recently, there has been an increasing demand for high-quality lossy compressors in scientific applications. This demand is particularly driven by the requirements of deep learning-based post-analysis, which necessitates higher image quality to achieve greater accuracy [8, 62]. For instance, in materials science, the texture information within images

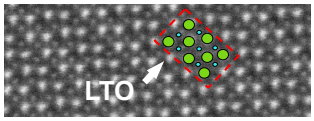

Figure 1. The identification of LTO in a material [30].

is crucial for identifying the type of material. Figure 1 shows a microscope image of Lithium Titanium Oxide (LTO) in a material. It is evident that texture information is essential for identifying the characteristics of specific materials, highlighting the importance of compressors that can preserve high texture quality. Similarly, in medical imaging, such as X-ray images, detailed texture information can significantly influence the final diagnosis [15]. These practical demands drive researchers to explore high-quality texture compressors.

1.2 Limitations of Existing Compressors and Goal

Classical image compressors use handcrafted transformations like Discrete Cosine Transform (DCT) in JPEG [56] to preserve low-frequency information, but these non-optimal transformations often result in artifacts in the reconstructed image. Most current scientific lossy compressors, such as SZ [34] and ZFP [35], designed for scientific data, fail to provide high image quality at high compression ratios. The absolute error control schemes used by these scientific compressors tend to introduce texture defects in images and lack effective support for an image's integers.

Recently, AI-based image compressors [3, 4, 13, 26, 31, 38, 61] have excelled in preserving image quality, especially at high compression ratio. AI-based compressors consist of two components: the learned-model transformation and lossless encoding. However, despite advancements in model structures to better retain image information, the lossless encoding process remains time-consuming and has been largely overlooked in the development of AI-based compressors. This results in AI-based compressors being significantly slower (over 10× slower according to evaluation) and thus impractical for many scientific domains necessitating low latency/instant feedback. We provide a detailed comparison of the advantages and disadvantages of these compressors

Table 1. Comparison of different types of compressors

Existing GPU Lossy Compressors	Good Quality In High Ratio?	High Throughput?	Keep Detail Texture?
Scientific Compressors	✗	✓	✗
Image Compressors	✓	✓	✗
AI-based Compressors	✓	✗	✓
ViSemZ (Ours)	✓	✓	✓

in Table 1. An ideal lossy compressor should satisfy the following criteria: (1) Extreme quality preservation; (2) High throughput that can saturate the overall data-intensive workflow; and (3) Ability to preserve details, such as texture, even at high compression ratios.

1.3 Our Solution: VISEMZ

To this end, we design VISEMZ, a high-performance AI-based compressor framework that preserves visual semantics, such as image textures, while delivering a high compression ratio. We address three primary challenges in our design: (1) **Irregular Sparsity in Inference Output.** The model inference output tends to be extremely irregular and sparse at the bit level, especially in high compression ratio cases. The rANS[22, 48] lossless encoding from State-of-the-art (SOTA) AI-based compressors struggle to handle this data feature efficiently. To overcome this, we propose a fast irregular redundancy removal process that efficiently pre-encodes this type of data. (2) **Extremely Slow CPU Lossless Encoding.** CPU-based lossless encoding, commonly used by AI-based compressors, suffers from poor performance due to the inherent slowness caused by sequential data dependency processing and GPU-CPU data transfer, as the inference output is located on the GPU. While a GPU-based lossless encoder can address these issues, it is challenging to design. The recurrence of memory offsets among thread blocks significantly reduces throughput, and the data partitioning strategy lowers the compression ratio. We introduce an improved GPU lossless encoding method that incorporates bitshuffle and a decoupled lookback prefix-sum to enhance performance while maintaining a high compression ratio. (3) **Sequential Data Processing.** Scientific discovery requires rapidly analyzing many images in a time-sensitive workflow. The dependency between different processes in a multi-image workflow limits concurrency in compression. We propose a pipelining optimization to decouple these dependencies, coordinate data streaming and compression workflows, enable asynchrony between CPU and GPU kernels, and reduce both I/O and memory footprint.

The main contributions of this paper are listed below:

- VISEMZ, the first high-performance compressor designed to preserve image visual semantics like texture.
- Design of an efficient irregular bit-level sparsity pre-encoding process with varied length integer truncation; a fast GPU lossless encoding specifically for AI-based compressors with bitshuffle and decoupled lookback prefix-sum; and a pipeline workflow that enables concurrent data loading and processing.
- Evaluation on several representative scientific and real-world datasets demonstrate that VISEMZ significantly improves compression throughput by 9.6× while preserving the same reconstructed data quality with similar compression ratios as other AI-based compressors. Compared to non-AI-based compressors, VISEMZ delivers the highest visualization and texture quality in similar compression ratios, as evidenced by scalar metrics such as PSNR, SSIM, and DISTS [21] while maintaining high performance with 1.9× speedup compared to nvJPEG.

2 Pinpointing Causes of Low Throughput in AI-based Compressors

AI-based compressors typically consist of three main components: pre-trained model inference, integer quantization, and lossless encoding, as illustrated in Figure 3. The model inference processes input images and outputs a multi-dimensional tensor. This is followed by quantization, which converts the floating-point values in the tensor to integers. Finally, the quantized data is moved from GPU memory to CPU memory, and lossless encoding concludes the compression process.

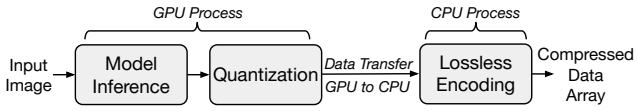

Figure 3. Demonstration of the AI-based compression workflow.

Model inference is the most computationally intensive part of the compression process, contributing to most of the compression time. However, recent developments have led to more lightweight and robust models that achieve comparable or superior performance with simplified structures. Additionally, advancements in GPUs have resulted in faster model inference in recent years [7, 49]. Figure 4 shows the increase in compression and decompression throughput on various GPUs over the past few years. Consequently, model inference is no longer the primary bottleneck for AI-based compressors and will continue to improve.

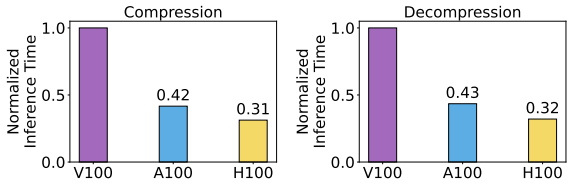

Figure 4. Compression and decompression time speedups on evolving generations of GPUs over recent years.

Despite significant efforts on model improvement, lossless encoding has received less attention. Both the earliest model [3] and the state-of-the-art model [38] use the same CPU implementation [22, 23] of rANS encoding [48]. The rANS encoding requires a counting process to calculate the appearance count for each input symbol (denoted as freq), and the cumulative distribution function (CDF) by cumulative summation of freq. The input data stream is then encoded in an arithmetic way by each symbol:

$$x_{n+1} = \left\lfloor \frac{x_n}{\text{freq}[s]} \right\rfloor \times \sum \text{freq} + \text{CDF}[s] + x_n \mod \text{freq}[s]$$

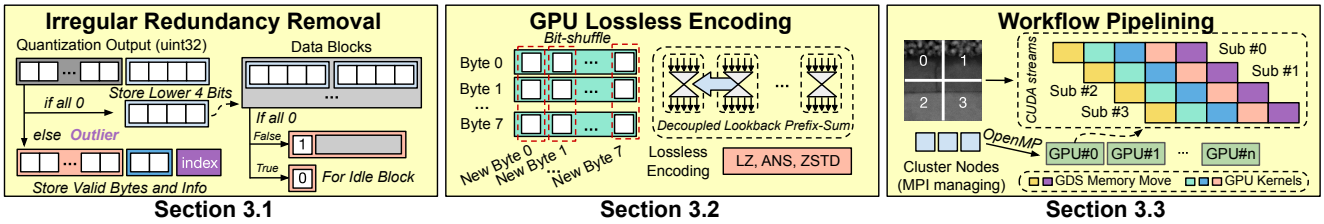

Figure 2. An abstraction of the key ideas in each section.

