GeoRedact: Towards automated redaction of privacy-sensitive geo-spatial information in full-motion videos

by

Samir Dharar

A thesis submitted to the School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of *Engineering* in *Computer Engineering*

Faculty Supervisor: Dr. Jonathan Anderson

Department of *Electrical and Computer Engineering*

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

Oct 2024

St. John's Newfoundland

Abstract

In recent years, the defense sector has seen significant advancements with the integration of drones that capture full-motion video (FMV) along with geospatial metadata. Byte-level analysis of these videos can disclose confidential mission information, highlighting the importance of protecting sensitive data from unauthorized access. While current redaction techniques often focus on visual elements, such as faces and license plates, the redaction of geospatial metadata has received less attention. This dissertation presents a systematic investigation into FMV redaction by introducing tools for metadata inspection and transformation, a novel approach for redacting geospatial metadata, and a new evaluation method, the Privacy-Utility Redaction Score (PURS), for assessing object detection models. The container parsers developed can efficiently extract and narrow down bytes of interest with minimal memory and CPU usage, and crucially, without memory leaks, making them well-suited for security-focused applications. The metadata redaction module enables selective redaction of user-specified metadata elements in Motion Imagery Standards Board (MISB)-compliant FMV. Additionally, the Privacy-Utility Redaction Score provides a metric for evaluating models based on their ability to redact private objects while ensuring public objects remain visible.

Acknowledgements

The past two years have been challenging yet rewarding, as I get to the point of writing the last few words of my thesis. First and foremost, I am indebted to my supervisor, Dr. Jonathan Anderson, for his invaluable support, guidance and encouragement throughout this journey. I am truly fortunate to have had the opportunity to work under his supervision.

I also want to extend my sincere appreciation to C-CORE and Mitacs for providing the resources and facilities which have contributed significantly to the development of this thesis.

Furthermore, I would like to offer my heartfelt gratitude to the university for providing the required intellectual atmosphere and academic opportunities that have been instrumental in my overall academic growth.

I want to thank my mother for her unconditional love, support and encouragement throughout my academic pursuits, even while being several miles away. Thanks to my father, who believed in my abilities and motivated me to continue my studies while he was in this world.

Thanks to my siblings for always asking for the progress of my thesis, mentoring me and teaching me how to manage my time well. I am truly blessed to have such a loving and understanding family.

Lastly, I want to acknowledge all the researchers whose invaluable work and publications helped improve my knowledge and lay the foundation for this work.

Contents

List of Figures

List of Tables

List of Abbreviations

AAC Advanced Audio Coding 58, 92

AP Average Precision 122

AVI Audio Video Interleave 91, 92

BER Basic Encoding Rules x, 49, 52–54, 95

BMP Bitmap Image file 92

CD Compact Disk 13

CLI Command Line Interface 57, 58

CNN Convolutional Neural Networks xi, 106, 113–116

CRT Cathode Ray Tube 9

CSV Comma Separated Value 42

CVE Common Vulnerabilities and Exposures 77

CWE Common Weakness Enumeration x, 77, 80, 81

DCT Discrete Cosine Transform 16–20

DETR Detection Transformer 132

DHL Dalsey, Hillblom and Lynn 3

DJI Da-Jiang Innovations 2, 4, 63, 94

DOCX Document Extended Format 89, 92

DoD Department of Defense 93

DVD Digital Video Disc 10

ECM Entitlement Control Message 46

EMM Entitlement Management Message 46

ES Elementary Stream 33, 47, 61

EXIF Exchangeable Image File Format 87

FAA Federal Aviation Administration 3

FHD Full High Definition 9, 10

FLAC Free Lossless Audio Codec 92

FMV Full-motion Video 4, 6, 31, 49, 85, 92, 94, 95, 97, 145, 147, 148

FN False Negative 119

FP False Positive 119

FPS frames per second 9, 10

GIS Geographic information system xi, 94, 97, 98, 100

GOP Group of Pictures 19–21, 27

GPAC GPAC Project on Advanced Content 57, 58, 67, 77, 80, 82, 83, 91

GPS Global Positioning System 5, 87, 148

GUI Graphical User Interface 57, 58

HD High Definition 9, 10

HOG Histogram of Oriented Gradients xi, 110–112

IDR Instantaneous Decoder Refresh 23, 27–30

IoU Intersection Over Union xi, 114, 117, 120–123, 132

ISO/IEC International Organization for Standardization/International Electrotechnical Commission xiii, 27, 28, 34, 36, 44, 46

ISR Intelligence, Surveillance and Reconnaissance 3, 4, 49, 85

ITU-R BT International Telecommunication Union - Radiocommunications Broadcasting Service (Television) 11

ITU-TT International Telecommunication Union - Telecommunication Standardization Sector xiii, 30, 46

JPEG Joint Photographic Experts Group ix, 15, 17, 18, 92

KLV Key Length Value x, xi, 45, 48–50, 52, 53, 61, 62, 71, 85, 94, 95, 97, 98, 145, 146

LL Left-to-right, leftmost derivation 33

LPDDR Low-Power Double Data Rate 132

LR Left-to-right, rightmost derivation in reverse 33

mAP Mean Average Precision xiv, 114, 122, 123, 132–134, 143, 144, 146

Microsoft COCO Common Objects in Context ix, xiv, 26, 27, 132–134, 137, 138

MISB Motion Imagery Standard Board x, 4, 5, 31, 49, 93, 94, 97, 99, 145

MISB ST 0601.17 Motion Imagery Standards Board ST 0601 UAS Datalink Local Set 49, 53, 93

MITRE Massachusetts Institute of Technology Research and Engineering 77

MP3 MPEG Audio Layer 3 15, 92

MP4 Moving Pictures Expert Group 4, Part 14 21–23, 31, 32, 34, 36, 38, 40, 55, 90, 92, 145

MPEG-TS Moving Picture Experts Group - Transport Stream x, xiii, 23, 31–33, 44–49, 52, 55, 56, 58, 59, 61, 65, 70–77, 92, 94, 95, 145

NAL Network Abstraction Layer xiii, 22, 27–30

NALU Network Abstraction Layer Unit ix, xiii, 27–29

NIST National Institute of Standards and Technology 77

NMS Non-maximum suppression 114

NSG National Geospatial-Intelligence Agency 93

OOP Object-oriented programming 38, 39

PDF Portable Document Format 88, 89, 92

PES Packetized Elementary Stream 33, 45–49, 52, 61, 145

PID Packet Identifier 45, 48, 52

PII personally identifiable information 121, 139

PIL Pillow 88

PMU Performance Monitoring Unit 65

PNG Portable Network Graphics 15, 92

PPS Picture Parameter Space 28, 29

PPTX PowerPoint Open XML Presentation 92

PURS Privacy Utility Tradeoff Score xi, 102, 123, 124, 142, 143, 146

QHD Quad High Definition 10

R-CNN Regions with Convolutional Neural Networks 103, 113–117, 134, 136, 137, 143

RBSP Raw Byte Sequence Payload 29

ResNet-50 Residual Network-50 132

RGB Red, Green, Blue ix, 8, 10–12, 18

RLE Run-Length Encoding 16

ROI Region of Interest 101, 102

RPN Region Proposal Networks 115

RTK Real Time Kinematics 4

RWTH Rheinisch-Westfälische Technische Hochschule 3

SD Standard Definition x, 10, 49

SPS Sequence Parameter Space ix, 28, 29

SSD Single Shot Multibox Detector 117, 132, 134, 136, 137, 143

STANAG 4906 NATO Standardization Agreement 4586 93

STCO Chunk offset Atom 38, 60, 61

STSS Sync sample Atom 38, 60, 61, 91

STSZ Sample Size Atom 38, 60, 61

SVM Support Vector Machine 114

TAR Tape ARchive 92

TLV Tag Length Value 50, 52, 53, 95

TP True Positive 119

TS Transport Stream 44, 45, 47, 48, 58, 59, 61, 72, 145

UAS Unmanned Aircraft Systems x, 3, 49, 52, 94

UHD Ultra High Definition 10

UML Unified Modeling Language x, 38, 39, 41, 47, 48

UTC Universal Time Coordinated 50

VCL Video Coding Layer 27–29

VGG Visual Geometry Group 117

VLC VideoLAN Client 80

WAV Waveform Audio File Format 15

WebP Web Picture Format 92

XLSX Excel Open XML Format Spreadsheet 92

XML Extensible Markup Language 58

YOLO You Only Look Once 103, 116, 132

Chapter 1

Introduction

According to a market report from Global Airborne - Intelligence, Surveillance and Reconnaissance (GA-ISR), the global consumer drones market is predicted to reach US $26.8 billion by the year 2026 [1]. The number of registered drones in the United States and Canada, as of the first quarter of 2023, is over 871,000 and 56,000, respectively [2]. Due to the increased safety concerns, both nations have created stringent regulations for using drones in their airspace. Unlike the US, where it is mandated to obtain a license to fly drones, in Canada, one may skip this requirement as long as the drone weighs less than 250 grams and the safety recommendations from Transport Canada are followed when flying the drone. As of 2023, there are approximately 307,000 registered drone pilots in the US and this count is a little over 90,000 in Canada [3].

A large proportion of drones today are primarily employed in industries such as agriculture, construction, logistics, and more. In collaboration with DJI, the *FarmBeats* project from Microsoft is an example of employing drone technology in agriculture [4]. The project aimed at flying drones over farmlands to survey and collect crucial data affecting crop production, such as the moisture in the soil or pest infestation and water scarcity. In logistics, companies such as Amazon and DHL have worked on integrating drone technology to transport goods by air between distribution centers and production sites. The *Parcelcopter* is a drone developed by DHL and researchers at RWTH Aachen to deliver pharmaceutical supplies to remote islands [5]. The *Prime Air* is a drone prototype developed by Amazon for delivering packages weighing up to 5 pounds, up to 15 miles, within 30 minutes or less [6]. Bechtel, the largest construction company in the United States, in collaboration with Skycatch, released a FAA-approved UAS project using drone technology to offer real-time surveying of construction sites while collecting environmental data such as air quality, temperature and more to enhance construction efficiency [7].

While drones have benefited humankind with their positive applications discussed above, we noticed instances where drones have been used to carry out unlawful activities. For instance, drones have been used as smuggling tools to transport illicit goods from Latin America to the United States across the border [8]. In some cases, drones were used to drop wire cutters in prison, which helped a prison break [9]. Attempts have been made to fly drones in restricted airspace to disrupt flights deliberately. In 2018, there was an instance in the United Kingdom where a dozen drones were flown in restricted airspace which caused delays in flights [10]. The late 1990s and early 2000s involved the allegedly attempted use of drones for terrorist activities by various militia groups [11]. More recently, we have noticed the use of consumer drones for air combat applications in warfare. The Russia-Ukrainian War has seen the most weaponized use of off-the-shelf drones for ISR roles. With a modest budget and a shortage in supply of military grade-drones, the Ukrainian forces turned towards using consumer-grade drones. The Ukrainian military forces have been allegedly found to use improvised versions of the DJI Mavic 3 and Matrice 300 RTK to form an army of ISR drones to conduct missions within Russian military space [12]. These drones were capable of surveilling the opposing troops and identifying their exact location to launch an attack. The drones have been modified to carry and drop air-delivery ammunition at enemy locations.

A collaborative study by researchers at the Ruhr University Bochum and CISPA (Center for IT-Security, Privacy and Accountability) in Germany revealed about 16 vulnerabilities in different versions of DJI drones [13]. The research proposed how it was feasible to alter the serial number of a DJI drone, a piece of critical information used by air authorities to track drone pilots. Similarly, the underlying tracking protocol, DroneID, used by DJI, was found to transmit the pilots' location in an unencrypted form when sending it to the authorities.

1.1 Motivation

Off-the-shelf consumer drones are not primarily designed for military ISR operations and thus lack the requisite security measures for such purposes. This deficiency renders them vulnerable to exploitation and the potential exposure of sensitive data. The widespread adoption of consumer drones, coupled with their inherent security vulnerabilities, underscores the need for in-depth research into the security aspects of these devices. This study aims to address a critical security concern by systematically examining the redaction of metadata in full-motion videos captured by drones.

Full-motion videos, which encompass audio, video, and metadata, are stored as interleaved data streams or packets. The tools proposed in this work intend to aid the process of producing full-motion video (FMV) and enabling access to the sensitive MISB metadata items for redaction using consumer-grade drones by distinguishing the bytes that constitute geospatial metadata and other such sensitive metadata items. However, these tools can also be integrated with systems targeting visual redaction as they offer access to raw frame data, in addition to underlying metadata items. While it is theoretically feasible to manually inspect these bytes using a Hex Editor, this approach demands a high level of expertise from analysts familiar with the intricate structure of full-motion videos [14]. Moreover, such manual inspection is both time-consuming and resource-intensive, relying heavily on human intervention.

There are several ways to identify the location of an individual by inspecting full-motion videos. In addition to examining the MISB metadata, it is possible to determine the GPS location by visually recognizing known structures, objects, or other elements shown in the video. To fully obfuscate GPS data from a given video, it is necessary to redact both types of information. This work primarily focuses on the redaction of geospatial metadata, while the redaction of image-based visual geo-location metadata is not explored. However, initial progress in visual redaction has been made by exploring various object detection approaches, which is a crucial task in protecting visually sensitive information.

Most consumer-grade drones capture video footage and geospatial metadata separately into different files, necessitating human post-processing that entails submitting the flight record logs to third-party websites for interpretation. This practice exposes confidential information to untrusted third parties. The necessary tools must thus be accessible to analyze and further transform the full-motion videos without having to manually inspect the lower-level bytes or upload sensitive metadata to third-party sites for inspection and further manipulation.

1.2 Thesis Outline

Our research investigates three key areas. First, we have proposed tools for the analysis and modification of full-motion video. Second, we have introduced a redaction technique for metadata in full-motion video. Third, we have conducted a quantitative review of methods for object detection and devised a privacy-centric metric for evaluating these algorithms, ensuring that data privacy remains a central focus throughout our work.

The outline of this thesis is as follows:

We begin by providing some background information in Chapter 2 which is essential for understanding the fundamentals of videos and FMV metadata. In chapter 3, we discuss the design and implementation of tools for parsing full-motion videos. Chapter 4 discusses a unique approach of redaction by proposing a metadata redaction library. Chapter 5 outlines a taxonomy of object detection approaches and their evaluation. Finally, we conclude our thesis by describing future research work in Chapter 6.

Chapter 2

Background

To effectively redact sensitive information from video footage, it is essential to first grasp several foundational concepts. This chapter addresses key background topics necessary for this understanding, including the structure of video [Section 2.1], video properties [Section 2.2], video compression techniques [Section 2.3], frame types [Section 2.4], and video containers and codecs [Section 2.5].

2.1 Anatomy of a Video

A *video* is a sequence of frames displayed rapidly and progressively on a screen. At least 24 frames must be shown each second for the human eye to perceive a succession of frames as a video [15].

A *frame* is a 2D array of pixels. In the context of a video file, a frame is one of the several still pictures arranged in chronological sequence [16]. The rapid succession of these frames creates the illusion of continuous motion, which the human eye perceives as seamless video playback.

A *pixel*, abbreviated for *picture element*, is the smallest unit of information present in a video [17]. In a grayscale frame, each pixel is represented as an 8-bit integer (or more), whereas in a color frame, each pixel is typically represented as a tuple of integers (e.g., RGB values). Pixels are the fundamental building blocks that combine to create a frame, and they are organized in a 2D array of pixels.

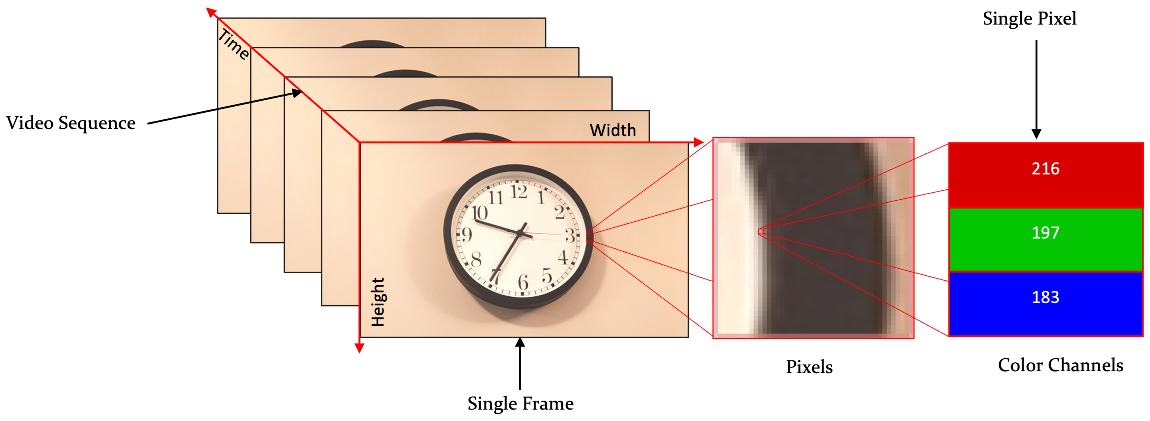


Figure 2.1: Relationship between video sequence, frame, and pixel

Figure 2.1 depicts authors visualization of the decomposition of a color video sequence into consecutive frames. Each frame is composed of pixels, which are represented as a tuple of RGB values. These frames are characterized by their height and width, collectively referred to as resolution, as defined in section 2.2.

2.2 Characteristics of a Video

A *frame size*, also known as *resolution*, is the number of pixels in each of the horizontal and vertical dimensions used to construct a video frame. One of the most widely utilized resolutions today is FHD, which stands for *full high-definition* and is denoted by 1920 x 1080 or simply 1080p, where *p* is abbreviated for *progressive scan*. There are two different approaches used for rendering the pixels visible in each new frame during video playback, namely *progressive scan* and *interlaced scan* [18].

In the progressive scanning approach, every pixel pertaining to a video frame is rendered sequentially from left-to-right, top-to-bottom, without omission of any pixel rows. The interlaced scanning method, on the contrary, renders pixels from left-to-right, top-to-bottom fashion, except it only renders alternating rows of pixels. In particular, it only renders odd rows of pixels in the first pass, followed by a swift rendering of even rows of pixels. This switch in rendering of odd and even row of pixels occurs at least 24 times per second, making us perceive it as if the entire image is rendered in a single pass.

This approach of transmitting half the pixels at a time reduced the required bandwidth by 50%, crucial for fitting within the limited broadcast bitrate. While this method was well-suited for CRT television displays, the primary reason for its use was to accommodate the constraints of broadcast bandwidth [19]. Although this approach significantly saved bandwidth, it suffered from flickering or blurring artifacts in scenes involving fast-moving objects. Conversely, progressive scan did not encounter such issues and was introduced for modern computer screens and HD screens. With CRTs becoming obsolete, progressive scan is anticipated to be employed in future video system standards[20].

An *aspect ratio* is the ratio of the width to the height of the given computer screen. Common aspect ratios include 16:9 and 4:3. Table 2.1 illustrates some common resolutions and their aspect ratios.

A *frame rate* is the number of frames captured through a sensor or displayed on a screen consecutively per second, also known as FPS. The framerate of a video highly affects the

Resolution	Standard	Aspect Ratio	Width	Height	Pixels
480p	SD	4:3	640	480	307,200
720p	HD	16:9	1,280	720	921,600
1080p	FHD	16:9	1,920	1,080	2,073,600
1440p	QHD	16:9	2,560	1,440	3,686,400
2160p	4K UHD	16:9	3,840	2,160	8,294,400
4320p	8K UHD	16:9	7,680	4,320	33,177,600

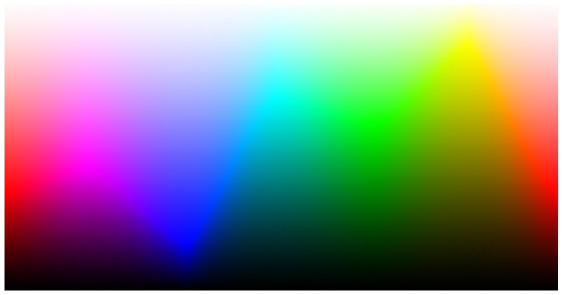
Table 2.1: Video Resolutions and Aspect Ratios

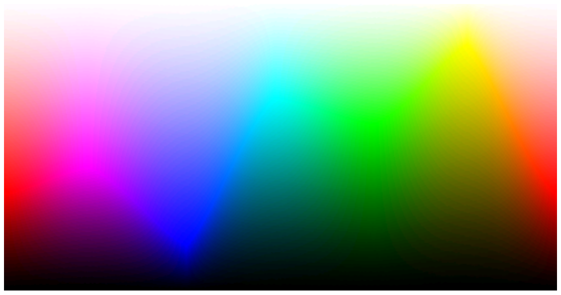
playback performance of a video. It is therefore common to use graphics processing units for intensive applications such as 3D rendering, HD video conferencing, and video games, where FPS plays a significant role.

