Efficient Learned Data Compression via Dual-Stream Feature Decoupling

Huidong Ma[1,2], **Xinyan Shi**[1], **Hui Sun**[1], **Xiaofei Yue**[3],
Xiaoguang Liu[1*], **Gang Wang**[1*], **Wentong Cai**[2]
[1] College of Computer Science, TMCC, SysNet, DISSec, GTIISC, Nankai University
[2] College of Computing and Data Science, Nanyang Technological University
[3] School of Computer Science and Technology, Beijing Institute of Technology
[*] Corresponding authors
{mahd, liuxg, wgzwp}@nbjl.nankai.edu.cn

Abstract

While Learned Data Compression (LDC) has achieved superior compression ratios, balancing precise probability modeling with system efficiency remains challenging. Crucially, uniform single-stream architectures struggle to simultaneously capture micro-syntactic and macro-semantic features, necessitating deep serial stacking that exacerbates latency. Compounding this, heterogeneous systems are constrained by device speed mismatches, where throughput is capped by Amdahl's Law due to serial processing. To this end, we propose a Dual-Stream Multi-Scale Decoupler that disentangles local and global contexts to replace deep serial processing with shallow parallel streams, and incorporate a Hierarchical Gated Refiner for adaptive feature refinement and precise probability modeling. Furthermore, we design a Concurrent Stream-Parallel Pipeline, which overcomes systemic bottlenecks to achieve full-pipeline parallelism. Extensive experiments demonstrate that our method achieves state-of-the-art performance in both compression ratio and throughput, while maintaining the lowest latency and memory usage. The code is available at https://github.com/huidong-ma/FADE.

1 Introduction

With the rapid evolution of the Internet and AI-generated content technologies, multi-source data (spanning text, multimedia, and scientific sequences such as genomes and floating-point data) is experiencing explosive growth at a pace far surpassing Moore's Law (Sun et al., 2024a,b, 2025c,b,a). This surge imposes tremendous pressure on data transmission bandwidth and storage infrastructure. Traditional lossless compression algorithms, represented by Gzip (Gailly and Adler, 1992), zstd (Collet, 2015), and others (Seward, 1996; Deutsch, 1996), rely primarily on heuristic dictionary matching (e.g., LZ77 (Ziv and Lempel, 1977)) or statistical coding (e.g., Huffman (Huffman, 1952), ANS (Duda, 2013)). However, they struggle to effectively capture the high-order semantic redundancy in complex data, resulting in limited compression capability.

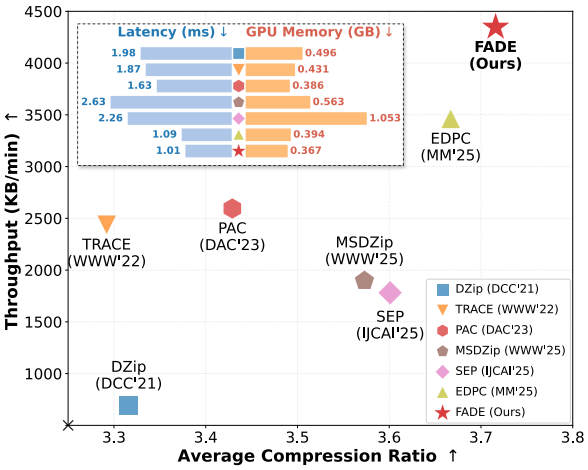

Figure 1: Trade-off between compression ratio and throughput. Top-right is better.

Recently, deep learning has revolutionized sequence modeling, enabling LDC to significantly outperform traditional methods in compression ratio (Sun et al., 2025b). Despite this progress, balancing precise probability modeling with system-level efficiency remains challenging due to two structural limitations: **First**, uniform single-stream architectures struggle to capture heterogeneous micro-macro patterns using unified parameters. Consequently, existing methods rely on deep Multilayer Perceptron (MLP) stacking to approximate complex distributions, a strategy that inevitably lengthens computational paths and severely exacerbates autoregressive decoding latency. **Second**, heterogeneous systems suffer from systemic throughput bottlenecks. The inherent speed mismatch between GPU probability generation and CPU arithmetic coding causes pipeline stalls, while autoregressive serial decoding remains strictly bound by Amdahl's Law (Amdahl, 1967), preventing parallel acceleration and restricting overall throughput.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7151–7164
July 2-7, 2026 ©2026 Association for Computational Linguistics

In this paper, we propose an efficient learned data compression method (**FADE**) that achieves superior compression ratios and high throughput, while maintaining low latency and GPU memory usage. Unlike existing methods, FADE reframes the modeling of complex dependencies by decoupling conventional deep serial processing into shallow parallel streams. Specifically, FADE employs a Dual-Stream Multi-Scale Decoupler (DMD) to disentangle features into a micro-syntactic Convolutional Neural Network (CNN) (LeCun et al., 2002; Ma et al., 2023a,b) branch and a macro-semantic MLP branch, and fuses these local and global features via the proposed Content-Adaptive Router. To ensure precise probability estimation, we incorporate a Hierarchical Gated Refiner (HGR) that leverages dynamic gating to inject stream-specific persistent memory for instance adaptation, while utilizing a high-capacity network to capture complex global dependencies. Furthermore, to break autoregressive serial constraints, we design a Concurrent Stream-Parallel Pipeline (CSPP) that hybridizes data parallelism with thread-safe, double-buffered temporal parallelism. Our contributions are summarized as follows:

- **Dual-Stream Multi-Scale Decoupler.** We propose the DMD, which decouples features into macro-semantics and micro-syntax, processes them concurrently via MLP and CNN branches, and fuses them using a content-adaptive router.
- **Hierarchical Gated Refiner.** We introduce the HGR, which performs coarse-to-fine refinement by constructing stream-aware context and achieving precise feature memorization and modeling to optimize the compression ratio.
- **Concurrent Stream-Parallel Pipeline.** We design the CSPP, which hybridizes data parallelism with thread-safe and double-buffered temporal parallelism, to achieve zero-wait processing and higher throughput.
- **SOTA Performance.** Extensive experiments on standard datasets demonstrate that FADE outperforms state-of-the-art methods in both compression ratio and throughput (see Figure 1).

2 Related Work

LDC methods typically combine a probability prediction model and an entropy coding algorithm. While the primary focus of current research lies in constructing accurate and lightweight probability models, a few recent works have targeted pipeline optimization to enhance throughput.

Neural Autoregressive Probability Models. Early research mainly leveraged Recurrent Neural Networks (RNNs) (Elman, 1990) and their variants to model sequential patterns. Specifically, LSTM-Compress (Knoll, 2017), NNCP (Bellard, 2019), DeepZip (Goyal et al., 2018), and DecMac (Liu et al., 2019) all adopted Long Short-Term Memory (LSTM) (Sepp and Jürgen, 1997) as their prediction model to capture long-range dependencies. To balance efficiency and performance, DZip (Goyal et al., 2021) proposed a semi-adaptive framework combining bootstrap and supporter models. Subsequently, the latest MSDLC (Ma et al., 2025b) improved modeling capability by introducing xLSTM (Beck et al., 2024). With technological evolution, methods based on Transformers (Vaswani et al., 2017) and Large Language Models (LLMs) have developed rapidly. NNCP v2 (Bellard, 2021) achieved excellent performance through relative positional encoding. TRACE (Mao et al., 2022b) significantly reduced inference latency by introducing a linear attention mechanism SLiM (Likhosherstov et al., 2021; Choromanski et al., 2020); LMIC (Delétang et al., 2023) and LLMZip (Valmeekam et al., 2023) established new state-of-the-art compression ratios leveraging pre-trained models but face enormous computational overhead. Hybrid ensembles like CMIX (Knoll, 2016) achieve exceptional compression performance but are practically limited by excessive computational complexity.

