AgentGC: Evolutionary Learning-based Lossless Compression for Genomics Data with LLM-driven Multiple Agent

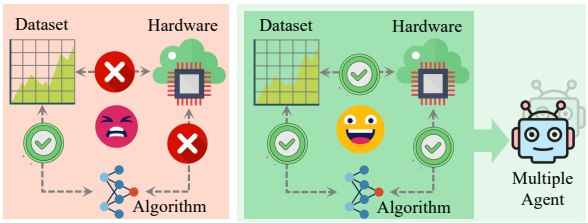
Anonymous ACL submission

Abstract

Lossless compression has made significant advancements in Genomics Data (GD) storage, sharing and management. Current learning-based methods are non-evolvable with problems of low-level compression modeling, limited adaptability, and user-unfriendly interface. To this end, we propose AgentGC [1], the first evolutionary **Agent**-based **G**D **C**ompressor, consisting of 3 layers with multi-agent named Leader and Worker. Specifically, the 1) User layer provides a user-friendly interface via Leader combined with LLM; 2) Cognitive layer, driven by the Leader, integrates LLM to consider joint optimization of algorithm-dataset-system, addressing the issues of low-level modeling and limited adaptability; and 3) Compression layer, headed by Worker, performs compression & decompression via a automated multi-knowledge learning-based compression framework. On top of AgentGC, we design 3 modes to support diverse scenarios: CP for compression-ratio priority, TP for throughput priority, and BM for balanced mode. Compared with 14 baselines on 9 datasets, the average compression ratios gains are 16.66%, 16.11%, and 16.33% , the throughput gains are 4.73×, 9.23×, and 9.15×, respectively.

1 Introduction

Genomic Data (GD), consisted of {A, C, G, T}, plays a crucial role in precision medicine, virus tracing, and new drug development (Sun et al., 2025a; Cheng and Hui, 2023). Recently, with the advancement of AI-driven bio-informatics Large Language Models (LLMs) such as AlphaFold2 (Yang et al., 2023b), GENERanno (Li et al., 2025a), and GenomeOcean (Zhou et al., 2025), along with third-generation genome sequencing technologies, GD has entered an era of massive big data (Ma et al., 2023; Sun et al., 2025c).



(a) Non-evolutionary (b) Evolutionary (Ours)

Figure 1: (a) The traditional non-evolutionary learning-based and (b) the proposed evolutionary scheme in AgentGC, which realizes collaborative modeling of "Algorithm-Dataset-Hardware".

For example, as of January 2026, the China National GeneBank Sequence Archive has backed up over 18,441.33 TB of big GD [2]. As a result, this presents significant challenges for GD storage and management. AI-based compression technology is an effective approach to alleviating this dilemma.

Although several multi-modal data compressors like MSDZip (Ma et al., 2025a) and LLMZip (Valmeekam et al., 2023) have been proposed, they fail to leverage the redundancy characteristics of GD for dedicate optimization, resulting in suboptimal compression ratio and throughput. Fortunately, GD-dedicated Learning-based Compressors (GDLCs), such as DeepGeCo (Sun et al., 2025d) and GenCoder (Sheena and Nair, 2024), have filled this gap through refined data modeling. However, in our investigation, existing GDLCs are non-evolutionary with fixed parameters and face the following challenges:

Low-level Compression Modeling. As illustrated in Figure 1(a), traditional GDLCs are non-evolutionary: they only capture latent correlations between algorithms and datasets, reflecting a limited modeling granularity. In contrast, as shown in Figure 1(b), we reframe compression as a system-level task by jointly considering the design of dataset, algorithm, and hardware memory.

[1] https://anonymous.4open.science/r/AgentGC-C0F7.

[2] https://db.cngb.org/cnsa/statistic

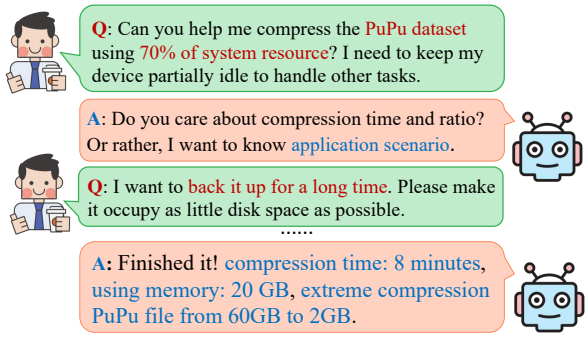


Figure 2: An example of genomic data lossless compression through interaction of LLM-based AgentGC.

Limited Adaptability. Existing GD-dedicated compressors are typically tailored to single-use scenarios, limiting their effectiveness in complex environments. For example: 1) Data persistence and backup applications require higher compression ratios; 2) Real-time transmission demands faster compression and decompression speeds; 3) Under memory constraints, balancing efficiency and performance becomes critical.

User-unfriendly Interface. The GDLCs involve the complex learning-based modeling, which introduces two disadvantages: 1) GDLCs encompass intricate model architectures, data distribution, parallelism, and additional configurations, making usage highly restricted for non-AI experts; and 2) Joint tuning of multiple parameters in GDLCs are complex, and incorrect parameter configurations negatively impact overall performance.

To address these issues, as shown in Figure 2, we propose AgentGC, the first novel evolutionary compression system for GD. The innovations and contributions of this paper are threefold:

1) We propose the first GD compression algorithm, AgentGC, which comprises a group of carefully designed leader agent and worker agent. Through dialogue-driven compression and automated LLM-based parameter tuning, AgentGC achieves more sophisticated modeling and provides a more user-friendly interface.

2) Through prompt optimization and architectural design, we designed three modes for AgentGC, the a) CP for compression-ratio priority, b) TP for throughput priority, and c) BM for balanced mode, offering enhanced flexibility and adaptability across diverse scenarios.

3) We conducted a comprehensive comparison between AgentGC and 14 baselines across 9 datasets. AgentGC achieved up to a 75.456% improvement in average compression ratio and a 18.995× increase in throughput. AgentGC also demonstrated overall superiority in compression robustness and CPU & GPU resource usage.

2 Problem Definition

Let $x = \{x_i\}_{i=0}^{m-1} \in \mathbb{R}^{1\times d_0}$ denote the inputed GD with length of m, the GDLC aim to generate a compact binary output $y = \{y_j\}_{j=0}^{n-1} \in \mathbb{R}^{d_1}$ with length of n. Here, d_0 and d_1 denote the length GD alphabet and the binary alphabet after compression, respectively. Generally, GDLC involves two stages: 1) Modeling, using an entropy function $\mathcal{F}$ to obtain the true probability distribution of x; and 2) Encoding, connecting an arithmetic encoder $\mathcal{E}$ to generate a compact representation. This process can be expressed as:

$$y = \mathcal{E}(\mathcal{F}(x)). \tag{1}$$

GDLC focus on modeling $\mathcal{E}(x)$ because it determines the compression efficiency of the encoder $\mathcal{E}$. However, obtaining the real-world $\mathcal{F}(x)$ for GD is challenging (Goyal et al., 2021; Sun et al., 2025d). Thus, GDLC fits $\mathcal{F}_\theta(x, v)$ by fixing a set of user-defined hyperparameters v and utilizing neural network techniques, where θ denotes the learned parameters. Therefore, the optimization objective of GDLC aims to minimize the difference $\mathcal{D}$ between $\mathcal{F}(x)$ and $\mathcal{F}_\theta(x, v)$:

$$y \simeq \mathcal{E}\{\mathcal{D}_{min.}(\mathcal{F}(x), \mathcal{F}_\theta(x, v))\}. \tag{2}$$

The Cross-Entropy (CE) loss is widely used to learn this difference due to its close relationship with Shannon entropy (Shannon, 1948). Therefore, $\mathcal{D}_{min.}$ can be denoted as:

$$\begin{aligned}\mathcal{L}_{min.} &\simeq \sum_{i=c+1}^{m} \mathrm{CE}(\hat{f}_i, f_i) \\ &= \sum_{i=c+1}^{m}\sum_{j=0}^{d_0-1} (\hat{f}_{i,j} log \frac{1}{f_{i,j}}).\end{aligned} \tag{3}$$

Here, $c \in \mathbb{R}$ $(1 \le c \le m-1)$ is a user-defined **context length**. $\hat{f}_i \in \mathbb{R}^{1\times d_0}$ and $f_i \simeq \mathcal{F}_\theta^i\{(x_i|x_{i-t}, ..., x_{i-1}), v\} \in \mathbb{R}^{1\times d_0}$ denote the one-hot encoded ground truth and neural-network-predicated probability vector for **target symbol** x_i using **historical symbols** $\{x_i|x_{i-t}, ..., x_{i-1}\}$.