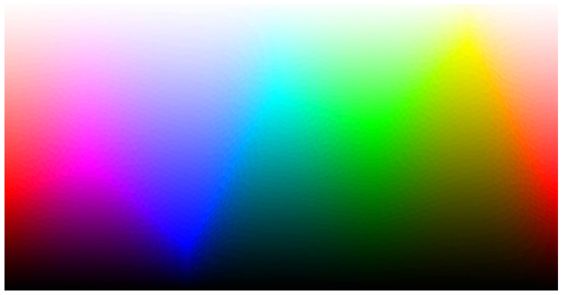
A *color space* offers a set of rules for mathematically expressing colors in digital form. Commonly-used color spaces are RGB and *YCbCr*. The RGB color space is divided into three color channels - red, green and blue. The values within each channel may vary based on the color depth of a given pixel. These channel values can be combined to illustrate the color of the pixel in question. For instance, a screen will display white color when the RGB value is 255, 255 and 255 and black when the value is 0, 0 and 0. An example color formation based on RGB color channels is depicted in Figure 2.1 and 2.2.

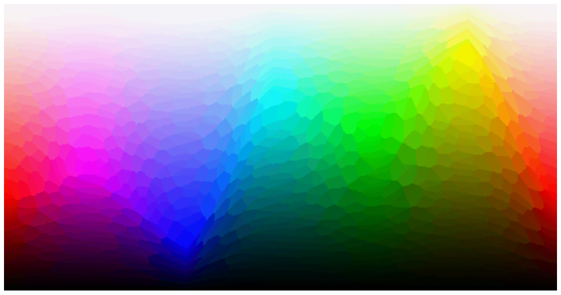
The YCbCr color space consists of one *Luminance (Y)* channel and two *Chrominance* or color channels, namely *Chroma Blue (Cb)* and *Chroma Red (Cr)*. The *Y* channel represents the brightness component of a pixel, which can range from 16 to 235. The chrominance channels, *Cb* and *Cr* represent blue and red color differences, respectively. The chrominance channels have values ranging from 16 to 240. For example, to render a pixel in white, a YCbCr value of 235, 128 and 128 is required, whereas black is 16, 128, 128. The YCbCr color space was originally designed for early SD Television Boxes, Blu-ray and DVD players with a dynamic color display range of 16 - 235, according to the ITU-R BT.601 specification [21]. Due to bandwidth and storage constraints, early displays considered every pixel with a color value less than 16 as black and any pixel with a color value more than 235 as white [22]. Figure 2.12 illustrates such a pixel color formation based on a combination of luminance and chrominance color channels.

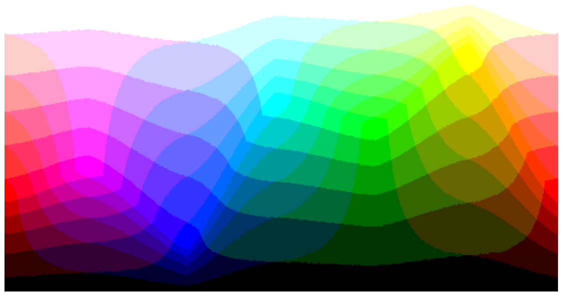
The *color depth* of a video, also known as *bits per pixel (bpp)*, is the number of bits utilized to encode a single pixel's color in a video frame. The higher the color depth values, the broader the range of color options available [23]. For instance, in the RGB color space, a frame with 24-bit color depth has three 8-bit integer values, each used for representing red, green and blue color channels [24]. Figure 2.2 illustrates an image with different color depth values. An image with a color depth of 1 will result in a monochromatic or black-and-white image. As we move higher up the color depth values, such as 32 bits, the image appears much more vibrant, exhibiting greater color details.

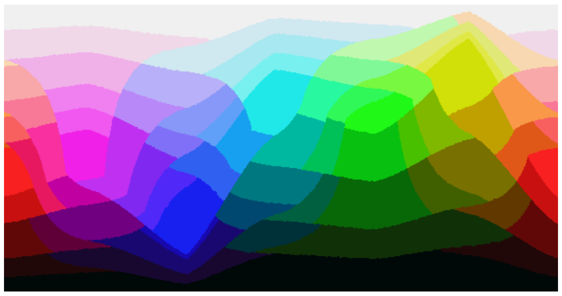

(a) 24 Bits

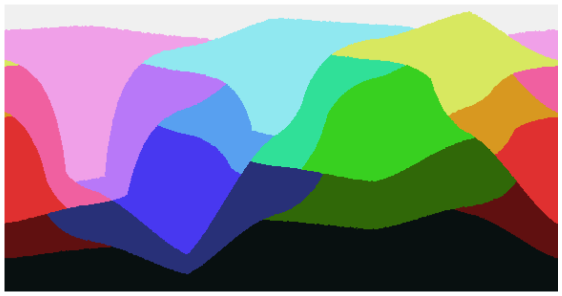

(b) 16 Bits


(c) 12 Bits


(d) 10 Bits


(e) 8 Bits


(f) 6 Bit


(g) 4 Bits

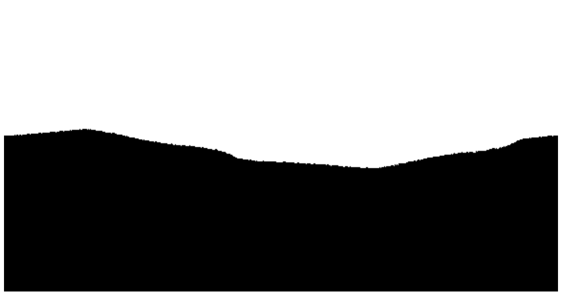

(h) 1 Bit

Figure 2.2: Image with varying RGB color depths (24b source image from Adobe Photoshop Color Palette)

The *bitrate* of a video is the amount of audiovisual information processed or transmitted per second. Video bitrate plays a crucial role in live video streaming applications. The video bitrate is expressed in terms of *bits/second* units and calculated according to equation 2.1.

$$\text{Bit Rate} = \left[\frac{\text{Width} \times \text{Height} \times \text{Color Depth} \times \text{Frames Per Second}}{\text{Compression Factor (described in section 2.3)}}\right] = bits/sec \tag{2.1}$$

The *audio sampling rate* is expressed in Hertz (1/s). It must be greater than or equal to the Nyquist rate to preserve the required audio information [25], which is twice the highest frequency of the originating signal [26]. While the lowest and highest frequency that humans can hear may vary from person to person, but a common approximation used between 20Hz - 20KHz [27]. A human ear may perceive audio at varying frequency differently depending on its ability to hear high or low frequency components. For example, the audio coming out of a telephone connection may sound very different from a CD-quality recording of a human voice. Table 2.2 illustrates common sampling frequencies in kHz applicable in different audio scenarios.

Sampling Rate	**Application**
8 kHz	Telephone Communications [28]
44.1 kHz	CD Audio [29]
48 kHz	Audio Tracks in Movies [30]
96 kHz	Studio Recordings [31]

Table 2.2: Sampling Rates for various applications

The properties of the video described in this section collectively influence the size of a video. Video frame height, width, color depth, and frame rate make up the visual component of the video, and sampling rate, bit rate and audio channels make up the audio component of the video. The relationship between key video properties may be summarized in equation 2.2, which calculates the absolute number of bytes that make up an uncompressed video frame.

$$F_S = F_H \times F_W \times C_D \times F_{PS} + S_R \times B_D \times A_{DC} \tag{2.2}$$

where:

F_S = frame size,

F_H = frame height,

F_W = frame width,

C_D = color depth,

F_{PS} = fps,

S_R = audio sampling rate,

B_D = audio bit rate,

A_{DC} = audio channels,

2.3 Compression in Video Frames

2.3.1 Compression Ratio

Compression techniques exploit the redundancies in video frames with an attempt to use as few bits as possible to render a frame. Video file size depends on the type of compression technique utilized. Video compression reduces the number of bits necessary to display a video to conserve bandwidth and storage space. It is measured in terms of the compression ratio shown in equation 2.3.

$$\text{Compression Ratio} = \left[\frac{\text{Original Stream Size}}{\text{Compressed Stream Size}}\right] \tag{2.3}$$

2.3.2 Compression Types

Lossless compression seeks to minimize the image size by re-encoding and storing the same image efficiently with no loss of information. As a result, this approach generates a compressed file that is reversible to its original form, maintaining image quality identical to the original image. This compression type is suitable where recovering the original data after decompression is desired. For example, lossless compression is commonly employed in image, file, and audio storage formats such as PNG, ZIP, and WAV.

The objective of *lossy* compression is to achieve the highest *compression ratio* feasible while keeping picture quality close to the original image. The resultant image is significantly smaller, making it suitable for applications where a loss of information is acceptable. JPEG and MP3 are two standard lossy compression algorithms used to store images and audio files. Figure 2.3 demonstrates an example for lossy and lossless compression on an input image taken from the *Cats and Dogs Dataset* by Microsoft [32].

Another important categorization of compression is Inter-frame and Intra-frame compression. *Intra-frame compression* identifies and exploits spatial redundancy in a video frame. Spacial redundancy refers to the repetitious nature of pixels within a single frame. It minimizes the use of repeating pixel by making pixels with the identical frequencies point to one location. Figure 2.3, for example, comprises repeated black and white pixel color values that make up the dog's face, as well as repeating green pixel color values that make up the grass on the ground.

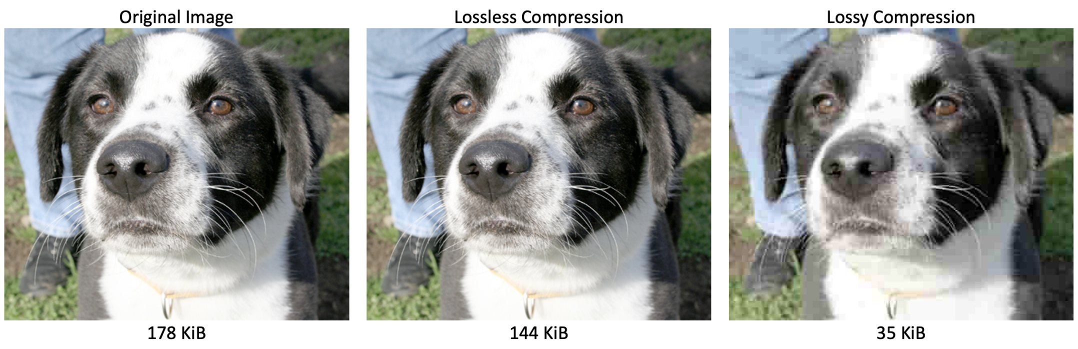


Figure 2.3: Lossy vs Lossless Compression

Inter-frame compression technique looks for temporal redundancies in a series of frames over time. Temporal redundancy in video frames refers to pixels that are repeated between two adjacent frames. The construction of the current frame depends on other reference frame(s).

2.3.3 Compression Algorithms

This section discusses two well-known intra-frame compression algorithms: Run-Length Encoding (RLE) and Discrete Cosine Transform (DCT). In *Run-Length Encoding*, pixel intensities are read from left to right while keeping track of pixels that have the same intensity [33]. Each time a new intensity pixel is observed, a new intensity count for the pixel is recorded. The RLE encoding is not ideal for compressing frames having pixels with frequent intensity value shifts, resulting in negative compression, where the compressed frame takes up more storage space than the uncompressed frame. Figure 2.4 gives an example of compression using Run-Length encoding, where the first pixel value of 0 is stored as 01, where 0 represents the pixel value and 1 indicates the count. Subsequently, a sequence of 11 is encoded as 12, a run of 000 is represented as 03, and a series of 1111 is stored as 14.

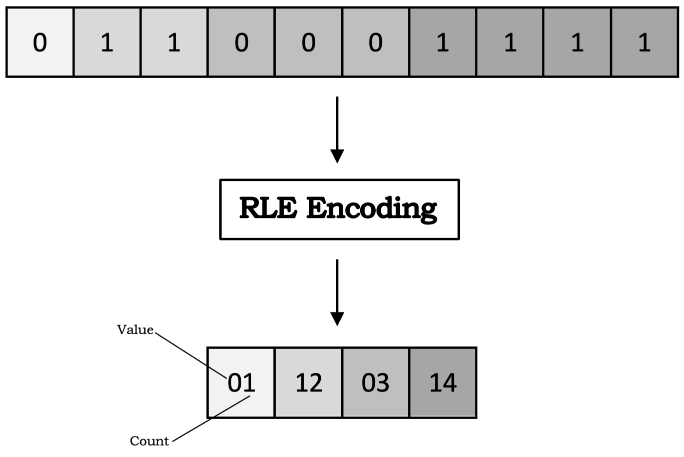


Figure 2.4: Run-Length-Encoding Compression

DCT transforms a spatial image into a frequency image representation. Pixel blocks represent a weighted sum of sinusoidal impulses, especially cosine waves with a range of frequencies. A larger weight will be given to the cosine wave representation of a macroblock that describes higher frequency variations and vice versa. Prior to calculating the weighted sum of all the signals, certain higher-frequency cosine waves are discarded to accomplish compression. Figure 2.5 shows various stages in JPEG compression including Discrete Cosine Transformation.

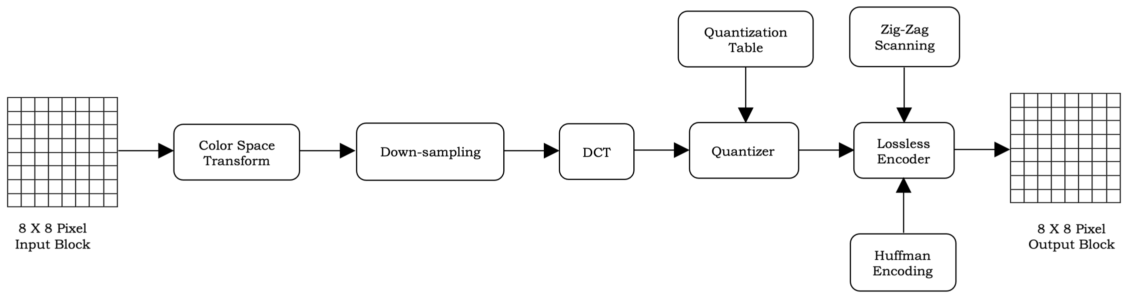


Figure 2.5: Discrete cosine transform in JPEG compression [34]

A picture in RGB color space is first transformed to the YCbCr color space. It is then partitioned into 8x8 pixel blocks, each separately encoded using DCT to produce exactly 64 cosine waves of varying frequencies. Each of the basis cosine waves have an associated numerical value called a DCT Coefficient that represents the measure of contribution for each cosine wave towards the 8x8 image block. The reconstruction of the 8x8 image block entails adding each cosine wave multiplied by the specified DCT constant. In JPEG compression, DCT coefficients are quantized i.e. trade-off between compression and quality of the image. The DCT transform matrix, which has a fixed set of elements responsible for generating DCT coefficients, is multiplied by the 8x8 input pixel matrix to obtain these coefficients.

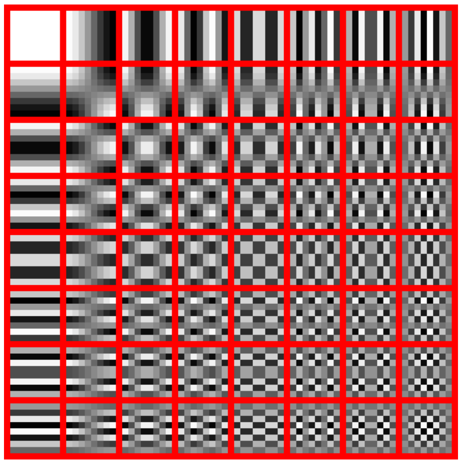


Figure 2.6: 64 Cosine waves represented using 64, 8x8 pixel bock patterns [35]

As shown in figure 2.6, the cosine wave's frequency rises from the top right corner and moves downward from left to right. The inputted pixel values are first centered or left shifted by a mid-point number depending on the sub-sampling ratio employed since a cosine wave oscillates between -1 and 1, centering around 0. The next step is quantization, which removes or reduces high-frequency DCT coefficients. Different applications may use custom quantization table values depending on the ratio of quality vs. image size we want to achieve. Each DCT coefficient is divided by the corresponding quantization table value and rounded to the nearest integer.

2.4 Keyframes and Delta Frames

Most modern video codecs (discussed in section 2.5) use a combination of three different kinds of frames: keyframe, predictive frame, and bidirectional frame. The order in which these frames appear in a video stream is indicated by a GOP, commonly referred to as a *Group of Pictures*, illustrated in figure 2.7.

A *keyframe* is an intra-coded image, often abbreviated *I-frame*. Keyframes are often placed at the beginning or end of a GOP sequence, as shown in figure 2.7. They are beneficial when transitioning between two different scenes in video footage, where there is minimal temporal redundancy. As a result, compression algorithms aim to utilize spatial redundancy in keyframes. For example, instead of storing individual pixels for the sky, only distinct pixels to render the sky can be reserved while disregarding repeating pixels. This compression type is mainly beneficial in keyframes as we cannot refer to another frame for pixel redundancies. In figure 2.7, for example, the B and P frames employ inter-frame compression, with the P-frame referring to the subsequent P and I-frames and the B-frames using both forward and backward referencing to obtain pixels from I and/or P frames. The macroblocks used in keyframes utilize the *intra-prediction* technique for their storage, in which a macroblock can only refer to another macroblock present in the same frame. A keyframe, in H.264 encoding, uses the Discrete Cosine Transform (DCT) (discussed in section 2.3.3) compression technique [36].

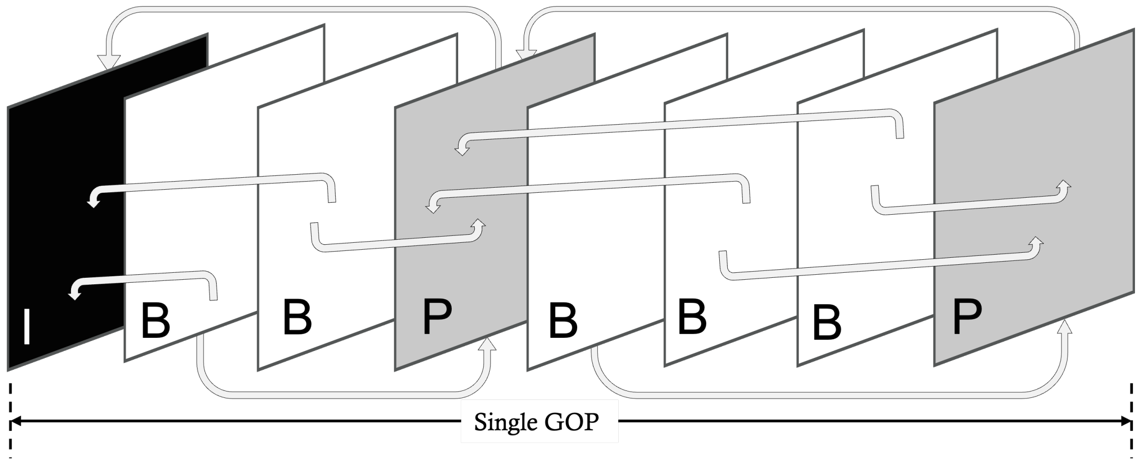


Figure 2.7: Type of Frames

The term *delta* refers to *difference* [37]. A *p-frame*, short for *predictive frame*, is a type of *delta frame* that only stores the pixel differences with reference to another frame. Thus, they are also referred to as *difference* frames. Since delta frames depend on other frames in the GOP sequence for rendering, they are not a standalone image. The macroblocks present in a predictive frame use the *inter-prediction* technique for their storage. In the inter-prediction approach, every macroblock within a predictive frame can refer to a macroblock present in another frame as long as it relates to an I-frame or a P-frame in the GOP sequence.

A *b-frame*, short for *bidirectional* frame, uses *macroblocks* that depend on inter-prediction approach: macroblocks can refer to macroblocks from an I-frame before it or a P-frame before or after it. A B-frame cannot rely on another bidirectional frame. For instance, an I-frame, which is depicted as a black square in Figure 2.7, does not access pixels from any neighboring frames; yet, it may have spatial redundancy in which a macroblock accesses another macroblock for its own rendering, but only inside the same I-frame.