The s denotes the current encoding symbol. The x denotes the encoded number. Although delivering a high compression ratio, the rANS encoding has dependencies between each input data, which makes this a sequential process. This creates a bottleneck in the overall compression and decompression workflow. The overhead of lossless encoding surpasses that of model inference, even though it is significantly less computation-intensive. Moreover, CPU-based lossless encoding requires data transfer from GPU to CPU to process the model inference output located in the GPU, further degrading performance. The quantization process, although not time-consuming, imposes extra memory overhead on the GPU due to its multi-kernel nature.

3 Design of VISEMZ

In this study, we introduce VISEMZ, a high-performance AI-based compressor designed to preserve the details in image data exceptionally well. A comparison of VISEMZ with AI-based compressors is demonstrated in Figure 5. VISEMZ holistically compresses or decompresses the output of model inference with kernel fusion technique, thereby mitigating the expensive memory footprint and significantly reducing high-latency global memory accesses.

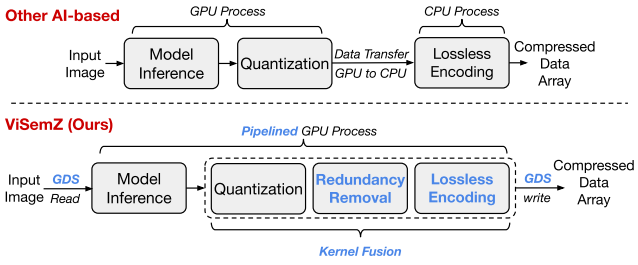

Figure 5. Comparison of VISEMZ with AI-based compressors. The blue text represents our proposed designs.

Overview of VISEMZ's Design. The design of VISEMZ encompasses three key components, each addressing a specific challenge. The irregular redundancy removal, discussed in Section 3.1, tackles Challenge ① **Irregular Sparsity in Inference Output**. The fast GPU-based lossless encoding, presented in Section 3.2, addresses Challenge ② **Extremely Slow CPU Lossless Encoding**. Finally, the efficient workflow pipelining, described in Section 3.3, mitigates Challenge ③ **Sequential Data Processing**. An overview/abstraction of the key ideas in each section is illustrated in Figure 2.

The subsequent subsections provide a detailed explanation of the key design elements in VISEMZ.

3.1 Irregular Redundancy Removal

Motivation for Removing Redundancy. In AI-based compressors, following model inference, the input image is transformed into several feature maps of floats, each containing information from the original image. The quantization process then rounds these floats into integers (quantization codes) for lossless encoding. Our evaluation demonstrates that irregular sparsity is a significant characteristic in the quantization results of AI-based compression model inference. We showcase an example by noting the value in a slice of the feature map in Figure 6. It is evident that most of quantization codes are 0, and other values are integers close to 0, resulting in sparsity at the integer-level and bit-level. In Figure 7, we further extract a data block from the example quantized feature map to illustrate the sparsity at the bit level. Due to the two's complement representation of the negative number, the bit-level sparsity is broken. Therefore, instead of using two's complement to store negative numbers, we propose a sign-shift representation, which saves the absolute value with a 1-bit left shift and moves the sign (0-bit for a positive number and 1-bit for a negative number) to the rightmost bit. It is obvious that the sparsity at the bit-level is non-normally distributed. This data feature challenges the traditional lossless encoding used in state-of-the-art solutions (e.g., rANS in CompressAI [6]) due to their generalized design for normal data. The challenges are (1) the lack of a dedicated design for bit-level data pattern encoding and (2) inefficient sparse data compression.

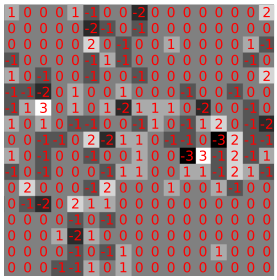

Figure 6. An example of quantized inference output for pixel values.

Varied Length Integer Truncation. To address this issue, we propose a truncation process that reduces data to fewer bits, removing redundancy and saving outliers separately. Outliers are data points that cannot be represented using the truncated form. Unlike existing methods that truncate integers to a fixed bit length and save outliers as full-length

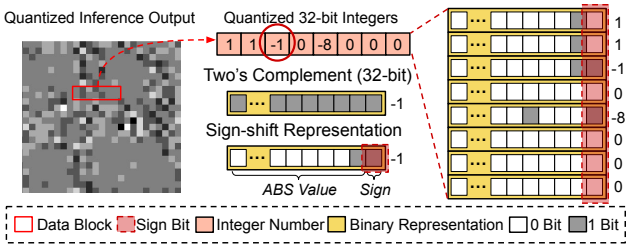

Figure 7. An example of the bits in the quantized inference output (pixels). The block is also used in Figure 8 and Figure 9.

integers, our method uses data-dependent truncation. The length of the truncated bits is decided based on the dataset characteristics. Additionally, we use varied-length outlier storage to avoid wasted bits in compressed data. Figure 8 illustrates our design. Our approach selects the number of truncation bits based on the data distribution. In this example, over 95% of integers fall within the range $[-7, 7]$ and can be represented with 4 bits, including a 3-bit truncation length and a 1-bit sign. We choose a power-of-2 truncation length to align the output as bytes. For outliers, instead of saving the full-length integer, we store only the valid byte(s) and use 2 bits in the index byte to indicate the length of the valid bytes. For example, 00 denotes 1 valid byte, while the remaining 6 bits indicate the index of the outlier within the data block. With this design, we typically use 2 bytes to store an outlier, whereas existing methods require 5 bytes (including 1 extra byte for the index)—more than the integer itself. Additionally, to maintain the fixed length of each data block in the truncated array, we use placeholder bits for the outliers.

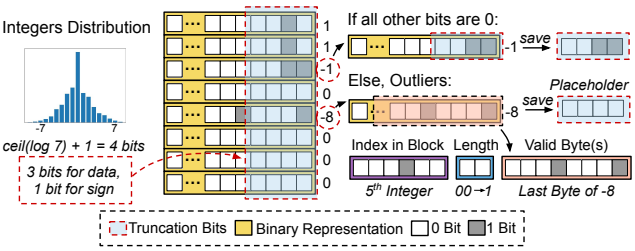

Figure 8. An example of the varied length integer truncation.

Fast Bit-Flag Scanning. While varied-length integer truncation significantly reduces redundant bits, it is inefficient for handling large amounts of 0-value data, as some bits (truncation length) are still preserved for 0 data. Additionally, outliers produce all 0 placeholder bits, as illustrated in Figure 8. To address this, we propose a chunk-based bit-flag scanning approach. Specifically, we first combine truncated integers into bytes, then aggregate these bytes into chunks, and use a single bit to indicate whether the entire chunk is composed of zeros. If the chunk is all zeros, we discard the data and use only the bit to denote it; otherwise, we mark the chunk with a bit-flag and write it back. However, when assigning data chunks to multiple GPU threads, this scan operation requires communication between threads to determine the result, necessitating cross-thread synchronization. This leads to sequential execution and impacts performance. Moreover, the operation involves intensive memory access, which can negatively affect performance due to memory access overhead and increase the memory footprint in the limited GPU shared memory (low-latency GPU memory). To mitigate these issues, we avoid using shared memory and instead use registers to temporarily store data. We map each data chunk to a warp and utilize the warp-level function *__any_sync* to facilitate direct communication between threads, thereby optimizing performance without incurring additional memory usage.

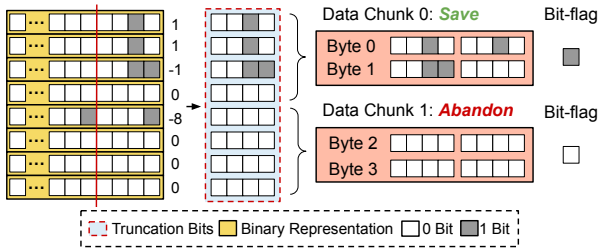

Figure 9. An example of the fast bit-flag scanning.

3.2 GPU-based Lossless Encoding

Sequential Nature of Existing AI-based Compressors' Lossless Encoding. Current SOTA AI-based compressors, following the approach established by [3], still use CPU-based rANS encoding [22], which includes a recurrence of input data stream as discussed in Section 2. This sequential characteristic of rANS encoding results in low throughput. This significant performance bottleneck motivates us to optimize the lossless encoder in VISEMZ.