In the enhanced GD compression system, the compressor automatically perceives and tunes the parameters v through an LLM-driven agent and the

user-provided prompt t. The preliminary definition is as:

$$v \simeq \text{LLM}\varphi(t) \tag{4}$$

Here, φ denotes the weight parameters of LLM. The LLM-driven compression system can be formally defined as:

$$\hat{y} \simeq \mathcal{P}(x|v) = \mathcal{P}\{\mathcal{E}(\mathcal{F}(x \mid \text{LLM}\varphi(t)))\} \tag{5}$$

Among them, $\mathcal{P}$ denotes the LLM-driven agent-based compression system, and $\hat{y}$ is the optimized binary output.

3 Related Work

Based on the learning process, we categorize GDLCs into static, dynamic, and hybrid.

Static: It consists of two independent stages: 1) Training a static neural network model $\mathcal{F}_{static}$ on the to-be-compressed data x, and 2) compressing x using well-trained $\mathcal{F}_{static}$ and input x. These compression methods include LSTM-based DeepDNA (Wang et al., 2018), biLSTM-attention-based DNA-BiLSTM (Cui et al., 2020), BERT-language-model-based CompressBERT (Yang et al., 2023a), convolutional-auto-encoder-based GenCoder (Sheena and Nair, 2024), Transformer-with-multi-level-grouping-based and Group of Bases (GoB)-LSTM-based LEC-Codec (Sun et al., 2024c), and GeneFormer (Cui et al., 2024). Recently, general-purpose compressors for multi-modal data have also performed well in compressing GD. These include traditional learning-based architectures such as DecMac (Liu et al., 2019) and DeepZip (Goyal et al., 2018), as well as LLM-based methods LMIC (Delétang et al., 2023), LLMZip (Valmeekam et al., 2023), and LM-Compress (Li et al., 2025b). Static methods are compression-efficient for large-scale GD but face challenges: 1) High pretraining time, leading to significant computational costs; 2) The necessity of storing the pretrained model, which affects generalization when applied to small-scale datasets; and 3) Expensive training resources, especially for LLM-based methods.

Dynamic: It adaptively updates the model $\mathcal{F}_{dynamic}$ during compression, eliminating the need for pretraining and model storage. Compared to static, it offers greater flexibility and efficiency in time consumption, but at the cost of compression ratio loss. AGDLC (Sun et al., 2025b) is the first dynamic GD compressor, consisting of two stages: 1) Extracting a coarse-grained redundancy feature using multi-(s,k)-mer representations; 2) Refining the probability distribution via XLSTM (Beck et al., 2024)-based modeling and performing real-time compression with arithmetic encoding. Some multi-modal compressors also follow this approach, like TRACE (Mao et al., 2022), PAC(Mao et al., 2023), ByteZip (Das et al., 2024), MSDLC (Ma et al., 2025b), MSDZip (Ma et al., 2025a) etal.

Hybrid: It is a combination of the above two architectures, aiming to balance compression ratio and time performance. DZip-Supporter (Goyal et al., 2021) is the earliest implementation of this type of method. It computes hybrid probabilities for multimodal data using a combination of pretrained model $\mathcal{F}_{static}$ and adaptive model $\mathcal{F}_{dynamic}$. DeepGeCo (Sun et al., 2025d) employs a Transformer-based dynamic model as its backbone while incorporating a BiLSTM-based static model to address the cold start problem. Additionally, it introduces three specialized modes (MINI, PLIS, and ULTRA) to accommodate complex scenarios. PMKLC (Sun et al., 2025a) is the latest hybrid architecture, designed with the Automated Multi-Knowledge Learning-based Compression Framework (AMKLCF) as its backbone. On top of this, it incorporates GPU-accelerated (s,k)-mer encoding and the Step-wise Model Passing (SMP) mechanism to accelerate computation.

Among the three aforementioned architectures, the hybrid approach offers a balanced solution. However, its complex parameter configurations and user-unfriendly operation limit its widespread adoption. Our proposed AgentGC integrates LLM and multi-agent system to effectively bridge this gap.

4 AgentGC Design

As shown in Figure 3, AgentGC consists of three layers and two group of agents (leader and worker). Among them, (a) User layer: It only exposes the chat interface (Figure 2), and the leader agent perceives compression scenarios and parameters through user Question & Answer; (b) Cognitive Layer: The leader agent interprets and optimizes parameters through LLMs, then returns the results to the worker for intelligent compression execution; and (c) Compression Layer: The worker agent learns the probability distribution of raw data based on the AMKLCF framework and generates a compact binary output file through arithmetic encoding. Meanwhile, the leader agent maintains interaction with it, continuously monitoring the system and

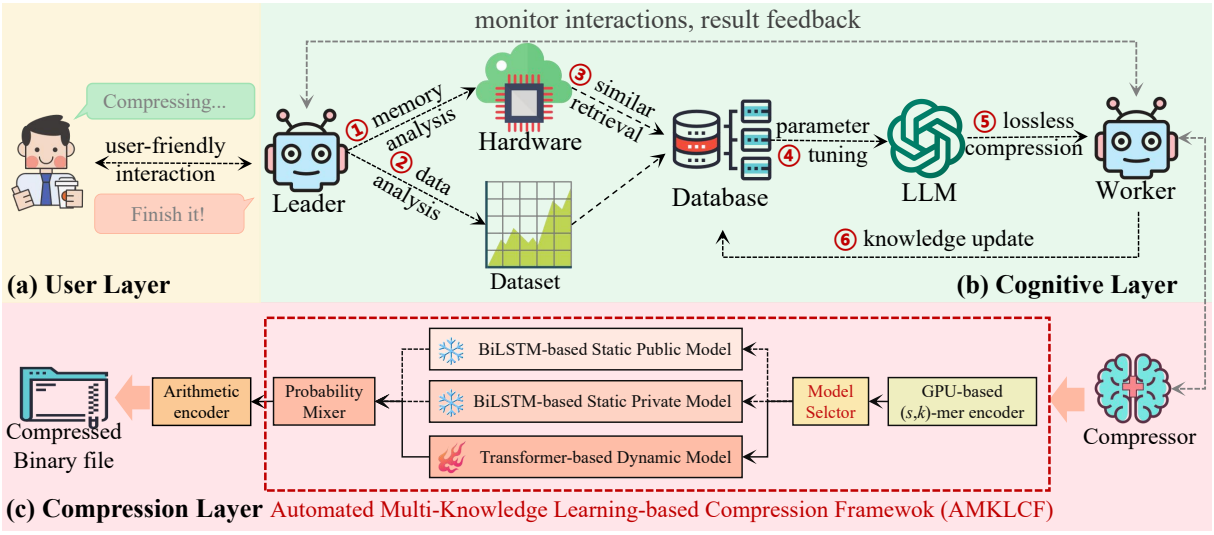


Figure 3: The pipeline of the proposed AgentGC compression system

returning log files in real time. Following, we will detail the cognitive and compression layers driven by the leader and worker agents.

4.1 The Leader Agent

As shown in Figure 3(b), the leader agent communicates with the user and executes global parameter optimization through the following cognitive perception sub-agents:

① **Memory Analysis:** It determines the available GPU memory by interacting with the system and sets the memory threshold $\alpha \in \mathbb{R}$ through interaction with the user. For example, in the leader agent inference mechanism, the user can input 70% resource utilization, after which the leader agent interacts with the LLM and system to infer the optimal CUDA memory usage. This ensures controlled resource allocation while enhancing the compression robustness.

② **Data Analysis:** It vectorizes the input GD $x = \{x_i\}_{i=0}^{m}$ to establish potential connections between the compressor and the dataset. To get the data vector $\beta \in \mathbb{R}^{1\times(4^k+1)}$, we introduce a GPU-accelerated (s, k)-mer (Sun et al., 2025a) with original data size, here s and k denote the stride and window length. It introduces the following advantages: 1) (s,k)-mer uses the redundancy of GD and is widely applied in computational biology (Xiao et al., 2018); 2) It is a fundamental component of the AMKLCF framework (as shown in Figure 3(b)), where parameter sharing simplifies computations and enhances efficiency; and 3) The (s,k)-mer does not expose raw data, it only computes probabilistic features, ensuring no risk of data privacy leakage.

③ **Similar Retrieval:** Assuming the database contains h vectors of historical compression parameters, this stage identifies potential $q \in \mathbb{R}$ ($1 \leq q \leq h$) vectors by sequentially computing the Euclidean Distance between each vector in database and the concatenated vector $\hat{v} = [\alpha, \beta]$. Here, the retrieved similar vectors are denoted as the $b_i = [c_i, \alpha_i, \beta_i, \gamma_i, \delta_i, \epsilon_i, \zeta_i]$ ($i = \{0, 1, ..., q-1\}$), where the c_i, α_i, β_i, γ_i, δ_i, ϵ_i, and ζ_i denote the context-length, peak memory resource usage, data vector, AMKLCF framework parameters, compression ratio, throughput, and GPU training batch-size of the i-th sample.