2.5 Video Codecs and Containers

2.5.1 Video Containers

The term *container* refers to a wrapper format that encapsulates audio streams, video streams and additional metadata such as subtitles. Common container formats include MP4 (*MPEG-4, Part 14*), AVI (*Audio-Video Interleaved*), and TS (*Transport Stream*). In full-motion videos, apart from the audiovisual data streams, a third stream is used to carry geospatial metadata, such as the real-time longitude, latitude and altitude of an aerial system. Similarly, many other systems utilize the third and succeeding streams to hold application-specific metadata. For example, the video footage captured through Parrot drones stores drone-specific metadata into one of the streams in MP4 container format [38].

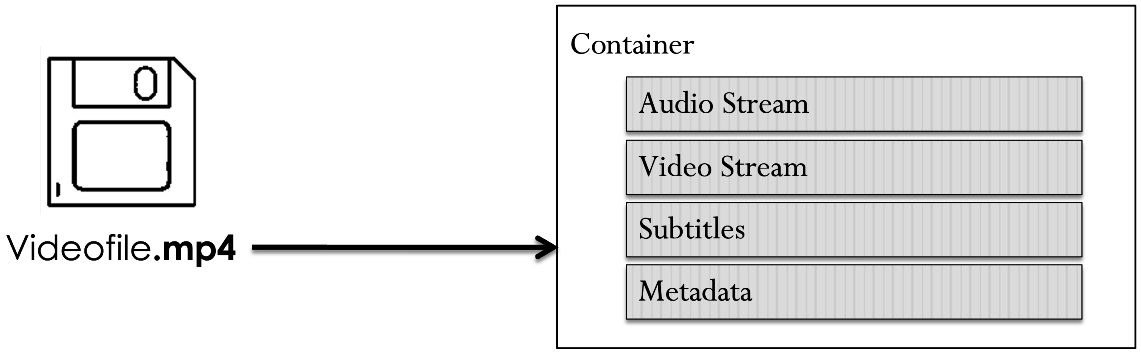


Figure 2.8: Video File Containers

2.5.2 Video Codecs

The term *codec* is a portmanteau of *coder, decoder*. It is a software component employed to encode and decode audio/video information. Figure 2.9 shows the video encoding process.

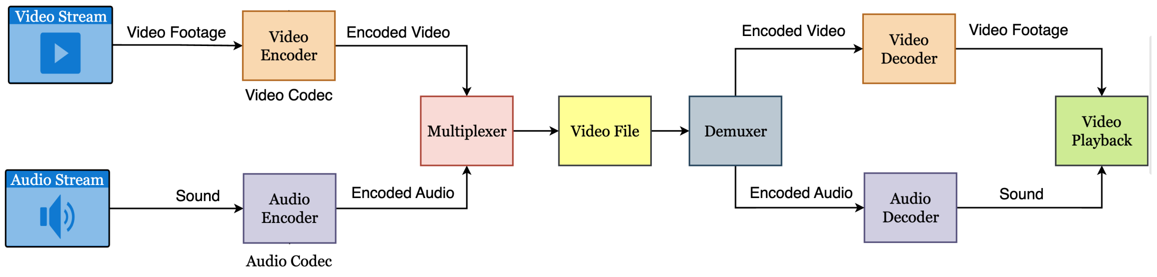


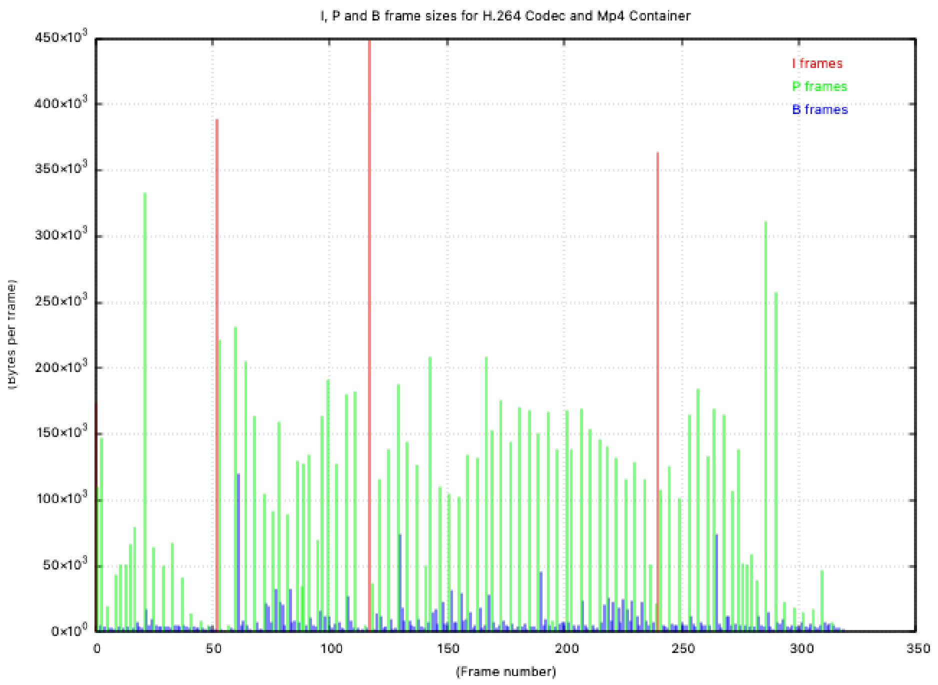
Figure 2.9: Video Encoding [39]

Codec-specific information typically begins at the payload level of video containers. For instance, the *mdat* atom in the MP4 container format stores codec-specific such as NAL units (discussed in section 2.6). Similarly, the elementary stream payload in MPEG-TS containers holds codec-specific information. For example, a slice of IDR coded picture (discussed in section 2.6).

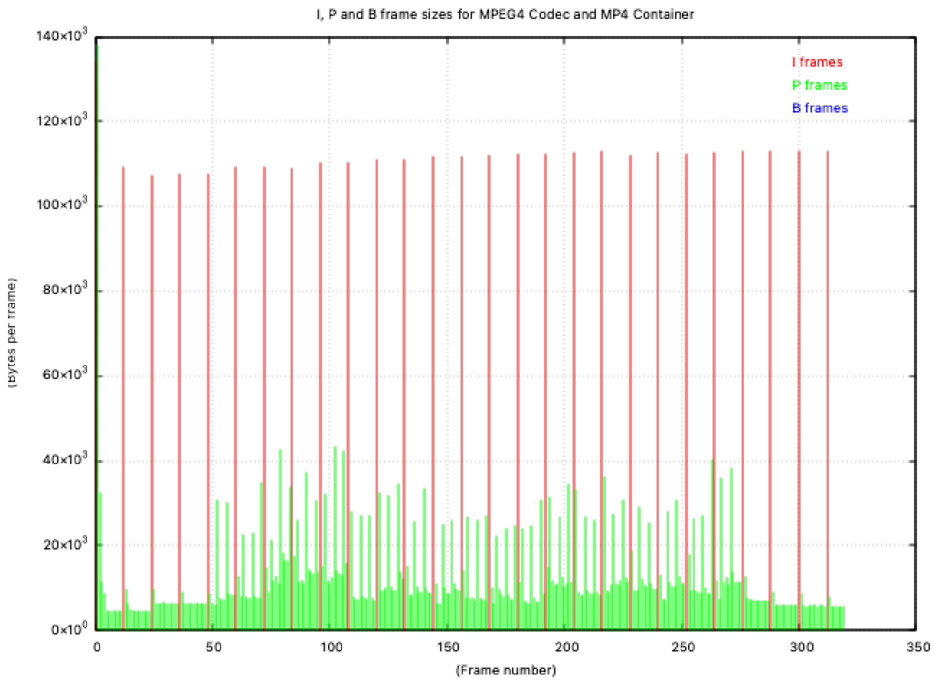
Container	MP4			
Codec	H.264	MPEG4	H.265	VP9
I-Frame Count	4	27	3	3
P-Frame Count	94	293	82	317
B-Frame Count	222	0	235	0

Table 2.3: I, P and B-frame count variation based on Codec Type changes

The number of frames in a video may vary depending on the kind of video codec employed for the storage of media streams. Table 2.3 depicts the change in frame count when a MP4 container is coupled with four different types of codecs. A shell script using libraries such as gnuplot, ffprobe, awk, and more was used for investigating the varying frame counts. For this analysis, four video files were utilized, each using the MP4 container combined with different codecs: H.264, H.265, MPEG4, and VP9. A bar graph of these observations are shown in Figure 2.10 and 2.11.

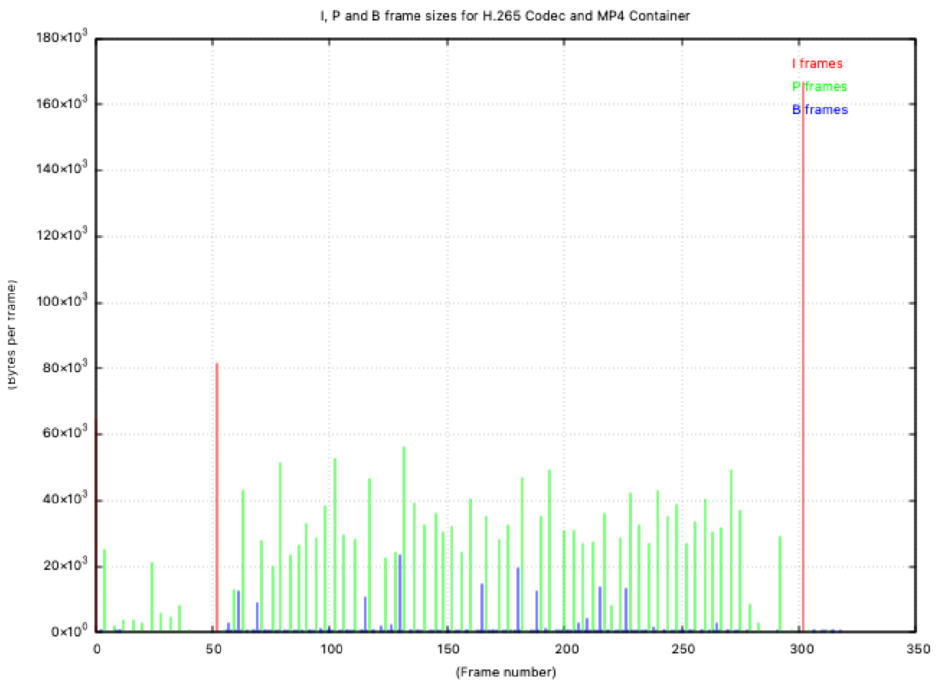


(a) H.264 Codec

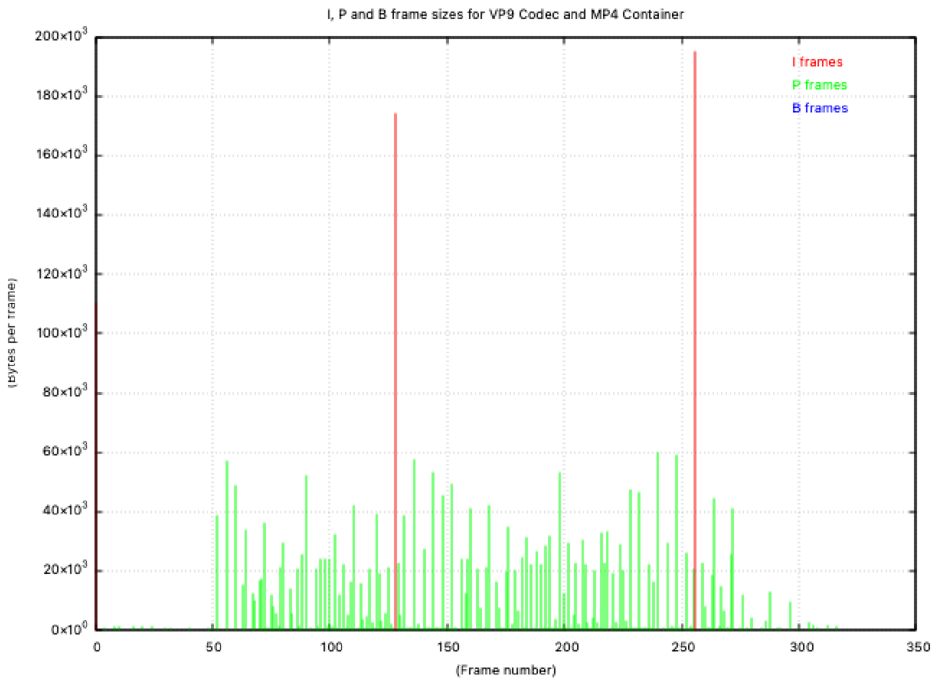


(b) MPEG4 Codec

Figure 2.10: Varying Codec Types: Frame number vs Bytes per frame



(a) H.265 Codec



(b) VP9 Codec

Figure 2.11: Varying Codec Types: Frame number vs Bytes per frame (Continued)

2.6 The H.264 Video Codec

2.6.1 Background

H.264 is the most widely-used video codec today, accounting for 82% of worldwide videos [40]. It utilizes the YCbCr color space and each frame is divided into several data units known as macroblocks of N$\times$N pixels. Each macroblock has 16$\times$16 pixel blocks for the luminance (Y) channel and two 8$\times$8 pixels blocks for chrominance, or the *Cb* and *Cr* channels. The brightness of the video is controlled by the luminance channel, while the chrominance channel determines each frame's color values.

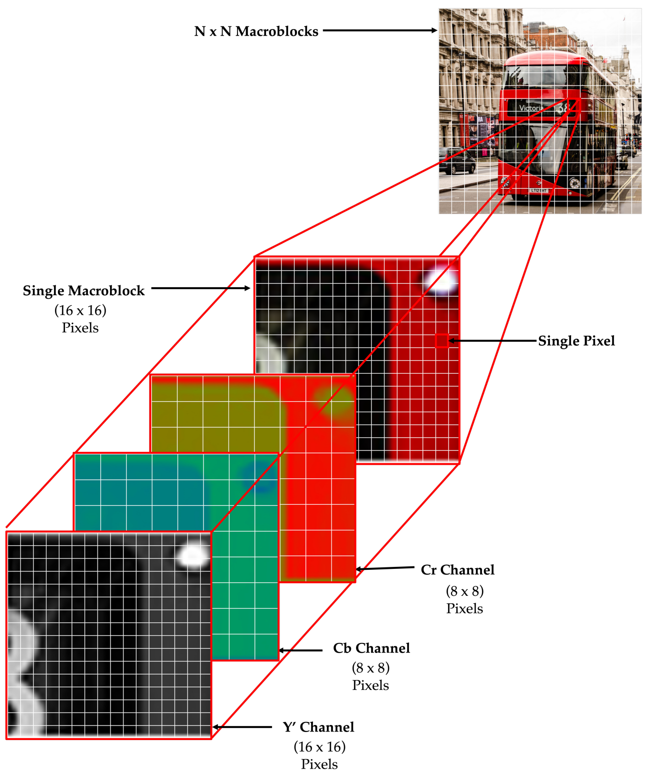


Figure 2.12: Relationship between frame, macroblock and pixels (reproduced from Microsoft COCO 2014 Dataset [41])

The author's visualization of macroblocks, frames and pixels based on a source image from Microsoft COCO 2014 Dataset [41] is shown in figure 2.12 illustrating subdivision of the video frame into macroblocks which are further represented in color channels.

2.6.2 ISO/IEC Specification

The ISO/IEC 14496-10:2020 standard specifies the requirements for storing H.264 codec-specific information [42]. Figure 2.13 illustrates the internal structure of an H.264 codec. The first level of abstraction is a series of network abstraction layer units, abbreviated for NALU. The information stored in a NAL unit may be categorized into two classes: Video Coding Layer (VCL) or non-Video Coding Layer (non-VCL). An example of NALU that belongs to VCL class is an *Instantaneous Decoder Refresh* (IDR) or a *Non-IDR picture*, which is a naming convention for a keyframe and delta frame in GOP sequence.

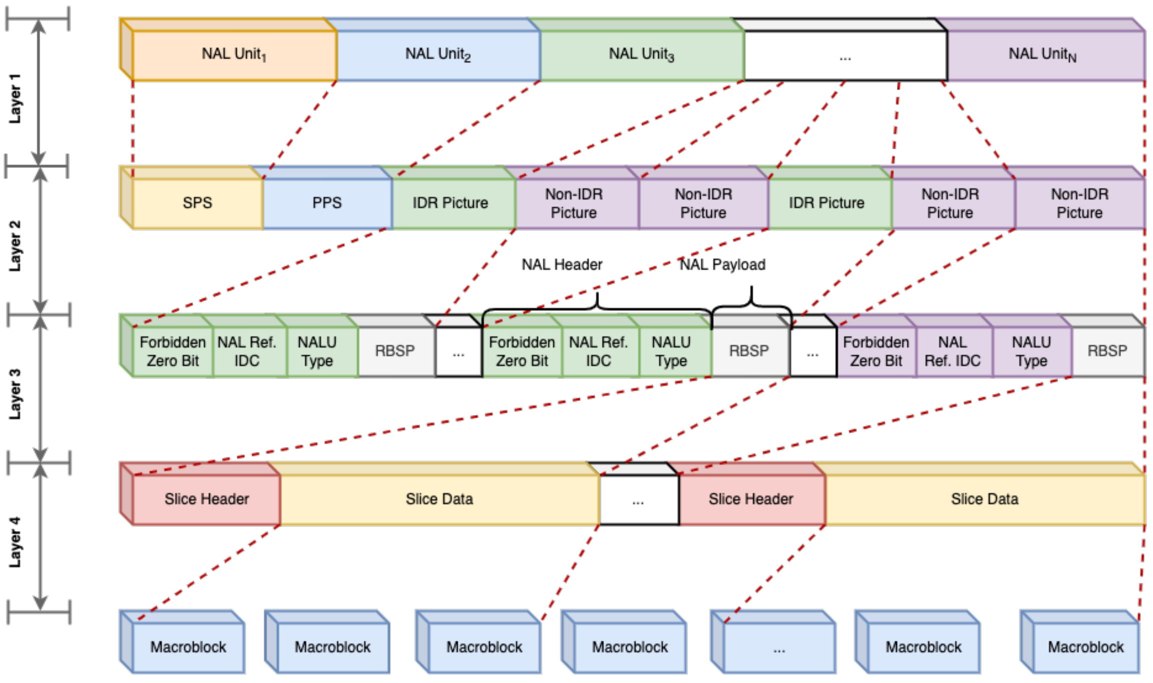


Figure 2.13: Data-structure of H.264-encoded video stream [42]

A *non-VCL class* typically includes decoding information. The *Sequence Parameter Set* (SPS) and *Picture Parameter Set* (PPS) are example NAL Units belonging to the non-VCL class storing decoding information. The SPS NAL unit holds information such as the type resolution or framerate, while PPS stores information such as of compression parameters used. The ISO/IEC specification for H.264 supports 31 different NAL unit types, as shown in table 2.5. In H.264, the start of a NALU can be identified by a fixed byte stream 0x00000001, followed by the NALU header. For example, the SPS NALU type will have a hex byte value of 0x67, indicating the NALU itself.

A NAL unit is made up of a fixed-size header followed by a variable-length payload. The one-byte NALU header holds three variables, as shown in the Table 2.4.

Variable	Size (in bits)
forbidden_zero_bit	1
nal_ref_idc	2
nal_unit_type	5

Table 2.4: NALU Header

The *forbidden_zero_bit* field is used to validate whether the given NALU has syntax violations or errors during transmission. A value of 0 indicates no syntax violations, while 1 suggests the presence of bit errors.

The *nal_ref_idc* field is used to identify whether the given NALU contains reference information necessary for other NALUs. A value of 00 indicates the absence of reference information for other non-IDR picture reference NALUs for inter-prediction, and discarding this NALU should not affect the rendering of referencing NALUs. A value greater than one indicates the presence of either a coded slice of IDR picture of encoding metadata such as

SPS or PPS, which are essential for reconstruction of reference NALUs such as a coded slice of non-IDR picture. For instance, a non-IDR frame (B) will have this field set to 00(0), an IDR frame (I) will have this value set to 01(1) and an encoding metadata NALU such as SPS will have this number set to 11 (3).

The *nal_unit_type* field identifies the type of NAL unit being transmitted. Figure 2.14 illustrates the NAL Header bytes for *Sequence Parameter Set* (SPS), a NAL unit type shown in Table 2.5.