Lightweight Architectures and Feature Refinement. To address the slow inference of deep networks, MLP-based lightweight compression architectures such as OREO (Mao et al., 2022a) and PAC (Mao et al., 2023) have become a research hotspot, substantially boosting speed through masking and caching mechanisms. Recent research has further explored MLP potential to enhance feature representation. MSDZip (Ma et al., 2025a) designed a local-global-deep mixing block to stabilize cold-start training. SEP (Wan et al., 2025) introduced a semantics enhancement module to capture complex intra-patch relationships. EDPC (Lu et al., 2025) proposed a dual-path framework and a latent transformation engine to enrich feature flow and reduce GPU memory usage.

Parallelism and System Optimization. Beyond model architecture, parallelism is key to improving throughput. In terms of data parallelism, MSDLC introduced a parallel expansion mapper for chunk-

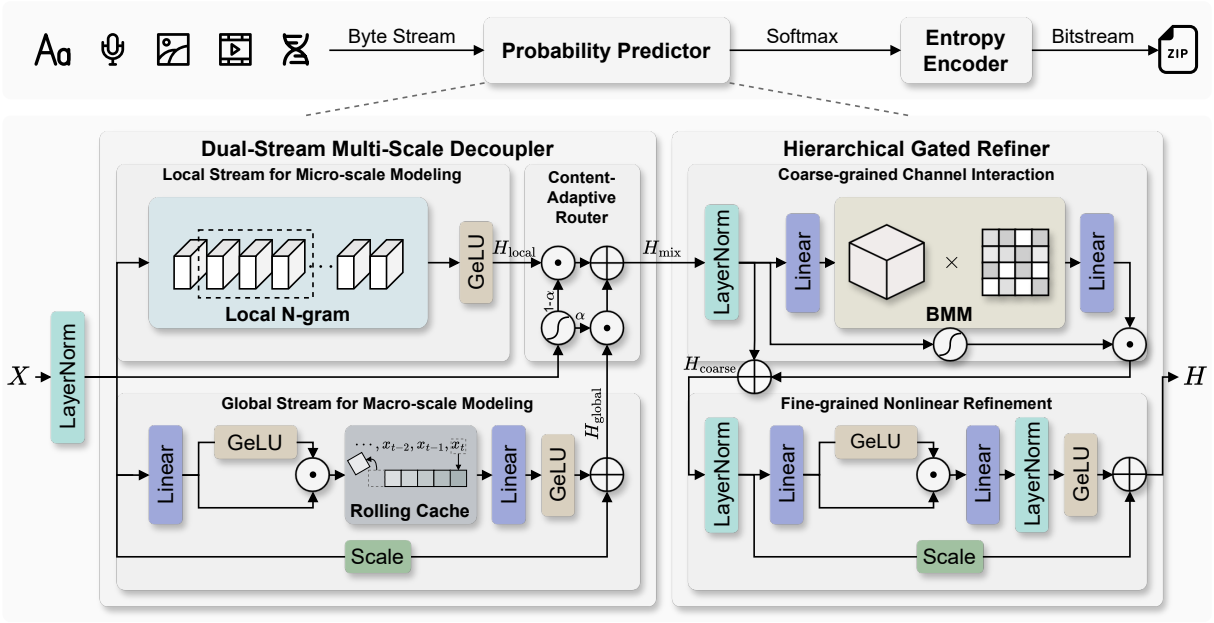

Figure 2: Overview of the proposed method. The embedded input $\boldsymbol{X}$ is disentangled into local and global contexts by the DMD, then fused and dynamically refined by the HGR to generate the final representation $\boldsymbol{H}$.

based data processing, while MSDZip proposed a stepwise-parallel strategy to accelerate large-scale data compression using multiple GPUs. Regarding pipeline optimization, SEP designed multi-stream pipelines, effectively masking I/O and transmission latencies. EDPC further decoupled probability prediction from arithmetic coding, realizing heterogeneous GPU-CPU parallelism.

3 Method

3.1 Preliminaries

Problem Formulation. LDC aims to map a discrete sequence of symbols $\boldsymbol{S} = \{x_1, \ldots, x_n\}$ into the shortest bitstream. According to Shannon's source coding theorem, the expected coding length is lower-bounded by the entropy $H(\boldsymbol{S}) = -\sum P(\boldsymbol{S}) \log_2 P(\boldsymbol{S})$. Since the joint probability decomposes as $P(\boldsymbol{S}) = \prod_t P(x_t|x_{<t})$, approaching this theoretical limit relies on the accurate estimation of the conditional probability $P(x_t|x_{<t})$.

Autoregressive Framework. As shown in Alg. 1, to approximate this theoretical limit, LDC adopts an autoregressive framework including two phases:

- **Compression Phase.** At step t, the network $\mathcal{M}$ processes the history context $x_{<t}$ to predict the conditional distribution $\hat{p}_t$. An entropy encoder (e.g., Arithmetic Coding (Witten et al., 1987)) then utilizes this probability estimate to compress the target symbol x_t into the bitstream.
- **Decompression Phase.** Operating as the inverse process while maintaining strict causality, the decoder employs the identical network $\mathcal{M}$ on the previously decoded context to reconstruct $\hat{p}_t$. Subsequently, the entropy coding algorithm recovers x_t from the bitstream and appends it to the history for the next iteration.

Building upon this autoregressive framework, we propose FADE. The overall architecture is illustrated in Figure 2. The subsequent sections will elaborate on the primary innovations within our predictor design.

3.2 Dual-Stream Multi-Scale Decoupler

Analysis. Information-theoretic studies (Shannon, 1948; Khandelwal et al., 2018) reveal that data sequences exhibit dual dependency patterns: micro-syntactic dependencies governed by local regularities (e.g., N-gram patterns) and macro-semantic dependencies spanning long-range context, empirically verified in Figure 3. Existing LDC methods primarily employ MLPs for rapid inference. However, the single-stream MLP inherently functions as a full-scale mixer, attempting to fit these heterogeneous features using a shared set of parameters. As illustrated in Figure 3 (c), this results in a diffuse saliency distribution that fails to capture sharp micro-syntactic fluctuations, leading to multi-scale interference. To compensate for this lack of specialized inductive bias, existing methods are often compelled to stack deeper layers to approximate complex distributions. While this strategy marginally improves representation, the increased computational depth forces a long sequential exe-

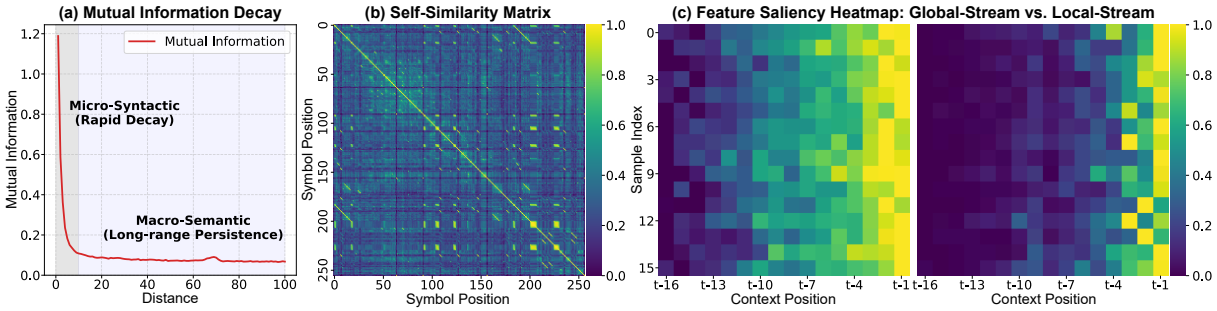

Figure 3: Verification of dual dependency patterns on Silesia. (a) Mutual information decay exhibits a sharp initial drop (micro-syntactic) followed by a persistent non-zero tail (macro-semantic). (b) The self-similarity matrix corroborates this observation via a prominent diagonal band and recurring off-diagonal blocks. (c) Feature saliency heatmaps of the Global and Local streams, illustrating the distinct patterns captured by each branch.

cution path, directly translating to higher latency.