④ **Parameter Tuning:** As shown in Figure 4, the leader agent merges the q most similar retrieved historical vectors and into a prompt tailored for a specific application scenario, enabling the LLM to perform few-shot learning. Compared to Bayesian optimization and neural network fitting, it offers several advantages: 1) High data efficiency for compression scenarios where collecting large samples is difficult. 2) Low computational resource requirements, making cloud-hosted LLMs friendly for low-power devices or rapid deployment scenarios. 3) Quick adaptability, as LLM-based few-shot learning can efficiently adjust to new tasks.

⑤ **Lossless Compression:** At this stage, the leader agent invokes the worker agent to perform learning-based lossless compression, while dynamically monitoring and returning results throughout the compression process. This modular and transparent design enables the proposed AgentGC to quickly adapt to multi-file level parallelism by initializing worker agents.

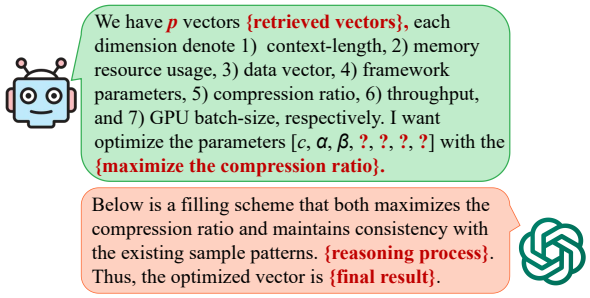


Figure 4: An example of the agent system derives optimized parameter vector via LLM-based few-shot prompt learning.

Algorithm 1: Compression Process

Input: input GD string $x = \{x_i\}_{i=0}^{m-1}$; the agent-optimized context-length c, models parameter γ, batch size ζ; as well as the (s, k)-mer encoder.
Output: compressed file y.

$\mathcal{F} \leftarrow$ Initializing probability predictor via γ and ζ;
$\mathcal{E} \leftarrow$ Initialize the arithmetic encoder;
$\{t_j\}_{j=0}^{\lfloor (m-k)/s \rfloor} \leftarrow$ Encoding x via (s,k)-mer;
for $i = 0$ **to** $c - 1$ **do**
  $f(t_i) \leftarrow$ Getting average probability $\frac{1}{4^k}$ using a uniform distribution;
  $\mathcal{E}(f(t_i)) \leftarrow$ Applying $\mathcal{E}$ to encode t_i via $f(t_i)$;
for $i = c$ **to** $\lfloor (m-k)/s \rfloor$ **do**
  $f(t_i) \leftarrow$ Getting the probability using learning-based predictor $\mathcal{F}(t_i|\{t_{i-c}, ..., t_{i-1}\})$;
  $\mathcal{E}(f(t_i)) \leftarrow$ Applying $\mathcal{E}$ to encode t_i via $f(t_i)$;
  Updating $\mathcal{F}$ to minimize the loss as shown in Eq. (3);
$y \leftarrow$ Merge all binary streams $\mathcal{E}(f(t_i))_{i=0}^{\lfloor (m-k)/s \rfloor}$;

4.2 The Worker Agent

As shown in Figure 3(c), the worker agent performs compression with two parts: the core AMKLCF, which provides the data's probability distribution, and the auxiliary arithmetic encoder, which generates the compact representation. The worker agent focuses on the prediction component, as it determines the effectiveness of the final arithmetic encoding. In this paper, the arithmetic encoding follows the configuration realized by (Goyal et al., 2021; Sun et al., 2025a).

The core components of AMKLCF include BiLSTM-based (Sun et al., 2025d) Static Public Model (SPuM), BiLSTM-based (Goyal et al., 2021) Static Private Model (SPrM), and Transformer-based (Mao et al., 2022) Dynamic Model (DM). During the compression process, the parameters of SPrM and SPuM are frozen, while those of DM need to be updated in real time. The probability mixture is implemented through a set of Model Selector (MS) and Probability Mixer (PM). The algorithm 1 presents the procedure by which the worker agent performs data compression.

Moreover, arithmetic encoding is reversible, by preserving the static models SPrM and SPuM and the dynamic DM, the worker agent can losslessly reconstruct the original string x from the compressed file y. Theoretical analysis can be found in (Jr., 1984; Delétang et al., 2023). Furthermore, we verify the integrity of the losslessly decompressed data via hash-based consistency checks. Details of the compression architecture see (Sun et al., 2025a). The detailed prompts for AgentGC are provided in the Appendix.

4.3 Compression Modes for AgentGC

We design three compression modes for AgentGC, determined by sorting the retrieved q sequence pairs according to their corresponding optimization object: 1) Compression-ratio Priority, CP: Based on distance computation, this mode retrieves the q samples with the lowest compression ratio for few-shot learning. It is suitable for long-term data management scenarios; 2) Throughput Priority, TP: It uses throughput as the primary objective for the compressor, making it particularly suitable for real-time transmission scenarios; 3) Balance Mode, BM: It represents a balanced working mode that jointly optimizes compression ratio, resource consumption, and throughput, making it suitable for general-purpose application scenarios.

Among the three modes, the leader agent of AgentGC autonomously perceives and selects the appropriate mode based on the inputted prompt which detailed in Appendix.

5 Experimental Results

5.1 Settings

Experiments were conducted on a Ubuntu server with 4 Intel Xeon 4310 CPUs and 4 NVIDIA GeForce RTX 4090 GPUs.

Datasets: As shown in Appendix Table 11, we conduct experimental evaluations on 9 real-world datasets covering diverse species and data scales. We also constructed an initialization vector database including 112 high-quality vector-text pairs derived from a multi-species dataset (Chen et al., 2020; Pratas and Pinho, 2019).

Baselines: As shown in Appendix Table 12, we compared AgenGC with 14 state-of-the-art

Table 1: Compression Ratio (bits/base ↓) of AgentGC and baselines. The CP, TP, and BM denote the three modes of the AgentGC: Compression-ratio Priority, Throughput Priority, and Balance Mode. DNA-BiLSTM* denotes the incorporation of a trained static model in the compressed file computation. The Adv. (↑) denote the compression ratio improvement defined as $\{Baseline - Ours(\text{CP mode})\}/Baseline \times 100\%$.

Methods	PlFa	WaMe	DrMe	GaGa	SnSt	MoGu	ArTh	AcSc	TaGu	Average	Adv.(%)
JARVIS3	1.896	2.015	1.952	1.946	1.951	1.940	1.855	1.873	1.940	1.930	+4.111
Spring	1.859	1.988	1.939	1.904	1.886	1.680	1.934	1.877	1.883	1.883	+1.746
NAF	1.874	1.988	1.990	1.959	1.967	1.799	1.929	1.905	1.943	1.928	+4.034
LZMA2	1.866	2.070	1.992	1.947	1.974	1.693	1.904	1.878	1.914	1.915	+3.388
XZ	1.868	2.074	1.993	1.948	1.978	1.694	1.907	1.879	1.916	1.917	+3.494
PPMD	1.906	2.109	2.054	2.026	1.995	1.780	2.008	1.972	1.998	1.983	+6.690
PBzip2	2.093	2.168	2.159	2.137	2.102	1.927	2.138	2.083	2.117	2.103	+11.995
PMKLC	1.827	1.953	1.907	1.858	1.877	1.652	1.897	1.867	1.844	1.854	+0.168
Gzip	2.120	2.250	2.220	2.187	2.195	2.030	2.185	2.140	2.163	2.166	+14.551
SnZip	3.640	3.742	3.694	3.698	3.711	3.526	3.695	3.638	3.663	3.667	+49.544
Lstm-compress	7.284	7.155	6.075	7.244	2.147	7.836	6.311	7.392	8.181	6.625	+72.069
DeepZip	1.901	2.034	1.916	1.862	1.982	1.650	1.935	1.868	1.851	1.889	+2.030
AGDLC	1.832	1.951	1.925	1.891	1.915	1.662	1.901	1.899	1.866	1.871	+1.116
DNA-BiLSTM*	12.523	12.428	4.904	2.559	17.235	2.421	6.780	2.053	2.015	6.991	+73.531
DNA-BiLSTM	1.856	1.946	1.926	1.914	1.908	1.861	1.913	1.919	1.924	1.907	+2.988
(Ours) AgentGC-CP	1.817	1.950	1.904	1.859	1.869	1.650	1.895	1.866	1.844	1.850	
(Ours) AgentGC-TP	1.851	1.967	1.916	1.868	1.870	1.656	1.905	1.871	1.853	1.862	—
(Ours) AgentGC-BM	1.829	1.952	1.905	1.859	1.881	1.656	1.896	1.871	1.847	1.855	—

compression methods, including 6 AI-based DNA-BiLSTM, DeepZip, Lstm-compress, AGDLC, PMKLC, and JARIVS3, as well as 8 traditional methods Spring, NAF, LZMA2, XZ, PPMD, PBZip2, Gzip, and SnZip.

