HORIZON-EIC-2022-PATHFINDERCHALLENGES-01

Project No. 101115115

Interoperable end-to-end **P**latform of scalabl**E** and sustain**A**ble high-th**R**oughput techno**L**ogies for **DNA**-based digital data storage

Deliverable D1.1
PEARL-DNA compression codec

WP1 – Data preparation system

Authors	Stephanie Kristin Schröder (LUH)
Lead participant	LUH
Delivery date	30 September 2024
Dissemination level	PU - Public
Type	DEM – Demonstrator, pilot, prototype

Version 1.0

European Innovation Council

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Innovation Council and SMEs Executive Agency (EISMEA). Neither the European Union nor the granting authority can be held responsible for them. Swiss participants in this project are supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract numbers 23.00332 and 23.00328.

Revision history

Author(s)	Description	Date
Stephanie Kristin Schröder (LUH)	Draft deliverable	16.09.2024
Tjaša Stare (BioSis)	Revision 1	23.09.2024
Stephanie Kristin Schröder (LUH)	Version 2	23.09.2024
Tjaša Stare (BioSis)	Revision 2	24.09.2024
Stephanie Kristin Schröder (LUH)	Version 3	25.09.2024
Anna Ziemele (accelCH)	Formatting and formal check	27.09.2024

Contents

Partner short names

No.	Organisation	Short name
1	**BioSistemika**	BioSis
2	**Gottfried Wilhelm Leibniz Universität Hannover**	LUH
3	**Imagene SA**	IMG
4	**Technische Hochschule Wildau**	THWi
5	**Haute École Specialisée de Suisse Occidentale**	HES-SO
6	**accelopment Schweiz AG**	accelCH

Abbreviations

Abbreviation	Term
AVC	Advanced Video Coding
CRA	Clean Random Access
EC	European Commission
EU	European Union
FFmpeg	Fast Forward Moving Picture Experts Group
GOP	Group of Pictures
HDR	High Dynamic Range
HEIF	High Efficiency Image Format
HEIF	High Efficiency Image File Format
HEU	Horizon Europe
HEVC	High Efficiency Video Coding
QP	Quantization Parameter
QPA	Perceptual QP Adaption
SDR	Standard Dynamic Range
VVC	Versatile Video Coding
VVenC	Versatile Video Encoder
WP	Work Package

Executive summary

Background

Deliverable D1.1 is part of work package 1 (WP1), which focuses on developing a data preparation system. D1.1 describes the development of a compression module that can reduce the amount of DNA that needs to be synthesized.

Objectives

One of the objectives of the PEARL-DNA project is to develop a compression module that can target specific data types (e.g., videos and images) and efficiently process generic data. This document will describe how the compression module was developed, including the integration of state-of-the-art compression solutions to form a lossless compression solution for generic data and a lossy compression solution targeting specific data types.

Methodology and implementation

The main methodology in developing the compression module integrates state-of-the-art compression solutions to form a lossless compression solution for generic data and a lossy compression solution targeting specific data types. The implementation proceeds in two phases: initially creating the lossless compression solution, followed by the development of the lossy solution tailored to specific data types. Finally, both compression solutions are merged into an adaptable PEARL-DNA compression codec.

Outcomes

The key findings of this deliverable are the development of a compression module that can target specific data types and efficiently process generic data and the integration of state-of-the-art compression solutions to form a lossless compression solution for generic data and a lossy compression solution targeting specific data types.

Impact

The expected impact of this deliverable is the reduction of the amount of DNA that needs to be synthesized by optimizing the process of DNA synthesis and making the process more cost efficient. Overall, this will contribute to the final goal of the PEARL-DNA project to develop a novel DNA-based digital data storage system.

Next steps

The next step is the development of the PEARL-DNA channel codec (D1.4), an error detection and correction (EDC) module that protects the stored data at multiple tiers.

1 Introduction

The PEARL-DNA Compression Codec is a critical component of the PEARL-DNA project. The codec is designed to reduce the amount of DNA that needs to be synthesized by compressing data using state-of-the-art compression solutions. The compression module is part of WP1, which focuses on developing a data preparation system.