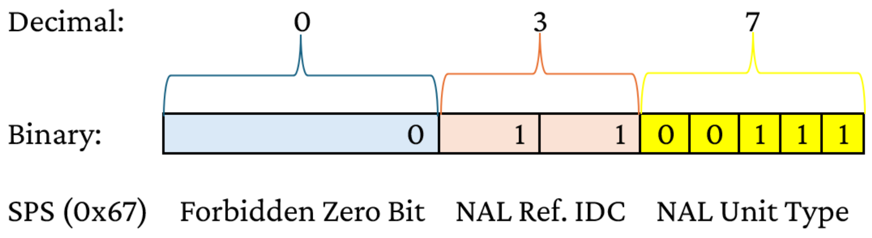


Figure 2.14: Example NALU (SPS) Header

The RBSP field, which stands for *Raw Byte Sequence Payload* maintains a sequence of slices that incorporate macroblocks which may contain pixel-specific (VCL) or decoding-specific (non-VCL) information.

This brings to a conclusion the background chapter, which introduced the fundamentals of images and videos while delving further into compression methods, container formats, and codecs. This chapter lays the groundwork for comprehending the ideas that are discussed in Chapter 3.

NAL Unit Type	Name of NAL Unit
0	Unspecified
1	Coded slice of a non-IDR picture
2	Coded slice data partition A
3	Coded slice data partition B
5	Coded slice of an IDR picture
6	Supplemental enhancement information (SEI)
7	Sequence parameter set
8	Picture parameter set
9	Access unit delimiter
10	End of sequence
11	End of stream
12	Filler data
13	Sequence parameter set extension
14	Prefix NAL unit
15	Subset sequence parameter set
16-18	Reserved
19	Coded slice of an auxiliary coded picture without partitioning
20	Coded slice extension
21	Coded slice extension for depth view components
22-23	Reserved
24-31	Unspecified

Table 2.5: NAL Unit Types (reproduced from ITU-TT Rec. H.264 [43])

Chapter 3

Tools for Full-Motion Video

This section details the tools developed to aid the objective of producing full-motion videos (FMV) from consumer-grade drones, focusing on the creation and application of MP4 and MPEG-TS parsers, including KLV (Key, Length, Value)/BER (Basic Encoding Rules) components. The motivation for developing MP4 parsers stems from the prevalent use of the MP4 container format among consumer-grade drone manufacturers. Given that MP4 is a commonly employed video format in these drones, it is necessary to develop a parser tailored to handle and process this format effectively. Conversely, the need to develop an MPEG-TS parser arises from the requirement to conform to the MISB standard, which mandates MPEG-TS as the container format for integrating various metadata. To ensure that consumer drone videos meet MISB compliance, incorporating the requisite metadata into the video footage is essential. Therefore, a MPEG-TS parser is crucial for achieving MISB-compliant full-motion videos by facilitating the integration of metadata with video content.

3.1 Parser Basics

Parsers are tools that help interpret raw bytes using specific rules and convert it into a more understandable form and vice versa. For example, in computers, parsers can take human language input and translate it into a machine-readable format. In video parsing, parsers analyze the structure of video files to extract useful information like metadata, frames, and streams. Parsers can be classified into two main categories: Top-Down Parsers and Bottom-Up Parsers [44].

3.1.1 Types of Parsers

3.1.1.1 Top-Down Parsers

Top-down parsers start from the general structure and work their way down to the details. They use a method called "left-most derivation," which begins with the highest-level structure and breaks it down into smaller parts. In video parsing, a top-down parser might start by looking at the overall format of a video file (like MP4) and then break it down into smaller parts like boxes (or atoms) in MP4, or packets in MPEG-TS. There are two main types of top-down parsers: Recursive Descent Parsers and Predictive Parsers.

Recursive Descent Parsers

These parsers try all possible ways to match the input with grammar rules, using a trial-and-error approach. If one way fails, they backtrack and try another. In an MP4 file, this parser would start with the "ftyp" box and then try to understand each nested box like moov and trak one by one.

Predictive Parsers

Also known as LL parsers, they use lookahead to decide which production to use, avoiding the need for backtracking. They start with the start symbol and expand the left-most non-terminal nodes until the entire parse tree is constructed. In an MP4 file, this parser uses lookahead to efficiently predict the next box type to parse.

3.1.1.2 Bottom-Up Parsers

Bottom-up parsers, also known as Shift-Reduce Parsers, start from the details and build up to the overall structure. They use a method called "reverse right-most derivation," meaning they start from the smallest parts and combine them to form the complete structure.

In video parsing, a bottom-up parser might start by reading individual bytes from a video stream and progressively build higher-level structures. For instance, in MPEG-TS parsing, the parser would start by reading individual ES Packets, then group them into PES packets, and eventually reconstruct the entire video frame. There are two main types of bottom-up parsers: LR Parsers and Operator Precedence Parsers.

LR Parsers

These parsers scan the input from left to right but construct the parse tree using the reverse of right-most derivation. They handle parsing by shifting bytes onto a stack and reducing them based on parsing rules. In MP4, this parser would shift bytes corresponding to box headers onto a stack and reduces them into complete boxes as their boundaries are identified.

Operator Precedence Parsers

These parsers are specialized for generating the parse tree using operator grammars, ensuring that no two consecutive non-terminals or epsilon appear on the right-hand side of any production. They are useful for parsing expressions with operators, managing precedence without needing complex backtracking.

The MP4 parser presented in this research primarily uses a Top-Down Recursive Predictive Parser approach without backtracking. However, when unknown items are encountered, a recursive descent parser with backtracking is used to identify and store these unknown atoms. Similarly, the proposed Transport Stream parser also employs a Top-Down Recursive and Predictive Parser approach.

3.2 MP4 Parser

3.2.1 Background

Our preliminary work concentrated on comprehending the underlying structure of major video container formats and codecs, as well as the video redaction techniques. The majority of object detection libraries are built with open-source libraries such as *OpenCV* and *FFmpeg*. According to the ISO/IEC 14496-14:2020 standard for MP4 containers, MP4 files have an atomic structure [45]. Figure 3.1 provides a graphical representation of the MP4 container format.

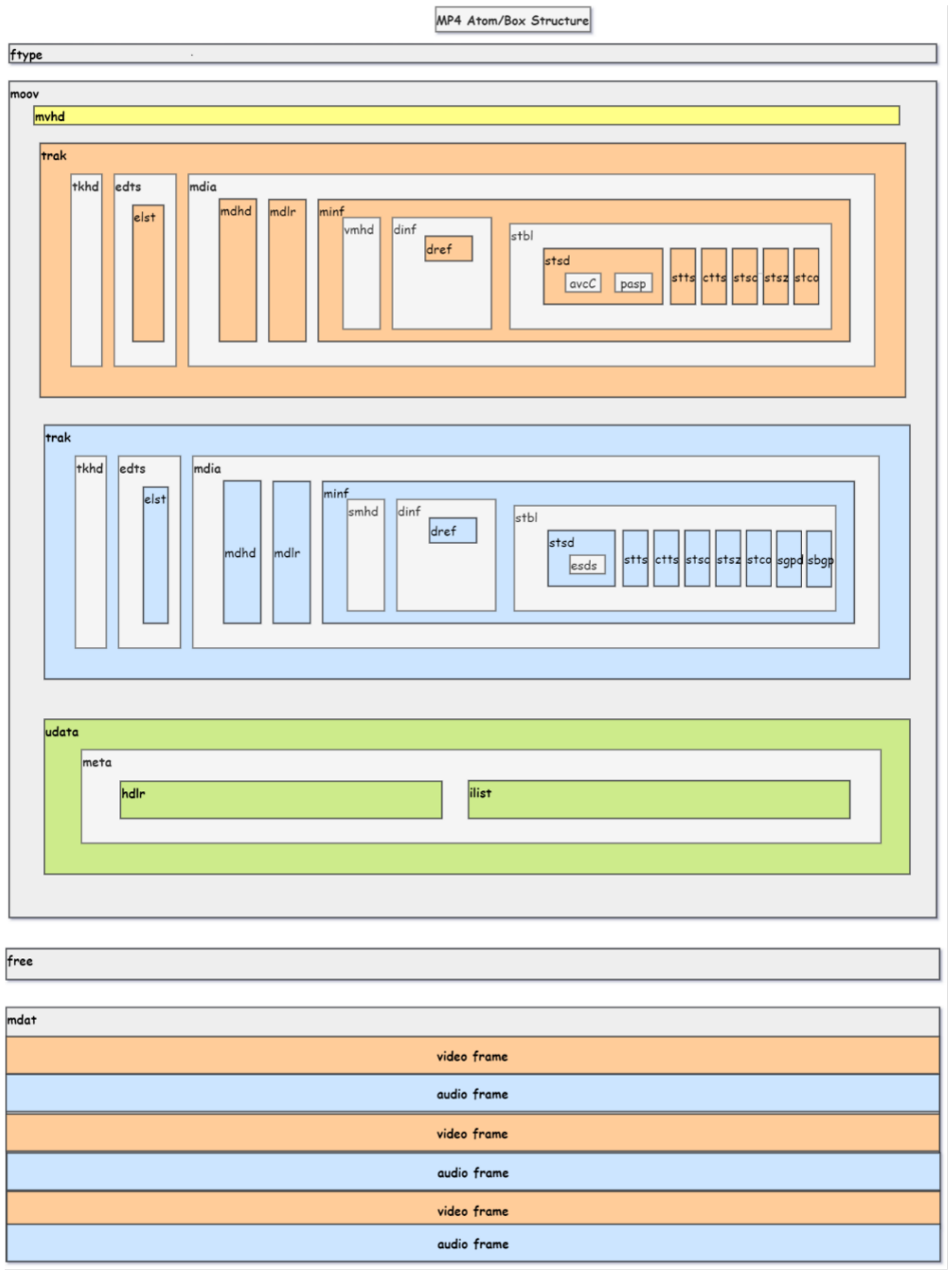


Figure 3.1: Structure of MP4 container

The multimedia data is stored in data units called *atoms*, which are also known as *Boxes*. These data units hold the underlying audio, video, and subtitle streams, as well as application-specific metadata. Each atom's first two fields are the *atom name* and the *atom size*, each holding 4 bytes of information. A pre-defined atom name denotes the beginning of a new atom, and the atom size determines the number of bytes needed to be read next to reach a sub-level atom. Atoms having varied sizes can be placed in any order except *ftyp*, which must always be at the root of the atom tree for media players to uniquely identify the file type. Inner-level atoms such as *moov*, *mdat*, *wide*, or any other custom atom can be encoded using any sequence.

A MP4 format is uniquely identified by the file type atom (ftyp). The *moov* atom is an abbreviation for movie header atom, and it contains metadata pertinent to various traks. For example, a MP4 video may have audio, video, and subtitle tracks. Further examination of lower-level atoms in trak shows a *sample table atom* entry containing the *chunk offset table*. This table contains direct offsets to individual audio and video frames stored in the *mdat* or *movie* data atom in an interleaved form.

3.2.2 Algorithm Design & Implementation

The parser library is implemented in the *Rust* with no dependency on external libraries or crates. Figure 3.2 presents the flowchart for the MP4 parsing algorithm, detailing the iterative process of atom parsing. The initial four bytes are read to identify the *atom name*, followed by parsing the subsequent four bytes to determine the *atoms size*. If the extracted atom name matches one of the predefined atom names as specified by ISO/IEC standards, the iterator is incremented by the size of the atom. During this step, the contents of the atom are stored, and the start and end offset locations are recorded for future reference.

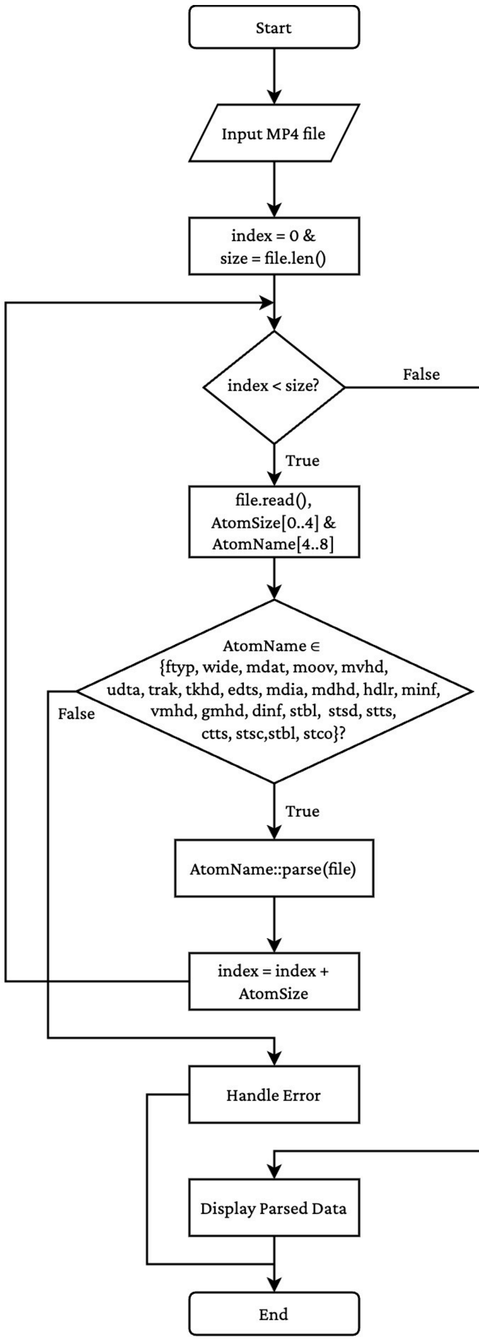

Start
Input MP4 file
index = 0 & size = file.len()
index < size?
False
True
file.read(), AtomSize[0..4] & AtomName[4..8]
AtomName ∈ {ftyp, wide, mdat, moov, mvhd, udta, trak, tkhd, edts, mdia, mdhd, hdlr, minf, vmhd, gmhd, dinf, stbl, stsd, stts, ctts, stsc,stbl, stco}?
False
True
AtomName::parse(file)
index = index + AtomSize
Handle Error
Display Parsed Data
End

Figure 3.2: Flowchart - MP4 Parsing Algorithm

The final output of this parsing process is a structured sequence of atoms, each represented by its corresponding start and end offset addresses. Figure 3.3 further demonstrates how essential metadata elements such as offset addresses (STCO), sync sample information (STSS), and frame sizes (STSZ) can be used to accurately pinpoint the location of an I-frame within the mdat atom.

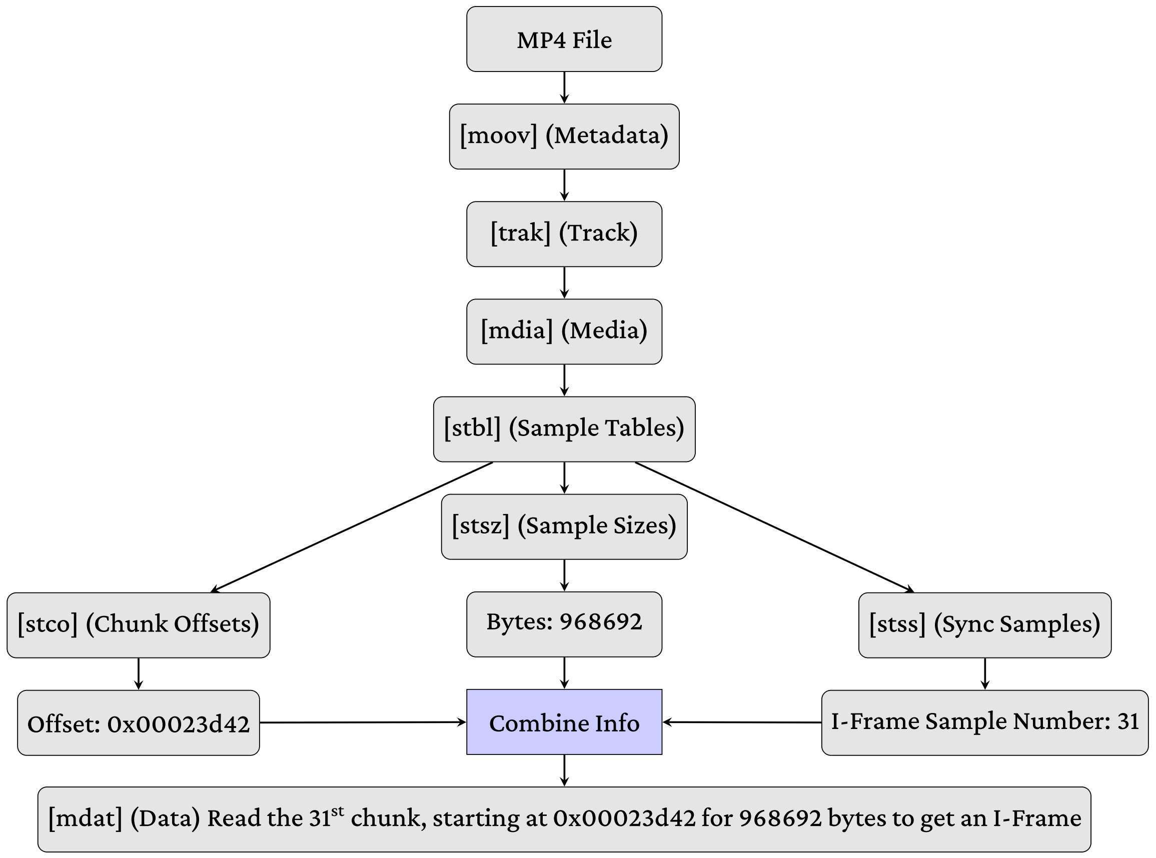


Figure 3.3: Fetching an I-frame using MP4 metadata

Figures 3.4 and 3.5 demonstrates the relationship between key components of the MP4 parser library using an *UML Class* diagram. Unlike standard OOP languages like Java that demonstrate direct inheritance using *Classes* and *Interfaces*, Rust gives developers the ability to accomplish similar goals with primitives such as *Structs* and *Traits*. Rust employs composition over inheritance that allows combining multiple structs to create more complex objects.

Somewhat similar to Java, which uses inheritance to create a hierarchy of classes and demonstrate a relationship between each of those classes. In other words, Java encourages an 'is-a' relationship, while Rust offers a 'has-a' relationship to achieve code re-usability. Thus, even though they both provide inheritance properties, they exhibit some key differences. Traits in Rust may define methods optionally implemented by the inheriting struct, while interfaces in Java define a contract that an implementing class must strictly adhere to.

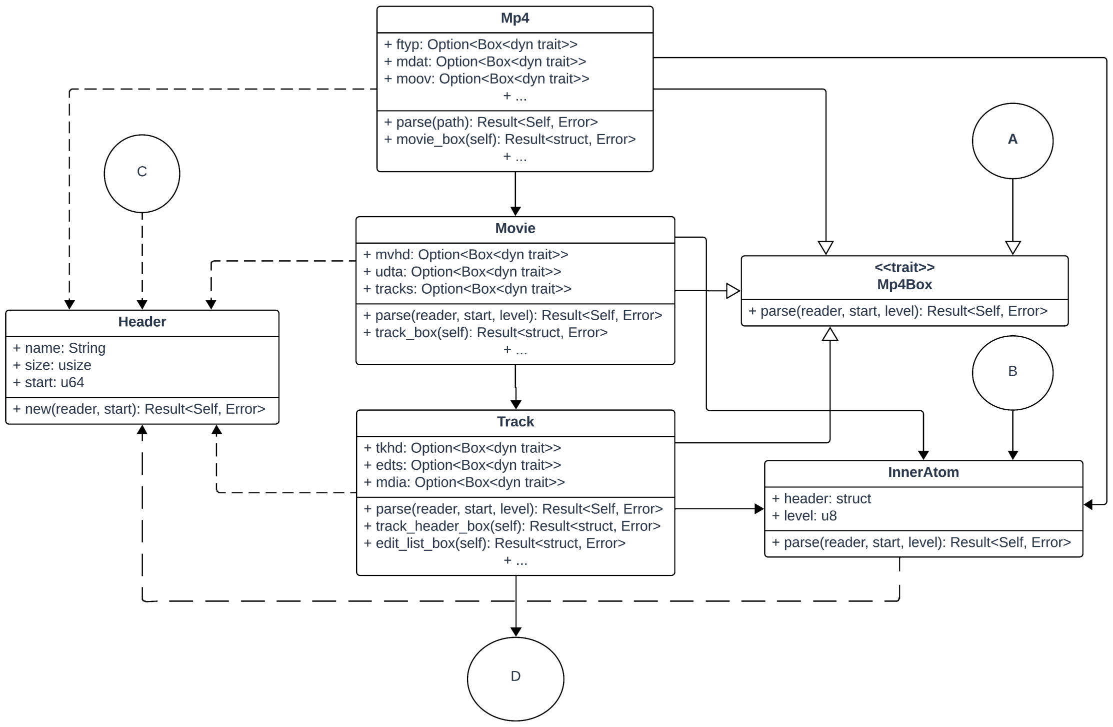

Mp4
+ ftyp: Option<Box<dyn trait>>
+ mdat: Option<Box<dyn trait>>
+ moov: Option<Box<dyn trait>>
+ ...
+ parse(path): Result<Self, Error>
+ movie_box(self): Result<struct, Error>
+ ...
C
A
Movie
+ mvhd: Option<Box<dyn trait>>
+ udta: Option<Box<dyn trait>>
+ tracks: Option<Box<dyn trait>>
+ parse(reader, start, level): Result<Self, Error>
+ track_box(self): Result<struct, Error>
+ ...
<<trait>>
Mp4Box
+ parse(reader, start, level): Result<Self, Error>
Header
+ name: String
+ size: usize
+ start: u64
+ new(reader, start): Result<Self, Error>
B
Track
+ tkhd: Option<Box<dyn trait>>
+ edts: Option<Box<dyn trait>>
+ mdia: Option<Box<dyn trait>>
+ parse(reader, start, level): Result<Self, Error>
+ track_header_box(self): Result<struct, Error>
+ edit_list_box(self): Result<struct, Error>
+ ...
InnerAtom
+ header: struct
+ level: u8
+ parse(reader, start, level): Result<Self, Error>
D

Figure 3.4: UML Class Diagram (Main) for MP4 Parser

For example, the *Mp4Box* trait provides a variety of methods without a method body, which is analogous to *Interfaces* in an OOP paradigm. Structures like *Mp4*, *InnerAtom*, *Movie*, and others implement one or more methods from the *Mp4Box* trait, similar to *Classes* in an OOP environment. The *implement* keyword in Rust, like the *implements* keyword in OOP, provides for the inheritance of parent trait methods and their implementation.