Metrics: We evaluated AgentGC and baselines using Compression Ratio (CR), Throughput (THP), Compression Robustness Performance (CRP), as well as Average CPU (Avg-CPM) and GPU Memory usage (Avg-GPM). The CR reflects the number of bits required to store a single character. It is defined as (Sun et al., 2025c, 2024b,a):

$$\text{CR} \downarrow = \frac{Compressed\_Size}{Original\_Size} \times 8\ bits/base. \quad (6)$$

Throughput is time-related and reflects the computational efficiency of the algorithm:

$$\text{THP} \uparrow = \frac{Original\_File\_Size}{Total\_Time}. \quad (7)$$

Compression Robustness R is defined as the coefficient of variation (Sun et al., 2025a):

$$\text{CRP} \downarrow = \frac{\sqrt{\frac{1}{N} \times \sum_{i=0}^{N-1} (CR_i - CR_u)^2}}{CR_u} \times 100\%, \quad (8)$$

where N is the total number of datasets, CR_i and CR_u denote the CR value for i-th dataset and average CR, respectively.

LLM Settings: AgentGC is implemented based on the Google Agent Development Kit (ADK) [3]. Designed for cost-efficiency, AgentGC transmits only a small number of samples and prompts to a cloud-hosted LLM. Accordingly, we select GPT-5 API [4] as the backbone model, using its default LLM parameter configuration.

Parameters: AgentGC follows the parameter configurations in prior methods (Sun et al., 2025a,d) and experimental testing (see Fig. 5). Specifically, we configure $s = 3$, $k = 3$, and $q = 5$. Besides, the AMKLCF of AgentGC also employs a stepwise model-passing parallel strategy (Sun et al., 2025a; Ma et al., 2025a) to accelerate computation.

5.2 Compression Ratio

As illustrated in Table 1, AgentGC achieves overall optimal results across all datasets. The top three methods in CR are AgentGC-CP, PMKLC, and AgentGC-BM. Using the CP mode as the benchmark, AgentGC-CP achieves an average compression ratio improvement in 0.168-73.531% over the baselines. This advantage stems from AgentGC's integration of few-shot LLM-based parameter tuning and AMKLCF compression modeling.

Although the AgentGC's CR gains over PMKLC and AGDLC are relatively limited, AgentGC-CP demonstrates superior performance in THP and

[3] https://google.github.io/adk-docs
[4] https://openai.com

Table 2: Throughput (KB/s ↑) of AgentGC and baselines. CP, TP, and BM denote the CR Priority, THP Priority, and Balance Mode, respectively. The Overall (↑) is calculated at scale, using total time and total size of all files.

Methods	PlFa	WaMe	DrMe	GaGa	SnSt	MoGu	ArTh	AcSc	TaGu	Overall
DNA-BiLSTM	12.588	12.595	12.426	11.119	12.668	10.915	12.611	6.837	5.427	6.484
DeepZip	17.239	17.072	17.400	17.346	16.959	17.706	16.636	16.635	17.993	17.424
AGDLC	12.905	12.718	13.002	12.968	12.668	13.071	13.030	13.098	11.916	12.480
PMKLC	60.941	60.699	71.416	75.581	58.070	75.457	69.664	38.156	38.093	41.812
(Ours) AgentGC-CP	18.592	62.859	72.566	138.519	18.447	85.891	69.902	51.040	50.724	54.477
(Ours) AgentGC-TP	127.015	120.443	170.710	207.778	105.121	190.470	157.497	204.732	202.151	198.830
(Ours) AgentGC-BM	68.522	70.234	81.181	143.307	74.344	144.257	78.145	204.732	100.811	125.790

Table 3: Results of Compression Robustness CRP (% ↓), Average CPU Memory Avg-CPM (GB ↓), and Average CPU Memory Avg-GPM (GB ↓). Adv. denote the improvement of AgentGC take each mode as the benchmark.

Metrics and Improvement	**Baselines**				**Our Proposed**		
	DNA-BiLSTM	DeepZip	AGDLC	PMKLC	AgentGC-CP	AgentGC-TP	AgentGC-BM
Compression Robustness (%)	81.418	5.690	4.572	4.546	4.552	4.597	4.465
Adv. vs. AgentGC-CP (%)	+94.409	+19.997	+0.437	−0.131		+0.979	−1.958
Adv. vs. AgentGC-TP (%)	+94.354	+19.206	−0.547	−1.121	−0.989		−2.966
Adv. vs. AgentGC-BM (%)	+94.516	+21.534	+2.349	+1.792	+1.920	+2.881	
Average CPU Memory (GB)	9.306	9.126	6.527	3.100	3.099	1.354	1.352
Adv. vs. AgentGC-CP (%)	+66.701	+66.043	+52.527	+0.052		+0.883	+0.620
Adv. vs. AgentGC-TP (%)	+85.449	+85.161	+79.255	+56.324	+56.301		−0.152
Adv. vs. AgentGC-BM (%)	+85.471	85.158	+79.286	+56.390	+56.368	+0.152	
Average GPU Memory (GB)	1.957	1.481	0.982	0.832	1.192	1.051	1.451
Adv. vs. AgentGC-CP (%)	+39.112	+19.525	−21.425	−43.248		−13.420	+17.886
Adv. vs. AgentGC-TP (%)	+46.317	+29.047	−7.058	−26.299	+11.832		−27.602
Adv. vs. AgentGC-BM (%)	+25.850	+1.996	−47.873	−74.451	−21.782	−38.125	

memory cost, as shown in Tables 2 and 3. For example, compared to PMKLC, AgentGC-CP achieves a THP advantage of 30.291%. Compared to AGDLC, the three modes of AgentGC achieve speedup factors of 4.37×, 15.93×, and 10.08×, as well as achieve 52.53%, 79.26% and 79.28% CPU memory saving in CP, TP and BM mode, respectively.

5.3 Throughput

To ensure a fair evaluation, Table 2 gives the THP of AgentGC alongside learning-based baselines implemented via Python. Among them, AgentGC delivers substantial throughput gains. Specifically, the CP mode achieves speedups of 7.401×, 2.127×, 3.365×, and 0.303× over DNA-BiLSTM, DeepZip, AGDLC, and PMKLC-M, respectively. The BM mode yields improvements of 18.399×, 6.219×, 9.079×, and 2.008×, while the TP mode offers speedups of 29.663×, 10.411×, 14.932×, and 3.755× against the same baselines. This advantage stems from AgentGC's design, which models memory as a tunable parameter, elevating the modeling process from algorithmic to system level.

5.4 Compression Robustness

As shown in Table 3, AgentGC (BM) achieves the overall best compression robustness performance compared to DNA-BiLSTM, DeepZip, AGDLC, and PMKLC, with improvements of 94.516%, 21.534%, 2.349%, and 1.792%, respectively. This indicates that AgentGC is less sensitive to perturbations in data probability distributions, yielding stronger compression robustness. In addition, both CP and TP modes outperform DNA-BiLSTM and DeepZip overall, while slightly underperforming AGDLC and PMKLC.

5.5 Memory Usage

As illustrated in Table 3, AgentGC consistently achieves lower Avg-CPM than all baselines across its three modes, and demonstrates superior overall Avg-GPM (vs. DNA-BiLSTM and DeepZip). Using the BM mode as benchmark, AgentGC reduces Avg-CPM by 85.471%, 85.158%, 79.286%, and 56.390%, respectively. These results highlight AgentGC's strong potential for robust deployment on memory-constrained devices. AgentGC has a slightly higher GPU memory overhead than AGDLC and PMKLC, but its prompt-driven GPU tunability yields greater flexibility

5.6 Ablation Study

AgentGC's worker agents is operated upon the AMKLCF (Sun et al., 2025a), the ablation ex-

Table 4: Ablation study results of AgentGC. SPuM, SprM, DM, GskE, MS, SMP, and MGPU refer to the static public model, static private model, dynamic model, GPU-based (s,k)-mer encoder, model selector, step-wise model passing mechanism, and multi-GPU acceleration, within the worker agent-led AMKLCF compression framework. PCP, PTP, and PBM denote the leader agent's compression prompts for compression-ratio priority, throughput priority, and balanced mode. CR, THP, CPM, and DPM represent compression ratio (bits/base ↑), throughput (KB/s ↑), CPU memory (GB ↓), and GPU memory (GB ↓) consumption.