Design. We propose the Dual-Stream Multi-Scale Decoupler (DMD). By implementing explicit feature decoupling, DMD processes features via parallel streams with distinct inductive biases. Crucially, this design simultaneously compensates for saliency dilution and replaces deep serial stacking with shallow parallel execution. Formally, given the input sequence embedding $\boldsymbol{X}_{\text{emb}} \in \mathbb{R}^{B\times T\times D}$ (with batch size B, time steps T, and embedding dimension D) and the flattened normalized input $\boldsymbol{X} \in \mathbb{R}^{B\times 1\times D_h}$ (where $D_h = T \times D$), the processing workflow is formulated as follows:

(1) Global Stream for Macro-scale Modeling. Dedicated to macro-semantic modeling, this stream employs a GeGLU-based Rolling Cache (Dauphin et al., 2017; Shazeer, 2020; Mao et al., 2023; Lu et al., 2025) to capture long-range dependencies. This design enhances the nonlinear expressivity of historical context while maintaining inference efficiency. Specifically, we maintain a latent cache $\boldsymbol{M} \in \mathbb{R}^{B\times 1\times D_{cache}}$, which is updated at step t by integrating the latest feature via a rolling operation:

$$\text{GeGLU}(\boldsymbol{X}_t) = \psi(\boldsymbol{X}_t\boldsymbol{W}_g) \odot (\boldsymbol{X}_t\boldsymbol{W}_v) \quad (1)$$

$$\boldsymbol{M}_t = \text{Roll}(\boldsymbol{M}_{t-1}, \text{GeGLU}(\boldsymbol{X}_t)) \quad (2)$$

where $\boldsymbol{X}_t \in \mathbb{R}^{B\times 1\times D}$ denotes the embedding of the latest symbol (i.e., the last D channels of the input $\boldsymbol{X}$), $\odot$ denotes element-wise multiplication, and ψ is the GeLU activation function (Hendrycks and Gimpel, 2016). The updated cache is then projected back into the output space to yield the global feature $\boldsymbol{H}_{\text{global}} \in \mathbb{R}^{B\times 1\times D_h}$:

$$\boldsymbol{H}_{\text{global}} = \psi(\boldsymbol{M}_t\boldsymbol{W}_m) + \lambda_m \cdot \boldsymbol{X} \quad (3)$$

where $\boldsymbol{W}_m \in \mathbb{R}^{D_{\text{cache}}\times D_h}$ denotes the projection matrix, and λ_m is a learnable residual scaling factor initialized to 1.

(2) Local Stream for Micro-scale Modeling. To address the multi-scale interference, we introduce a lightweight Local Stream serving as a micro-syntactic decoupler. This branch employs a 1D convolution (Bai et al., 2018) to impose a strong local inductive bias, yielding the local feature $\boldsymbol{H}_{\text{local}} \in \mathbb{R}^{B\times 1\times D_h}$:

$$\boldsymbol{H}_{\text{local}} = \text{Flatten}(\psi(\text{Conv}(\text{LN}(\boldsymbol{X}_{\text{emb}})))) \quad (4)$$

where $\text{LN}(\cdot)$ denotes Layer Normalization (Ba et al., 2016). As illustrated in Figure 3 (c), this branch exhibits a sharply localized response pattern. It precisely captures micro-syntactic N-gram patterns while filtering out long-range noise, thereby successfully offloading the syntactic matching task from the global stream.

(3) Content-Adaptive Router. To achieve dynamic fusion of multi-scale features, we introduce a Content-Adaptive Router. This module generates routing weights $\boldsymbol{\alpha} \in \mathbb{R}^{B\times 1\times D_h}$ conditioned on the input context via a matrix $\boldsymbol{W}_r \in \mathbb{R}^{D_h\times D_h}$:

$$\boldsymbol{\alpha} = \sigma(\boldsymbol{X}\boldsymbol{W}_r) \quad (5)$$

where σ represents the Sigmoid activation function. The final fused representation $\boldsymbol{H}_{\text{mix}} \in \mathbb{R}^{B\times 1\times D_h}$ is computed as:

$$\boldsymbol{H}_{\text{mix}} = \boldsymbol{\alpha} \odot \boldsymbol{H}_{\text{global}} + (1 - \boldsymbol{\alpha}) \odot \boldsymbol{H}_{\text{local}} \quad (6)$$

3.3 Hierarchical Gated Refiner

Analysis. While the DMD effectively integrates multi-scale features along the temporal dimension, its reliance on globally shared parameters limits its adaptability to the non-stationary feature distribution shifts inherent in online compression. In real-world scenarios, instance-specific context exhibits highly heterogeneous channel interaction patterns. Consequently, shared weights fail to achieve deep

instance adaptation. Crucially, merely increasing network depth to capture these variations is ineffective; without selective filtering, it amplifies noise, compromising the probability estimation.

Design. To tackle these challenges, we introduce the Hierarchical Gated Refiner (HGR). This module adopts a cascaded strategy transitioning from coarse-grained channel interaction to fine-grained nonlinear refinement, facilitating deep instance adaptation by selectively enhancing high-order semantic features while suppressing noise propagation. Formally, given $\boldsymbol{H}_{\text{mix}} \in \mathbb{R}^{B\times 1\times D_h}$:

(1) Coarse-grained Channel Interaction. HGR captures global correlations via Block Matrix Multiplication. Crucially, since we employ online adaptation with stateful batching, the batch index k corresponds to a fixed data stream. Thus, we define $\boldsymbol{W}_U \in \mathbb{R}^{B\times d_h\times d_h}$ as a persistent memory, where the k-th slice evolves via backpropagation to capture unique patterns. This effectively achieves sample-adaptive channel mixing, thereby tailoring channel interactions to each specific input. We denote the resulting adaptive feature as $\boldsymbol{H}_{\text{a}} \in \mathbb{R}^{B\times 1\times D_{\text{h}}}$:

$$\boldsymbol{H}_{\text{a}} = \text{Up}(\text{BMM}(\text{Down}(\text{LN}(\boldsymbol{H}_{\text{mix}})), \boldsymbol{W}_U)), \quad (7)$$

where $\text{Down}(\cdot)$ and $\text{Up}(\cdot)$ denote the dimensionality-reducing and expanding projections (mapping between D_h and d_h), respectively (Lu et al., 2025). To mitigate noise accumulation from high-order interactions, HGR employs a content-aware self-gating mechanism to selectively suppress irrelevant features, yielding $\boldsymbol{H}_{\text{coarse}}$ via a matrix $\boldsymbol{W}_c \in \mathbb{R}^{D_h\times D_h}$:

$$\boldsymbol{H}_{\text{coarse}} = (\boldsymbol{H}_{\text{a}} \odot \sigma(\boldsymbol{H}_{\text{mix}}\boldsymbol{W}_c)) + \lambda_c \cdot \boldsymbol{H}_{\text{mix}}, \quad (8)$$

(2) Fine-grained Nonlinear Refinement. Building upon the coarse-grained interaction, we further conduct element-wise feature refinement via GeGLU and a projection matrix $\boldsymbol{W}_f \in \mathbb{R}^{D_e\times D_h}$, yielding the expanded representation $\boldsymbol{H}_{\text{expand}}$ and the final output $\boldsymbol{H} \in \mathbb{R}^{B\times 1\times D_h}$:

$$\boldsymbol{H}_{\text{expand}} = \text{GeGLU}(\text{LN}(\boldsymbol{H}_{\text{coarse}})) \quad (9)$$

$$\boldsymbol{H} = \psi(\text{LN}(\boldsymbol{H}_{\text{expand}}\boldsymbol{W}_f)) + \lambda_o \cdot \boldsymbol{H}_{\text{coarse}} \quad (10)$$

3.4 Concurrent Stream-Parallel Pipeline

Analysis. While advanced pipelines (Wan et al., 2025; Lu et al., 2025) mask device heterogeneity, they suffer from a critical limitation: Asymmetry of Parallelism. Existing methods accelerate compression but often revert to strictly serial execution during decompression due to autoregressive causality (i.e., x_t depends on $x_{<t}$). This results in a performance imbalance where decompression lags significantly behind.