Concerning polymorphism property, Java offers both compile-time and run-time Polymorphism through method overloading and method overriding, respectively. Rust, on the other hand, encourages ad-hoc polymorphism using traits that support static dispatch, where depending on the type, the actual implementation is determined at compile-time.

For each atom type identified in MP4 file, the two standard fields are the *atom name* and *atom size*, with certain atoms incorporating sub-level atoms. For instance, the *ftyp* and *free* atom does not contain sub-level atoms, whereas the *moov* atom holds underlying sub-level atoms. This information helps decide *Template Method* as the appropriate design pattern for the parser library. To gather header information like the atom name and size, we construct a *parse* method utilizing skeleton code and the *Template Method* design pattern. Depending on the kind of atom, implementing the *parse* method may incorporate atom-specific logic. For instance, while implementing the Template Method *parse* for the *Movie* struct, it might be necessary to include additional code to parse sub-level atoms like trak.

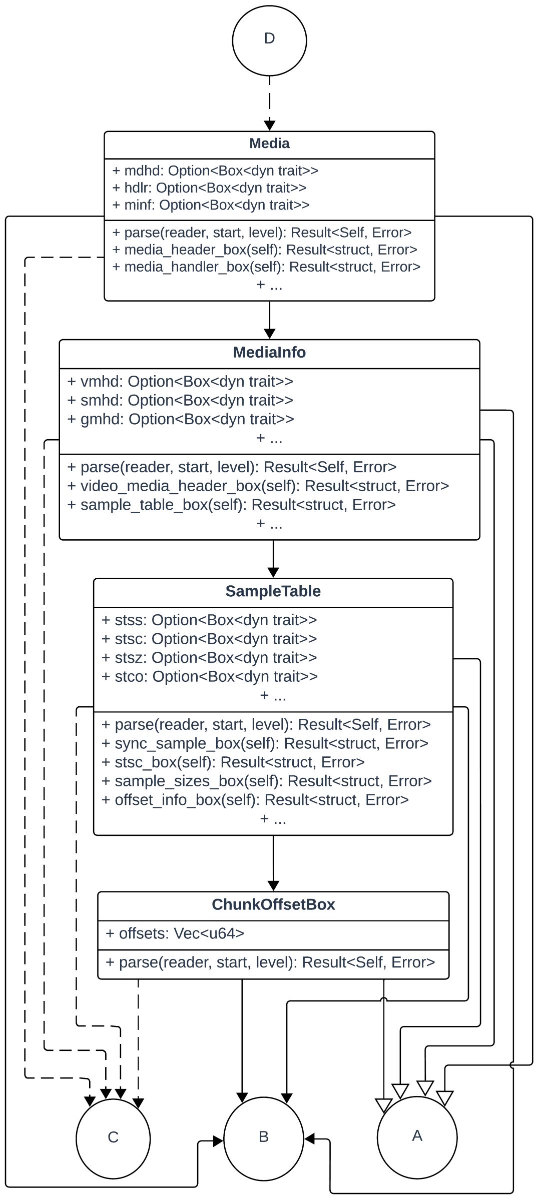

D
Media
+ mdhd: Option<Box<dyn trait>>
+ hdlr: Option<Box<dyn trait>>
+ minf: Option<Box<dyn trait>>
+ parse(reader, start, level): Result<Self, Error>
+ media_header_box(self): Result<struct, Error>
+ media_handler_box(self): Result<struct, Error>
+ ...
MediaInfo
+ vmhd: Option<Box<dyn trait>>
+ smhd: Option<Box<dyn trait>>
+ gmhd: Option<Box<dyn trait>>
+ ...
+ parse(reader, start, level): Result<Self, Error>
+ video_media_header_box(self): Result<struct, Error>
+ sample_table_box(self): Result<struct, Error>
+ ...
SampleTable
+ stss: Option<Box<dyn trait>>
+ stsc: Option<Box<dyn trait>>
+ stsz: Option<Box<dyn trait>>
+ stco: Option<Box<dyn trait>>
+ ...
+ parse(reader, start, level): Result<Self, Error>
+ sync_sample_box(self): Result<struct, Error>
+ stsc_box(self): Result<struct, Error>
+ sample_sizes_box(self): Result<struct, Error>
+ offset_info_box(self): Result<struct, Error>
+ ...
ChunkOffsetBox
+ offsets: Vec<u64>
+ parse(reader, start, level): Result<Self, Error>
C
B
A

Figure 3.5: UML Class Diagram (Continued) for MP4 Parser

3.2.3 Evaluation

Rust's primitive benchmarking crate criterion.rs was employed to assess the performance of the parser library [46]. The crate allows running multiple benchmarks and comparing the statistics between current and previous runs. Performance information may be seen as graphs created with the GNUPlot tool or as sample readings saved in a CSV file. The *criterion* benchmarking script comprised four stages: warmup, measurement, analysis and comparison. The warmup phase involves repeatedly executing the parser library set warmup period to populate caches and hardware. If the programmer specifies no warmup time, a warmup period of 3 seconds is utilized by default. Without this warmup, evaluation script will produce highly skewed data points for the first few iterations. We used a warm-up period of 10 seconds, followed by 50 standard evaluation runs, considering our means were in the order of 3 seconds. Based on the sample iterations count set, the benchmark is conducted against the parser utility in the measurement stage while keeping track of the execution times.

File Size	**Mean (μs)**	**Median (μs)**	**Std. Dev. (μs)**
3.9 MB	109.476	108.917	16.954
457.6 MB	109.618	109.563	15.346
968.4 MB	108.831	108.917	15.023
1.54 GB	108.710	109.042	15.002
2.07 GB	109.283	108.896	17.366

Table 3.1: Criterion Evaluation Results for MP4 Parser

The data gathered from the measurement step are used to compute several statistics, including Mean, Median, Standard Deviation, and more, in the Statistics stage. The statistics are then provided to the user through statistical calculations. Every second benchmarking result is evaluated in relation to the initial benchmark. The user is presented with logs and plots that show the statistical variance. In addition to writing the assessments, adjustable confidence levels are also reported. If the programmer fails to specify a confidence level, a default confidence level of 95% is applied during benchmarking. The sample time vs number of iterations Box Plot in Figure 3.6, demonstrates how the processing time of the MP4 parser varies with different file sizes over linear scale.

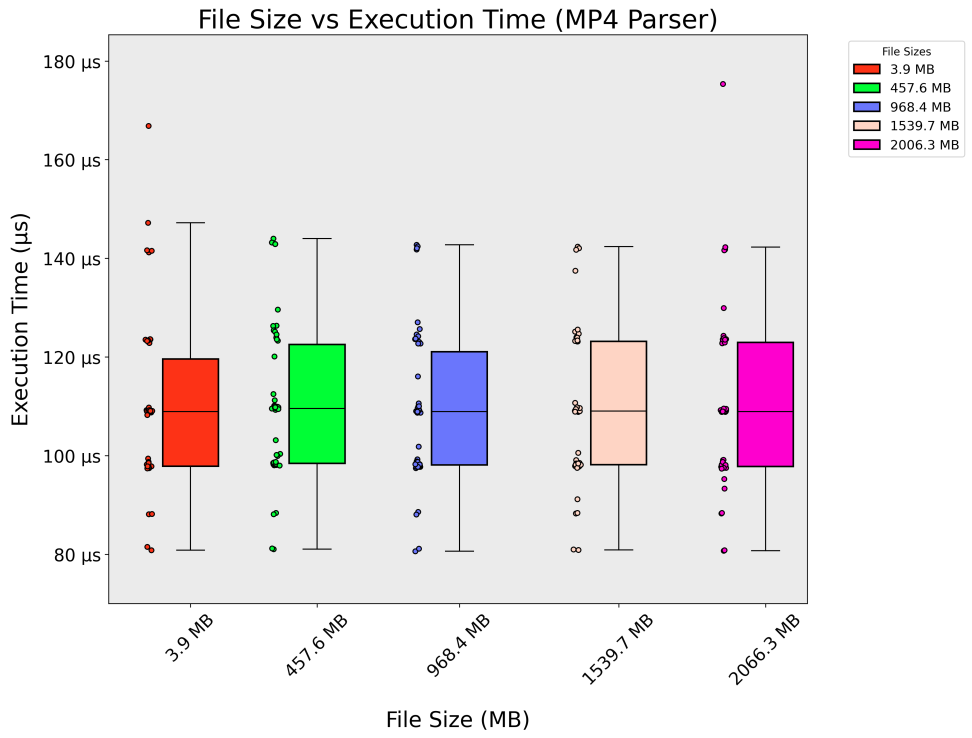


Figure 3.6: Box Plot for Execution Time vs Varying MP4 file size

The mean execution times ranging from 109.48 μs for a 3.9 MB file to 109.29 μs for a 2.07 GB file, indicating that average parsing time is almost constant with respect to different file sizes. The median execution times, closely aligned with the mean, range from 108.92 μs

to 108.90 μs, highlighting the consistency of parsing times across different file sizes. The standard deviation values remain low, gradually increasing from 16.95 μs for the smallest file to 17.36 μs for the largest, suggesting that the variability in execution time is minimal and does not significantly change as file size increases. The statistical calculations with upper and lower bounds (in Microseconds) are provided in Table 3.1.

Unlike MPEG-TS, which requires parsing complete packets that include both header and payload to extract required metadata, the MP4 parser does not require traversing the entire file. The critical atom metadata is extracted without the need to parse the mdat atom, which contains the bulk of the payload data. With almost constant parsing time, relatively low and stable standard deviation values across all file sizes suggest effective resource management and consistent I/O operations, ensuring predictable and reliable parsing performance even as file complexity and size increase.

3.3 MPEG-TS Parser

3.3.1 Background

The MPEG-TS container format is described by the ISO/IEC 13818-1 standard[47, 48]. The three prevalent packet types that may be found in a MPEG-TS container format are seen in Figure 3.7. A *transport stream*, a *packetized elementary stream*, and an *elementary stream* are the three packets used to transmit the video stream. To create a *transport stream* packet, an elementary stream is split into 188-byte chunks, each having 4 bytes of header metadata and a variable-length payload. An optional 8-byte *adaptation field* is present between the TS header and payload and includes information on clock references for time synchronization.

The *sync* byte, which carries the magic-number 0x47 to signify the start of a TS packet, is used to synchronize TS packets. Another significant piece of metadata in the TS header is the PID, or *packet identifier*, which is essential for determining what data the current TS packet's payload includes. For instance, a PID of 0x258 indicates that the payload of the supplied packet contains geospatial KLV metadata.

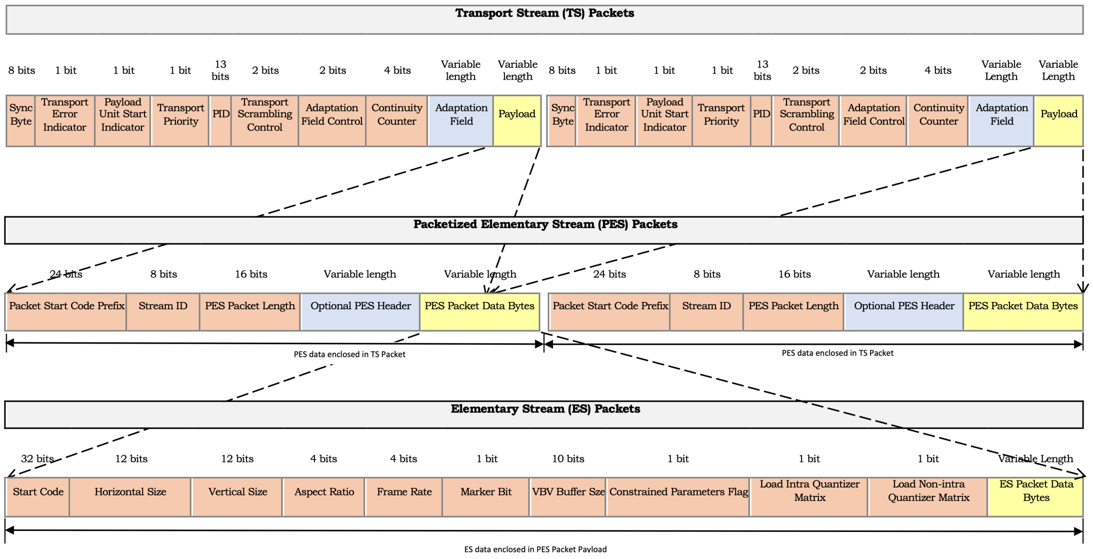


Figure 3.7: Structure of MPEG-TS file

The bytes associated with a *PES packet* are contained in the TS packet's payload section, depending on whether the *payload unit start indication* (pusi) flag is set. Similar to TS packets, PES packets include distinct Header and Payload fields. The *packet start code prefix* is the first three bytes of the PES header and must always include the hex value 0x00 00 00 01 in order to be valid. This 3-byte prefix is followed by a 1-byte *Stream ID* used to identify the kind of data in the packet. For instance, a Stream ID between 0xC0-0xDF and 0xE0-0xEF, respectively, will indicate the availability of audio and video streams. On the other hand, a Stream ID of 0xFC confirms that the current stream is a metadata stream. Other Stream IDs accessible in MPEG-TS are shown in Table 3.2 below.

Stream_ID		Stream Coding
Hex	Binary	
0xBC	1011 1100	program_stream_map
0xBD	1011 1101	private_stream_1
0xBE	1011 1110	padding_stream
0xBF	1011 1111	private_stream_2
0xCx	110x xxxx	ISO/IEC 23008-3 audio stream number 'x xxxx'
0xEx	1110 xxxx	Rec. ITU-TT H.265 video stream number 'xxxx'
0xF0	1111 0000	ECM_stream
0xF1	1111 0001	EMM_stream
0xF2	1111 0010	ISO/IEC 13818-6_DSMCC_stream
0xF3	1111 0011	ISO/IEC_13522_stream
0xF4	1111 0100	Rec. ITU-TT H.222.1 type A
0xF5	1111 0101	Rec. ITU-TT type B
0xF6	1111 0110	Rec. ITU-TT H.222.1 type C
0xF7	1111 0111	Rec. ITU-TT H.222.1 type D
0xF8	1111 1000	Rec. ITU-TT H.222.1 type E
0xF9	1111 1001	ancillary_stream
0xFA	1111 1010	ISO/IEC 14496-1_SL-packetized_stream
0xFB	1111 1011	ISO/IEC 14496-1_FlexMux_stream
0xFC	1111 1100	metadata stream
0xFD	1111 1101	extended_stream_id
0xFE	1111 1110	reserved data stream
0xFF	1111 1111	program_stream_directory

Table 3.2: Stream ID assignments (from ISO/IEC 13818-1:2018)

The payload portion of the PES packet contains an *elementary stream* packet with a 4-byte start code of 0x00 00 01 B3. The header bytes of a PES packet contains essential data, such as the horizontal and vertical measurements necessary to determine a video's *resolution*.

The *frame rate*, *bit rate*, and *aspect ratio* of the video are among the extra critical metadata that may be found in an ES header.

3.3.2 Algorithm Design & Implementation

Our MPEG-TS parser library is also written in Rust, offering an efficient iterator over different packets found in MPEG-TS containers. Figure 3.8 demonstrates the relationship between key components of the MPEG-TS parser library using a UML diagram.

The iterator first determines if we have reached the end of the file by comparing the current address against the size of the input MPEG-TS file. Using a loop that terminates if the first byte from the read bytes is not the hexadecimal value 0x47, the first 188 bytes of the input file are read into a buffer with a capacity of 188 bytes. This hexadecimal number represents the commencement of a Transport Stream packet. The *payload unit start indicator* and *adaptation field control* flags are typically extracted after this by reading the first 4 bytes of the TS Header to verify (1) whether the payload comprises the start of a PES packet, (2) whether an adaptation field is present.

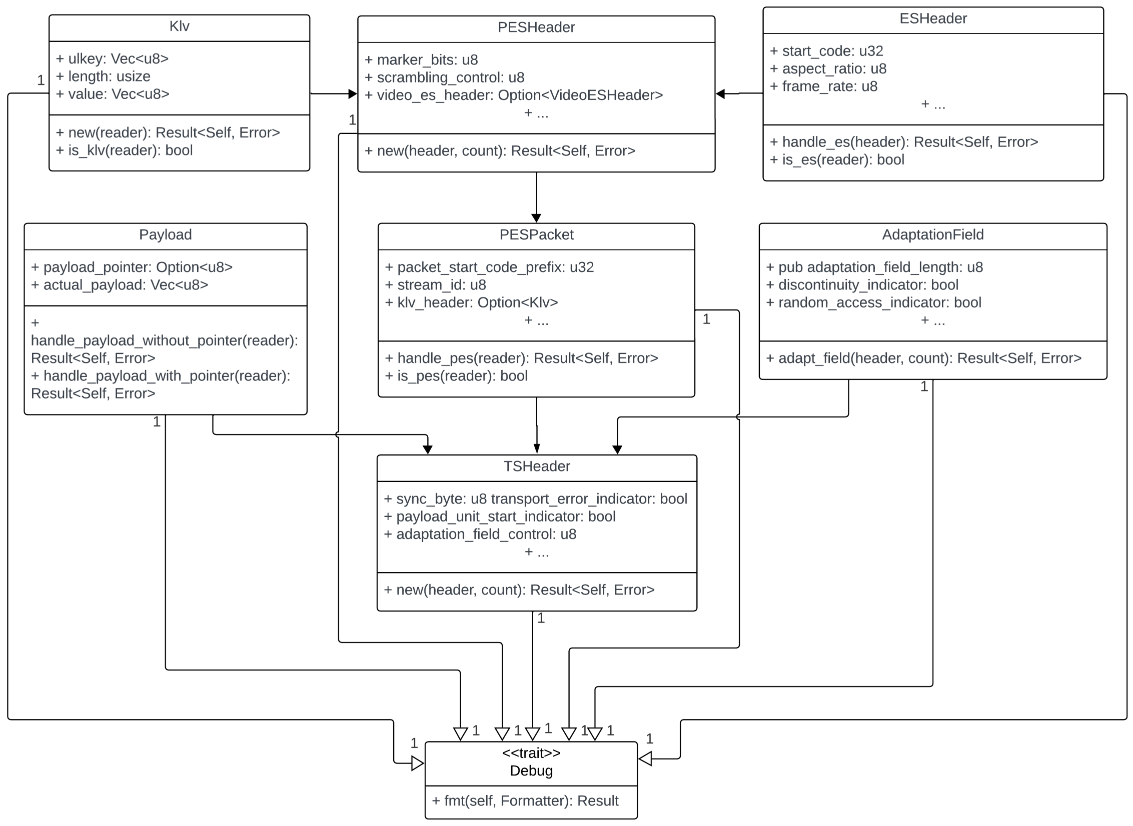

Klv
+ ulkey: Vec<u8>
+ length: usize
+ value: Vec<u8>
+ new(reader): Result<Self, Error>
+ is_klv(reader): bool
PESHeader
+ marker_bits: u8
+ scrambling_control: u8
+ video_es_header: Option<VideoESHeader>
+ ...
+ new(header, count): Result<Self, Error>
ESHeader
+ start_code: u32
+ aspect_ratio: u8
+ frame_rate: u8
+ ...
+ handle_es(header): Result<Self, Error>
+ is_es(reader): bool
Payload
+ payload_pointer: Option<u8>
+ actual_payload: Vec<u8>
+ handle_payload_without_pointer(reader): Result<Self, Error>
+ handle_payload_with_pointer(reader): Result<Self, Error>
PESPacket
+ packet_start_code_prefix: u32
+ stream_id: u8
+ klv_header: Option<Klv>
+ ...
+ handle_pes(reader): Result<Self, Error>
+ is_pes(reader): bool
AdaptationField
+ pub adaptation_field_length: u8
+ discontinuity_indicator: bool
+ random_access_indicator: bool
+ ...
+ adapt_field(header, count): Result<Self, Error>
TSHeader
+ sync_byte: u8 transport_error_indicator: bool
+ payload_unit_start_indicator: bool
+ adaptation_field_control: u8
+ ...
+ new(header, count): Result<Self, Error>
<<trait>>
Debug
+ fmt(self, Formatter): Result
1

Figure 3.8: UML Class Diagram for MPEG-TS parsing library

The PES parsing function returns the information from the PES Header and Payload. To further distinguish the type of data being read in the subsequent phase, the PID field of the PES packet is utilized. For instance, a PID of 0x258 indicates that geospatial KLV data is present in the payload of the current packet. The parser also displays the offset address of each TS and PES packet. This information might be beneficial for a visual redaction tool to narrow down the bytes and pixels associated with a certain object in the scene so that redaction can be applied.