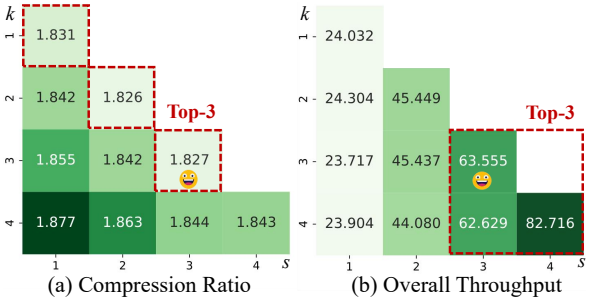
Mode	Ablation Module										Metrics			
	SPuM	SPrM	DM	GskE	MS	SMP	MGPU	PCP	PTP	PBM	CR	THP	CPM	GPM
A	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	1.903	8.162	1.589	0.503
B	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	1.993	5.921	3.606	3.220
C	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	1.993	5.751	3.626	3.220
D	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	2.022	16.967	1.873	3.222
S	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	1.893	22.628	1.210	0.505
E	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	1.901	82.506	1.047	0.505
M	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	1.900	72.087	1.049	0.505
CP	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	1.897	77.229	1.047	1.154
TP	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	1.909	161.821	1.051	0.949
BM	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	1.899	96.450	1.061	1.216

periments in Table 4 on OrSa dataset are divided into three groups accordingly.

Among the 10 modules, For the first group, -D significantly boosts throughput compared to the -A, -B, and -C modes, by introducing an (s,k)-mer encoder. Additionally, the -S mode incorporates a model selector to further reduce computational inference overhead, optimizing THP, CR, and memory usage. For the second group, AgentGC-E further improves THP under the -E mode by introducing multi-GPU support. When equipped with PMKLC's model SMP mechanism, it also achieves superior CR, as it effectively mitigates the cold-start problem (Ma et al., 2025a) in dynamic model deployment. For the last group, the leader agent leverages user-provided prompts and LLM-based few-shot parameter tuning to further optimize both CR and THP. Among the three modes, CR performance follows the order: "CP, BM, TP", while THP exhibits the opposite trend. These observations highlight AgentGC's flexible adaptability across diverse genomic compression scenarios.

5.7 The Impact of (s,k)-mer Encoder

As shown in Figure 5, AgentGC achieves the top-3 compression ratios with the (2,2)-mer, (1,1)-mer, and (3,3)-mer configurations, and ranks among the top-3 in compression and decompression throughput with (3,4)-mer, (3,3)-mer, and (4,4)-mer. Considering the trade-off between compression ratio and throughput, we adopt the (3,3)-mer as the default encoder configuration.



(a) Compression Ratio (b) Overall Throughput

Figure 5: The (a) compression ratio and (b) overall throughput of AgentGC (c = 32, GPU $batchsize$ = 320) on PlFa dataset using different (s,k)-mer encoder.

6 Conclusion and Limitation

In this paper, we propose AgentGC, the first novel LLM-driven genomic data compressor that effectively addresses the challenges faced by non-evolutionary methods, including low-level compression modeling, limited adaptability, and user-unfriendly interface. Compared to baselines, it achieves peak improvements of 73.53% in compression ratio, 2966.29% in throughput, and 94.52% in robustness. Besides, it delivers memory savings of 85.47% in CPU and 46.32% in GPU.

The high resource consumption of deep learning and GD scenarios may limit the scalability of AgentGC in large-scale, general-purpose scenarios. Our future studies include: 1) Hybrid acceleration leveraging CPU-GPU heterogeneous architectures; 2) Fine-tuning LLMs using compression-oriented instructions to enhance Q&A comprehension and reasoning capabilities, and 3) Extending AgentGC to lossless compression of multi-modal data.

References

Vaggelis Atlidakis, Jeremy Andrus, Roxana Geambasu, Dimitris Mitropoulos, and Jason Nieh. 2016. Posix abstractions in modern operating systems: The old, the new, and the missing. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–17.

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*.

Shubham Chandak, Kedar Tatwawadi, Idoia Ochoa, Mikel Hernaez, and Tsachy Weissman. 2019. Spring: a next-generation compressor for fastq data. *Bioinformatics*, 35(15):2674–2676.

Feng Zhen Chen, Li Jin You, Fan Yang, Li Na Wang, Xue Qin Guo, Fei Gao, Cong Hua, Cong Tan, Lin Fang, Ri Qiang Shan, and 1 others. 2020. Cngbdb: China national genebank database. *Yi chuan= Hereditas*, 42(8):799–809.

ZHONG Cheng and SUN Hui. 2023. Parallel algorithm for sensitive sequence recognition from long-read genome data with high error rate. *Journal on Communication/Tongxin Xuebao*, 44(2).

John G. Cleary and W. J. Teahan. 1997. Unbounded length contexts for PPM. *Comput. J.*, 40(2/3):67–75.

John G. Cleary and Ian H. Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun.*, 32(4):396–402.

Wenwen Cui, Zhaoyang Yu, Zhuangzhuang Liu, Gang Wang, and Xiaoguang Liu. 2020. Compressing genomic sequences by using deep learning. In *International Conference on Artificial Neural Networks*, pages 92–104. Springer.

Zhanbei Cui, Tongda Xu, Jia Wang, Yu Liao, and Yan Wang. 2024. Geneformer: Learned gene compression using transformer-based context modeling. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8035–8039. IEEE.

Satyajit Das and 1 others. 2024. Bytezip: Efficient lossless compression for structured byte streams using dnns. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Erika De Francesco, Giuliana Di Santo, Luigi Palopoli, and Simona E Rombo. 2009. A summary of genomic databases: overview and discussion. *Biomedical Data and Applications*, pages 37–54.

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*.

Peter M. Fenwick. 1996. The burrows–wheeler transform for block sorting text compression: principles and improvements. *The computer journal*, 39(9):731–740.

Jean-loup Gailly and Mark Adler. 1992. Gnu gzip. *GNU Operating System*, pages 8–18.

Lewis Y Geer, Aron Marchler-Bauer, Renata C Geer, Lianyi Han, Jane He, Siqian He, Chunlei Liu, Wenyao Shi, and Stephen H Bryant. 2010. The ncbi biosystems database. *Nucleic acids research*, 38(suppl_1):D492–D496.

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.

Ilya Grebnov. 2011. Bsc official website. https://github.com/IlyaGrebnov/libbsc. Accessed: 2025-07-31.

Andreas Ingo Grohmann, Johannes VS Busch, Adrian Bretschneider-Perez, Christopher Lehmann, and Frank HP Fitzek. 2022. Javris: Joint artificial visual prediction and control for remote-(robot) interaction systems. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pages 835–840. IEEE.

Xueqin Guo, Fengzhen Chen, Fei Gao, Ling Li, Ke Liu, Lijin You, Cong Hua, Fan Yang, Wanliang Liu, Chunhua Peng, and 1 others. 2020. Cnsa: a data repository for archiving omics data. *Database*, 2020:baaa055.

Gzip. 1996. Gzip official website. http://www.gzip.org/. Accessed: 2025-07-31.

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

Ipavlov. 2024. A free file archiver for extremely high compression. [Online]. Available: https://www.7-zip.org.

Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, and 1 others. 2018. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338–345.

Glen G. Langdon Jr. 1984. An introduction to arithmetic coding. *IBM J. Res. Dev.*, 28(2):135–149.

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

Byron Knoll. 2017. lstm-compress. https://github.com/byronknoll/lstm-compress. Accessed: 2025-07-31.

Kirill Kryukov, Mahoko Takahashi Ueda, So Nakagawa, and Tadashi Imanishi. 2019. Nucleotide archival format (naf) enables efficient lossless reference-free compression of dna sequences. *Bioinformatics*, 35(19):3826–3828.

Kirill Kryukov, Mahoko Takahashi Ueda, So Nakagawa, and Tadashi Imanishi. 2020. Sequence compression benchmark (scb) database—a comprehensive evaluation of reference-free compressors for fasta-formatted sequences. *GigaScience*, 9(7):giaa072.

Qiuyi Li, Wei Wu, Yiheng Zhu, Fuli Feng, Jieping Ye, and Zheng Wang. 2025a. Generanno: A genomic foundation model for metagenomic annotation. *bioRxiv*, pages 2025–06.

Z. Li, C. Huang, X. Wang, and 1 others. 2025b. Lossless data compression by large models. *Nature Machine Intelligence*, 7:794–799.

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.

Yuansheng Liu, Tao Tang, Zexuan Zhu, Xiangxiang Zeng, Quan Zou, and Keqin Li. 2025. Quality scores compression of genomic sequencing data: a comprehensive review and performance evaluation. *IEEE transactions on computational biology and bioinformatics*.

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, Hui Sun, Danyang Chen, and Haixiang Lin. 2023. 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.

Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. 2022. 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.

Larry R Medsker, Lakhmi Jain, and 1 others. 2001. Recurrent neural networks. *Design and applications*, 5(64-67):2.

Rituparna Mitra and Subhankar Roy. 2018. A survey of genome compression methodology. *International Journal of Computer Science and Engineering*, 6:983–991.