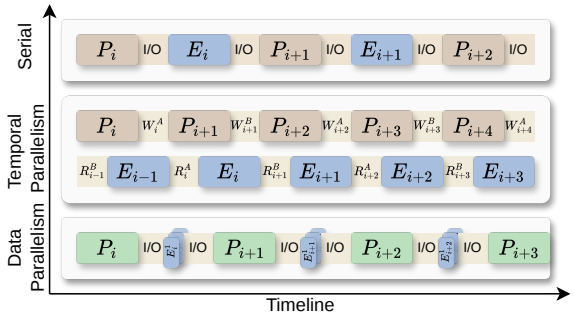

Figure 4: Illustration of execution strategies: Serial, Temporal Parallelism, and Data Parallelism. P and E denote the probability distribution prediction executed on the GPU and the entropy coding process performed on the CPU, respectively. $W^{A/B}$ and $R^{A/B}$ represent writing to and reading from Buffer A/B, respectively.

Design. To address this, we propose the Concurrent Stream-Parallel Pipeline (CSPP), a framework that synchronizes execution strategies across temporal and data dimensions (Figure 4).

(1) Temporal Parallelism. To bridge the speed mismatch, we implement an asynchronous pipeline with thread-safe ping-pong buffering. This design decouples producer-consumer threads into isolated memory regions. Unlike single-buffer queues prone to locking overhead, our zero-copy pointer swapping strategy eliminates memory contention. Crucially, this allows the GPU to continuously prefetch the next chunk while the CPU processes the current one, masking transmission latency.

(2) Data Parallelism. To resolve the autoregressive bottleneck, we tailor the data parallelism strategy specifically for the autoregressive workflow via a micro-step mechanism. We partition the input stream into N independent sub-streams to circumvent sequential dependency. While each substream maintains internal causality, we orchestrate N workers to execute them concurrently via a dual-barrier protocol. This effectively transforms the complexity from strictly serial $O(B)$ to parallel $O(B/N)$, boosting overall throughput to match the efficiency of compression.

In the compression phase, we integrate both Temporal and Data Parallelism to maximize throughput. In the decompression phase, due to autoregressive causality, we rely exclusively on Data Parallelism.

Dataset	Type	Size (MB)	Hartley (H_0)
Enwik9	text	954	7.69
LJSpeech	audio	281	8.00
TestImages	image	449	8.00
UVG	video	890	7.79
CESM	float	954	8.00
DNACorpus	genome	654	2.00
Silesia	heterogeneous	202	8.00

Table 1: Statistical information of the datasets. $H_0 = \log|\mathcal{A}|$, where $\mathcal{A}$ represents the alphabet set.

4 Experiments

4.1 Setup

Dataset. We employs representative datasets spanning 7 distinct domains, including Enwik9 (Mahoney, 2006), LJSpeech (Ito and Johnson, 2017), TestImages (Rawzor, 2008), UVG (Mercat et al., 2020), CESM (Zhao et al., 2020), DNACorpus (Pratas and Pinho, 2019), and Silesia (Deorowicz, 1985). Details are provided in Table 1.

Baselines. We compare our algorithm with 10 baselines, including 4 classic traditional algorithms: Gzip, 7z (Pavlov, 1999), zstd, and PBZip2 (Gilchrist, 2003); and 6 advanced online LDC algorithms: DZip, TRACE, PAC, MSDZip, EDPC, and SEP (based on PAC). Notably, to ensure a fair comparison at the algorithmic level, we forgo the multi-GPU setups used in MSDZip and SEP and instead evaluate all methods on a single GPU using PyTorch's default kernels.

Metrics. In this paper, we evaluate performance using Compression Ratio (CR) and Throughput (TP).

Settings. To ensure fair comparison, we set the batch size to 512 for all algorithms; all other hyperparameters follow their default settings. Consistent with the advanced method EDPC (Lu et al., 2025), we set the time steps to 16 and the embedding dimension to 32.

All experiments were conducted on a server equipped with an AMD EPYC 7402 24-Core Processor, and 4 × NVIDIA GeForce RTX 4090 GPUs. The server runs Ubuntu 22.04.5 LTS.

4.2 Results

4.2.1 Compression Ratio

We evaluate the CR of all methods on 7 datasets in Table 2. Overall, LDC methods significantly outperform traditional approaches. Among baselines, DZip and TRACE are constrained by limited dependency modeling or attention approximations. While PAC and MSDZip improve performance via masking strategies, and EDPC achieves a strong average CR of 3.667, FADE outperforms these approaches, establishing a new state-of-the-art with an average CR of 3.716. Benefiting from the macro-micro feature decoupling and hierarchical gated refinement, FADE captures multi-scale features more precisely, yielding remarkable gains on Enwik9 (6.288) and Silesia (5.400).

4.2.2 Throughput

Table 3 shows that FADE achieves the highest total throughput of 4347 KB/min, outperforming baselines by margins ranging from 25.6% to 525.5%. This disparity stems from pipeline architectures: conventional methods (e.g., PAC) are capped by serial stop-and-wait overheads, while EDPC is bottlenecked by serial decompression (2856 KB/min) despite parallel compression. In contrast, FADE utilizes CSPP to achieve full-pipeline parallelism. Crucially, by breaking autoregressive constraints

Method	Venue	Enwik9	LJSpeech	TestImages	UVG	CESM	DNACorpus	Silesia	Average↑
		text	audio	image	video	float	genome	hete.	
Traditional Compressor									
Gzip	-	3.100	1.168	1.359	1.578	1.369	3.685	3.133	2.199
7z	-	4.689	1.370	1.670	1.887	1.829	4.450	4.352	2.892
PBZip2	-	3.936	1.363	1.723	2.054	1.413	3.805	3.878	2.596
zstd	-	4.249	1.238	1.524	1.819	1.404	4.276	4.008	2.645
Learned Compressor									
DZip	DCC'21	5.758	1.257	2.146	2.456	2.488	4.448	4.661	3.316
TRACE	WWW'22	5.142	1.783	2.290	2.336	2.696	4.278	4.517	3.292
PAC	DAC'23	5.815	1.734	2.380	2.416	2.230	4.440	4.987	3.429
MSDZip	WWW'25	5.987	1.853	2.386	2.411	2.765	4.459	5.149	3.573
SEP	IJCAI'25	6.129	1.858	2.376	2.425	2.859	4.443	5.120	3.601
EDPC	MM'25	6.176	1.879	2.392	2.520	2.910	4.472	5.321	3.667
FADE (Ours)	-	**6.288**	**1.880**	**2.402**	**2.603**	**2.939**	**4.503**	**5.400**	**3.716**

Table 2: Compression ratios ↑ of all 11 compressors on 7 datasets. Bold values denote the best results.