Problems with Existing GPU Lossless Encoders. Conversely, GPU-based compressors decorrelate data dependencies and enable parallel execution. However, three key issues hinder efficient GPU lossless encoding: (1) Thread blocks in CUDA execute in parallel, causing each thread block to be unaware of the cumulative compressed size in previous thread blocks, leading to undetermined offsets in the compressed output. To solve this, a prefix-sum, which calculates the summation of all previous elements, is necessary. However, a plain GPU implementation of prefix-sum causes sequential processing, requiring each thread block to wait until its predecessor finishes (synchronization), which breaks GPU concurrency and leads to performance drops; (2) the compression ratio of GPU-based compressors is generally smaller than CPU-based ones [63], due to the common data block partition strategy for parallel processing. This strategy partitions the input data stream into independent data blocks, enabling parallel processing but decreasing the compression ratio by breaking the pattern between data blocks; (3) multiple kernel launches lead to kernel launch overhead and extra

memory accesses, increasing memory footprint. We address each of these challenges with specific design solutions.

Prefix-Sum via Decoupled Lookback. Inspired by [40], we propose a highly integrated decoupled lookback prefix-sum to solve inefficient synchronization issues by utilizing computing resources while waiting for synchronization. A comparison of the plain solution vs our solution is demonstrated in Figure 10. In the plain solution, after each thread block finishes the compression for its allocated data block, it starts to wait for its immediate predecessor to finish the compression and pass the accumulated compressed size for all the previous thread blocks to the current thread block to let it know the offset in the final compressed output. Unlike the plain implementation that forces each thread block to synchronize with its immediate predecessor, our method decouples this dependency by allowing each thread block to look back at all its predecessors, starting with the closest one, to calculate the offset itself instead of waiting for its immediate predecessor.

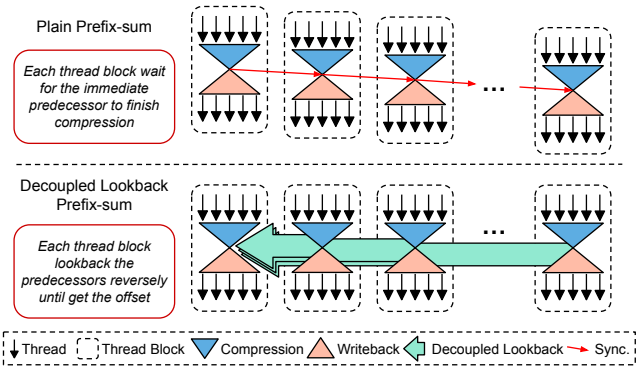

Figure 10. Comparison of plain prefix-sum and our solution.

We use an example in Figure 11 to better illustrate this concept. Each thread block maintains two parameters: the local compressed size (LCS) and the global inclusive offset (GIO). After a thread block finishes compressing its allocated data block, it writes its compressed size to the LCS, copies it to the GIO, and looks back at its predecessors in reverse order while maintaining an exclusive summation (ES), which is initially set to 0. If the compression process in a predecessor is not yet finished, the current thread block will pause and wait. Otherwise, if the compression process is finished but the predecessor's GIO has not yet been updated (i.e., it is still equal to the LCS, except for the first thread block), the predecessor's LCS is added to the current thread block's ES. This process repeats for all preceding thread blocks until one with an updated GIO is found. The value of this GIO is then added to the current ES, which determines the current thread block's offset in the final compressed output. Finally, the thread block updates its own GIO by adding the ES.

We further analyze the process by examining two thread blocks to enhance understanding. For thread block 2, after completing compression, it begins by looking back at its

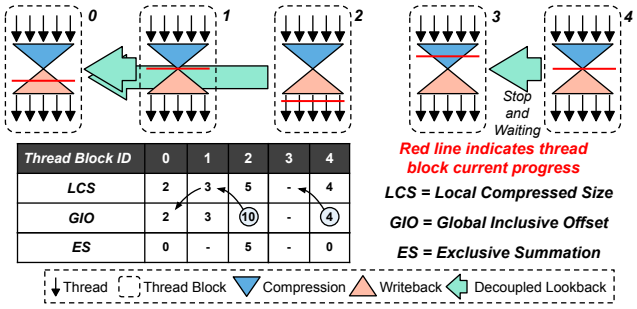

Figure 11. An example of the decoupled lookback prefix-sum.

predecessors, starting with thread block 1. It observes that GIO_1 (the GIO of thread block 1) has not been updated yet, so it adds $LCS_1 = 3$ to ES_2 (resulting in $ES_2 = 0+3$) and proceeds to the next predecessor, thread block 0. It then finds that this is the first thread block, so it adds $GIO_0 = 2$ to ES_2 (resulting in $ES_2 = 0+3+2 = 5$), leading to a calculated memory offset for thread block 2 of $ES_2 = 5$. Finally, it updates its own GIO ($GIO_2 = GIO_2 + ES_2$). For thread block 4, after completing compression, it begins to look back but finds that thread block 3 has not yet finished compression. Therefore, it waits until LCS_3 is generated before continuing the process.

Bitshuffle. Although varied-length integer truncation significantly reduces redundancy, some redundancy remains at the bit level. For example, in Figure 8, the truncation bits are still sparse column-wise, but this feature is not detectable by lossless encoders since they process data streams in units of bytes. This limitation motivates us to propose an additional scheme to address this sparsity, making it recognizable for subsequent lossless compression. Inspired by [39], we use bitshuffle, an algorithm that reorganizes the dataset bitwise by gathering the n-th bits of all the bytes in the data chunk. Figure 12 provides an example of bitshuffle. However, the bitshuffle operation requires intensive memory access to the same byte due to bit-level operations, making it time-consuming. Drawing on insights from [63], we utilize shared memory in GPUs to store and shuffle the data. Shared memory is a programmable memory space integrated into the GPU's L1 cache, which makes our design faster than a standard implementation using GPU global memory. Additionally, we employ the warp-level function *__ballot_sync* to shuffle the data, as it facilitates register-level direct communication within a warp, further improving performance. Moreover, the GPU's shared memory is divided into 32 banks, with each consecutive word (4 bytes) belonging to a different bank. Simultaneous access to the same bank by threads within the same warp causes sequential memory access, known as bank conflict. To avoid this, we allocate 4 bytes to each thread for shuffling, which further enhances performance. It is worth noting that bitshuffle is more efficient when the data contains large areas of similar values, a characteristic common in smooth scientific datasets. Therefore, we apply bitshuffle as an optional step.

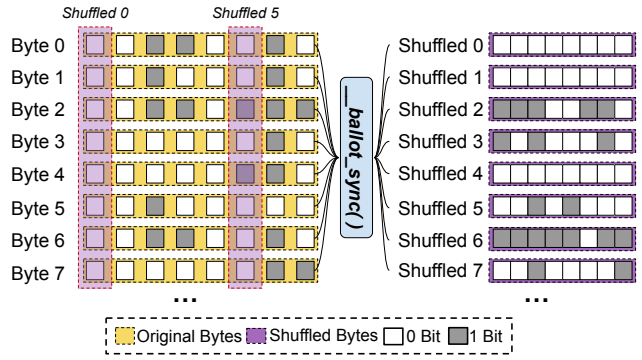

Figure 12. An example of the Bitshuffle implementation.

Table 2. Comparison of lossless encoders on the sample images.

encoder	comp ratio	throughput (GB/s) comp.	decomp.
ANS	36.91	94.72	185.25
LZ	63.06	3.98	28.53
Zstd	162.93	1.81	5.76

General Lossless Encoder. Finally, we apply a general GPU-based lossless compressor to further compress the data. Available lossless encoders can be broadly categorized into dictionary-based (e.g., the LZ family) and entropy-based (e.g., ANS) encodings. This study undertakes a comparative analysis of several lossless encoders from both categories to identify the most suitable solution. We evaluated three state-of-the-art lossless encoding methods: GPU-based ANS encoding from nvCOMP[41], GPU-based LZ encoding (GPULZ[64]), and a hybrid approach combining entropy and dictionary encoding methods, specifically ZSTD from nvCOMP[41]. Our evaluation, summarized in Table 2, highlights the distinct advantages of these encoders: the ANS encoder offers the highest throughput but the lowest compression ratio, while ZSTD achieves the highest compression ratio but with the lowest throughput. GPULZ provides a balanced approach. The user can choose a lossless compressor based on the specific application scenario. We use ZSTD as the default option (also used in our evaluation).