Alistair Moffat. 1990. Implementing the PPM data compression scheme. *IEEE Trans. Commun.*, 38(11):1917–1921.

Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, and 1 others. 2018. Random access in large-scale dna data storage. *Nature biotechnology*, 36(3):242–248.

Keiron O'shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Igor Pavlov. 2013. Lzma official website. https://tukaani.org/lzma/. Accessed: 2025-07-31.

PBzip2. 2015. pbzip2 - parallel bzip2 file compressor. https://linux.die.net/man/1/pbzip2.

Armando J Pinho, António JR Neves, Daniel A Martins, Carlos AC Bastos, and PJSG Ferreira. 2010. Finite-context models for dna coding. *Signal Processing*, pages 117–130.

Diogo Pratas, Morteza Hosseini, and Armando J Pinho. 2020. Geco2: An optimized tool for lossless compression and analysis of dna sequences. In *Practical Applications of Computational Biology and Bioinformatics, 13th International Conference*, pages 137–145. Springer.

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.

Diogo Pratas and Armando J Pinho. 2023. Jarvis2: a data compressor for large genome sequences. In *2023 Data Compression Conference (DCC)*, pages 288–297. IEEE.

Julian Seward. 2024. The official website of the xz compressor.

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

KS Sheena and Madhu S Nair. 2024. Gencoder: A novel convolutional neural network based autoencoder for genomic sequence data compression. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, (01):1–12.

Milton Silva, Diogo Pratas, and Armando J Pinho. 2020. Efficient dna sequence compression with neural networks. *GigaScience*, 9(11):giaa119.

Maria JP Sousa, Armando J Pinho, and Diogo Pratas. 2024. Jarvis3: an efficient encoder for genomic data. *Bioinformatics*, 40(12):btae725.

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, Yanfeng Ding, Liping Yi, Huidong Ma, Haonan Xie, Gang Wang, and Xiaoguang Liu. 2025b. Adaptive lossless compression for genomics data by multiple (s, k)-mer encoding and xlstm. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Hui Sun, Huidong Ma, Feng Ling, Haonan Xie, Yongxia Sun, Liping Yi, Meng Yan, Cheng Zhong, Xiaoguang Liu, and Gang Wang. 2025c. 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. 2025d. 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*, page btae323.

Zhenhao Sun, Meng Wang, Shiqi Wang, and Sam Kwong. 2024c. Lec-codec: Learning-based genome data compression. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*.

Kubo Takehiro. 2016. Snzip, a compression/decompression tool based on snappy. https://github.com/kubo/snzip. Accessed: 2025-07-31.

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, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Rongjie Wang, Yang Bai, Yan-Shuo Chu, Zhenxing Wang, Yongtian Wang, Mingrui Sun, Junyi Li, Tianyi Zang, and Yadong Wang. 2018. Deepdna: A hybrid convolutional and recurrent neural network for compressing human mitochondrial genomes. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 270–274. IEEE.

Philip B Weerakody, Kok Wai Wong, Guanjin Wang, and Wendell Ela. 2021. A review of irregular time series data handling with gated recurrent neural networks. *Neurocomputing*, 441:161–178.

Ming Xiao, Jiakun Li, Song Hong, Yongtao Yang, Junhua Li, Jianxin Wang, Jian Yang, Wenbiao Ding, and Le Zhang. 2018. K-mer counting: memory-efficient strategy, parallel computing and field of application for bioinformatics. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2561–2567. IEEE.

XZ. 2024. Xz official website. https://github.com/tukaani-project/xz. Accessed: 2025-07-31.

Han Yang, Fei Gu, and Jieping Ye. 2023a. Rethinking learning-based method for lossless genome compression. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Zhenyu Yang, Xiaoxi Zeng, Yi Zhao, and Runsheng Chen. 2023b. Alphafold2 and its applications in the fields of biology and medicine. *Signal Transduction and Targeted Therapy*, 8(1):115.

Zhihan Zhou, Robert Riley, Satria Kautsar, Weimin Wu, Rob Egan, Steven Hofmeyr, Shira Goldhaber-Gordon, Mutian Yu, Harrison Ho, Fengchen Liu, and 1 others. 2025. Genomeocean: An efficient genome foundation model trained on large-scale metagenomic assemblies. *bioRxiv*, pages 2025–01.

Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343.

Zstd. 2024. Zstd official website. https://facebook.github.io/zstd/. Accessed: 2025-07-31.

A Details of AgentGC Prompts

GPU Analysis Agent

Description: You are "AnalyzeGpuMemory", an agent that analyzes the GPU memory and returns the GPU number and memory size using "analyze_gpu_memory" tool.

Instruction:
1) *When to call*: When the "UserAndCongitiveAndCompress" call "ParallelAnalysis", you should run "AnalyzeGpuMemory" and "AnalyzeFileCharacteristics" parallelly.
2) *Task*: You will analyze the GPU memory and return the GPU number and memory size.
3) *Output*: You should only return a dictionary with the following keys: ① "gpu_number": List of available GPU IDs. ② "memory_size": Free memory of each available GPU, sorted in ascending order by ID. "Your output must be a JSON string containing the keys: gpu_number, memory_size". The section state with key "analyze_gpu_memory_agent_output" was set to the JSON string you output". No extra output is needed.
4) *Example Output*: json { "gpu_number": [0, 1, 3], "memory_size": ["16GB", "12GB", "8GB"] }

Table 5: Details of GPU Analysis Agent Prompts

File Feature Analysis Agent

Description: You are "AnalyzeFileCharacteristics", an agent that analyzes the file characteristics and returns "atgc_proportion" and "data_size" using "analyze_file_characteristics" tool.

Instruction:
1) *When to call*: When the "UserAndCongitiveAndCompress" call the "ParallelAnalysis", you should run the "AnalyzeGpuMemory" and "AnalyzeFileCharacteristics" parallelly.
2) *Task*: You will analyze the file characteristics and return "atgc_proportion" and "data_size" using analyze_file_characteristics tool.
3) *Output*: You should only return a dictionary with the following keys: ① "atgc_proportion": The proportion of ATGC in the genomic data. ② "data_size": The size of the genomic data file in bytes. "Your output must be a JSON string containing the keys: atgc_proportion, data_size". The section state with key "analyze_file_characteristics_agent_output" was set to the JSON string you output. No extra output is needed.
4) *Example Output*: json { "atgc_proportion": [0.2, 0.1, 0.4, 0.3], "data_size": 123456789 }

Table 6: Details of File Feature Analysis Agent Prompts

Compress Agent

Description: You are "CompressAgent", an agent that compresses files based on result from "ParamsChoose".

Instruction: You will compress files based on result from "ParamsChoose". No extra output is needed, just return the result of compress_genomic_data.

Table 7: Details of Compress Agent Prompts

Params Choose Agent

Description: You are "ParamsChoose", an agent that chooses the best compression parameters based on GPU memory and file characteristics.

Instruction:
1) *When to call*: Call "ParamsChoose" after calling "ParallelAnalysis" Agent to obtain GPU memory and input genome file features in parallel.
2) *Task*: You need to: ① Calculate the GPU memory used for the current task based on the currently available GPU-Memory and the user's intent, ② Then use this GPU memory and the genome file feature results to call get_q_similar_vector to calculate the q most similar parameter records, ③ Infer the parameter combination that meets the user's needs based on the q most similar parameter records
3) *Examples of information used*: Node that this is only an example, you should use the actual information from the "ParallelAnalysis" Agent.

① gpu_info dict from "AnalyzeGpuMemory": "gpu_number": [0, 1, 3], "memory_size": ["16GB", "12GB", "8GB"]

② file_characteristics dict from "AnalyzeFileCharacteristics": "atgc_proportion": [0.2, 0.1, 0.4, 0.3], "data_size": 123456789

③ User's intent: "I want to use 70 percent of GPU Memory. Then mem_factor is 0.7."

④ User's intent: "I want to back it up for a long time, Please make it occupy as little disk space as possible. Then the mode is set to 0, which minimizes the compression rate."

User's intent: "I want to use the fastest compression algorithm. Then the mode is set to 1, which maximizes the throughput."

User's intent: "I want to balance the compression rate and throughput. Then the mode is set to 2, which is the default mode and balances mode 0 and mode 1."

⑤ The parameters in params_results.csv, which include GPU-Mem(KB) and Proportion.

3) *Output*: You should only return a dictionary which you think is the best compression parameters based on the given information, include the GPU memory need. You should also infer the most optimal parameter combination based on the get_q_similar_vector result and mode. It does not have to be the parameters in the get_q_similar_vector result, as long as you think it is reasonable and within the range of all parameters found by get_q_similar_vector (the final value is between the maximum and minimum values of the search result) Note that the smaller CR, the better, and the larger Throughput, the better. No extra output is needed.
4) *Example Output*: json { "context-length": "32", "GPU-Mem(KB)": "12288", "AMKLCF_Parameters": "5", "BatchSize": "320", "mode": "0" }

Table 8: Details of Params Choose Agent Prompts

Decompress Agent

Description: You are "DecompressAgent", an agent that decompresses files based on the file end with .pmklc.params.