The PEARL-DNA Compression Codec is designed to be adaptable, capable of targeting specific data types such as video and images, and efficiently processing generic data. It has been tested on various file types, including text, images, and videos. The compression and decompression statistics are provided in Table 2, which shows the compression time, decompression time, original size, compressed size, memory usage, and PSNR values for each file type.

2 Requirements

The compression codec is a critical component of the project, and its requirements are multifaceted. Firstly, it is expected to be adaptable, capable of targeting specific data types such as images and videos, and efficiently processing generic data. The codec should provide lossless compression, reducing and optimally eliminating all redundant information contained in the input data. This is crucial for maintaining data integrity and ensuring the compressed data can be accurately restored to its original form. Additionally, the codec should offer lossy compression on target file types, which includes reducing irrelevant data and allowing for a trade-off between data quality and compression ratio. The compression codec should be able to shift priority towards specific dimensions depending on the specific usage scenario, ensuring that it can be optimized for different applications and use cases. Furthermore, the codec should be designed to work seamlessly with other project components, such as the error-correction functionality and the data-to-DNA encoding module, to ensure a cohesive and efficient data storage and retrieval system.

Compared to de-facto standard compression methods such as ZIP, the PEARL-DNA Compression Codec offers several distinct advantages. While traditional formats apply a generic approach to compression, the PEARL-DNA codec is specialized and adaptable to the type of data being processed, such as video and image files. It utilizes advanced tools like VVenC for lossy video compression and Pillow for images while leveraging Zstandard (Zstd) for lossless compression of generic data. This allows it to eliminate redundant data with precision for lossless needs and strike a balance between compression ratio and data quality for lossy scenarios. Moreover, the codec is designed to integrate smoothly with critical project components like error correction, making it an efficient and versatile solution compared to more limited, general-purpose formats like RAR and ZIP.

3 Choice of Codecs

The PEARL-DNA Compression Codec uses VVenC [1] for lossy video compression, Pillow [2] for lossy image compression in the High-Efficiency Image File Format (HEIF), and Zstandard (Zstd) [3] for lossless compression of other data types (Table 1). Other video file types can be compressed by converting them into YUV format before compression.

Table 1: Codec and supported file types

File type	Supported file types	Codec
Video	YUV (.yuv)	VVenC
Image	PNG (.png), JPEG (.jpg, .jpeg)	HEIF
Other	all other file types	Zstandard

3.1 VVenC Video Compression

Selecting a suitable video compression codec is crucial for efficient video content storage and transmission. VVenC is an implementation of the Versatile Video Coding (VVC) [4] standard. In this context, VVenC video compression offers a compelling solution. Published in 2020, VVC is the successor to AVC (2004) and HEVC (2013) [5], making it a state-of-the-art video codec. With its high compression ratio of 150:1 to 1000:1, VVenC is well-suited for applications with limited storage space. The codec's ability to discard non-essential data during compression significantly reduces file size, making it an attractive option for video applications where visual quality is paramount. Furthermore, VVenC's enhanced motion vector prediction, greater partitioning flexibility, and improved intra-prediction enable a reduction of bit rate by approximately 50% over HEVC at the same quality level. [4]

3.2 HEIF Image Compression

For image compression, the High Efficiency Image Format (HEIF), which utilizes the High Efficiency Video Coding (HEVC) [5] standard for encoding, presents a viable alternative to traditional image formats. First published in 2015, HEIF offers a lossy compression with a high compression ratio of 20:1 to 40:1, and up to 100:1. This results in significantly smaller file sizes compared to JPEG, making HEIF an ideal choice for applications where image storage space is a concern, such as mobile devices or web applications. Furthermore, HEIF's versatility in supporting a wide range of image types, including photographs and graphics, adds to its appeal. With its ability to achieve approximately 50% better compression than JPEG, HEIF is an attractive option for applications where image storage space is limited. [5]

3.3 Zstandard Compression

Zstandard compression offers a robust solution for compressing generic files. Although not explicitly mentioned in the project context, Zstandard's lossless compression capabilities make it a suitable choice for applications where data integrity is crucial. Its fast compression and decompression speeds, with compression speeds of several hundred MB/s and decompression speeds of over 1 GB/s on modern hardware, make it an attractive option for applications where speed is critical, such as data centers or cloud storage applications. [6]

4 Installation Guide

4.1 System Requirements and Recommendations

- Linux, macOS, or Windows operating system
- FFmpeg installed (see installation guide below)
- CMake installed (for building VVenC and FFmpeg on Linux)
- Python installed
- If you are using Windows, it is recommended to install the Windows Linux Subsystem (WSL)
- It is recommended to use Anaconda for the installation.