Integration through Kernel Fusion. Despite the various optimizations, each one is implemented in a separate kernel. This use of multiple GPU kernels necessitates additional read/write operations on intermediate data, which introduces overhead and increases the demand for memory to store temporary results. To address these issues, we explore kernel fusion, which merges multiple kernels into a single kernel function. This approach eliminates the need for temporary buffers and reduces unnecessary memory access, thereby enhancing performance. However, kernel fusion requires access to the GPU kernel's source code, so this optimization is applicable primarily to open-source compressors.

3.3 Workflow Pipelining

Necessity of a Pipeline Architecture. AI-driven compression systems typically consist of two primary components, as outlined in Section 2: model inference and lossless encoding. In addition to these processes, the workflow incorporates data loading and writing operations. These components face certain concurrency challenges: (1) The model inference process involves numerous GPU kernels that are interdependent and cannot be executed in parallel. This limitation may lead to sub-optimal GPU resource utilization, particularly with smaller-sized input images. (2) Additionally, the processes of data loading and writing can be executed concurrently with kernel operations, a functionality inherently supported by CUDA. By launching GPU operations such as memory copy or kernels to different CUDA streams, they can execute concurrently when computing resources are sufficient. Therefore, a well-designed pipeline is essential to manage these tasks efficiently.

Challenges in Pipeline Implementation. Implementing a pipeline, however, is non-trivial and may still encounter performance issues due to several factors: (1) AI-based compressors are typically implemented using PyTorch, which is constrained by Python's Global Interpreter Lock (GIL), preventing multi-threading. As a result, image reading and saving the compressed output cannot be parallelized. Although libraries like DALI [42] enable parallel data loading in Python, they do not support parallel writing. (2) Even with a pipeline, the image reading process requires the CPU to create an additional temporary bounce buffer, leading to memory and time overhead.

Multi-threading with LibTorch and OpenMP. To overcome the limitations of multi-threading in Python, we utilize the LibTorch library [47], the C++ API for PyTorch, to transition model inference from Python to C++. We also leverage OpenMP [10] to initialize multiple threads for managing CUDA streams and parallelizing data reading and writing. As illustrated in Figure 13, each thread processes an independent input image, launching GPU operations on its designated CUDA stream and independently handling the reading and writing of input and output data.

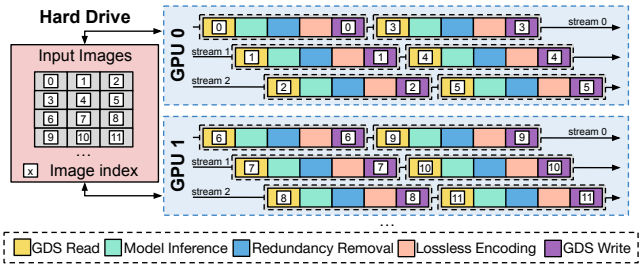

Figure 13. The pipeline design of our framework.

GPUDirect Storage (GDS) Adoption. To eliminate the overhead associated with CPU bounce buffers, we advocate for the integration of GPUDirect Storage (GDS) technology [24]. GDS facilitates direct data transfers between GPU memory and storage via direct memory access (DMA), bypassing the need for data to pass through the CPU. This significantly enhances the efficiency of GPU data reading and offloading.

4 Evaluation

> **Argument.** We demonstrate that ViSemZ achieves excellent results across each individual metric and, overall, offers an attractive tradeoff between them.

4.1 Experimental Setup

Platforms. In our study, we employ a singular node derived from a High-Performance Computing (HPC) cluster to run most of our evaluation (except for the multi-node scale up). This node is configured with dual AMD EPYC 7742 Central Processing Units (CPUs), each featuring 64 cores operating at a frequency of 2.25GHz. Additionally, it is equipped with four NVIDIA Ampere A100 Graphics Processing Units (GPUs), each offering 108 Streaming Multiprocessors (SMs) and a memory capacity of 40GB. The software environment on this platform includes CentOS 7.4 as the operating system and version 12.3.0 of the CUDA toolkit.

Table 3. Real-world datasets used in the evaluation.

DATASETS	FIELD DATA SIZE dimensions	#FIELDS examples(s)
MATERIAL SIMULATION	19.60 MB	48 in total
PRISM [18]	369×369	combo_2_ABF
MICROSCOPE	100.66 MB	32 in total
STEM [50]	1,024×1,024	LMO_STO
IMAGE COMPRESSION	251.60 MB	30 in total
CLIC [55]	1,365×2,048	bridge, lion
PHOTOGRAPHY	28.31 MB	24 in total
KODAK [32]	512×768	kodim01, kodim02
MEDICAL X-RAY	360 MB	30 in total
COVID [15]	2,000×2,000	COVID19_471
CLIMATE SIMULATION	1360.80 MB	70 in total
CESM [14]	1,800×3,600	CLDICE, RELHUM
COSMOLOGY SIMULATION	4.71 MB	6 in total
NYX [45]	512×512	vx, vy
PETROLEUM EXPLORATION	9.68 MB	18 in total
RTM [27]	449×449	snapshot_1200

Datasets. Our evaluation and comparison are based on the analysis of eight distinct datasets relevant to real-world compression tasks. These datasets span a variety of fields, showcasing the versatility of our approach. The specifics of these datasets are meticulously cataloged in Table 3, providing a detailed overview for further analysis.

Baselines. In this study, we compare our proposed method with state-of-the-art (SOTA) GPU-based lossy compression algorithms. We include nvJPEG [43], a GPU implementation of the classic JPEG image compression algorithm [56]. Although we intended to include nvJPEG2000 [44], the GPU implementation of the JPEG2000 algorithm [53], we were unable to deploy it due to a lack of administrative permissions. Consequently, we used the JPEG2000 implementation from PIL [46] to evaluate data quality only. Additionally, we include pure GPU-based scientific compressors: cuSZ[16], cuZFP [17], FZ-GPU [63], and Bitcomp, as implemented in the nvCOMP library [41].

We select two AI-based compressors and build ViSemZ using their models to demonstrate the generality of our method: Balle2016 [3], the pioneering AI-based compressor, and Liu2023 [38], a state-of-the-art (SOTA) transformer-based compressor. We use Balle2016 and Liu2023 to refer to the original works. Both compressors are implemented using CompressAI [6], with Balle2016 integrated into it and Liu2023 built upon it[37]. Both models utilize the CPU rANS lossless encoding [22] from CompressAI. For Balle2016, we use the default 8 quality levels, and for Liu2023, we use 6 quality levels as specified on their official site [37], with $N = 64$ and a λ range of [0.05, 0.025, 0.013, 0.0067, 0.0035, 0.0025]. We refer to our work with these models as ViSemZ(Balle2016) and ViSemZ(Liu2023).

4.2 Evaluation Metrics

Our evaluation framework encompasses four primary metrics to assess the performance of compression algorithms. These metrics are meticulously chosen to cover the essential aspects of compression efficiency and effectiveness:

Compression Ratio & Bitrate: This metric is a fundamental measure in compression research, quantifying how effectively data size is reduced. Compression ratio is calculated as the ratio of the original data size to the compressed data size, with higher values indicating more efficient compression, i.e., more information is compacted relative to the original. Another related measure is the bitrate, or bits per pixel (bpp), which reflects the average number of bits used per pixel after compression. Bitrate is calculated by dividing the compressed size by the number of pixels.

Compression End-to-end Time: This metric reflects the performance of a compressor, highlighting the temporal efficiency of the compression algorithm. Although some work only evaluates the kernel execution time of GPU compressors, we choose to evaluate the end-to-end time for a more practical evaluation.

Image Quality Metrics: Assessing distortion is important for evaluating the quality of data reconstruction after compression, especially for lossy compressors. We mainly use the Peak Signal-to-Noise Ratio (PSNR) and the Structural Similarity Index Measure (SSIM) to measure distortion. Additionally, since we focus on preserving the visual meaning of images, we include a new AI-based metric called DISTS [21], which uses machine learning to better quantify the texture-preserving ability of different compressors.

Visualization Quality: The ability of a compression algorithm to maintain the visual integrity of data post-compression is paramount, especially for applications reliant on visual data analysis, such as scientific visualization.