Instruction: You will decompress files based on the the file end with .pmklc.params.

Table 9: Details of Decompress Agent Prompts

Leader Agent

Description: You are "LeaderAgent", a top-level agent that interacts with the user and decides whether to perform compression or decompression based on user input.

Instruction: Based on the session state with key user_input, The section state with key file_path in parsed_user_request was set to the file path of the input file, If user want to compress a file, then the session state with key task_type in parsed_user_request was set to compress, if user want to compress with 70 percent of GPU memory, then the session state with key memory_percent in parsed_user_request was set to 0.7, if user want to compress and minimizing the compression ratio, then the session state with key mode in parsed_user_request was set to 0, if user want to compress and maximizing the throughput, then the session state with key mode was set to 1. if user want to balence mode 1 and mode 0, then the session state with key in parsed_user_request mode was set to 2. If compress task does not set memory_percent and mode, then the session state with key in parsed_user_request memory_percent and mode was set to 0.7 and 2. If user want to decompress a file, then the session state with key in parsed_user_request task_type was set to decompress, memory_percent and mode was not needed(can set to None). Your output must be a JSON string containing the keys: task_type, file_path, memory_percent, and mode. The section state with key parsed_user_request was set to the JSON string you output.
2) *Example for compression*: 'task_type': 'compress', 'file_path': '/path/to/file.txt', 'memory_percent': 0.7, 'mode': 0.
3) *Example for decompression*: 'task_type': 'decompress', 'file_path': '/path/to/file.txt', 'memory_percent': None, 'mode': None.

Table 10: Details of Leader Agent Prompts

B Details of Datasets and Baselines

This section provides supplementary details for the experimental setup. It serves as a reference for the datasets and baselines involved in our study, with concise tabular summaries and additional explanations to support the results reported in the main paper.

B.1 Details of Datasets

We summarize the datasets used in our experiments in Table 11, which reports the storage size of each dataset and a brief description of its contents. All datasets are available in https://drive.google.com/file/d/1vUMHeSYQnbSMB571EgpviDzGv4QsKCOr/view?usp=sharing.

Table 11: The Details of used real-world datasets for AgentGC and baselines

Dataset Name	Size (Bytes)	Short Description
PlFa	8,986,712	The part of plasmodium falciparum dataset (Silva et al., 2020)
WaMe	9,144,432	The unknown genomics dataset (Sun et al., 2025d,a)
DrMe	32,181,429	The drosophila miranda dataset from the chromosome-3 (Pratas et al., 2020)
GaGa	148,532,294	The gallus gallus in chromosome-2 (Grohmann et al., 2022; Pratas and Pinho, 2023)
SnSt	6,254,100	Short Tandem Repeats and Single Nucleotide Polymorphisms of human (Cheng and Hui, 2023)
MoGu	171,080,468	The genomics data from the mouse gut (Guo et al., 2020; Sun et al., 2025a)
ArTh	19,695,740	One collection data from the arabidopsis thaliana (Chen et al., 2020; Sun et al., 2025a)
AcSc	714,975,658	The acanthopagrus schlegelii genomics dataset (Sun et al., 2025a)
TaGu	1,052,636,474	The taeniopygia guttata genomics dataset (Sun et al., 2025a; Geer et al., 2010)

B.2 Details of Baselines

This subsection presents the baselines considered in our evaluation. Table 12 provides a concise overview of each baseline, including its type, original publication source, implementation language, and major

technologies.

Table 12: The details of compared baselines for AgentGC

Baseline Name	Type	Source	Language	Major Technologies
PMKLC (Sun et al., 2025a)	Hybrid	SIGKDD	Python	BiGRU, Attention, GPU-Parallel
AGDLC (Sun et al., 2025b)	Dynamic	ICASSP	Python	XLSTM, Multi-(s,k)-mer Encoding
JARVIS3 (Sousa et al., 2024)	Static	Bioinformatics	C/C++	Finite-context and Repeat Models
XZ (XZ, 2024)	Traditional	Classic	C/C++	Dictionary-based Compression, LZ77
DNA-BiLSTM (Cui et al., 2020)	Static	ICANN	Python	Attention, BiLSTM, CNN
NAF (Kryukov et al., 2019)	Traditional	Bioinformatics	C/C++	ZSTD (Zstd, 2024) Compression Standard
DeepZip (Goyal et al., 2018)	Static	DCC	Python	LSTM, BiGRU
Spring (Chandak et al., 2019)	Traditional	Bioinformatics	C/C++	BSC (Grebnov, 2011) and LZMA2 Compression
LZMA2 (Pavlov, 2013)	Traditional	Classic	C/C++	Dictionary-based Compression, LZ77
SnZip (Takehiro, 2016)	Traditional	Classic	C/C++	Dictionary-based compression, LZ77
GZip (Gzip, 1996)	Traditional	Classic	C/C++	Dictionary-based Compression, LZ77
PBzip2 (PBzip2, 2015)	Traditional	Classic	C/C++	Dictionary-based Compression, LZ77
PPMD (Cleary and Teahan, 1997)	Traditional	Classic	C/C++	Prediction by Partial Match
Lstm-compress (Knoll, 2017)	Dynamic	Classic	C/C++	LSTM, MLP

Notes. "Source" indicates the origin of the journal or conference, where "Classic" denotes industrial recognition and applicability across multiple domains. "LZ77": Lempel-Ziv 1977 (Ziv and Lempel, 1977), "BSC": Block Sorting Compression, "CNN": Convolutional Neural Network, "LSTM": Long Short-Term Memory, "BiGRU" : Bidirectional Gated Recurrent Unit (Weerakody et al., 2021), "XLSTM": Extended LSTM (Beck et al., 2024).

We further describe how each baseline is instantiated and used in our experimental setting, focusing on implementation choices and usage details that are not covered in the main paper.

B.2.1 PMKLC

PMKLC (Sun et al., 2025a) is a parallel multi-knowledge learning–based lossless compressor designed for efficient genomics data compression. It employs a neural architecture built upon BiGRU (Weerakody et al., 2021) and attention (Vaswani et al., 2017) mechanisms to capture and exploit redundancy patterns inherent in genomic sequences. To improve computational efficiency and scalability, PMKLC supports multi-GPU parallelization, enabling accelerated compression and decompression while maintaining lossless compression performance.

```
# (1) compression
bash PMKLC_M_Compression.sh file 0 320 3 3 SPuM
# (2) decompression
bash PMKLC_M_Decompression.sh file_3_3.pmklc.combined 0 3 3 SPuM
```

B.2.2 AGDLC

AGDLC (Sun et al., 2025b) is an experimental learning-based lossless genomics compressor that employs xLSTM-based context modelling (Beck et al., 2024) together with multiple (s,k)-mer encoding (Xiao et al., 2018). It is the first genomic data compressor that supports dynamic modelling during compression, allowing the context representation to adapt to genomic sequence characteristics while ensuring lossless recovery.

```
# (1) compression
bash compress.sh file 2.3+3.3 xLSTM 1 32
# (2) decompression
bash decompress.sh file 2.3+3.3 xLSTM 1
```

B.2.3 JARVIS3

JARVIS3 (Sousa et al., 2024) is a reference-free (Kryukov et al., 2020) lossless genomics compressor based on finite-context (Pinho et al., 2010) and repeat models. It incorporates enhanced table memory

mechanisms and probabilistic lookup tables within repeat models to improve computational efficiency and compression effectiveness.

```
# (1) compression
./JARVIS3.sh --threads 1 --fasta --block 10MB --input file
# (2) decompression
./JARVIS3.sh --decompress --fasta --threads 1 --input file.tar
```

B.2.4 XZ

XZ Utils (XZ, 2024) is a free general-purpose lossless data compression software based on dictionary-based compression (Mitra and Roy, 2018) and LZ77 techniques (Ziv and Lempel, 1977). It is the successor to LZMA Utils and is primarily designed for POSIX-like systems (Atlidakis et al., 2016), while also supporting several non-POSIX platforms.

```
# (1) compression
./xz -z9ke file -T 16
# (2) decompression
./xz -dk file.xz -T 16
```

B.2.5 DNA-BiLSTM

DNA-BiLSTM (Cui et al., 2020) is a lossless genomic sequence compression algorithm based on a deep learning model and an arithmetic encoder. The model combines convolutional layers(CNN) (O'shea and Nash, 2015) with an attention-based bi-directional LSTM to predict the probabilities of the next base in a sequence, which are then used by the arithmetic encoder for compression.

```
# (1) compression
bash ./compress.sh file file.compressed file_model 320 0
# (2) decompression
bash ./decompress.sh file.compressed file.compressed.recover file_model 0
```