4.2 Installation with Anaconda

If you are using Linux, run the following command before continuing to update all packages:

- sudo apt update

4.2.1 Clone the Repository

Open a terminal or command prompt and navigate to the directory where you want to install the module.

Run the following command to clone the repository and its submodules:

- git clone --recursive https://www.tnt.uni-hannover.de:3000/schroeder/PEARL-DNA_compression-module.git

Navigate to the cloned repository directory:

- cd PEARL-DNA_compression-module

4.2.2 Create and Activate the Conda Environment

- conda env create -f environment.yml
- conda activate comp_codec

4.2.3 Build VVenC

Use this command to install CMake:

- conda install gcc gxx make cmake

Build VVenC with the following commands:

  - cd external/vvenc/
  - mkdir build
  - cd build
  - cmake..
  - make

4.2.4 Install FFmpeg

Linux:

- cd external/ffmpeg
- ./configure --disable-x86asm
- make -j

macOS (using Homebrew):

- /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
- brew install ffmpeg

Windows:

- winget install ffmpeg

4.3 Installation without Anaconda

If you are using Linux, run the following command before continuing to update all packages

- sudo apt update

4.3.1 Clone the Repository

Open a terminal or command prompt and navigate to the directory where you want to install the module.

Run the following command to clone the repository and its submodules:

- git clone --recursive https://www.tnt.uni-hannover.de:3000/schroeder/PEARL-DNA_compression-module.git

Navigate to the cloned repository directory:

- cd PEARL-DNA_compression-module

4.3.2 Create and Activate the Virtual Environment

Run the following command to create and activate the virtual environment:

- Linux:
  - sudo apt install python3-venv
  - python3 -m venv .venv
  - source .venv/bin/activate
- macOs:
  - python3 -m venv .venv
  - source .venv/bin/activate
- Windows
  - python -m venv .venv
  - .venv\Scripts\activate

4.3.3 Install Required Packages

Run the following command to install the required packages:

- pip3 install -r requirements.txt

4.3.4 Build VVenC

Linux:

- sudo apt install cmake
- sudo apt install build-essential

macOS (using Homebrew):

- brew install cmake

Windows: Download the installer from the official CMake website

Build VVenC with the following commands:

  - cd external/vvenc/
  - mkdir build
  - cd build
  - cmake..
  - make

4.3.5 Install FFmpeg

Linux:

- cd external/ffmpeg
- ./configure –disable-x86asm
- make -j

macOS (using Homebrew):

- /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
- brew install ffmpeg

Windows:

- winget install ffmpeg

5 Usage Guide

The PEARL-DNA Compression Module can be used to compress and decompress files using the following commands:

- **Compression**: python3 main.py -i <input_file> -o <output_file> [-p <preset>]
  - <input_file>: input file to be compressed
  - <output_file>: output file after compression
  - <preset>: preset only for video compression (slow, medium, fast)
- **Decompression**: python3 main.py -d -i <input_file> -o <output_file>
  - <input_file>: input file to be decompressed
  - <output_file>: output file after decompression

5.1 Usage Examples

- Compress a video file using the "fast" preset:
  - python3 main.py -i test_files/video.yuv -o test_files/video.yuv.hevc -p fast
- Compress an image file:
  - python3 main.py -i test_files/image.png -o test_files/image.png.heif
- Compress a text file:
  - python3 main.py -i test_files/text.txt -o test_files/text.txt.zst
- Decompress a video file:
  - python3 main.py -d -i test_files/video.yuv.hevc -o test_files/video.yuv.hevc.yuv
- Decompress an image file:
  - python3 main.py -d -i test_files/image.png.heif -o test_files/image.png.heif.png
- Decompress a text file:
  - python3 main.py -d -i test_files/text.txt.zst -o test_files/text.txt.zst.txt

5.2 Verification

To verify the result of lossless compression (Zstandard), use the following command:

- diff <original_file> <decompressed_file>

The results of the lossy compression of videos (VVenC) can be evaluated in the following ways:

- PSNR: A PSNR value exceeding 38 dB is considered to have good quality with minor degradation that is typically imperceptible to the human eye. The PSNR is automatically displayed during the video compression.
- Visual inspection: A visual inspection is a good way to assess subjective quality. Look for common artifacts such as blocking, blurring, ringing, or banding.

