APPENDIX

A Related Work

A.1 Data Compression Methods

Traditional Data Compression Methods.
Traditional data compression techniques have been extensively studied and applied in various domains. Lossless compression methods, such as Huffman coding [Moffat, 2019], arithmetic coding [Rissanen and Langdon, 1979], and Lempel-Ziv-Welch (LZW) compression [Nelson, 1989], aim to reduce data size without losing any information. These methods achieve high compression ratios but may be computationally expensive. On the other hand, lossy compression techniques, including discrete cosine transform (DCT) [Khayam, 2003] and wavelet transform [Zhang and Zhang, 2019], sacrifice some information to achieve higher compression ratios. These methods are commonly used in image and video compression but may result in quality degradation. However, conventional compressors like Gzip [Deutsch, 1996] and Zstandard [Collet and Kucherawy, 2018]exhibit limited performance when dealing with multi-modal data streams, owing to their dictionary-based approach.

Deep Learning in Data Compression.
The performance of traditional compressors is limited when confronted with multi-modal data streams. To address this, a potential solution is to develop a multi-specialized compressor system capable of handling diverse data types. However, scalability becomes a significant hurdle for these specialized systems when it comes to cloud deployment. In recent years, a number of deep-learning-based compression algorithms have been proposed to improve compression ratios. These approaches view the compression task as a sequential modeling problem, where the historical symbols (i.e., the "sequence") are used as input to estimate the probability of the incoming symbol (i.e., the "target symbol"). While some of these algorithms, such as DecMac [Liu *et al.*, 2019] and Dzip [Goyal *et al.*, 2021], use Recurrent Neural Networks to capture long-term dependencies, others, such as NNCP [Bellard, 2019] and TRACE [Mao *et al.*, 2022], utilize transformers to achieve precise probability estimation. Despite their effectiveness, the computational burden of these deep-learning-based compressors renders them impractical for most applications. For example, NNCP's dictionary size of 16384 and its 55M transformer model lead to a compression rate of only 0.48 2.04 kB/s. As a solution, bye-stream compressors have been proposed to treat each byte as a symbol and obtain a smaller dictionary. TRACE can achieve a compression rate of 15.87 kB/s, while PAC offers an even higher compression rate of 31.12kb/s with PyTorch and 71.42 kB/s with C++ LibTorch.

Parallelization Techniques in Data Compression.
Efficient parallelization is a critical factor in achieving faster data compression speeds. Various parallelization techniques have been explored in data compression, such as multi-threading, multi-core processing, and GPU acceleration [Blanchard and Tanner, 2013; Huang *et al.*, 2020; Simek and Asn, 2008; Li *et al.*, 2019]. These techniques can significantly reduce processing time by distributing the workload across multiple computing units. For instance, multi-threading enables multiple threads to work concurrently and share resources, thereby increasing overall processing speed. Similarly, multi-core processing partitions the workload across multiple cores, allowing the system to perform multiple tasks simultaneously. GPU acceleration leverages the parallel processing capabilities of GPUs to offload the computation-intensive tasks from CPUs, resulting in faster data processing. In our approach, we employ a multi-stream mechanism [Sourouri *et al.*, 2014; Li *et al.*, 2014] to parallelize the key processes involved in data compression, maximizing GPU utilization and further enhancing speed. It is worth noting that the multi-stream approach has been used in some lossy deep learning data compression domains [Díaz *et al.*, 2019] but has not been applied to general-purpose lossless data compression.

GPU Memory Optmization

The vDNN [Rajbhandari *et al.*, 2020]optimizes the high memory usage of feature maps during the training process in an academic paper style. Unused feature maps are immediately released, and those still needed are temporarily moved to the CPU side. Simultaneously, multiple streams are utilized to overlap computation and communication.Zero[Rhu *et al.*, 2016] implemented offloading of parameters, gradients, and optimizer data at different levels. These studies reduced memory usage but increased communication between devices, leading to GPU computation waiting for data and decreased GPU utilization.

Recomputing, also known as checkpointing[Chen *et al.*, 2016], only remembers the output of each segment and discards all intermediate results within each segment. In the backward propagation process, the discarded results are recalculated at the segment level. The memory optimization techniques mentioned above themselves result in dynamic, irregular allocation requests, leading to higher memory fragmentation.