Method	Pipeline (Cmp. | Decmp.)	Throughput (KB/min)↑ Cmp.	Decmp.	Total	Impr. (%)	FLOPs↓ (G)	Params↓ (M)	Latency↓ (ms)	PGMU↓ (GB)
DZip	Serial | Serial	466	1365	695	525.5	16.56	26.18	1.98	0.496
TRACE	Serial | Serial	2755	2187	2438	78.3	9.13	**2.37**	1.87	0.431
PAC	Serial | Serial	2898	2349	2595	67.5	**4.33**	8.48	1.63	0.386
MSDZip	Serial | Serial	1988	1814	1897	129.2	6.52	12.72	2.63	0.563
SEP	Parallel | Serial	1954	1636	1781	144.1	5.41	10.57	2.26	1.053
EDPC	Parallel | Serial	4391	2856	3461	25.6	7.10	13.84	1.09	0.394
FADE (Ours)	Parallel | Parallel	**4571**	**4144**	**4347**	-	7.83	15.20	**1.01**	**0.367**

Table 3: Efficiencies of LDC methods on Silesia. PGMU represents Peak GPU Memory Usage.

Module	Base +	CR↑	TP (KB/min)↑ Cmp.	Decmp.	Total
DMD	MLP	3.412	6353	4853	5502
	CNN	4.086	5600	4372	4910
HGR	CCI w/o gate	4.408	4983	3995	4435
	CCI w/ gate	4.565	4755	3836	4246
	FNR	5.400	3730	3188	3438
CSPP	CSPP	5.400	4571	4144	4347

Table 4: Ablation study on the progressive integration of proposed components on Silesia.

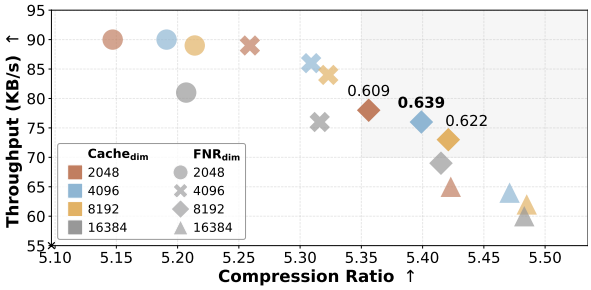

Figure 5: Analysis of different settings on Silesia.

via Concurrent Data Parallelism, FADE reduces decoding complexity from $O(B)$ to $O(B/N)$. This boosts decompression throughput to 4144 KB/min (45.1% higher than EDPC), ensuring balanced and maximized system efficiency.

4.2.3 Model Performance

Table 3 compares the computational efficiency of all models. The RNN-based DZip exhibits the highest FLOPs and parameter count, while TRACE suffers from high FLOPs. MSDZip and SEP show high Latency and PGMU. Notably, despite higher parameter count due to DMD and HGR, FADE achieves the lowest Latency and PGMU. This efficiency stems from the parallel execution of DMD branches and the use of fewer, computationally denser modules (large matrix operations) rather than deep stacking. This design minimizes kernel launch overhead, enabling superior compression and faster inference simultaneously.

4.3 Ablation Studies

4.3.1 Effectiveness of Components

To investigate the efficacy of each component, we conduct a progressive ablation study on the Silesia dataset, starting from the baseline model. Detailed results are provided in Table 4. Building on the MLP baseline, DMD utilizes local convolutions to resolve multi-scale interference, elevating the CR from 3.412 to 4.086. Subsequently, the integrated HGR employs coarse-grained channel interaction and fine-grained nonlinear refinement to facilitate deep instance adaptation while effectively suppressing noise propagation, significantly propelling the CR to 5.400. However, the associated computational overhead reduces the TP to 3438 KB/min. Finally, by applying the CSPP we resolve this system bottleneck via fine-grained parallel optimization. This restores the Total TP to 4347 KB/min (representing a substantial 26.4% gain) while preserving optimal compression performance, achieving a superior balance between efficiency and effectiveness.

4.3.2 Impact of Hidden Dimensions

We determine optimal model capacity via a grid search on Silesia, varying the hidden dimensions of Rolling Cache ($\text{Cache}_{\text{dim}}$) and FNR ($\text{FNR}_{\text{dim}}$) from 2048 to 16384. As shown in Figure 5, larger dimensions improve CR at the cost of latency. To identify the optimal trade-off among Pareto candidates, we calculate the Weighted Normalized Score which is defined as:

$$\text{Score} = \omega \cdot \frac{\text{CR} - \text{CR}_{\text{min}}}{\text{CR}_{\text{max}} - \text{CR}_{\text{min}}} + (1-\omega) \cdot \frac{\text{TP} - \text{TP}_{\text{min}}}{\text{TP}_{\text{max}} - \text{TP}_{\text{min}}} \quad (11)$$

where $\omega = 0.5$ balances the metrics, and min/max denote search space extremes. Under constraints of CR ≥ 5.30 and TP ≥ 70 KB/s, the combination of $\text{Cache}_{\text{dim}}$=4096 and FNR_{dim}=8192 yields the highest score (0.639). Consequently, we adopt this configuration as the default for all evaluations.

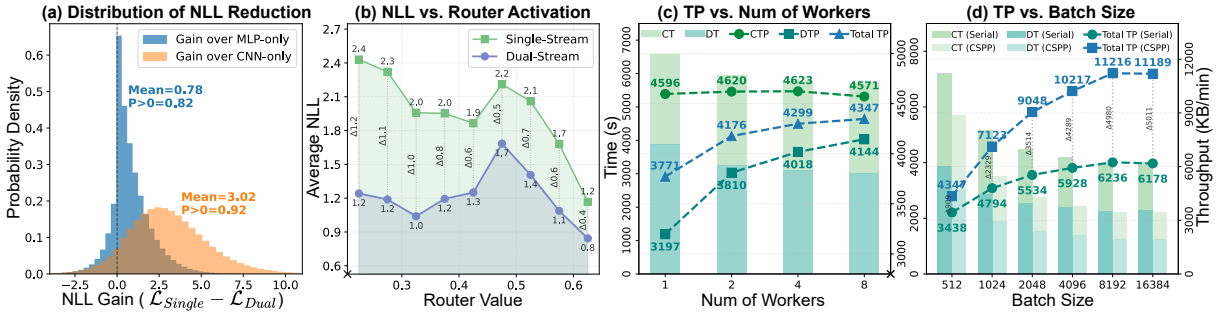

Figure 6: (a-b) NLL characteristics of the dual-stream architecture. (c-d) Impact of worker count and batch size on throughput. CT/DT and CTP/DTP denote Running Time and Throughput of Cmp./Decmp.

4.3.3 Single-Stream vs. Dual-Stream

To quantify the efficacy of the Dual-Stream Multi-Scale Decoupler, we employ Negative Log-Likelihood (NLL) to assess distribution fitting capability. The average NLL is defined as

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{T}\sum_{t=1}^{T} \ln P(x_t|x_{<t}) \tag{12}$$

Figure 6 (a) illustrates the NLL gains ($\Delta_{\text{NLL}} = \mathcal{L}_{\text{Single}} - \mathcal{L}_{\text{Dual}}$) of the dual-stream architecture over single-stream baselines. The significant gain over the MLP baseline confirms the local stream effectively offloads micro-syntactic tasks. Meanwhile, the advantage over CNN indicates that N-gram features alone are insufficient for modeling complex semantics. Thus, the dual-stream design establishes a benchmark unattainable by isolated architectures. Furthermore, Figure 6 (b) reveals the correlation between router activation and prediction difficulty, exhibiting an inverted-U trend. Both ends correspond to confidence zones with low NLL, demonstrating accurate routing to specialized branches. Conversely, peak NLL concentrates in the central ambiguity zone. Notably, the gap in the left region validates the local branch's decisive role in correcting micro-blind spots of the global model.