The results of the lossy compression of images (HEIF) can be evaluated in the following ways:

- PSNR: A PSNR value exceeding 38 dB is considered to have good quality with minor degradation that is typically imperceptible to the human eye. The PSNR for images is not automatically calculated during the compression.
- Visual inspection: A visual inspection is a good way to assess subjective quality. Look for common artifacts such as blocking, blurring, ringing, or banding.

5.3 Supported File Types

- Uncompressed:
  - Video: YUV (.yuv)
  - Image: PNG (.png), JPEG (.jpg,.jpeg)
  - Other: all other file types
- Compressed:
  - Video: HEVC (.hevc)
  - Image: HEIF (.heif)
  - Other: Zstandard (.zstd)

6 Code Structure

6.1 Detailed Module Description

6.1.1 Video Codec

This module is designed to handle both video compression and decompression using VVenC for compression and FFmpeg [4] for decompression, incorporating customizable preset configurations from YAML files. The primary functionality revolves around two main operations: compressing a video file and decompressing it back to its original format.

The vvc_compress function facilitates the compression of a video file. It accepts three parameters: the input file path, the output file path, and a preset that defines the compression settings. The available presets are "slow," "medium," and "fast," each of which corresponds to a different YAML configuration file stored in the project's directory. The selected preset's settings are loaded dynamically from YAML files, which specify compression parameters like size, frame rate, bitrate, and more. Once loaded, the compression process is triggered using VVenC, a video encoder designed for high-efficiency video coding (HEVC).

The vvc_decompress function performs video decompression. It accepts an input video file and an output path, applies decompression settings from a predefined YAML configuration, and generates a decompressed output file. The decompression process is handled using FFmpeg.

Two helper functions support these core functions, _exec_vvenc_compression and _exec_ffmpeg_decompression. These handle the execution of the video compression and decompression commands using subprocesses, ensuring that the appropriate parameters are passed based on the preset configurations. Both functions raise an exception if the subprocess fails, allowing for error handling and debugging.

Key advantages of this module include its flexibility through YAML-based configuration, which allows users to easily modify or extend compression and decompression presets.

6.1.2 Image Codec

This Python module uses the Pillow library and the pillow_heif extension to compress and decompress image files in the High Efficiency Image File Format (HEIF). It comprises two primary functions:

heif_compress to compress images into HEIF format and heif_decompress to revert HEIF files back to standard image formats.

The heif_compress function is designed to take a standard image file, compress it into the HEIF format using the highest quality settings, and then save the compressed file. It accepts two parameters: input_path, the file path of the source image, and output_path, the destination file path for the compressed image. The function opens the source image, converts it to HEIF format, saves it to the specified output path, and then confirms the operation's success with a printed message detailing the input and output paths.

The heif_decompress function decompresses HEIF images back into their original format. It also requires two parameters: input_path, where the compressed HEIF image is located, and output_path, where the decompressed image will be saved. The function reads the HEIF file, converts it back to a standard image format, saves it, and outputs a success message indicating the completion of the process along with the file paths involved.

This module depends on Pillow for general image handling tasks and pillow_heif for specific HEIF file manipulation. HEIF is particularly useful in environments where storage efficiency is essential—such as mobile applications, image archives, or web platforms dealing with high-resolution images—due to its ability to maintain high-quality images at roughly half the file size of equivalent JPEG images.

6.1.3 Zstandard Codec

This module provides functionality for compressing and decompressing files using the Zstandard compression algorithm. Zstd is a modern, high-performance compression library known for its fast compression speeds and high compression ratios, making it suitable for a variety of use cases where both performance and space savings are essential.