4.3 Evaluation of Compression Quality

In our analysis, a key comparative tool is the evaluation of the rate-distortion curves, which graphically represent the relationship between quality, measured in Peak Signal-to-Noise

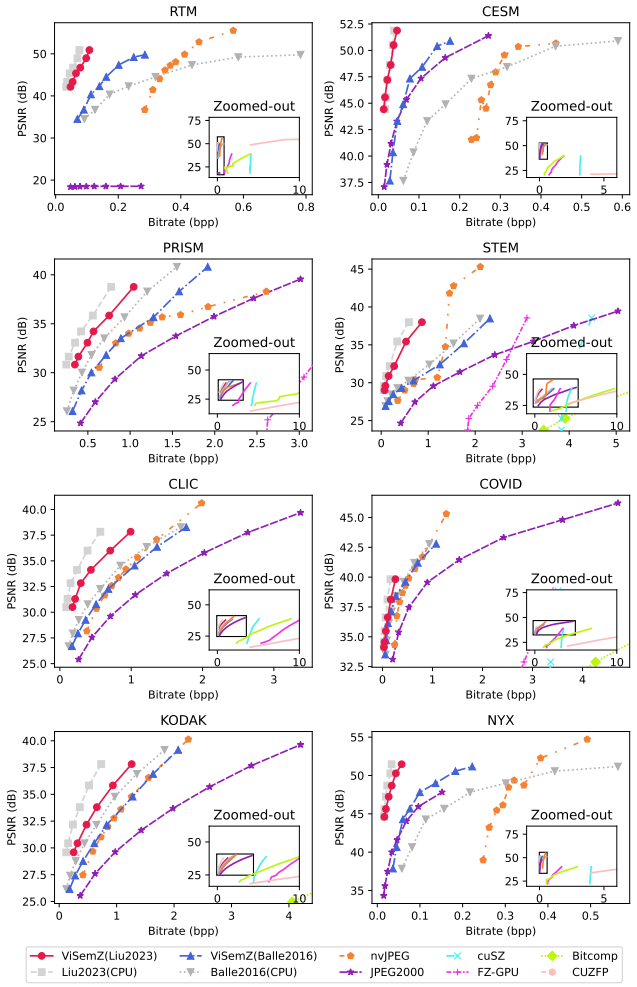

Figure 14. Rate-distortion plot showing PSNR (quality, higher is better) and bitrate (compression efficiency, lower is better). A zoomed-in view of the low bitrate region is provided, with the full plot shown in the bottom right corner of each subfigure.

Ratio (PSNR), and bitrate across various compressors, which is widely used in compression studies [1, 19, 33, 51, 54, 63]. These curves serve as a direct indicator of compression efficiency, underlining how well each compressor maintains image quality at different levels of data compression. To construct these curves, we adjust the settings of each compressor, employing a range of configurations to modulate the image quality and, subsequently, observe the variations in PSNR. The rate-distortion map is shown in Figure 14. The Y-axis denotes PSNR, indicating quality (higher is better), and the X-axis denotes the bitrate, indicating compression efficiency (lower is better). The better rate-distortion curve will be oriented toward the top left direction.

Comparison with Balle2016 and Liu2023. The comparative analysis reveals notable similarities between the rate-distortion curves of our approach and those of the AI-based compressors. Notably, VISEMZ demonstrates similar or even superior efficiency in certain datasets, such as RTM, CESM, STEM, COVID, and NYX, compared to Balle2016 and Liu2023. This improvement is attributed to the implementation of an irregular redundancy-removal process applied to the inference output. However, in other datasets, a gap remains between our work and the original AI-based compressors. In these cases, the images are more complex, resulting in less bit-level sparsity and a higher bitrate for VISEMZ.

Comparison with non-AI-based Compressors. Compared to scientific compressors, VISEMZ(Liu2023) demonstrates superior results across various datasets. This advantage is primarily due to the reliance of scientific compressors on basic operations to achieve higher compression throughput and facilitate error control. For example, floating-point truncation, as used in cuZFP, often results in a lower compression ratio for a given PSNR. Regarding image compressors, the superiority of AI-based methods over traditional compressors is well-documented [3, 38]. Our work maintains this advantage, even in pure GPU implementations, through optimizations for compression ratio, such as redundancy removal and bitshuffle.

> **Observation I.** In terms of compression quality, our work achieves the highest PSNR at similar bitrates compared to non-AI-based compressors. Evaluation shows that VISEMZ delivers up to 20× lower bitrate at a similar PSNR compared to the best non-AI-based compressor, nvJPEG.

4.4 Evaluation of Compression Performance

We then evaluate the end-to-end compression time, as it is more practical for real-world applications. This comparison involves averaging each compressor's performance across various settings, given that different compressors employ different error-control schemes. The results are presented in Table 4. The top number represents the end-to-end time in milliseconds (lower is better), while the bottom number (along with the bar) indicates the speedup relative to nvJPEG.

Comparison with Balle2016 and Liu2023. Thanks to the optimizations, VISEMZ surpasses CompressAI across all tested datasets, achieving an average speedup of 2.6× and 5.2× over Balle2016 and Liu2023, respectively. An image data stream will further utilize our pipeline design, achieving a 9.6× speedup, as evaluated in §4.6.

Comparison with nvJPEG. Observations reveal that our framework demonstrates higher efficiency, achieving an average speedup of 1.9× over nvJPEG. This enhancement can be attributed to nvJPEG's hybrid computational approach, leveraging both CPU and GPU resources, wherein certain processes with data dependencies, such as lossless encoding, are CPU-bound. In contrast, our framework is fully deployed on GPUs, limiting CPU only to launch CUDA kernels.

Comparison with Scientific Compressors. Our analysis shows that scientific compressors achieve low end-to-end

Table 4. Compression/decompression time of different compressors (AI-based compressors underlined). The top number is end-to-end time in milliseconds (lower the better). The bottom number (along with the bar) indicates speedup relative to nvJPEG (higher is better). ViSemZ(B) and ViSemZ(L) denote ViSemZ(Balle2016) and ViSemZ(Liu2023). Note ViSemZ's multi-image compression (pipelining) is ≈ 3.1× faster.

(a) The compression performance.

	RTM	CESM	PRISM	STEM	CLICK	COVID	CODAK	NYX
nvJPEG	833.9	863.81	834.77	855.82	871.84	872.29	839.35	280.49
	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×
cuSZ	505.26	908.88	488.77	633.71	779.34	1087.29	569.68	563.28
	1.65 ×	0.95 ×	1.71 ×	1.35 ×	1.12 ×	0.80 ×	1.47 ×	0.50 ×
FZ-GPU	487.11	645.93	482.69	535.72	559.88	639.21	489.75	488.78
	1.71 ×	1.34 ×	1.73 ×	1.60 ×	1.56 ×	1.36 ×	1.71 ×	0.57 ×
Bitcomp	288.67	346	281.39	307.97	315.73	345.42	288.11	282.31
	2.89 ×	2.50 ×	2.97 ×	2.78 ×	2.76 ×	2.53 ×	2.91 ×	0.99 ×
CUZFP	408.87	507.98	400.62	427.41	448.6	505.77	388.91	411.25
	2.04 ×	1.70 ×	2.08 ×	2.00 ×	1.94 ×	1.72 ×	2.16 ×	0.68 ×
Balle2016	1307.2	1722.89	1272.81	1422.05	1562.84	1731.11	1316.02	1262.86
	0.64 ×	0.50 ×	0.66 ×	0.60 ×	0.56 ×	0.50 ×	0.64 ×	0.22 ×
ViSemZ(B)	544.12	679.33	527.33	556.38	560.2	572.6	527.21	531
	1.53 ×	1.27 ×	1.58 ×	1.54 ×	1.56 ×	1.52 ×	1.59 ×	0.53 ×
Liu2023	2143.33	2971.36	1841.06	2132.47	2296.27	2743.2	1863.71	1831.83
	0.39 ×	0.29 ×	0.45 ×	0.40 ×	0.38 ×	0.32 ×	0.45 ×	0.15 ×
ViSemZ(L)	227.77	852.24	382.15	491.81	343.5	709.91	392.37	400.46
	3.66 ×	1.01 ×	2.18 ×	1.74 ×	2.54 ×	1.23 ×	2.14 ×	0.70 ×

(b) The decompression performance.