The earliest method[Veldema and Philippsen, 2012] to eliminate fragmentation was data movement, where non-contiguous memory usage was moved to contiguous regions to eliminate fragmentation. However, this approach led to a significant amount of data movement, increased system load, and poorer support for large arrays.Metis [Xu *et al.*, 2015]introduced pre-unloading analysis, attempting to allocate memory blocks with similar lifecycles to adjacent regions, making the space more contiguous upon release. While effective in reducing fragmentation in loop-intensive programs, its analysis falls short for complex dynamic graph computations in deep learning models.Recent Glake[Guo *et al.*, 2024] research focuses on reallocating fragments generated during runtime from the hardware abstraction layer of the PyTorch framework, achieving higher memory utilization rates. However, this research is currently limited to single-stream scenarios.

B EXPERIMENTAL DETAILS

B.1 Datasets

We evaluate the proposed SEP framework by conducting experiments on seven real-world datasets(heterogeneous), including:

- **Book** First 1000M byte of BookCorpus, which is a large collection of free novel books written by unpublished authors that contains 11,038 books in 16 different sub-genres [Zhu *et al.*, 2015].
- **Enwik9** First 1000M byte of the English Wikipedia. Enwik9 contains 243,426 article titles, of which 85,560 are links, and the rest are regular articles [Mahoney, 2011].
- **Float** Scientific IEEE 754 64-Bit Double-Precision Floating-Point Datasets [Burtscher and Ratanaworabhan, 2008].
- **Sound** The ESC-50 dataset is a labeled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification [Piczak, 2015].
- **Image** 100000 pictures from ImageNet. The ImageNet dataset contains 14,197,122 annotated images according to the WordNet hierarchy [Deng *et al.*, 2009].
- **Backup** A disk backup from personal PC [Mao *et al.*, 2023].
- **Silesia** The intention of the Silesia corpus is to provide a data set of files that covers the typical data types used nowadays. The sizes of the files are between 6MB and 51MB [Deorowicz, 2007].

B.2 Performance Evaluation

Data Transfer Hiding.

In GPU computation with CUDA, overlapping techniques are employed to allow for concurrent execution of computation and data transfer, rather than following a sequential process. GPUs with a single copy engine are found to utilize PCIe in half-duplex mode for explicit memory transfers, thereby constraining the overlapping capability to one direction only. However, when device-mapped host memory is employed, full-duplex operations are achieved. GPUs equipped with dual copy engines, such as Nvidia Tesla A800, can achieve complete overlap in bidirectional computation and communication through the utilization of different streams. With these mechanism of GPU, we design multi-stream pipelines to support scalable compression tasks. It is worth mentioning that due to the longer training and calculation time, the time of data transfer hiding is not significant.

Arithmetic Encoder(AE) Computation& I/O Hiding

We break down the compression task into smaller parts and extensively analyzed performance using profiler tools [Rauschmayr *et al.*, 2022]. Our analysis reveal that, aside from the training phase, the AE computation time stands out. This lead us to consider utilizing the multi-core capabilities of CPUs to tackle the mutual-blocking issues among different hardware devices. We develop a queue-based process scheduling strategy to ensure the concurrent execution of AE and model training.

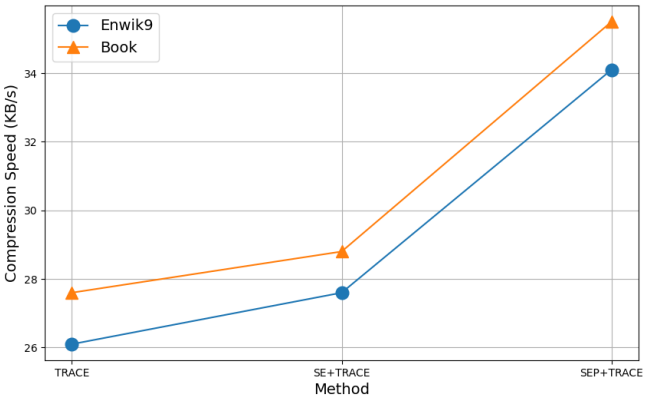


Figure 1: Different compression speed on TRACE, SE+TRACE and SEP+TRACE

Performance Details

To navigate the complexity and offer an evaluation of the performance of data compression process, we break down the complete compression process and analyze the time consumption of each part. The process encompasses PCIe transfers, disk I/O, and the concurrent execution of both AE computation and model training on the CPU and GPU.