The zstd_compress function compresses a file located at the specified input_path and saves the compressed version to the output_path. This process begins by reading the file in binary mode and ensuring all the contents are loaded into memory. The function then initializes a Zstandard compressor using the ZstdCompressor object from the Zstandard library. The input data is compressed into a more compact form and subsequently written to the output file, which is also opened in binary mode. A success message is printed upon completion, indicating the original and compressed file paths.

The zstd_decompress function handles the inverse operation, decompressing a previously compressed file. The function reads the input file in binary mode to load the compressed data into memory. It then initializes a Zstandard decompressor using the ZstdDecompressor object, which performs the decompression process. The decompressed data is written in its original form in the output file. Like the compression function, a success message is printed once the decompression is completed.

6.2 Configuration

6.2.1 Video Compression Presets

The video compression configuration presets outline the three settings for video compression using the VVenC encoder, specifically optimized for three different presets: faster, medium, and slower, named fast, medium, and slow.

The video compression is performed on a video file with a resolution of 1920x1080, which represents a Full HD format. The temporal rate of the input file is set at a high frame rate of 120 frames per second (fps), with a frame scale of 1. However, a fractional framerate is defined as 60/1 for the final encoding, resulting in a smooth 60 frames-per-second output.

The video format is YUV 4:2:0, which indicates that the color space used for the input is YUV with chroma subsampling at 4:2:0, reducing the color information to save bandwidth and compression

space. This configuration also supports two variations of YUV 4:2:0: 8-bit and 10-bit, though the config defaults to 8-bit unless specified otherwise.

The VVenC presets are critical to the configuration. They change across the three configurations (faster, medium, and slower). These presets control the trade-off between compression speed and efficiency. A faster preset prioritizes encoding speed over compression efficiency, while a slower preset spends more time compressing the video to achieve better efficiency and potentially better visual quality.

The quantization parameter (QP) is set to 32, which controls the level of compression. The higher the value, the more compression is applied at the expense of video quality. In this case, 32 is a moderately high compression level, balancing file size and quality. The bitrate and maxrate are set to 0, indicating that no specific bit rate control is used, allowing for an unconstrained variable bit rate (VBR) approach.

There is an option for Perceptual QP Adaptation (QPA), enabled with a value of 1, meaning the encoder adapts the QP values based on the perceptual importance of different parts of the frame, likely leading to a better visual quality by preserving more detail where the viewer is likely to notice.

The refreshsec parameter is set to 1, meaning the intra-frame (or refresh frame) period occurs every second. This is complemented by the refresh type set to CRA, indicating that open GOP (group of pictures) structures are being used, where CRA (Clean Random Access) frames are used for intra-refresh instead of IDR (Instantaneous Decoder Refresh) frames. This results in a balance between efficient compression and the ability to seek randomly within the video. HDR and SDR modes are disabled.

In summary, the main difference between the *fast*, *medium*, and *slow* configurations lies in the preset value, which adjusts the encoding speed and efficiency. The faster preset will produce a quicker encode with less computational load but may sacrifice compression quality. On the other hand, the slower preset will result in a more refined compression process, potentially yielding higher video quality and lower file size at the cost of encoding time.

6.2.2 Video Decompression Preset

The video decompression preset is tailored to FFmpeg and aims to preserve the highest quality during the process. It employs an experimental mode to allow more flexible features, setting the compliance level to "-2." This means it can handle less common formats or advanced options that might not be fully standardized, offering greater adaptability in video processing.

The video is decoded using the "rawvideo" codec, which produces uncompressed video frames. By opting for raw video, the configuration ensures that no additional compression is applied after decompression, maintaining the original quality of the video content. This results in an output as close as possible to the source material.

For the pixel format, "yuv420p" is chosen, a widely used format that balances quality and file size. In this format, the luminance (brightness) data is stored at full resolution, while the chrominance (color) data is subsampled, reducing the overall amount of color information. This efficient subsampling method preserves a high level of visual quality, making it a standard in many video processing workflows.