3.3.3 KLV Parser

3.3.3.1 Background

Government agencies, such as the *US Department of Defense*, routinely use drones for ISR missions that produce full-motion video imagery. FMV analysts are frequently required to manually analyze and evaluate the video stream recorded during such ISR missions [49].

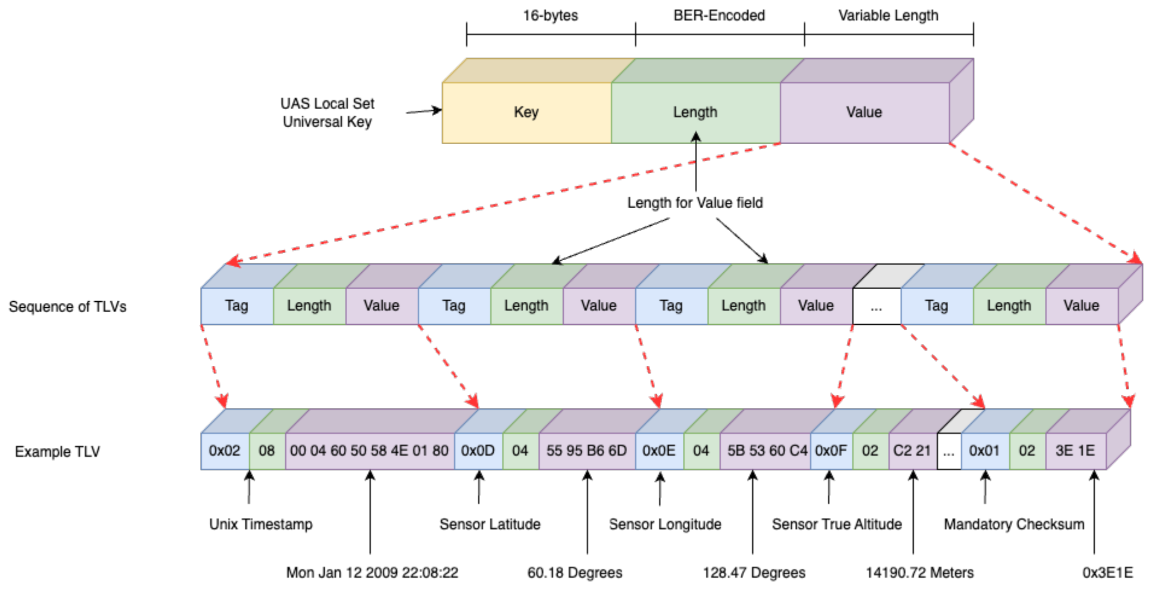


Figure 3.9: MISB Standard (SD) 0601.8 - UAS Datalink Local Set KLV Packet

To encode MISB-compliant full-motion videos, the KLV encoding described in MISB ST 0601.17 specification is implemented. In the payload component of an MPEG-TS PES packet, a KLV packet can be encountered. To uniquely identify the beginning of a KLV packet, a fixed-size *key* is immediately embedded within the KLV packet. In the standard, this *key* is referred to as the *UAS Local Set Universal Key*. The BER standard discussed in the preceding section 3.3.4 is then used to encode a *length* field. Figure 3.9 shows a collection of KLV packets, including various geospatial metadata components. The *value* field within a KLV packet may be interpreted as a *payload* field comprising a sequence of *TLV sub-packets*, which stand for *tag-length-value*. Unlike the *key* field, the *tag* field in a KLV packet is connected with a specific geospatial metadata item. For example, the metadata fields *checksum* and *Unix timestamp* are represented by *tag* values 1 and 2, respectively, and are also deemed required tags by the standard.

For example, *tag* 0x02 in Figure 3.9 signifies the *Unix Timestamp*, followed by 0x08 which represents the *length* of the given *tag* and finally the *value* field holding the hexadecimal bytes 0x00046050584E0180. Converting this number to decimal yields the Unix timestamp, i.e. 1,231,798,102,000,000 in seconds, which can be represented into human readable time as Monday, January 12, 2009, 22:08:22. A Unix epoch is a way of representing and storing time in operating systems. The beginning of the Unix timekeeping system was defined as 00:00:00 in UTC on January 1, 1970. The decimal number above is the number of seconds that have elapsed while keeping this Unix epoch as a reference. Similarly, for the tag value of 0x0E, we can obtain the sensor longitude from its hex representation 0x5B5360C4 using the Equation 3.1.

$$\begin{aligned}
\text{Longitude} &= \left[\frac{\text{Maximum longitude in degrees}}{\text{Maximum hex bytes}}\right] \times \text{Longitude hex bytes} \\
&= \left[\frac{360}{4{,}294{,}967{,}295}\right] \times 1{,}532{,}190{,}916 \qquad (3.1) \\
&= 128.47^{\circ}
\end{aligned}$$

The sensor altitude can be derived by reading the tag value 0x0F, which has a length of

0x02, indicating the altitude field. The value field is 0xC221, which translates to 49,697 in decimal representation. To compute the altitude in meters, Equation 3.2 can be used:

$$\begin{aligned}\text{Altitude} &= \left[\frac{\text{Altitude Range}}{\text{Unit Range}} \times \text{Value Bytes}\right] - \text{Offset}\\ &= \left[\frac{65{,}535}{49{,}697} \times 1{,}532{,}190{,}916\right] - 900 \qquad (3.2)\\ &= 14{,}190.72 \text{ meters}\end{aligned}$$

Where the Altitude Range is the difference between the maximum and minimum altitude levels: 19000-(-900) = 19900, Unit Range is the maximum value for a 16-bit unsigned integer (65,535) and Offset is the minimum altitude value, which is -900 meters. The altitude value of 14,190.72 with cm precision with altitude values ranging in 14Kms can be overly precise. The recommended resolution for an altitude sensor is approximately 0.3 meters [50]. This means the sensor can reliably measure and report altitude changes of at least 0.3 meters.

To ensure the final result is an integer and a multiple of 0.3, we can select the closest integer value such that its least significant digit is divisible by 3. This approach aligns with rounding to a multiple of 0.3. For example, the closest integer value to our result is 14,190 (since 90 is divisible by 3), which means 14,190 is divisible by 0.3, resulting in: $14,190 \div 0.3 = 47,300$. Therefore, the altitude value rounded to the nearest multiple of 0.3 meters, ensuring it is an integer, is: 14,190 meters.

3.3.3.2 Algorithm Design & Implementation

The KLV parser extends the MPEG-TS parser presented in the preceding section 3.3. In full-motion videos, the payload section of a PES packet is intended to hold codec-specific information. For example, an H.264 video stream, Advanced Audio Coding (AAC) audio stream and KLV metadata stream may be contained in the payload of a video, audio and metadata PES packet [43]. The KLV parser decodes PES packets with the PID set to 0x258 to indicate that the packet is a KLV packet. After the correct PID has been determined, we examine whether the initial 16 bytes of the payload correspond to the *UAS Datalink Local Set Key*.

If a match is found, the first bit of the following byte is read to determine if the *length* of the KLV packet is encoded using the *short* or *long* BER encoding standards. The *bitreader* crate is used to parse singular bits [51]. If the BER encoding type flag is set to 0, the following seven bits are parsed to determine the *length* of the KLV packet. If this flag is set to 1, the *length* of the *length* field is determined by parsing the remaining 7 bits. Depending on whether the length field is short or long BER encoded, the iterator is incremented with the final *length* value. The final length is then utilized to parse the value bytes containing the underlying geospatial metadata encoded in the TLV format. The KLV redaction technique described in chapter 4 expands this work by considering individual TLV bytes when obfuscating the value bytes. Figure 4.2 depicts an example of a KLV packet output when a full-motion movie is provided as input to the library.

3.3.4 BER Parser

In a full-motion video, the *length* value from both KLV and TLV packets stream is encoded in BER [52, 53]. The motivation behind writing the BER parser library was to decode the underlying Value field from KLV and TLV that held various metadata items defined by MISB ST 0601.17 specification, including the geospatial metadata. The *basic encoding rules* (ber) specify a set of guidelines for encoding data in binary form for transmission to another system regardless of the underlying hardware or platform. Information can be encoded in BER format in three different ways: *short*, *long*, and *indefinite* BER encoding, as shown in Figure 3.10.

	7	6	5	4	3	2	1	0
Short Form	0	Length of Value field						
Long Form	1	Length of Length field						
Indefinite Form	0	0	0	0	0	0	0	0

Figure 3.10: Basic Encoding Rules

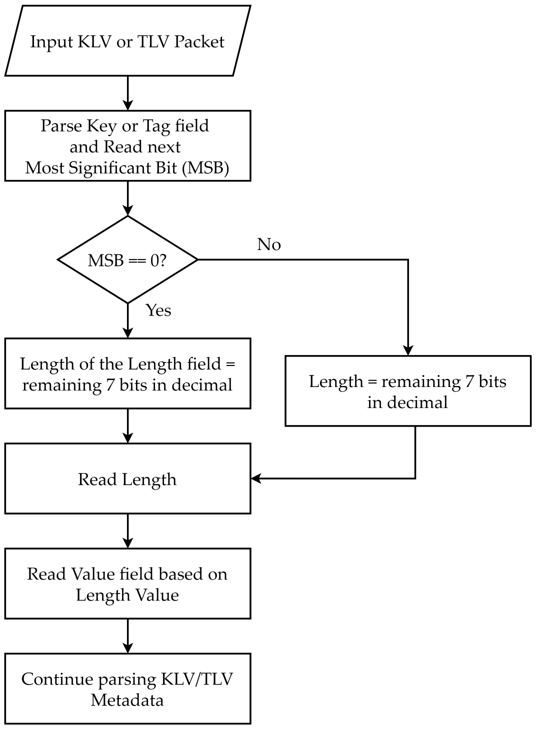

Input KLV or TLV Packet
Parse Key or Tag field and Read next Most Significant Bit (MSB)
MSB == 0?
No
Yes
Length of the Length field = remaining 7 bits in decimal
Length = remaining 7 bits in decimal
Read Length
Read Value field based on Length Value
Continue parsing KLV/TLV Metadata

Figure 3.11: BER Parser

Figure 3.11 displays the BER encoding technique. A *short* version of BER encoding is used to encode data whose length can be expressed with 7 bits. The seventh index bit of the octet is set to zero to indicate that this is a short form of encoding. The remaining 7 bits, from 6 to 0, are used to determine the *length* of the *value* field. When a length value cannot be expressed using 7 bits, a *long* variant of BER encoding is employed. The leftmost bit of the octet, or bit 7, is set to 1, while the following 7 bits reflect the *length* of the *length* field. The third type of BER encoding is termed *indefinite* form, and it involves setting all the bits in an octet to zero and ending the *value* field with two NULL bytes.

3.3.5 Evaluation

We followed the identical procedures as those outlined in the evaluation section for the MP4 parser library to assess performance of the MPEG-TS parser library, i.e., by using Criterion benchmarking crate by Rust. The Box Plot shown in Figure 3.12 demonstrates how the processing time of the parser varies with different file sizes. MPEG-TS parser exhibits a clear correlation between file size and parsing time, with execution times increasing significantly as file sizes grow over linear scale.

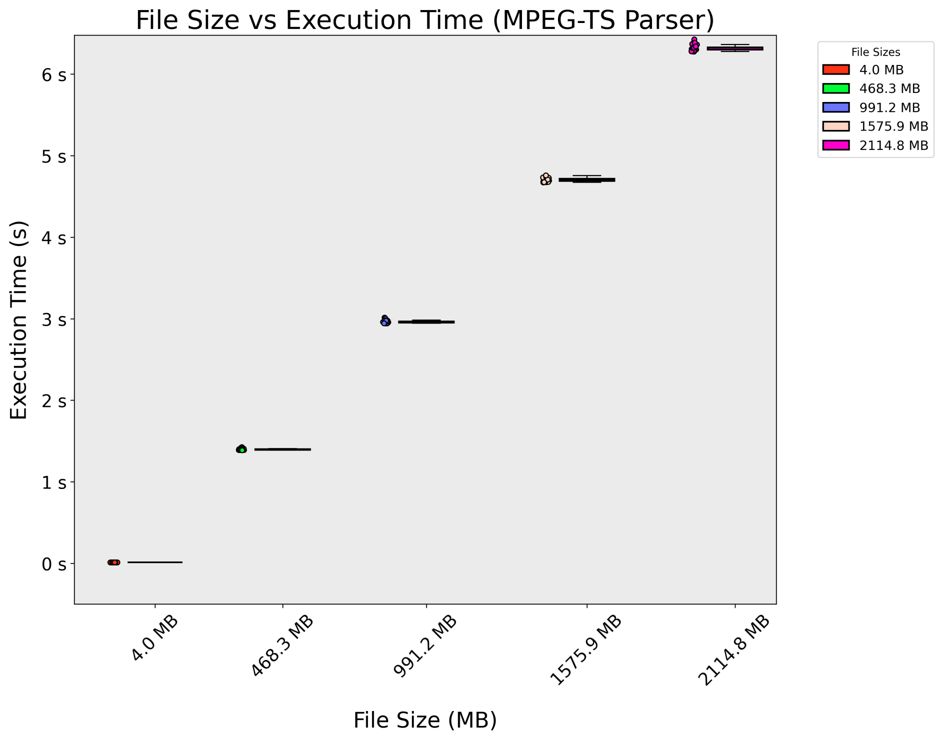


Figure 3.12: Box Plot for Execution Time vs Varying MPEG-TS file sizes

Table 3.3 presents key performance statistics of the MPEG-TS parser library, focusing on the mean, median, and standard deviation of processing times (in Milliseconds) for different file sizes.

File Size	Mean (s)	Median (s)	Std. Dev. (s)
4 MB	0.012	0.012	0.000
468.3 MB	1.398	1.396	0.008
991.3 MB	2.962	2.959	0.014
1.58 GB	4.704	4.701	0.021
2.11 GB	6.315	6.305	0.031

Table 3.3: Criterion Evaluation Results for MPEG-TS Parser

The mean execution times range from 0.012s (4 MB file) to 6.315s (2.11 GB file). Similarly, the median times follow this trend, increasing from 0.012s to 6.305s. The standard deviation ranges from 0s to 0.031s, reflecting increasing variability in execution times, especially for larger files. The MPEG-TS files are structured in the form of packets that include both headers and payloads, necessitating comprehensive parsing of each packet to extract relevant information. This requirement, combined with the larger amount of data in bigger files, leads to longer execution times and greater variability as file size increases.

3.4 Comparative Analysis

Given that MP4 and MPEG-TS are a widely adopted container format, several tools have been developed to analyze, dissect, and manipulate the audio, video, and metadata stored underneath. These tools differ in terms of their offerings, from basic metadata extraction to complex video analytics. However, they do not fully address the specific requirement of arbitrarily seeking bytes of interest. This section explores some of the well-known container parsers and benchmarks them against the tools proposed in this research.

3.4.1 Related Tools

The candidate tools for comparison were selected primarily from GitHub repositories, with additional insights drawn from a review of parsing techniques discussed in journal and conference articles. While these publications provided valuable theoretical foundations, they did not consistently include accessible or fully operational code, making GitHub the main source of practical implementations. We prioritized tools that compiled successfully and included user documentation. We chose some tools for their popularity and active development, as indicated by high numbers of stars, forks, and recent commits. Others were selected for their unique capabilities in parsing metadata relevant to our research, even if they had less community engagement. By including both widely-used and niche tools, we aimed to provide a well-rounded comparison.

Tools relevant to MP4 videos are listed below.

atomicparsley A command-line tool for analyzing metadata in MP4 files. It supports various operations such as reading, parsing, setting metadata tags [54].

Bento4 is a comprehensive video processing toolkit for working with mp4 files, offering features for both reading and writing MP4 files. Bento4 offers both, a library written in C++ and multiple Command Line Interface (CLI) tools such as mp4dump, mp4info, mp4split, etc., for different purposes [55].

qtfaststart is a command-line utility for optimizing MP4 files for web streaming by rearranging the file structure. It moves the metadata and offset information to the beginning of the file, improving playback start times for streaming scenarios [56].

GPAC is a framework containing several tools, in the form of library, CLI and GUI to process, convert, and analyze multimedia content. It supports parsing of several container formats, including MP4. The mp4box command-line utility from GPAC has been used during comparisons [57].

MediaInfo is a tool for metadata retrieval from different container formats, with about 37 languages for metadata support. It is cross-platform, written in Pascal and C++, offered as a library, command-line utility and a GUI variations [58].

ExifTool is a Perl library and CLI tool for reading, writing, and editing metadata from different types of multimedia files including images and videos [59].

FFmpeg is a well-rounded collection of CLI tools using libav [7] as library underneath for handling video processing tasks such as transcoding, streaming, editing, and compression. FFmpeg is open-source, platform-independent, and supports over 100 different codecs for video encoding and decoding [60].

Tools for working with MPEG-TS video are outlined below.

m2pb is developed by Google, written in C++, this parser library extracts and analyzes MPEG-TS streams, allowing users to selectively print specific metadata fields using corresponding flags [61].

mpeg-ts-media is a C++ library designed to handle MPEG-TS containers with H.264 and AAC streams. It parses the metadata structure and additionally de-muxes audio and video streams into separate files [62].

mpeg2tsparser is a simple MPEG-TS parser library that extracts metadata from TS packets along with their position offsets. It only parses Boolean based metadata fields [63].

mpeg_parser is another library that only parses TS packets from MPEG-TS streams, outputting metadata in an XML-formatted file for easy analysis [64].

mpegts-basic-parser is written in C, this lightweight parser is designed for basic parsing of MPEG-TS files, primarily focusing on inspecting metadata from audio and video streams

[65].

TS_Parser is a minimalistic MPEG-TS library that parses basic TS packet information along with offset addresses [66].

mpeg-ts-parser is an extension of the TS_Parser library, it supports parsing high-level header metadata for TS packet and additional streams such as subtitles [67].

3.4.2 Identifying Gaps in existing Tools

This subsection highlights the deficiencies found in existing parsers and the specific requirements they did not fulfill. It discusses how these gaps in security, performance, and functionality prompted us to write custom parsers that better suited our needs. Table 3.4 and 3.5 highlight some of the key differences in existing parsers vs parsers proposed in this research.

3.4.2.1 Generic Requirements

When investigating existing parsers, we noted some projects on GitHub where the program calls tool binaries like ffprobe or ExifTool directly within the code. This approach is not the best practice for several reasons. Firstly, invoking external binaries introduces performance overhead from process creation and inter-process communication, which degrades application efficiency. Secondly, running external binaries leads to higher consumption of system resources, such as memory and CPU, compared to using an integrated library embedded within the application.

Furthermore, managing dependencies between tool binaries and the application becomes more complex, increasing the likelihood of compatibility issues and making setup and maintenance more cumbersome. Finally, external binaries offer less flexibility and customization options than libraries, limiting the ability to tailor functionality to specific needs. Therefore, we require the parser to be available as a library to facilitate easy integration with various applications and systems.

Our research also highlighted the importance of access to packet offsets, which are crucial for accurately locating sensitive metadata that needs to be secured. Cryptographic filesystem like ADAPT (Advanced Detection and Prevention of Tampering) can use these offset addresses as input to preserve the integrity of the metadata.