We employ the TRACE model with semantice enhancement block(SE+TRACE) for experiments on the ENWIK9 dataset. Using the Profiler performance analysis tool and time-split statistics. A total rounds of the entire compression process is 1953000. The cumulative time for each part is:

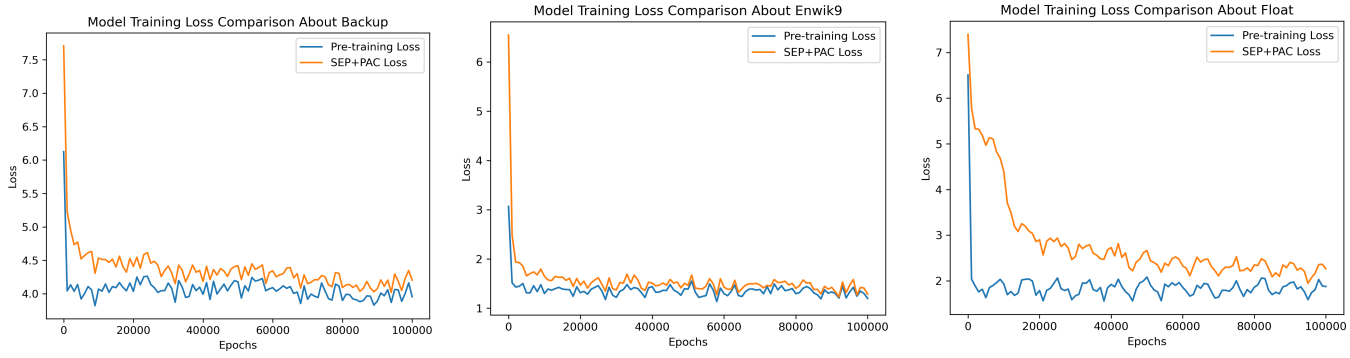
- $H2D_{\text{total}} = 185.51251745224s$
- $Train_{\text{total}} = 22873.427916765213s$
- $D2H_{\text{total}} = 326.0187122821808s$
- $I/O_{\text{total}} = 3.95140819549560547s$
- $AE_{\text{total}} = 13119.367810964584s$
- $ALL_{\text{total}} = 36507.025225400925s$

where $H2D_{\text{total}}$ refers to time consumption of host-to-device, $Train_{\text{total}}$ refers to time consumption of model training, $D2H_{\text{total}}$ refers to time consumption of device-to-host, AE_{total} refers to time consumption of arithmetic encoder, I/O_{total} refers to time consumption of disk reading and writing, and ALL_{total} refers to the total time consumption of the entire compression process.

From the results, it is evident that compressed models typically have a longer training time. The durations for AE are also relatively extended, while IO, H2D and D2H take shorter periods. We also conduct experiment with SEP+TRACE on ENWIK9 and the total time consumption is 28687s. It can be seen that we have hidden most of the AE and CPU-GPU data transfers time. The reason for not completely hiding all AE times may be due to some additional operations.

B.3 Pre-training Models

In the main paper, we have conducted a thorough analysis of the methodologies and outcomes associated with our pre-training model. This section is primarily focused on demon-



(a) Backup (b) Enwik9 (c) Float

(d) Image (e) Silesia (f) Sound

Figure 2: Comparision of Pre-training model and original model(SEP+PAC) on different Datasets

strating the effectiveness of the pre-training model. As evident from the Figure 2, it is clear that our model, upon being pretrained with a limited datasets, exhibits an enhanced capability to adapt swiftly to new data. This is particularly notable during the initial phases of compression, where the model rapidly reduces loss and improves the compression ratio without compromising on speed. The observed trends highlight the model's proficiency in learning from a constrained set of initial data, thereby setting a strong precedent for its application in scenarios where data availability is limited.

References

[Bellard, 2019] Fabrice Bellard. Lossless data compression with neural networks. *URL: https://bellard. org/nncp/nncp. pdf*, 2019.

[Blanchard and Tanner, 2013] Jeffrey D Blanchard and Jared Tanner. Gpu accelerated greedy algorithms for compressed sensing. *Mathematical Programming Computation*, 5(3):267–304, 2013.

[Burtscher and Ratanaworabhan, 2008] Martin Burtscher and Paruj Ratanaworabhan. Fpc: A high-speed compressor for double-precision floating-point data. *IEEE transactions on computers*, 58(1):18–31, 2008.

[Chen *et al.*, 2016] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. (arXiv:1604.06174), April 2016. arXiv:1604.06174 [cs].

[Collet and Kucherawy, 2018] Yann Collet and Murray Kucherawy. Zstandard compression and the application/zstd media type. Technical report, 2018.

[Deng *et al.*, 2009] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[Deorowicz, 2007] Sebastian Deorowicz. Silesia compression corpus, 2007.

[Deutsch, 1996] Peter Deutsch. Gzip file format specification version 4.3. Technical report, 1996.

[Díaz *et al.*, 2019] María Díaz, Raúl Guerra, Pablo Horstrand, Ernestina Martel, Sebastián López, José F López, and Roberto Sarmiento. Real-time hyperspectral image compression onto embedded gpus. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(8):2792–2809, 2019.