6.3 Documentation

The repository's documentation is focused on providing clear and concise instructions. It includes Python docstrings that describe the purpose and usage of functions, detailing the expected arguments and their types. The error handling in the code emphasizes robust and reliable usage patterns, ensuring that users are informed about potential issues and how to handle them. Overall, the repository's documentation aims for easy usability and maintainability by providing thorough explanations and

usage examples. Additionally, the README includes detailed installation and usage instructions, usage examples, and statistical tests.

7 Performance

The performance of the compression codec varies significantly across different file types and presets. For text files, the codec compresses a 479-byte file to 317 bytes in a brief 0.05 seconds and decompresses almost as quickly, using minimal memory. Image compression shows more resource intensity, compressing a 1 MB file to 375 kB in 0.47 seconds and decompressing in 0.22 seconds (Table 2).

Video compression performance is dependent on the preset used. The 'slow' preset achieves good compression (from 890 MB to 1.8 MB) but takes significantly longer (over 32 minutes) than the other presets. Therefore, it is more suitable for scenarios where compression time is less critical, but high compression and high quality (PSNR of 39.7340) are essential. The 'medium' and 'fast' presets offer quicker compression times. The 'medium' preset offers a slightly lower compression (from 890 MB to 1.9 MB) than the 'slow' preset but is significantly faster than the 'slow' preset (under 2 minutes). It also results in slightly lower quality (PSNR of 39.6224). The 'fast' preset is particularly notable for its rapid compression time of under 12 seconds and results in the highest compression (from 890 MB to 1.5 MB), though at the cost of a slight drop in quality (PSNR of 38.8153). These presets show a varying demand for memory resources, highlighting the codec's adaptability to different processing and storage capabilities (Table 2).

Table 2: Compression and decompression statistics

File Type	Preset	Compression Time	Decompression Time	Original Size	Compressed Size	Memory Compression	Memory Decompression	PSNR
Text	-	0.05s	0.04s	479 B	317 B	20 MB	19.7 MB	-
Excel	-	1.92s	0.06s	14 kB	3.3 kB	23 MB	21.5 MB	-
PDF	-	0.06s	0.04s	2.3 MB	2.2 MB	27.5 MB	26.3 MB	-
Image	-	0.47s	0.22s	1 MB	375 kB	76.31 MB	29.53 MB	-
Video	slow	1924.86s	2.05s	890 MB	1.8 MB	2.26 GB	838.09 MB	39.7340
Video	medium	108.29s	1.94s	890 MB	1.9 MB	2.21 GB	788.91 MB	39.6224
Video	fast	11.72s	1.90s	890 MB	1.5 MB	1.85 GB	886.73 MB	38.8153

8 Future work

After the development of the compression codec, future work in WP1 will focus on developing the PEARL-DNA channel codec (D1.4), an error detection and correction (EDC) module, and a storage format module. The EDC module will protect the stored data at multiple tiers, including a dynamic global error correction code, a dynamic regional error correction code, and a static local error detection scheme.

9 References

[1] "VVenC," [Online]. Available: https://github.com/fraunhoferhhi/vvenc. [Accessed 13 09 2024].

[2] "Pillow," [Online]. Available: https://pillow.readthedocs.io/en/stable/. [Accessed 13 09 2024].

[3] "Zstandard," [Online]. Available: https://github.com/facebook/zstd. [Accessed 13 09 2024].

[4] "ITU-T H.266: Versatile video coding," 2020. [Online]. Available: https://www.itu.int/ITU-T/recommendations/rec.aspx?id=15648&lang=en.

[5] "ITU-T H.265: High efficiency video coding," 2013. [Online]. Available: https://www.itu.int/itu-t/recommendations/rec.aspx?rec=14107.

[6] "VVenC wiki," [Online]. Available: https://github.com/fraunhoferhhi/vvenc/wiki. [Accessed 14 09 2024].

[7] "High Efficiency Image File Format (HEIF)," [Online]. Available: https://nokiatech.github.io/heif/. [Accessed 13 09 2024].

[8] "Zstandard," [Online]. Available: http://facebook.github.io/zstd/. [Accessed 15 09 2024].

[9] "FFmpeg," [Online]. Available: https://www.ffmpeg.org/. [Accessed 13 09 2024].