4.3.4 Scalability and Generalizability of CSPP

We investigate the impact of worker count and batch size on throughput, shown in Figure 6 (c) and (d). First, with batch size fixed at 512, increasing workers significantly increases both decompression TP (DTP) and total TP. The growth exhibits diminishing marginal returns, peaking at 4347 KB/min with 8 workers, which verifies the critical role of Data Parallelism in accelerating decoding. Conversely, CTP remains largely insensitive to worker count due to its reliance on Temporal Parallelism. Notably, a slight decline in CTP is observed at 8 workers, attributed to increased scheduling overhead and resource contention from managing threads. Consequently, we adopt 8 workers as the default for FADE. Second, fixing worker count at 8, both serial and parallel total TP increase with batch size. However, the parallel configuration exhibits much steeper growth, widening the performance gap against the serial baseline. Throughput saturates at a batch size of 8192, reaching a peak of 11,216 KB/min. This scalability confirms the efficacy of CSPP in maximizing hardware utilization.

Pipeline	TRACE	PAC	MSDZip	SEP	EDPC
Standard	2438	2595	1897	1781	3461
w/ CSPP (Ours)	3130	3260	2295	2046	4257
Impr. (%)	28.38	25.63	20.98	14.88	23.00

Table 5: Analysis of the CSPP across LDC methods.

To validate CSPP as a generic framework, we integrated it into baselines. As shown in Table 5, CSPP delivers consistent gains ranging from 14.88% to 28.38% across all methods. Notably, it accelerates serial baselines by enabling temporal and data parallelism. Crucially, CSPP even boosts the parallel-optimized EDPC by 23.00%, proving its portability in resolving residual bottlenecks.

5 Conclusion

In this paper, we propose FADE, a general-purpose lossless data compressor that establishes a new state-of-the-art. FADE incorporates the Dual-Stream Multi-Scale Decoupler to decouple features and integrates the Hierarchical Gated Refiner for precise refinement. Furthermore, we propose the Concurrent Stream-Parallel Pipeline, which resolves the serial processing bottleneck and significantly boosts throughput. Experiments demonstrate that FADE achieves superior CR compared to baselines, while simultaneously maintaining the highest throughput and lowest GPU memory usage.

Limitations

Currently, Compression Ratio and Throughput stand as the paramount metrics in modern data compression. The design philosophy of FADE prioritizes these core objectives to meet stringent practical deployment demands. To eliminate the prohibitive serial processing bottleneck, FADE transitions strategically from a conventional deep serial architecture to a shallow parallel dual-stream framework via feature decoupling, hierarchical gated refinement, and a parallel pipeline. This architectural shift results in a marginal increase in FLOPs and parameters compared to LDC baselines other than DZip, as shown in Table 3. We consider this a deliberate trade-off necessary to achieve maximal parallelism. Crucially, this theoretical increase in computational cost does not impede real-world efficiency; as evidenced by our experiments, FADE maintains the lowest inference Latency and Peak GPU Memory Usage, successfully translating parallel computational capacity into superior speed.

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China under Grants (62272252, 62272253) and the China Scholarship Council (CSC) scholarship program. We used the large language model Gemini solely for grammar checking and language polishing.

References

Gene M Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. 2024. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*.

F Bellard. 2019. Nncp: Lossless data compression with neural networks. https://bellard.org/nncp/.

Fabrice Bellard. 2021. Nncp v2: Lossless data compression with transformer. *Preprint at Fabrice Bellard https://bellard. org/nncp/nncp_v2. pdf*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, and 1 others. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Yann Collet. 2015. Zstandard fast real-time compression algorithm.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 933–941. PMLR.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, and 1 others. 2023. Language modeling is compression. *arXiv preprint arXiv:2309.10668*.

Sebastian Deorowicz. 1985. Silesia corpus. https://sun.aei.polsl.pl// sdeor/index.php?page=silesia.

L Peter Deutsch. 1996. Deflate compressed data format specification version 1.3. RFC 1951, IETF. https://www.rfc-editor.org/rfc/rfc1951.

Jarek Duda. 2013. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Jean-loup Gailly and Mark Adler. 1992. Gzip: The GNU zip compression utility. http://www.gzip.org/. Accessed: 2025-01-03.

Jeff Gilchrist. 2003. pbzip2 - parallel bzip2 file compressor. https://compression.ca/pbzip2/.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. 2018. Deepzip: Lossless data compression using recurrent neural networks. *arXiv preprint arXiv:1811.08162*.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. 2021. Dzip: improved general-purpose lossless compression based on novel neural network modeling. In *2021 data compression conference (DCC)*, pages 153–162. IEEE.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

David A Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Keith Ito and Linda Johnson. 2017. The lj speech dataset. https://keithito.com/LJ-Speech-Dataset.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp nearby, fuzzy far away: How neural language models use context. *arXiv preprint arXiv:1805.04623*.

Byron Knoll. 2016. Cmix. https://github.com/byronknoll/cmix.

Byron Knoll. 2017. lstm-compress. https://github.com/byronknoll/lstm-compress.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 2002. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Valerii Likhosherstov, Krzysztof M Choromanski, Jared Quincy Davis, Xingyou Song, and Adrian Weller. 2021. Sub-linear memory: How to make performers slim. *Advances in Neural Information Processing Systems*, 34:6707–6719.

Qian Liu, Yiling Xu, and Zhu Li. 2019. 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.

Zeyi Lu, Xiaoxiao Ma, Yujun Huang, Minxiao Chen, Bin Chen, Baoyi An, and Shu-Tao Xia. 2025. Edpc: Accelerating lossless compression via lightweight probability models and decoupled parallel dataflow. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 7268–7276.

Huidong Ma, Hui Sun, Liping Yi, Yanfeng Ding, Xiaoguang Liu, and Gang Wang. 2025a. Msdzip: Universal lossless compression for multi-source data via stepwise-parallel and learning-based prediction. In *Proceedings of the ACM on Web Conference 2025*, pages 3543–3551.

Huidong Ma, Hui Sun, Liping Yi, Xiaoguang Liu, and Gang Wang. 2025b. Multi-source data lossless compression via parallel expansion mapping and xlstm. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Huidong Ma, Cheng Zhong, Danyang Chen, Haofa He, and Feng Yang. 2023a. cnnlsv: detecting structural variants by encoding long-read alignment information and convolutional neural network. *BMC bioinformatics*, 24(1):119.

Huidong Ma, Cheng Zhong, Hui Sun, Danyang Chen, and Haixiang Lin. 2023b. ricme: Long-read based mobile element variant detection using sequence realignment and identity calculation. In *International Symposium on Bioinformatics Research and Applications*, pages 165–177. Springer.

Matt Mahoney. 2006. Large text compression benchmark. https://www.mattmahoney.net/dc/textdata.html.

Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. 2022a. Accelerating general-purpose lossless compression via simple and scalable parameterization. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 3205–3213.

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

Yu Mao, Jingzong Li, Yufei Cui, and Chun Jason Xue. 2023. Faster and stronger lossless compression with optimized autoregressive framework. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE.

Alexandre Mercat, Marko Viitanen, and Jarno Vanne. 2020. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM multimedia systems conference*, pages 297–302.

Igor Pavlov. 1999. 7z official website. https://www.7-zip.org/.

Diogo Pratas and Armando J Pinho. 2019. A dna sequence corpus for compression benchmark. In *Practical Applications of Computational Biology and Bioinformatics, 12th International Conference*, pages 208–215. Springer.

Rawzor. 2008. Image compression benchmark. http://imagecompression.info/test_images/.

Hochreiter Sepp and Schmidhuber Jürgen. 1997. Long short-term memory. *Neural Computation MIT-Press*.

Julian Seward. 1996. The official website of the xz compressor.