Our analysis indicates that traditional parsers are not often written using memory-safe languages, thereby posing security risks due to a known vulnerabilities associated with these languages. Lack of patching and community interest also make these types of libraries less desirable from a security perspective. To mitigate these risks, we seek parsers written in memory-safe languages that prevent common programming errors such as buffer overflows and memory leaks.

Tool Features	**atomicparsley**	**Bento4**	**qtfaststart**	**gpac**	**mediainfo**	**exiftool**	**FFmpeg**	**Proposed MP4 Parser**
Uses Memory-Safe Language								✓
Available as a Library		✓		✓	✓	✓	✓	✓
Basic metadata parsing	✓	✓	✓	✓	✓	✓	✓	✓
STCO Box Parsing				✓				✓
STSS Box Parsing				✓				✓
STSZ Box Parsing				✓				✓
Access to Box Offsets	✓						✓	✓
Handling unknown Boxes	✓							✓
Acceptable Update Frequency					✓	✓	✓	✓
Known Vulnerabilities	✓	✓		✓	✓	✓	✓	n/a

Table 3.4: Proposed MP4 Parser vs Existing MP4 Parsers

3.4.2.2 Requirements specific to MP4 Parsers

For MP4 parsing, we needed a parser library capable of handling the parsing of STCO, STSS, and STSZ boxes. Parsing the STCO box is essential because it provides offset addresses to specific frames, enabling efficient navigation within the file. Parsing the STSS box helps identify the type of frames, such as I-frames or P-frames, which is important for visual information redaction. The STSZ box parsing is necessary to determine the size of the frames. Additionally, handling unknown boxes is vital since different drone vendors may add custom metadata to their footage, requiring flexibility and adaptability beyond standard atoms.

Tool Features	m2pb	mpeg-ts-media	mpeg2ts parser	mpeg_parser	mpegts-basic-parser	TS_Parser	mpeg-ts-parser	Proposed MPEG-TS Parser
Uses Memory-Safe Language								✓
Available as a Library	✓	✓	✓	✓	✓	✓	✓	✓
Basic metadata parsing	✓	✓	✓	✓	✓	✓	✓	✓
Access to Packet Offsets	✓	✓	✓			✓	✓	✓
TS Header Parsing	✓	✓	✓		✓	✓	✓	✓
PES Header Parsing	✓							✓
KLV Stream Parsing	✓							✓
Stream De-muxing		✓						✓
Acceptable Update Frequency		✓				✓		✓
Known Vulnerabilities								n/a

Table 3.5: Proposed MPEG-TS Parser vs Existing MPEG-TS Parsers

3.4.2.3 Requirements specific to MPEG-TS Parsers

For MPEG-TS parsing, we required a parser library capable of handling Transport Stream (TS) headers, Packetized Elementary Stream (PES) headers, and KLV streams. Parsing the TS header helps us identify the type of data contained in each packet, while PES header parsing is crucial it encapsulates the Elementary Stream (ES) that holds the actual KLV data. Parsing KLV streams is essential for accessing the sensitive drone metadata. Additionally, we needed a parser with optional support for stream de-muxing to separate KLV data from video footage, allowing for tailored redaction applications based on the type of stream.

3.4.3 Evaluation

For this comparison, we employed three primary metrics: execution time, memory usage, and CPU usage to comprehensively evaluate the performance and efficiency of the container parsers.

Execution Time refers to the total duration required by the container parser to complete processing of a video file. This metric is crucial for understanding responsiveness of the parsers, especially when dealing with large video files. We measured the execution time of each parser using Hyperfine, a benchmarking tool that provides precise timing measurements.

Memory Usage indicates the amount of memory allocated and used by the parser during the processing of a video file. Efficient memory usage is essential for the stability of the parsers, particularly when handling high-resolution videos or multiple files concurrently. To measure memory usage and safety, we used two tools: Valgrind with Memcheck and Heaptrack. These tools helps detecting total memory usage along with memory leaks by tracking all memory allocations and de-allocations.

CPU Usage measures how intensively the parser utilizes CPU resources while processing the given video file. High CPU usage can impact the overall system performance, especially if the parser is run alongside other applications. By analyzing CPU usage, we can determine the computational efficiency of the parsers and identify any potential bottlenecks. We measured CPU usage using Perf, a powerful performance analysis tool that collects and reports performance data from the CPU.

3.4.4 Experimental Setup

The tests were conducted on a standardized system configuration to ensure uniform testing conditions and consistent hardware resource utilization across all benchmarks. The test machine is equipped with specifications specified in table 3.6 below.

Component	Specification
Processor	Intel Core i5-12600K, 10 cores, 3.7 GHz
RAM	32 GB DDR4
Storage	1 TB SATA SSD
Graphics Card	NVIDIA GeForce RTX 3060
Operating System	Windows 11 (64-bit)

Table 3.6: Hardware and Software Specs for Test Environment

For the benchmarking process, each container parser was evaluated using a consistent input: drone video footage captured from the DJI Mini 2. This was to ensure that the testing conditions are uniform and that performance measurements are both accurate and comparable across different parsers.

3.4.4.1 Benchmarking Tools

We evaluated performance metrics, including execution times, CPU utilization, and memory usage, using the benchmarking tools outlined in this section. Additionally, we assess the parsers from a memory safety standpoint by investigating potential memory leaks to ensure robust and reliable operation.

Hyperfine is a command-line tool that performs statistical analysis on the execution times of a program by using metrics such as mean, median, standard deviation, etc [68]. It supports warm-up runs, useful for programs with significant disk I/O, as it helps mitigate variations caused by disk cache states, e.g. warm vs cold. For this benchmark, we configured the number of runs to be 10 and warmup runs to be 3. Hyperfine supports benchmarking shell commands, e.g., ls, cat, etc., as well as program binaries across different platforms and programming languages. It additionally supports benchmarking multiple binaries simultaneously.

Valgrind is a powerful memory analysis tool used to detect memory leaks, illegal memory accesses, and other memory-related errors [69]. Valgrind operates by running the binary within its own sandbox environment, injecting instrumentation code for memory profiling without us needing to modify the original code. We utilized its Memcheck tool to track memory usage and detect leaks. Memcheck captures use of malloc() and free() function calls to track memory allocations.

Heaptrack is a heap memory profiler that logs memory allocations and de-allocations, offering insights into memory usage patterns and identifying memory leaks [70]. By using Heaptrack, we recorded detailed information about heap memory usage, including peak memory usage and allocation patterns. Heaptrack leverages the LD_PRELOAD mechanism to inject a specially crafted shared object into the given program that monitors system calls and functions related to memory management. It overloads functions like malloc() and free() to capture backtraces and logs runtime information about shared libraries.

Perf is a performance analysis tool that collects data on CPU usage, such as clock cycles, instructions executed, cache hits/misses, and context switches [71]. It helps identify CPU-related bottlenecks by tracking both hardware and software system events. The perf_events interface, allows monitoring of system events occurred during program execution. These events are gathered from sources such as the processor's Performance Monitoring Unit (PMU) or kernel interfaces, and include metrics like context switches, page faults and more. The low-level system events that Perf has access to may vary depending on the processor type and model.

3.4.5 Analysis and Results

This subsection discusses the assessment outcomes from benchmarking tools used to assess candidate parsers for both MP4 and MPEG-TS containers. By examining these metrics, we aim to understand the effectiveness and efficiency of each parser, highlighting their strengths and areas for improvement.

3.4.5.1 MP4 Parsers

The results from assessments for MP4 parsers are outlined below.

Execution Time Comparative execution times for MP4 parsers are shown in Figure 3.13. A higher execution time, as seen with ffprobe (139.80 ms), suggests more extensive processing or inefficiencies, either due to additional features or less optimized code. In contrast, a lower execution time, like that of AtomicParsley (14.10 ms) and Bento4 (15.40 ms), indicates more efficient processing, possibly due to optimized algorithms. The proposed MP4 parser, with an execution time of 29.70 ms, shows competitive performance, balancing speed and functionality effectively. This makes it a viable option for applications that require efficient parsing without compromising on speed.

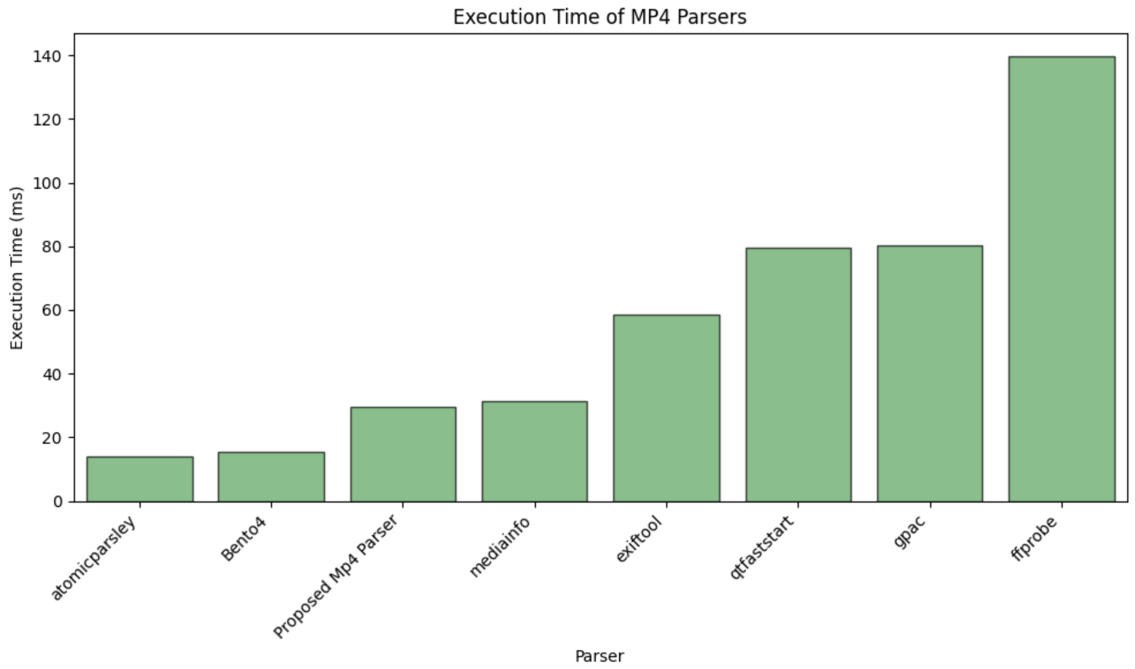


Figure 3.13: Execution Times across MP4 Parsers using Hyperfine

CPU Usage CPU usage statistics for MP4 parsers are depicted in Figure 3.14. Despite the Proposed MP4 Parser extracting comparatively wider array of metadata, its CPU time usage is exceptionally low at 1.57 ms, making it the fastest among all other parsers. This indicates that the proposed parser performs comprehensive and detailed metadata parsing without clogging up the CPU, ensuring efficient resource utilization. In contrast, qtfaststart (91 ms) and exiftool (69.46 ms) exhibit much higher CPU time usage, suggesting more intensive processing requirements or less optimized algorithms. AtomicParsley (1.64 ms) and Bento4 (1.72 ms) also show low CPU time usage, similar to the proposed parser, indicating efficient processing with minimal resource consumption. However, the proposed parser still slightly outperforms them, showcasing its superior optimization.

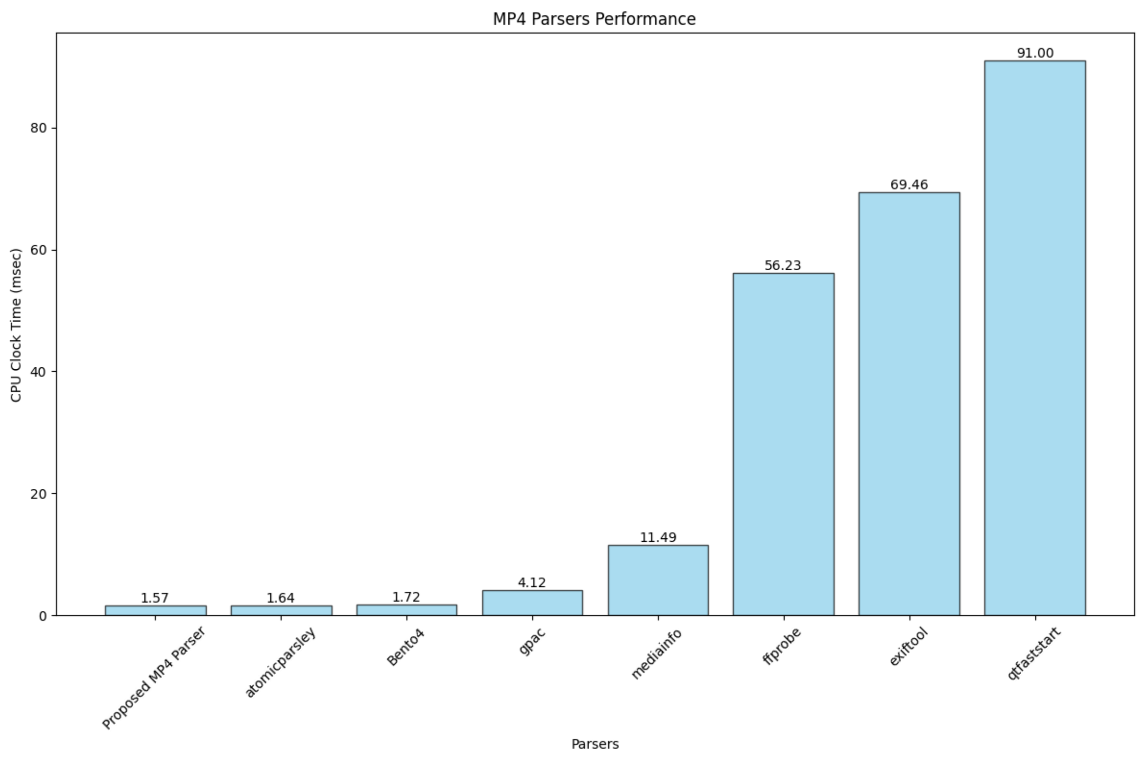


Figure 3.14: CPU Usage Statistics for MP4 Parsers using Perf

Other parsers like GPAC (4.12 ms), MediaInfo (11.49 ms), and ffprobe (56.23 ms) fall between these extremes, with moderate CPU time usage. While they do not match the efficiency of the proposed parser, they still offer reasonable performance, likely balancing processing complexity and optimization. Overall, the proposed MP4 parser's ability to handle extensive metadata while maintaining the lowest CPU time usage highlights its optimized design and effective balance between throughput and efficiency. This makes it an excellent choice for users seeking a lightweight, efficient, and comprehensive MP4 parsing tool.

Memory Usage and Leaks The memory usage results for various MP4 parsers reveal significant differences in their resource efficiency, focusing on total heap usage, memory leakage, and peak heap memory consumption. Detailed memory metrics for MP4 parsers are illustrated in Table 3.7.

Parsers	In Use at Exit	Total Heap Usage	Definitely Lost	Indirectly Lost	Possibly Lost	Still Reachable
Proposed Mp4 Parser	0 B	13.09 KB	0 B	0 B	0 B	0 B
ffprobe	50.37 KB	4.19 MB	0 B	0 B	0 B	48.40 KB
exiftool	12.52 MB	48.87 MB	28.56 KB	59.23 KB	12.43 MB	3.00 KB
mediainfo	0 B	9.76 MB	0 B	0 B	0 B	0 B
qtfaststart	435.59 KB	27.97 MB	0 B	0 B	0 B	435.69 KB
Bento4	0 B	105.92 KB	0 B	0 B	0 B	0 B
atomicparsley	0 B	82.95 KB	0 B	0 B	0 B	0 B

Table 3.7: Memory Usage Statistics for MP4 Parsers using Valgrind

Total Heap Usage The Proposed MP4 Parser demonstrates impressive efficiency with a total heap usage of 13,405 bytes, making it the lowest among the parsers. This indicates that it handles memory usage very efficiently, especially when compared to parsers like qtfaststart (29,326,034 bytes) and exiftool (51,240,784 bytes), which exhibit significantly higher total heap usage. These higher figures suggest that these parsers are more memory-intensive, possibly due to poor memory management.

Total Memory Leaked Analysis of memory leaks detected in MP4 parsers is shown in Figure 3.15. In terms of memory leakage, the Proposed MP4 Parser excels with 0 bytes of memory leaked, indicating that it has no memory management issues and does not leave any allocated memory unreleased. In contrast, exiftool has the highest memory leakage at 13.15 MB, followed by mediainfo (1.50 MB) and qtfaststart (404.99 KB). These figures suggest that these parsers may have issues with memory management, potentially leading to inefficient use of system resources.

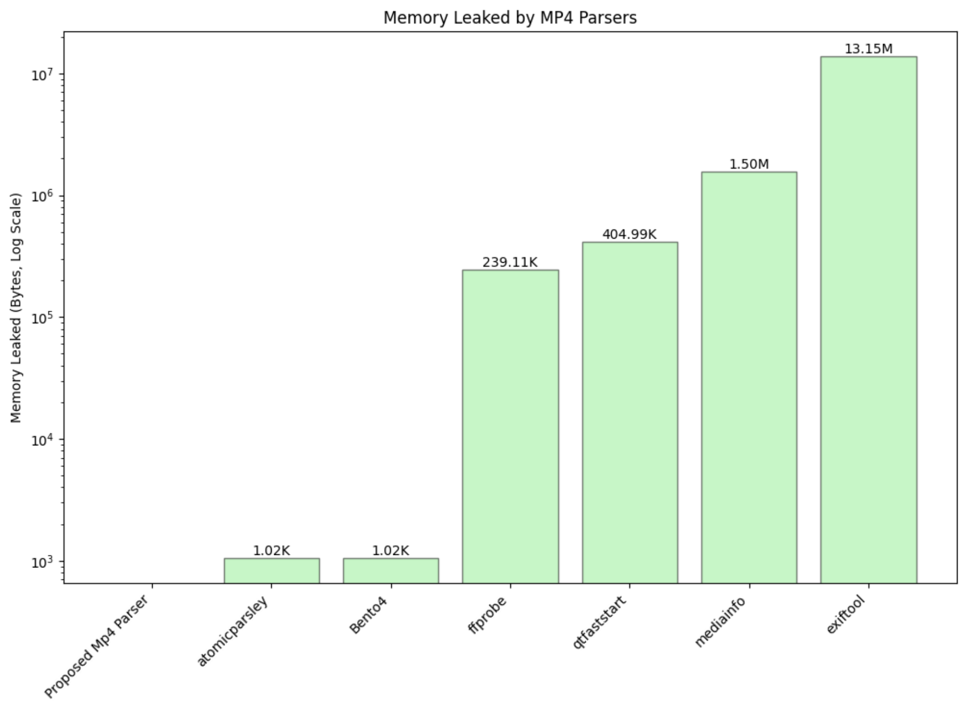


Figure 3.15: Memory Leak Statistics for MP4 Parsers using Heaptrack

Peak Heap Memory Consumption Peak memory usage by MP4 parsers is highlighted in Figure 3.16. The Proposed MP4 Parser also shows low peak heap memory consumption at 77.99 KB, which is notably lower compared to other parsers like qtfaststart (5.10 MB) and exiftool (14.94 MB). This indicates that the proposed parser maintains a lower memory footprint even at its peak usage, demonstrating effective memory management. atomicparsley (79.42 KB) and Bento4 (103.50 KB) also exhibit low peak memory consumption, but the proposed parser's performance remains superior in terms of minimizing peak memory usage.