	RTM	CESM	PRISM	STEM	CLICK	COVID	CODAK	NYX
nvJPEG	870.2	1034.65	876.19	917.46	944.3	1019.32	1019.32	307.19
	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×	1.00 ×
cuSZ	499.4	925.7	485.3	633.87	785.51	1093.45	566.56	556.33
	1.74 ×	1.12 ×	1.81 ×	1.45 ×	1.20 ×	0.93 ×	1.80 ×	0.55 ×
FZ-GPU	406.12	675.29	488.92	500.3	551.14	629.97	447.11	463.58
	2.14 ×	1.53 ×	1.79 ×	1.83 ×	1.71 ×	1.62 ×	2.28 ×	0.66 ×
Bitcomp	8.73	61.94	5.29	21.09	30.22	58.68	7.83	6.6
	99.66 ×	16.70 ×	165.69 ×	43.50 ×	31.23 ×	17.36 ×	130.11 ×	46.55 ×
CUZFP	392.6	464.6	391.64	415.11	423.08	463.51	382.19	416.94
	2.22 ×	2.23 ×	2.24 ×	2.21 ×	2.23 ×	2.20 ×	2.67 ×	0.74 ×
Balle2016	1315.63	2815.52	1316.82	1946.29	2378.34	3242.97	1393.84	1365.71
	0.66 ×	0.37 ×	0.67 ×	0.47 ×	0.40 ×	0.31 ×	0.73 ×	0.22 ×
ViSemZ(B)	475.71	491.33	481.56	567.59	583.4	609.87	480.04	488.67
	1.83 ×	2.11 ×	1.82 ×	1.62 ×	1.62 ×	1.67 ×	2.12 ×	0.63 ×
Liu2023	1981.5	3862.07	1839.6	3354.34	3082.4	4558.33	1952.83	1848.33
	0.44 ×	0.27 ×	0.48 ×	0.27 ×	0.31 ×	0.22 ×	0.52 ×	0.17 ×
ViSemZ(L)	223.67	522.9	271	418.74	434.75	613.45	281.4	289.53
	3.89 ×	1.98 ×	3.23 ×	2.19 ×	2.17 ×	1.66 ×	3.62 ×	1.06 ×

compression times across various datasets. This efficiency is primarily attributed to their use of highly parallelizable algorithms for data processing. For example, Bitcomp employs quantization for error management, a highly parallelizable process, while CUZFP utilizes floating-point truncation to maintain a fixed bitrate, which also benefits from straightforward parallelization. These characteristics result in average speedups of 1.51× for Bitcomp and 1.06× for CUZFP compared to our approach, ViSemZ(Liu 2023). However, it is important to note that these gains in processing speed often come at the cost of reconstructed data quality and lower compression ratios, a trade-off that is discussed in Section 4.3. Due to our optimized GPU design, our method even outperforms some scientific compressors such as cuSZ and FZ-GPU, achieving speedups of 1.47× and 1.17× respectively.

Observation II. In terms of compression time, our work (ViSemZ(Liu2023)) achieves performance comparable to non-AI-based compressors, with an average speedup of 1.9× compared to nvJPEG.

4.5 Evaluation of Reconstructed Data Visualization

In this analysis, we examine the visualization quality of images reconstructed by various compression methods. To ensure a fair comparison, we align the compression ratios of the compared compressors as closely as possible with our technique. If matching our compression ratio is not feasible, we use the highest achievable compression ratio for the other compressors. The compression ratio is presented in bits per pixel (bpp) in the figure, calculated as the compressed file size divided by the number of pixels. In addition to bpp, we use three key metrics to evaluate the quality of the reconstructed data: PSNR, SSIM, and DISTS, as mentioned in Section 4.2. For all these metrics, higher values indicate better quality.

The visual outcomes of this comparison are encapsulated in Figures. 15. These visual representations reveal that compressors, particularly those not based on AI methodologies, introduce noticeable artifacts in the images at lower bitrates. For instance, JPEG tends to induce block-wise segmentation artifacts, whereas scientific lossy compressors often result in the loss of texture within the images. Such degradation is undesirable for subsequent analytical processes. These artifacts can considerably diminish the accuracy of subsequent analyses, such as material science applications where classifier accuracy in post-analysis is paramount.

In contrast, our approach (ViSemZ(Liu2023)) exhibits superior performance in maintaining the integrity of image details without introducing noticeable artifacts. This is corroborated by PSNR, SSIM, and DISTS outcomes, where our work surpasses other lossy compressors in these quality assessments under similar compression ratios.

Observation III. In terms of visualization quality, our work (ViSemZ(Liu2023)) best preserves the details, as demonstrated not only in the reconstructed images but also in the metrics PSNR, SSIM, and DISTS.

4.6 Optimization Breakdown Study

Workflow Pipelining. We begin by evaluating the performance improvement achieved through our pipeline design. We simulate a practical scenario using all evaluation datasets as an input stream across four CUDA streams. The original implementation of Balle2016 serves as our baseline for comparison. To distinguish the impact of different optimizations, we define ViSemZ_1 as ViSemZ(Balle2016) with only the redundancy removal and GPU lossless encoding optimizations, and ViSemZ_2 as ViSemZ_1 with the additional pipeline

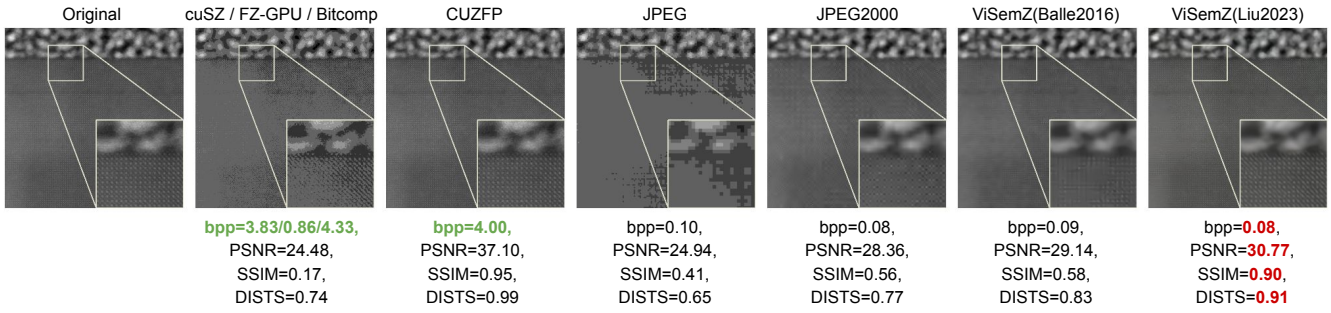

Figure 15. The comparison visualizes reconstructed images from various compressors. We tune the bpp for other compressors to be similar to ViSemZ(Liu2023); for cases where this is not possible, we use the minimum achievable bpp instead (cuSZ, FZ-GPU, Bitcomp, and CUZFP).

Table 5. Breakdown speedup of our pipeline design.

	Balle2016	ViSemZ_1	ViSemZ_2
comp.	1×	3.1×	9.6×
decomp.	1×	2.5×	8.3×

optimization. The results, shown in Table 5, indicate a significant performance improvement. The pipeline design results in an average speedup of 3.1× in compression speed compared to its non-pipelined counterpart and 9.6× compared to Balle2016. Decompression shows similar trends. These results demonstrate the effectiveness of our pipeline design in efficiently handling larger data volumes.

Redundancy Removal. Next, we evaluate the optimization of redundancy removal using the Liu2023 model. The GPU implementation, which utilizes warp-level functions and optimized memory access patterns, achieves high efficiency, with an average throughput of 78.8 GB/s, as shown in Figure 16. Table 6 presents the compression ratio results. Our redundancy removal method achieves an average compression ratio of 94.34 across all datasets, effectively compressing the data before the lossless encoding process.

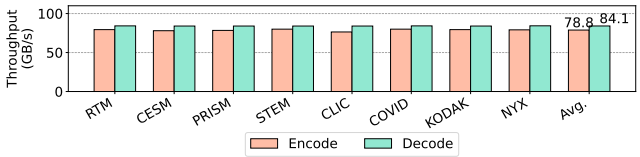

Figure 16. Throughput of the redundancy removal process.