Claude E Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Noam Shazeer. 2020. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Hui Sun, Yanfeng Ding, Liping Yi, Huidong Ma, Gang Wang, Xiaoguang Liu, Cheng Zhong, and Wentong Cai. 2025a. Pmklc: Parallel multi-knowledge learning-based lossless compression for large-scale genomics database. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 2725–2734.

Hui Sun, Huidong Ma, Feng Ling, Haonan Xie, Yongxia Sun, Liping Yi, Meng Yan, Cheng Zhong, Xiaoguang Liu, and Gang Wang. 2025b. A survey and benchmark evaluation for neural-network-based lossless universal compressors toward multi-source data. *Frontiers of Computer Science*, 19(7):1–16.

Hui Sun, Huidong Ma, Yingfeng Zheng, Haonan Xie, Meng Yan, Cheng Zhong, Xiaoguang Liu, and Gang Wang. 2024a. Lrcb: A comprehensive benchmark evaluation of reference-free lossless compression tools for genomics sequencing long reads data. In *2024 Data Compression Conference (DCC)*, pages 584–584. IEEE.

Hui Sun, Liping Yi, Huidong Ma, Yongxia Sun, Yingfeng Zheng, Wenwen Cui, Meng Yan, Gang Wang, and Xiaoguang Liu. 2025c. Genomics data lossless compression with (s, k)-mer encoding and deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 12577–12585.

Hui Sun, Yingfeng Zheng, Haonan Xie, Huidong Ma, Cheng Zhong, Meng Yan, Xiaoguang Liu, and Gang Wang. 2024b. Pqsdc: a parallel lossless compressor for quality scores data via sequences partition and run-length prediction mapping. *Bioinformatics*, 40(5):btae323.

Chandra Shekhara Kaushik Valmeekam, Krishna Narayanan, Dileep Kalathil, Jean-Francois Chamberland, and Srinivas Shakkottai. 2023. Llmzip: Lossless text compression using large language models. *arXiv preprint arXiv:2306.04050*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, and Łukasz Kaiser. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Meng Wan, Rongqiang Cao, Yanghao Li, Jue Wang, Zijian Wang, Qi Su, Lei Qiu, Peng Shi, Yangang Wang, and Chong Li. 2025. Sep: A general lossless compression framework with semantics enhancement and multi-stream pipelines. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence. IJCAI*, pages 3326–3334.

Ian H Witten, Radford M Neal, and John G Cleary. 1987. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540.

Kai Zhao, Sheng Di, Xin Lian, Sihuan Li, Dingwen Tao, Julie Bessac, Zizhong Chen, and Franck Cappello. 2020. Sdrbench: Scientific data reduction benchmark for lossy compressors. In *2020 IEEE international conference on big data (Big Data)*, pages 2716–2724. IEEE.

Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343.

A Algorithm Description

The procedure of the online LDC method is outlined in Algorithm 1. The model requires no pre-training; instead, parameters are initialized randomly and updated via backpropagation at each step (Line 9), which is synchronously replicated by the decoder to guarantee lossless reconstruction.

Algorithm 1: Compression Process of LDC method

Input: Byte Stream $\boldsymbol{S} = \{x_i\}_{i=0}^{|\boldsymbol{S}|-1}$, Time Step t
Output: Compressed File Φ

1 $P \leftarrow$ Initialize the Probability Predictor;
2 $E \leftarrow$ Initialize the Arithmetic Encoder;
3 **for** $i = 0$ **to** $t - 1$ **do**
4 $\quad p(x_i) \leftarrow$ Average probability $\frac{1}{256}$;
5 $\quad \epsilon(x_i) \leftarrow$ Apply E to encode x_i according to $p(x_i)$;
6 **for** $i = t$ **to** $|\boldsymbol{S}| - 1$ **do**
7 $\quad p(x_i|x_{i-t}, \ldots, x_{i-1}) \leftarrow$ Get probability of x_i using P;
8 $\quad \epsilon(x_i) \leftarrow$ Apply E to encode x_i according to $p(x_i)$;
9 $\quad$ Backpropagate to update P to minimize the loss;
10 Write binary data $\{\epsilon(x_i)\}_{i=0}^{|\mathcal{S}|-1}$ to the file Φ;

B Dataset Description

The detailed descriptions and links of used multi-source datasets are shown in Table 6.

Dataset	Type	Description	Link
Enwik9	text	First 10^9 bytes of the English Wikipedia dump on 2006.	Page
LJSpeech	audio	First 10,000 files of the LJSpeech audio dataset.	Page
TestImages	image	A classical 8-bit benchmark dataset for image compression evaluation.	Page
UVG	video	The video ShakeNDry from the UVG benchmark featuring 1080p 8-bit YUV format.	Page
CESM	float	First 10^9 bytes of floating-point data from the CESM-ATM climate dataset.	Page
DNACorpus	genome	A corpus of DNA sequences from 15 different species.	Page
Silesia	heterogeneous	A heterogeneous corpus of 12 files covering various file formats.	Page

Table 6: Descriptions and links of multi-source datasets.

C Detailed Information of Baselines

The implementation details and characteristics of the baselines are show in Table 7.

Method	Ref.	Version	Language	Methods	Link
Traditional Compressor					
Gzip	(Gailly and Adler, 1992)	1.10	C/C++	LZ77, HC	Page
7z	(Pavlov, 1999)	24.08	C/C++	LZ77, AC	Page
PBZip2	(Gilchrist, 2003)	1.1.13	C/C++	BWT, HC	Page
zstd	(Collet, 2015)	1.5.6	C/C++	LZ77, HC	Page
Learned Compressor					
DZip	(Goyal et al., 2021)	1.0	Python	RNN, AC	Page
TRACE	(Mao et al., 2022b)	1.0	Python	Transformer, AC	Page
PAC	(Mao et al., 2023)	1.0	Python	MLP, AC	Page
MSDZip	(Ma et al., 2025a)	1.0	Python	MLP, AC	Page
SEP	(Wan et al., 2025)	1.0	Python	MLP, AC	Page
EDPC	(Lu et al., 2025)	1.0	Python	MLP, AC	Page
FADE	-	1.0	Python	MLP, CNN, AC	Page

Table 7: Implementation details and characteristics of the baseline methods. **LZ77**: repeated strings are coded by offset and length of previous occurrence; **HC**: Huffman Coding; **BWT**: Burrows-Wheeler Transform; **AC**: Arithmetic Coding.

D More Experimental Results

D.1 Progressive Ablation Analysis via Loss Trends

In Section 4.3.1, we quantitatively validated the effectiveness of each component using CR. To provide a more intuitive verification, we further analyze the validation loss trajectories throughout the inference process, as visualized in Figure 7. The pure MLP baseline exhibits the highest entropy and significant volatility, highlighting the inherent instability of relying solely on global features. Notably, introducing the CNN-based local stream triggers a sharp reduction in loss, confirming the validity of the dual-stream decoupling strategy. Furthermore, the gated CCI variant consistently outperforms the non-gated version, proving that the gate effectively filters noise. Finally, the integration of the FNR achieves the lowest loss floor and the most stable convergence trajectory.