[Goyal *et al.*, 2021] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Dzip: Improved general-purpose loss less compression based on novel neural network modeling. In *2021 Data Compression Conference (DCC)*, pages 153–162. IEEE, 2021.

[Guo *et al.*, 2024] Cong Guo, Rui Zhang, Jiale Xu, Jingwen Leng, Zihan Liu, Ziyu Huang, Minyi Guo, Hao Wu, Shouren Zhao, Junping Zhao, and Ke Zhang. Gmlake: Ef-

ficient and transparent gpu memory defragmentation for large-scale dnn training with virtual memory stitching. In *ASPLOS*. arXiv, January 2024. arXiv:2401.08156 null.

[Huang *et al.*, 2020] Yu Huang, Yan Li, Zhaofeng Zhang, and Ryan Wen Liu. Gpu-accelerated compression and visualization of large-scale vessel trajectories in maritime iot industries. *IEEE Internet of Things Journal*, 7(11):10794–10812, 2020.

[Khayam, 2003] Syed Ali Khayam. The discrete cosine transform (dct): theory and application. *Michigan State University*, 114(1):31, 2003.

[Li *et al.*, 2014] Hao Li, Di Yu, Anand Kumar, and Yi-Cheng Tu. Performance modeling in cuda streams—a means for high-throughput data processing. In *2014 IEEE international conference on big data (big data)*, pages 301–310. IEEE, 2014.

[Li *et al.*, 2019] Jiaojiao Li, Jiaji Wu, and Gwanggil Jeon. Gpu acceleration of clustered dpcm for lossless compression of hyperspectral images. *IEEE Transactions on Industrial Informatics*, 16(5):2906–2916, 2019.

[Liu *et al.*, 2019] Qian Liu, Yiling Xu, and Zhu Li. Decmac: A deep context model for high efficiency arithmetic coding. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 438–443. IEEE, 2019.

[Mahoney, 2011] Matt Mahoney. Large text compression benchmark, 2011.

[Mao *et al.*, 2022] Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. Trace: A fast transformer-based general-purpose lossless compressor. In *Proceedings of the ACM Web Conference 2022*, pages 1829–1838, 2022.

[Mao *et al.*, 2023] Yu Mao, Jingzong Li, Yufei Cui, and Chun Xue. Faster and stronger lossless compression with optimized autoregressive framework. In *60th Design Automation Conference (DAC 2023): From Chips to Systems-Learn Today, Create Tomorrow*, 2023.

[Moffat, 2019] Alistair Moffat. Huffman coding. *ACM Computing Surveys (CSUR)*, 52(4):1–35, 2019.

[Nelson, 1989] Mark R Nelson. Lzw data compression. *Dr. Dobb's Journal*, 14(10):29–36, 1989.

[Piczak, 2015] Karol J Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1015–1018, 2015.

[Rajbhandari *et al.*, 2020] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. page 1–16, Atlanta, GA, USA, November 2020. IEEE.

[Rauschmayr *et al.*, 2022] Nathalie Rauschmayr, Sami Kama, Muhyun Kim, Miyoung Choi, and Krishnaram Kenthapadi. Profiling deep learning workloads at scale using amazon sagemaker. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3801–3809, 2022.

[Rhu *et al.*, 2016] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, page 1–13, October 2016.

[Rissanen and Langdon, 1979] Jorma Rissanen and Glen G Langdon. Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162, 1979.

[Simek and Asn, 2008] Vaclav Simek and Ram Rakesh Asn. Gpu acceleration of 2d-dwt image compression in matlab with cuda. In *2008 Second UKSIM European Symposium on Computer Modeling and Simulation*, pages 274–277. IEEE, 2008.

[Sourouri *et al.*, 2014] Mohammed Sourouri, Tor Gillberg, Scott B Baden, and Xing Cai. Effective multi-gpu communication using multiple cuda streams and threads. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 981–986. IEEE, 2014.

[Veldema and Philippsen, 2012] Ronald Veldema and Michael Philippsen. Parallel memory defragmentation on a gpu. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, page 38–47, Beijing China, June 2012. ACM.

[Xu *et al.*, 2015] Shijie Xu, Qi Guo, Gerhard Dueck, David Bremner, and Yang Wang. Metis: a smart memory allocator using historical reclamation information. In *ECOOP '15: European Conference on Object-Oriented Programming ECOOP 2015*, page 1–9, Prague Czech Republic, July 2015. ACM.

[Zhang and Zhang, 2019] Dengsheng Zhang and Dengsheng Zhang. Wavelet transform. *Fundamentals of image data mining: Analysis, Features, Classification and Retrieval*, pages 35–44, 2019.

[Zhu *et al.*, 2015] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.