Table 6. The compression ratio of redundancy removal.

Quality	RTM	CESM	PRISM	STEM	CLIC	COVID	KODAK	NYX
6	91.87	163.43	18.55	16.77	15.62	36.75	13.44	119.21
5	106.08	187.73	23.59	22.35	19.34	52.10	16.18	153.07
4	128.88	211.99	29.49	36.77	27.79	78.11	22.50	189.78
3	148.03	261.77	35.25	50.73	32.09	114.01	25.44	228.61
2	178.11	297.14	43.59	81.25	42.05	164.00	33.91	275.05
1	192.18	323.91	47.90	97.40	49.59	195.68	40.39	301.36

Bitshuffle. As an optional module, when using the Liu2023 model with $\lambda = 0.0025$, bitshuffle achieves an average improvement of 8% in the compression ratio across the CESM, STEM, and COVID datasets.

Decoupled Lookback Prefix-Sum. Our optimized decoupled lookback approach provides an average speedup of 2.5× compared to the plain prefix-sum across the evaluation datasets, as shown in Figure 17, effectively eliminating the prefix-sum as a bottleneck in the GPU process.

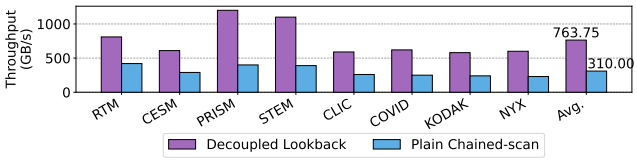

Figure 17. Comparison of prefix-sum performance.

Multi-node and Multi-GPU Speedup. We have also implemented multi-node and multi-GPU versions of ViSemZ. Using the Balle2016 model, the speedup results are shown in Figure 18. Due to the independence of image compression tasks, the speedup is nearly optimal for both multi-node and multi-GPU setups, making this approach highly suitable for large-scale scientific applications.

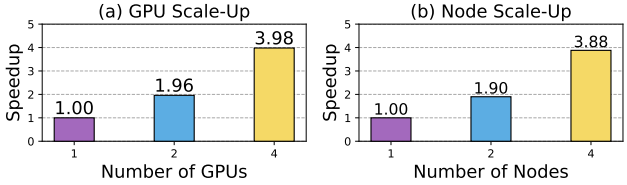

Figure 18. Normalized speedup of multi-node and multi-GPU.

5 Conclusion and Future Work

In this study, we introduce ViSemZ, an efficient AI-based GPU lossy compression algorithm. Leveraging an irregular redundancy removal process, an optimized GPU lossless encoder, and pipelined data processing, ViSemZ delivers excellent compression efficacy, characterized by a high compression ratio, low end-to-end time, and exceptionally high-quality reconstructed data. Evaluations demonstrate that our work outperforms non-AI-based compressors while achieving a significant speedup compared to AI-based compressors, making it the best choice for efficient compression with high detail preservation.

Despite its successes, ViSemZ has not yet addressed improvements in model inference, which represents another critical aspect of our framework. Moving forward, we aim to adopt strategies such as model pruning and mixed-precision inference to enhance performance. Additionally, we plan to further refine the algorithm for optimal performance on specific scientific datasets.

References

[1] M Ainsworth, O Tugluk, B Whitney, and S Klasky. 2017. MGARD: A Multilevel Technique for Compression of Floating-Point Data. In DRBSD-2 Workshop at Supercomputing.

[2] Sarah Akers, Elizabeth Kautz, Andrea Trevino-Gavito, Matthew Olszta, Bethany E Matthews, Le Wang, Yingge Du, and Steven R Spurgeon. 2021. Rapid and flexible segmentation of electron microscopy data using few-shot machine learning. npj Computational Materials 7, 1 (2021), 187.

[3] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. 2016. End-to-end optimized image compression. arXiv preprint arXiv:1611.01704 (2016).

[4] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. Variational image compression with a scale hyperprior. arXiv preprint arXiv:1802.01436 (2018).

[5] Steven VW Beckwith, Massimo Stiavelli, Anton M Koekemoer, John AR Caldwell, Henry C Ferguson, Richard Hook, Ray A Lucas, Louis E Bergeron, Michael Corbin, Shardha Jogee, et al. 2006. The Hubble ultra deep field. The Astronomical Journal 132, 5 (2006), 1729.

[6] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. 2020. Compressai: a pytorch library and evaluation platform for end-to-end compression research. arXiv preprint arXiv:2011.03029 (2020).

[7] Deep Learning GPU Benchmarks. [n. d.]. https://lambdalabs.com/gpu-benchmarks.

[8] Neelanjan Bhowmik, Jack W Barker, Yona Falinie A Gaus, and Toby P Breckon. 2022. Lost in compression: the impact of lossy image compression on variable size object detection within infrared imagery. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 369–378.

[9] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. 2019. Use cases of lossy compression for floating-point data in scientific data sets. The International Journal of High Performance Computing Applications 33, 6 (2019), 1201–1220.

[10] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. 2001. Parallel programming in OpenMP. Morgan kaufmann.

[11] Jin Chen, Xiaolin Zhu, James E Vogelmann, Feng Gao, and Suming Jin. 2011. A simple and effective method for filling gaps in Landsat ETM+ SLC-off images. Remote sensing of environment 115, 4 (2011), 1053–1064.

[12] Xinyu Chen, Jiannan Tian, Ian Beaver, Cynthia Freeman, Yan Yan, Jianguo Wang, and Dingwen Tao. 2024. FCBench: Cross-Domain Benchmarking of Lossless Compression for Floating-Point Data. Proceedings of the VLDB Endowment 17, 6 (2024), 1418–1431.

[13] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. 2020. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 7939–7948.

[14] Community Earth System Model (CESM) Atmosphere Model. 2019. http://www.cesm.ucar.edu/models/. Online.

[15] COVID: An covid lung X-ray image dataset. [n. d.]. https://www.kaggle.com/datasets/prashant268/chest-xray-covid19-pneumonia.

[16] cuSZ: A CUDA-Based Error-Bounded Lossy Compressor for Scientific Data. [n. d.]. https://github.com/szcompressor/cuSZ.

[17] cuZFP: Experimental CUDA port of zfp compression. [n. d.]. https://github.com/LLNL/zfp/tree/feature/cuda_support.

[18] Luis Rangel DaCosta, Hamish G Brown, Philipp M Pelz, Alexander Rakowski, Natolya Barber, Peter O'Donovan, Patrick McBean, Lewys Jones, Jim Ciston, MC Scott, et al. 2021. Prismatic 2.0–Simulation software for scanning and high resolution transmission electron microscopy (STEM and HRTEM). Micron 151 (2021), 103141.

[19] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In 2016 IEEE International Parallel and Distributed Processing Symposium. IEEE, Chicago, IL, USA, 730–739.

[20] Sheng Di, Jinyang Liu, Kai Zhao, Xin Liang, Robert Underwood, Zhaorui Zhang, Milan Shah, Yafan Huang, Jiajun Huang, Xiaodong Yu, et al. 2024. A Survey on Error-Bounded Lossy Compression for Scientific Datasets. arXiv preprint arXiv:2404.02840 (2024).

[21] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P Simoncelli. 2020. Image quality assessment: Unifying structure and texture similarity. IEEE transactions on pattern analysis and machine intelligence 44, 5 (2020), 2567–2581.

[22] Jarek Duda. [n. d.]. https://github.com/rygorous/ryg_rans.

[23] Jarek Duda. 2013. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. arXiv preprint arXiv:1311.2540 (2013).

[24] GPUDirect Storage. [n. d.]. https://docs.nvidia.com/gpudirect-storage/index.html.

[25] Pascal Grosset, Christopher Biwer, Jesus Pulido, Arvind Mohan, Ayan Biswas, John Patchett, Terece Turton, David Rogers, Daniel Livescu, and James Ahrens. 2020. Foresight: analysis that matters for data reduction. In 2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society, 1171–1185.

[26] Dailan He, Ziming Yang, Weikun Peng, Rui Ma, Hongwei Qin, and Yan Wang. 2022. Elic: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 5718–5727.