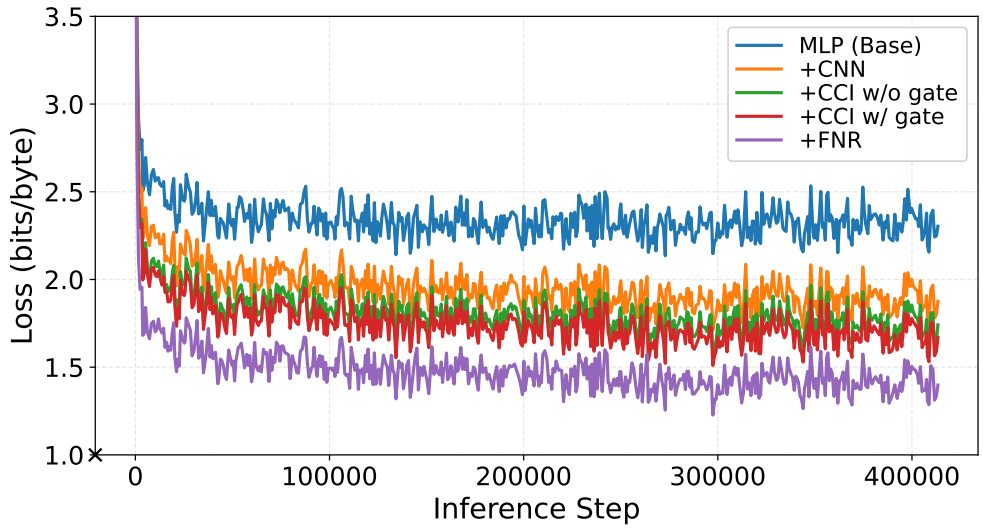

Figure 7: Model loss trajectories of progressive ablation study.

D.2 Analysis of Content-Adaptive Router

To verify the effectiveness of the Content-Adaptive Router, we visualize the dynamic variation of the routing weight α in the first 200 inference steps. As shown in Figure 8 (a), the router value exhibits high-frequency fluctuations (ranging from 0.278 to 0.546) rather than converging to a static constant. This rapid oscillation confirms the router's sensitivity to micro-contextual changes, allowing it to adjust the fusion strategy symbol-by-symbol. Consistent with these statistics, the distribution in Figure 8 (b) displays a unimodal pattern concentrated around this mean value. This indicates a general preference for the Local Stream while retaining the flexibility to incorporate global context when necessary.

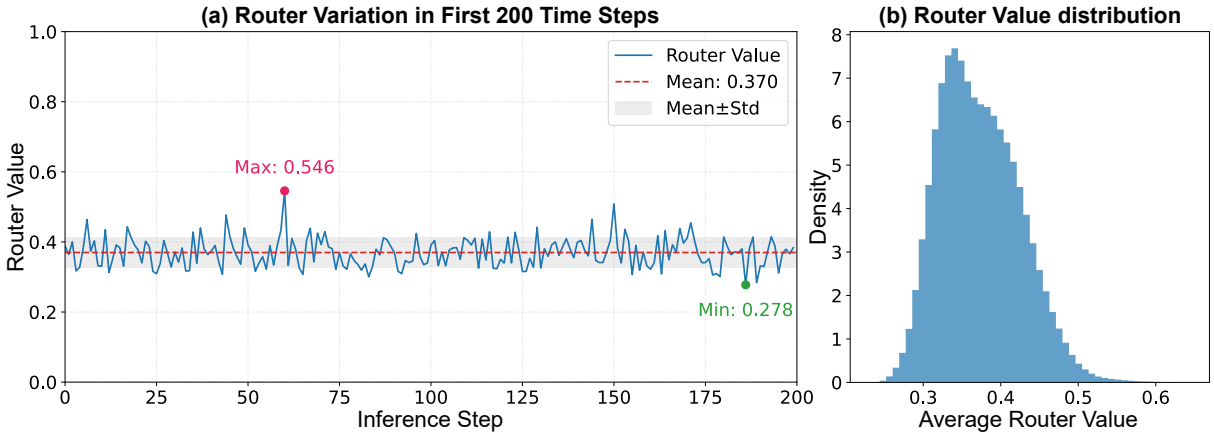

Figure 8: Analysis of the content-adaptive router value dynamics.

D.3 Impact of Batch Size on CR

To ensure a fair comparison with existing baselines, we set the default batch size to 512 for the primary evaluation in Section 4, consistent with their standard settings. Furthermore, we extend our investigation to analyze the impact of batch size on the CR across multi-source datasets, with detailed results presented in Table 8. The results indicate that larger batch sizes generally favor compression performance. Specifically, the model achieves the optimal average CR of 3.755 and 3.754 at batch sizes of 4096 and 8192, respectively. Breaking this down by domain, datasets like Enwik9, UVG, and DNACorpus peak at a batch size of 8192. In contrast, the heterogeneous dataset Silesia achieves its best compression at a smaller batch size of 1024, followed by a gradual decline.

Batch Size	Enwik9	LJSpeech	TestImages	UVG	CESM	DNACorpus	Silesia	Avg. CR	FLOPs	PGMU
	text	audio	image	video	float	genome	hete.		(G)	(GB)
512 (default)	6.288	1.880	2.402	2.603	2.939	4.503	5.400	3.716	7.83	0.367
1024	6.365	1.884	2.407	2.613	2.952	4.515	5.409	3.735	15.65	0.509
2048	6.423	1.888	2.410	2.623	2.951	4.548	5.390	3.748	31.31	0.796
4096	6.465	1.888	2.411	2.631	2.954	4.566	5.371	3.755	62.61	1.375
8192	6.491	1.887	2.409	2.643	2.948	4.568	5.335	3.754	125.22	2.532
16384	6.486	1.883	2.404	2.639	2.936	4.561	5.268	3.740	250.44	4.847

Table 8: Impact of batch size on compression ratio and performance.

To investigate the root cause of this divergence, we visualized the local entropy variations for Enwik9 and Silesia in Figure 9. As observed, Enwik9 exhibits consistent high-frequency fluctuations, indicating a stationary data distribution. Conversely, Silesia displays abrupt jumps and distinct blocky patterns, reflecting its non-stationary and highly heterogeneous nature. This suggests that for stationary sequences, larger batch sizes provide stable global statistics that enhance the HGR's refinement capability. However, for non-stationary data, excessive batch expansion dilutes local distinctiveness, making smaller batches more effective for capturing rapid distribution shifts.

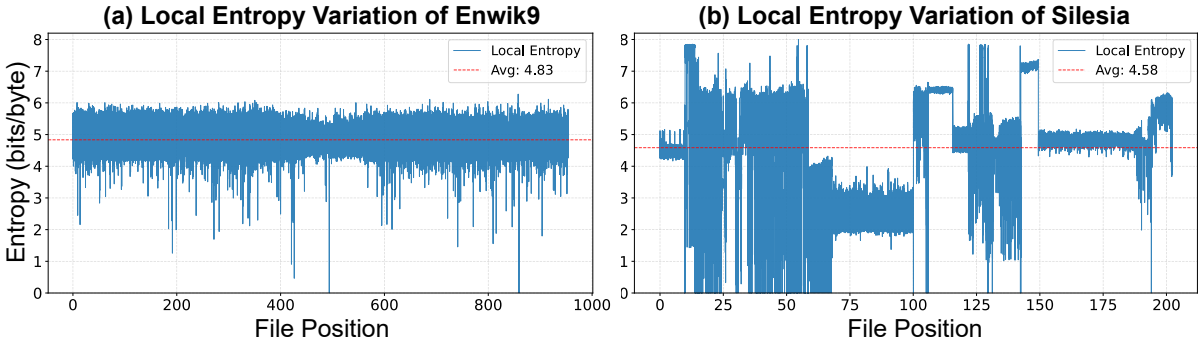

Figure 9: Analysis of datasets via local entropy.

Notably, this scaling benefit is not unbounded. When the batch size increases further to 16,384, the CR degrades across all datasets compared to 8192. This is attributed to context fragmentation, where the cumulative overhead from cold starts outweighs the statistical benefits. Consequently, for practical deployments, we recommend setting the batch size to 4096 or 8192 (contingent on hardware capacity) to achieve an optimal equilibrium between superior compression density and maximum throughput.