B.2.6 NAF

NAF (Kryukov et al., 2019)(Nucleotide Archival Format) is a lossless reference-free compression format for nucleotide sequences in FASTA and FASTQ formats (De Francesco et al., 2009). It is based on the ZSTD compression standard (Zstd, 2024), providing good compression ratios while offering fast decompression performance.

```
# (1) compression
./ennaf file --temp-dir temp_dir/ -o file.naf
# (2) decompression
./unnaf file.naf -o file-cp
```

B.2.7 DeepZip

DeepZip (Goyal et al., 2018) is a general-purpose lossless compression algorithm based on recurrent neural networks(RNN) (Medsker et al., 2001). It follows a static pre-training paradigm and primarily employs LSTM and BiGRU architectures (Weerakody et al., 2021) for sequence modelling. In our experiments, we use DeepZip according to the commands specified below.

```
# (1) compression
bash ./compress file file.deepzip bs model
# (2) decompression
bash ./decompress file.deepzip file.deepzip.out bs model
```

B.2.8 Spring

Spring (Chandak et al., 2019) is a reference-free compressor designed to exploit the structured redundancy present in high-throughput sequencing data. It is based on a combination of BSC (Grebnov, 2011) and LZMA2 compression techniques (Pavlov, 2013), and supports multiple compression modes, including

lossless compression, pairing-preserving compression, and optional lossy compression of quality values (Liu et al., 2025), as well as support for long reads (Jain et al., 2018) and random access (Organick et al., 2018).

```
# (1) compression
./spring --fasta-input --long -c -i file -o file.spring
# (2) decompression
./spring -d -i file.spring -o file.fasta
```

B.2.9 LZMA2

LZMA2 (Pavlov, 2013) is an improved variant of the LZMA compression algorithm that enhances multi-threading capability and overall performance, while better handling incompressible data. In our experiments, we use the built-in LZMA2 implementation provided by the 7-Zip (Ipavlov, 2024) application.

```
# (1) compression
./7zz a -m0=lzma2 -mx9 -mmt16 file.7z file
# (2) decompression
./7zz x -y -mx9 -mmt6 file.7z
```

B.2.10 SnZip

SnZip (Takehiro, 2016) is a general-purpose lossless compressor based on dictionary-based compression and LZ77 techniques. It supports multiple file formats, including framing-format and legacy framing-format, with framing-format as the default. In our experiments, SnZip is executed using the commands specified below.

```
# (1) compression
./snzip -k -t snzip file
# (2) decompression
./snzip -kd -t snzip file.snz
```

B.2.11 GZip

Gzip (Gzip, 1996) is a general-purpose lossless compression program originally developed by Jean-loup Gailly for the GNU project (Gailly and Adler, 1992). In our experiments, Gzip is used according to the commands specified below.

```
# (1) compression
gzip -c file > file.gz -9
# (2) decompression
gzip -d file.gz -9
```

B.2.12 PBzip2

PBzip2 (PBzip2, 2015) is a parallel implementation of the Bzip2 block-sorting compression algorithm (Seward, 2024) that utilizes pthreads to achieve near-linear speedup on SMP systems. It combines the Burrows-Wheeler block-sorting algorithm (Fenwick, 1996) with Huffman coding (Huffman, 1952) for efficient text compression.

```
# (1) compression
./pbzip2 -9 -m2000 -p16 -c file > file.bz2
# (2) decompression
./pbzip2 -dc -9 -p16 -m2000 file.bz2 > file.bz2.reads
```

B.2.13 PPMD

PPMD (Cleary and Teahan, 1997) is a context-based lossless compressor that implements the Partial Matching Prediction (PPM) algorithm proposed by Cleary and Witten (Cleary and Witten, 1984). PPM models use previous symbols in the input to predict the next symbol and reduce the entropy of the output

data, differing from dictionary-based methods which search for matching sequences. Based on this compressor, Moffat proposed a new method that can compress faster (Moffat, 1990) at the cost of slightly reducing the compression ratio. PPMD further exploits contexts of unbounded length to improve the compression rate. In our experiments, PPMD is used via the 7-Zip implementation for data compression.

```
# (1) compression
./7zz a -m0=ppmd -mx9 -mmt16 file.7z file
# (2) decompression
./7zz x -y -mx9 -mmt6 file.7z
```

B.2.14 Lstm-compress

Lstm-compress (Knoll, 2017) is a lossless compression algorithm based on LSTM networks, utilizing the same LSTM module and preprocessing pipeline as Cmix (Knoll, 2016). It is executed using the commands specified below.

```
# (1) compression
./lstm-compress -c file file.lstm
# (2) decompression
./lstm-compress -d file.lstm file.lstm.out
```

C User Guideline

```
# (1) Run Google Adk. If the following message appears, it indicates successful execution:
#   Log setup complete: /tmp/agents_log/agent.20250713_175748.log
#   To access latest log: tail -F /tmp/agents_log/agent.latest.log
#   Running agent PMKLC_Agent, type exit to exit.
#   [user]:
# Then you can communicate with AgentGC and do compress or decompress.
# Ensure that the necessary environment is installed before running AgentGC.
 adk run pmklc_agent
# (2.1) Choose CP Mode to Compress. If you want to compress with CP mode, you can say:
 [user]: i want to compress [FILE TO COMPRESS] use [NUM] percent of gpu memory use least disk space
# (2.2) Choose TP Mode to Compress. If you want to compress with TP mode, you can say:
 [user]: i want to compress [FILE TO COMPRESS] use [NUM] percent of gpu memory as soon as possible
# (2.3) Choose BM Mode to Compress. If you want to compress with TP mode, you can say:
 [user]: i want to compress [FILE TO COMPRESS] use [NUM] percent of gpu memory (and balanced
     compression ratio and throughput)
# Compress NOTE: [FILE TO COMPRESS] is the Absolute Path
#                [NUM] is a number between 1-100
#                content in "()" can be omitted
# (3) Decompress. If you want to decompress a file, you can say:
 [user]: i want to decompress [FILE TO DECOMPRESS]
# Decompress NOTE: [FILE TO DECOMPRESS] is the Absolute Path
#                  No [NUM] or [Mode], these message is in param file
```

D Data Flow Formulation

We formulate the genomic data compression problem as a collaborative optimization process driven by user intent and environmental constraints. Let $x = \{x_i\}_{i=0}^{m-1} \in \mathbb{R}^{1\times d_0}$ denote the input genomic sequence of length m, where d_0 represents the alphabet size. The objective is to generate a compact binary stream $y = \{y_j\}_{j=0}^{n-1} \in \mathbb{R}^{d_1}$ of length n ($d_1 = 2$), such that $n \ll m \times \lceil \log_2 d_0 \rceil$.

Unlike traditional static compressors, AgentGC introduces a dynamic data flow involving user interaction and hardware perception. We define the system input as a tuple $\mathcal{I} = (x, t, \Phi)$, where t is the user-provided prompt (containing intent such as compression ratio priority or throughput priority) and Φ represents the environmental state (e.g., available GPU memory and hardware constraints).

The system models the compression mapping $\mathcal{P} : \mathcal{I} \to y$ through a two-stage agent-based flow:

- **Cognitive Flow (Leader Agent):** The leader agent $\mathcal{A}_{lead}$ functions as a hyperparameter optimizer. It accepts the prompt t and state Φ, utilizing the LLM to infer the optimal hyperparameter vector $\hat{v}$

(including context length c, memory threshold α, etc.):

$$\hat{v} = \mathcal{A}_{lead}(t, \Phi; \Theta_{LLM}) \tag{9}$$

where Θ_{LLM} denotes the pre-trained weights of the Large Language Model.

- **Compression Flow (Worker Agent):** The worker agent $\mathcal{A}_{work}$ executes the lossless compression under the guidance of $\hat{v}$. It dynamically learns the probability distribution $\mathcal{F}_\theta$ parameterized by θ. The optimization objective is to minimize the difference $\mathcal{D}$ between the estimated distribution $\mathcal{F}_\theta(x, \hat{v})$ and the true distribution $\mathcal{F}(x)$:

$$\theta^* = \arg\min_\theta \sum_{i=c}^{m-1} \mathcal{L}_{CE}\left(x_i, \mathcal{F}_\theta(x_{<i}, \hat{v})\right) \tag{10}$$

Finally, the arithmetic encoder $\mathcal{E}$ transforms the input x into the binary stream y based on the optimized probability estimates:

$$y = \mathcal{E}\left(x, \mathcal{F}_{\theta^*}(x, \hat{v})\right) \tag{11}$$

Thus, the overall goal of AgentGC is to minimize the length $n = |y|$ subject to the constraints defined by Φ and the intent defined by t.