[27] Sian Jin, Sheng Di, Jiannan Tian, Suren Byna, Dingwen Tao, and Franck Cappello. 2022. Improving Prediction-Based Lossy Compression Dramatically Via Ratio-Quality Modeling. In 2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, 2494–2507.

[28] Sian Jin, Sheng Di, Frédéric Vivien, Daoce Wang, Yves Robert, Dingwen Tao, and Franck Cappello. 2024. Concealing Compression-accelerated I/O for HPC Applications through In Situ Task Scheduling. In EuroSys 2024.

[29] Sian Jin, Dingwen Tao, Houjun Tang, Sheng Di, Suren Byna, Zarija Lukic, and Franck Cappello. 2022. Accelerating parallel write via deeply integrating predictive lossy compression with HDF5. arXiv preprint arXiv:2206.14761 (2022).

[30] Tiffany C Kaspar, Seungbum Hong, Mark E Bowden, Tamas Varga, Pengfei Yan, Chongmin Wang, Steven R Spurgeon, Ryan B Comes, Pradeep Ramuhalli, and Charles H Henager Jr. 2018. Tuning piezoelectric properties through epitaxy of La2Ti2O7 and related thin films. Scientific reports 8, 1 (2018), 3037.

[31] Jun-Hyuk Kim, Byeongho Heo, and Jong-Seok Lee. 2022. Joint global and local hierarchical priors for learned image compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 5992–6001.

[32] KODAK: An lossless, true color image dataset. [n. d.]. https://r0k.us/graphics/kodak/.

[33] Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2018. An Efficient transformation scheme for lossy data compression with point-wise relative error bound. In CLUSTER. IEEE, Belfast, UK, 179–189.

[34] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M Gok, Jiannan Tian, Junjing Deng, Jon C Calhoun, Dingwen Tao, et al. 2022. Sz3: A modular framework for composing prediction-based error-bounded lossy compressors. IEEE Transactions on Big Data 9, 2 (2022), 485–498.

[35] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. IEEE Transactions on Visualization and Computer Graphics 20, 12 (2014), 2674–2683.

[36] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez.

2017. A survey on deep learning in medical image analysis. Medical image analysis 42 (2017), 60–88.
[37] Jinming Liu, Heming Sun, and Jiro Katto. [n. d.]. https://github.com/jmliu206/LIC_TCM.
[38] Jinming Liu, Heming Sun, and Jiro Katto. 2023. Learned image compression with mixed transformer-cnn architectures. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 14388–14397.
[39] Kiyoshi Masui, Mandana Amiri, Liam Connor, Meiling Deng, Mateus Fandino, Carolin Höfer, Mark Halpern, David Hanna, Adam D Hincks, Gary Hinshaw, et al. 2015. A compression scheme for radio data in high performance computing. Astronomy and Computing 12 (2015), 181–190.
[40] Duane Merrill and Michael Garland. 2016. Single-pass parallel prefix scan with decoupled look-back. NVIDIA, Tech. Rep. NVR-2016-002 (2016).
[41] nvCOMP: A library for fast lossless compression/decompression on the GPU. [n. d.]. https://github.com/NVIDIA/nvcomp.
[42] NVIDIA. [n. d.]. https://github.com/NVIDIA/DALI.
[43] nvJPEG. [n. d.]. https://docs.nvidia.com/cuda/nvjpeg/index.html.
[44] nvJPEG2000. [n. d.]. https://docs.nvidia.com/cuda/nvjpeg2000/userguide.html.
[45] NYX. [n. d.]. https://amrex-astro.github.io/Nyx/.
[46] Pillow. [n. d.]. https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html#jpeg-2000.
[47] PyTorch. [n. d.]. https://pytorch.org/cppdocs/installing.html.
[48] rANS. [n. d.]. https://github.com/rygorous/ryg_rans.
[49] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE, 446–459.
[50] Steven Spurgeon. 2024. Images and Labels for Segmentation Studies in Microscopy. https://doi.org/10.5281/zenodo.10909552
[51] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In 2017 IEEE International Parallel and Distributed Processing Symposium. IEEE, Orlando, FL, USA, 1129–1139.
[52] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2019. Optimizing lossy compression rate-distortion from automatic online selection between SZ and ZFP. IEEE Transactions on Parallel and Distributed Systems 30, 8 (2019), 1857–1871.
[53] David S Taubman, Michael W Marcellin, and Majid Rabbani. 2002. JPEG2000: Image compression fundamentals, standards and practice. Journal of Electronic Imaging 11, 2 (2002), 286–287.
[54] Jiannan Tian, Sheng Di, Kai Zhao, Cody Rivera, Megan Hickman Fulp, Robert Underwood, Sian Jin, Xin Liang, Jon Calhoun, Dingwen Tao, et al. 2020. cuSZ: An Efficient GPU-Based Error-Bounded Lossy Compression Framework for Scientific Data. In Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques. 3–15.
[55] George Toderici, Wenzhe Shi, Timofte Radu, Lucas Theis, Johannes Balle, Eirikur Agustsson, Nick Johnston, and Mentzer Fabian. 2020. Workshop and Challenge on Learned Image Compression (CLIC2020). http://www.compression.cc
[56] Gregory K. Wallace. 1992. The JPEG still picture compression standard. Commun. ACM 34, 4 (1992), 30–44.
[57] Daoce Wang, Pascal Grosset, Jesus Pulido, Tushar M Athawale, Jiannan Tian, Kai Zhao, Zarija Lukic, Axel Huebl, Zhe Wang, James Ahrens, et al. 2024. A High-Quality Workflow for Multi-Resolution Scientific Data Reduction and Visualization. arXiv preprint arXiv:2407.04267 (2024).
[58] Daoce Wang, Jesus Pulido, Pascal Grosset, Sian Jin, Jiannan Tian, James Ahrens, and Dingwen Tao. 2022. TAC: Optimizing Error-Bounded Lossy Compression for Three-Dimensional Adaptive Mesh Refinement Simulations. In Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing. 135–147.
[59] Daoce Wang, Jesus Pulido, Pascal Grosset, Sian Jin, Jiannan Tian, Kai Zhao, James Ahrens, and Dingwen Tao. 2024. TAC+: Optimizing Error-Bounded Lossy Compression for 3D AMR Simulations. IEEE Transactions on Parallel and Distributed Systems 35, 3 (2024), 421–438. https://doi.org/10.1109/TPDS.2023.3339474
[60] Daoce Wang, Jesus Pulido, Pascal Grosset, Jiannan Tian, Sian Jin, Houjun Tang, Jean Sexton, Sheng Di, Kai Zhao, Bo Fang, et al. 2023. AMRIC: A Novel In Situ Lossy Compression Framework for Efficient I/O in Adaptive Mesh Refinement Applications. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 1–15.
[61] Yueqi Xie, Ka Leong Cheng, and Qifeng Chen. 2021. Enhanced invertible encoding for learned image compression. In Proceedings of the 29th ACM international conference on multimedia. 162–170.
[62] Eddie Yan, Liang Luo, and Luis Ceze. 2021. Characterizing and Taming Resolution in Convolutional Neural Networks. In 2021 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 189–200.
[63] Boyuan Zhang, Jiannan Tian, Sheng Di, Xiaodong Yu, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. 2023. FZ-GPU: A Fast and High-Ratio Lossy Compressor for Scientific Computing Applications on GPUs. arXiv preprint arXiv:2304.12557 (2023).
[64] Boyuan Zhang, Jiannan Tian, Sheng Di, Xiaodong Yu, Martin Swany, Dingwen Tao, and Franck Cappello. 2023. GPULZ: Optimizing LZSS Lossless Compression for Multi-byte Data on Modern GPUs. In Proceedings of the 37th International Conference on Supercomputing. 348–359.
[65] Zhe Zhu and Curtis E Woodcock. 2014. Continuous change detection and classification of land cover using all available Landsat data. Remote sensing of Environment 144 (2014), 152–171.
[66] Maxim Ziatdinov, Ondrej Dyck, Artem Maksov, Xufan Li, Xiahan Sang, Kai Xiao, Raymond R Unocic, Rama Vasudevan, Stephen Jesse, and Sergei V Kalinin. 2017. Deep learning of atomically resolved scanning transmission electron microscopy images: chemical identification and tracking local transformations. ACS nano 11, 12 (2017), 12742–12752.