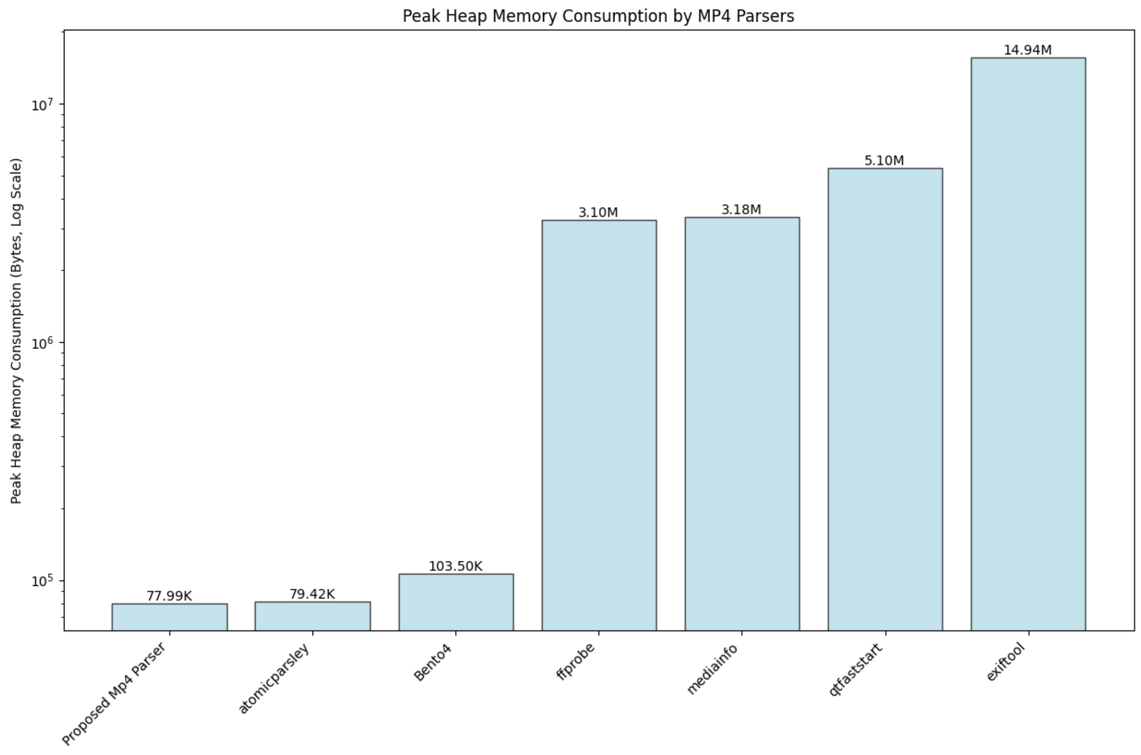


Figure 3.16: Peak Memory Usage Statistics for MP4 Parsers using Heaptrack

Overall, the proposed MP4 parser's low total heap usage, zero memory leakage, and minimal peak heap memory consumption highlight its efficient and safer memory management. This contrasts sharply with other parsers that exhibit higher memory usage and leakage, showcasing the strength of proposed parser in resource efficiency and stability.

3.4.5.2 MPEG-TS Parsers

The outcomes of the MPEG-TS parser evaluations are summarized below.

Execution Time Figure 3.17 compares execution times across MPEG-TS parsers. The proposed MPEG-TS Parser displays a notably higher execution time at 32741.20 ms. This significantly larger number is indicative of the ability of parser to recognize and parse a much wider range of metadata from the native MPEG-TS container and the KLV metadata stream. We observed much lower execution times when running our parser on an MPEG-TS file without a KLV stream.

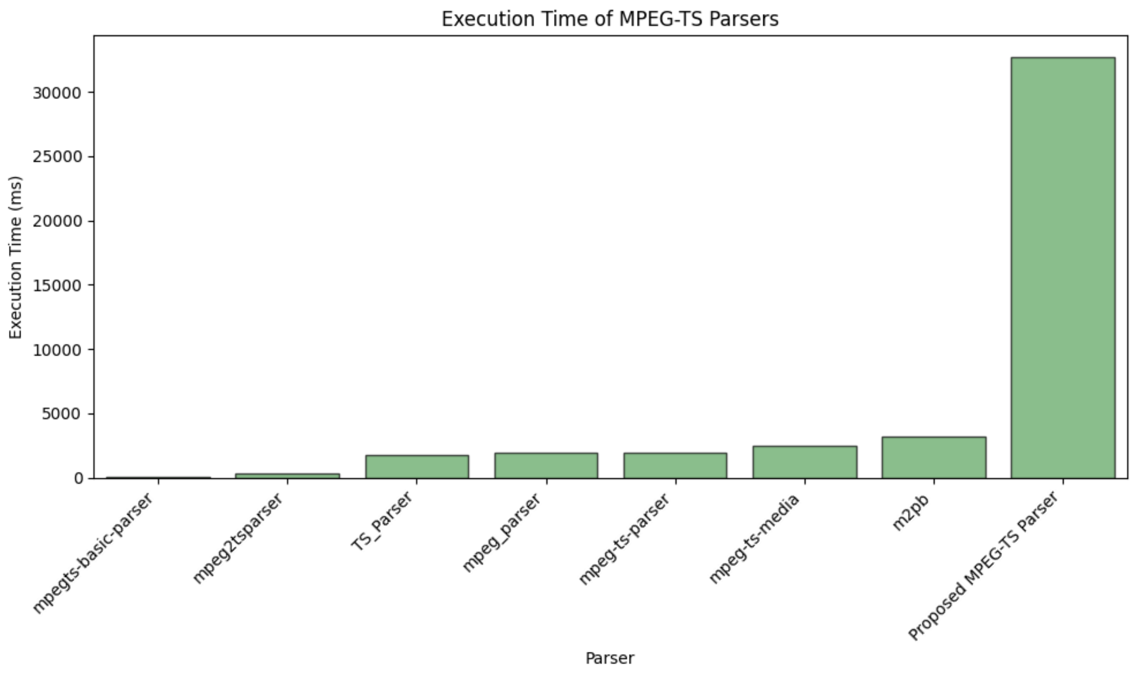


Figure 3.17: Execution Times across MPEG-TS Parsers using Hyperfine

Other parsers, such as mpeg2tsparser (360.10 ms), TS_Parser (1746.50 ms), and mpeg-ts-parser (1961.20 ms), focus on displaying only portion of metadata, leading to shorter execution times. These parsers might have been developed with intentions to parse small set of metadata, while excluding others, resulting in faster processing but less detailed output. Parsers like mpeg-ts-media (2466.50 ms) and mpegts-basic-parser (46.00 ms) do not support parsing of the third stream containing KLV metadata, which further reduces their processing scope and execution time.

Additionally, mpeg_parser (1917.60 ms) only selectively parses certain types of packets, e.g., Transport Stream (TS) packets. This selective parsing approach contributes to its faster execution time to more comprehensive parsers. In contrast, the trade-off in execution time demonstrates the capability a parser to handle complex and detailed metadata, making it an invaluable tool for those needing exhaustive parsing and metadata extraction.

CPU Usage Figure 3.18 displays CPU usage statistics for MPEG-TS parsers. Despite the Proposed MPEG-TS Parser taking longer to process a wide range of data, its CPU time usage is relatively low at 984.9 ms, indicating that it manages resources efficiently while handling extensive metadata. In contrast, parsers like m2pb (8836.44 ms) and mpeg-ts-media (9548.63 ms) exhibit much higher CPU time usage, suggesting that they demand more computational resources for processing, likely due to more complex or less optimized algorithms.

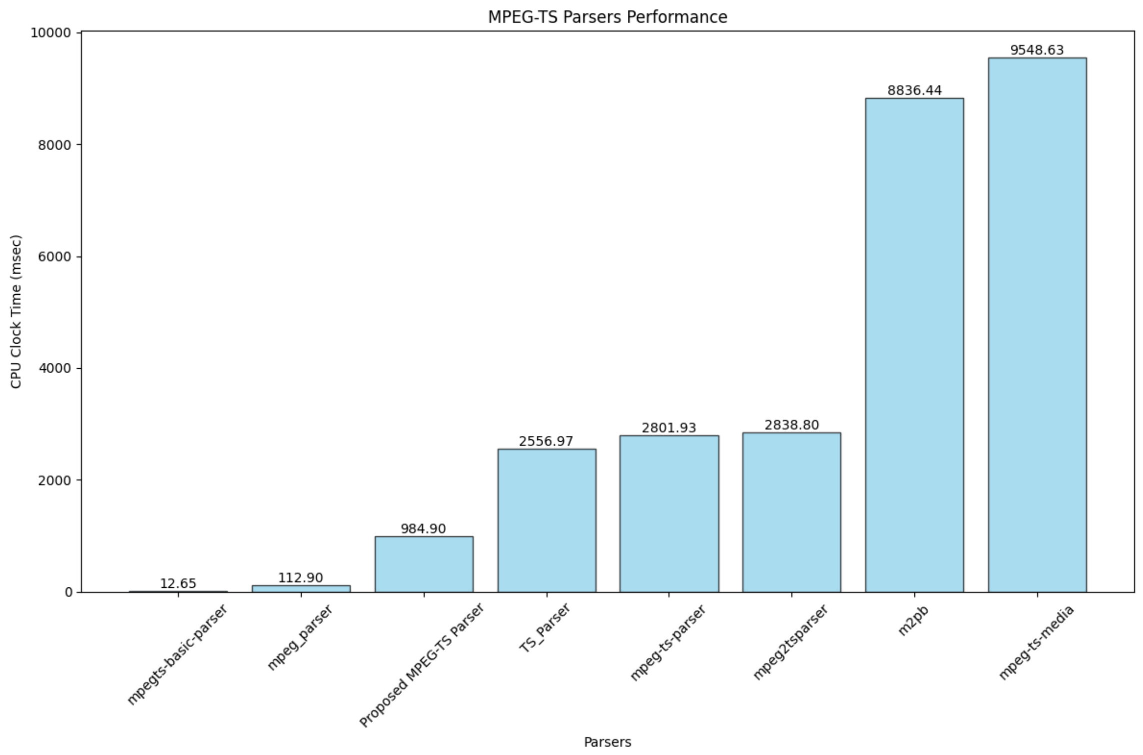


Figure 3.18: CPU Usage Statistics for MPEG-TS Parsers using Perf

Parsers such as mpeg2tsparser (2838.8 ms) and mpegts-basic-parser (12.65 ms) show varying levels of CPU efficiency. While mpeg2tsparser has moderate CPU time usage, indicating a balance between performance and resource consumption, mpegts-basic-parser stands out with the lowest CPU time usage, reflecting highly efficient processing but potentially focusing on only basic metadata. TS_Parser (2556.97 ms) and MPEG-TS-parser (2801.93 ms) fall into the middle range, with CPU time usage suggesting a trade-off between processing capabilities and efficiency.

Overall, while the proposed MPEG-TS parser has a higher execution time compared to the most efficient parsers like mpegts-basic-parser, its usage is considerably lower than the most resource-intensive parsers. This suggests that it effectively balances detailed processing with efficient resource management, making it a robust tool for handling complex MPEG-TS data without excessive CPU demand.

Memory Usage and Leaks The memory usage results for various MPEG-TS parsers highlight significant differences in their efficiency, focusing on total heap usage, memory leakage, and peak heap memory consumption. Table 3.8 presents the memory metrics for MPEG-TS parsers.

Total Heap Usage The Proposed MPEG-TS Parser shows a total heap usage of 100,099,451 bytes, which is notably lower compared to other parsers such as m2pb (2,049,947,457 bytes) and mpeg-ts-parser (762,131,084 bytes). This lower total heap usage suggests that the proposed parser is efficient in memory management, utilizing significantly less memory than more resource-intensive parsers. Parsers like mpeg2tsparser (266,296 bytes) and TS_Parser (6,064 bytes) much efficient memory usage, but the proposed parser remains superior in terms of managing Peak Memory Consumption (discussed below) with lower memory overhead.

Parsers	In Use at Exit	Total Heap Usage	Definitely Lost	Indirectly Lost	Possibly Lost	Still Reachable
Proposed MPEG-TS Parser	0 B	95.46 KB	0 B	0 B	0 B	0 B
mpeg-ts -parser	726.74 MB	726.82 MB	188 B	0 B	726.74 MB	0 B
TS_Parser	0 B	5.92 KB	0 B	59.23 KB	12.43 MB	3.00 KB
mpegts-basic -parser	386.10 KB	386.10 KB	128.04 KB	128.02 KB	0 B	130.04 KB
mpeg_parser	944 B	80.92 KB	0 B	0 B	0 B	944 B
mpeg2tsparser	0 B	260.05 KB	0 B	0 B	0 B	0 B
mpeg-ts-media	0 B	775.40 KB	0 B	0 B	0 B	0 B
m2pb	99.92 KB	1,954.98 MB	0 B	0 B	0 B	99.92 KB

Table 3.8: Memory Usage Statistics for MPEG-TS Parsers using Valgrind

Total Memory Leaked Examination of memory leaks in MPEG-TS parsers can be found in Figure 3.19. The Proposed MPEG-TS Parser excels with 0 bytes of memory leaked, indicating impeccable memory management with no residual memory left unreleased. This contrasts sharply with mpeg-ts-parser (762.05 MB), which has the highest memory leakage, suggesting substantial issues with memory management and potential inefficiencies. Other parsers, such as m2pb (103.44 KB) and mpeg2tsparser (266.30 KB), also show some memory leakage, highlighting areas where resource management could be improved.

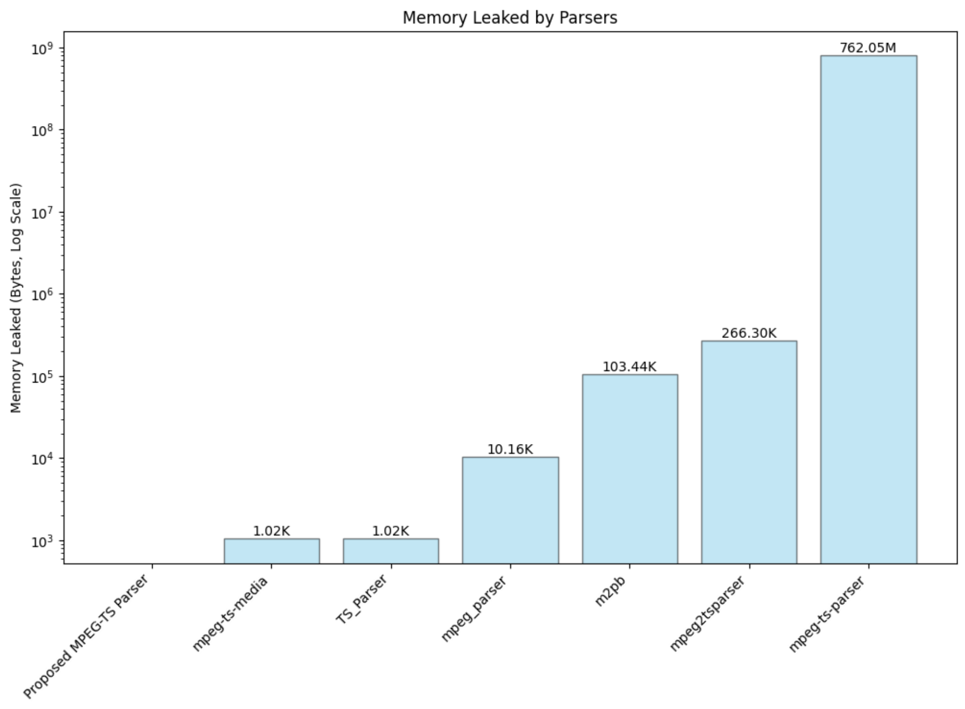


Figure 3.19: Memory Leak Statistics for MPEG-TS Parsers using Heaptrack

Peak Heap Memory Consumption Figure 3.20 showcases peak memory usage by MPEG-TS parsers. In terms of peak heap memory consumption, the Proposed MPEG-TS Parser maintains a low figure of 74.44 KB, indicating that it does not require excessive memory even at its highest usage. This is in sharp contrast to mpeg-ts-parser (762.13 MB) and m2pb (506.96 KB), which exhibit significantly higher peak memory usage. Parsers like mpeg_parser (82.86 KB) and TS_Parser (78.77 KB) also demonstrate efficient memory usage, but the proposed parser's low peak memory consumption highlights its effective memory management under peak conditions.

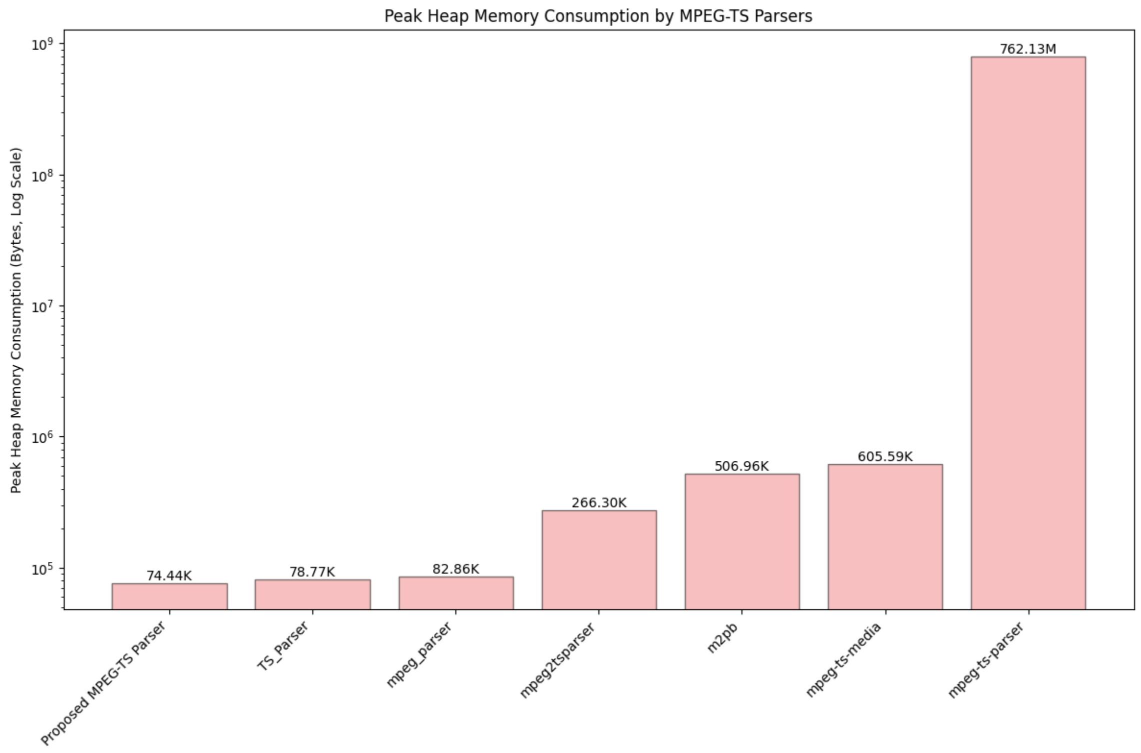


Figure 3.20: Peak Memory Usage Statistics for MPEG-TS Parsers using Heaptrack

Overall, the proposed MPEG-TS parser stands out for its efficient memory usage, low total heap usage, zero memory leakage, and minimal peak memory consumption. This contrasts with other parsers that exhibit higher memory demands and leakage, showcasing the proposed parser's strength in maintaining efficient and stable memory management.

3.4.6 Review of Known Vulnerabilities

This subsection provides an overview of known vulnerabilities found in MP4 and MPEG-TS parsers. It reviews frequency, severity and categorization of each vulnerability. By analyzing these known issues, we aim to identify potential risks and discuss the benefits of using Rust – a memory safe programming language, for parser development.

3.4.6.1 Data Preperation

The data presented in this analysis was collected from two major vulnerability databases: MITRE and NIST [72]. These databases offer comprehensive, publicly accessible information on vulnerabilities and exposures. MITRE's database focuses on the identification and categorization of vulnerabilities, while NIST provides detailed information on their severity, impact, and remediation [73].

We created a custom dataset by extracting CVEs (Common Vulnerabilities and Exposures) that specifically affected MP4 and MPEG-TS container parsers. We particularly collected information such as publish dates, CWE-ID, CVE numbers and severity. CVE numbers are unique identifiers assigned to publicly known security vulnerabilities. They serve as a standard reference for security researchers and professionals to share data across community end users.

CWEs categorize vulnerabilities into specific types of weaknesses, making it easier to understand common vulnerability patterns and prioritizing fixes based on the nature of the weakness. In our analysis, we identified the following CWEs affecting video container parsers over the past couple of decades:

3.4.6.2 Data Analysis

Vulnerability Distribution Figure 3.21 presents the trend of Vulnerability discovery over time. There is a general trend of increasing vulnerabilities over time from 2004 to 2023, with notable peaks in 2022 and 2021. Year 2022 had the highest count of vulnerabilities overall (70), with Bento4 and GPAC contributing significantly. Followed by year 2021, with a high count (64), mainly attributed to GPAC. Other parsers like Adobe Flash Player, RealNetworks

CWE-ID	Description
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-122	Heap-based Buffer Overflow
CWE-125	Out-of-bounds Read
CWE-189	Numeric Errors
CWE-190	Integer Overflow or Wraparound
CWE-191	Integer Underflow (Wrap or Wraparound)
CWE-264	Permissions, Privileges, and Access Controls
CWE-369	Divide By Zero
CWE-399	Resource Management Errors
CWE-400	Uncontrolled Resource Consumption
CWE-401	Missing Release of Memory after Effective Lifetime
CWE-404	Improper Resource Shutdown or Release
CWE-415	Double Free
CWE-416	Use After Free
CWE-476	NULL Pointer Dereference
CWE-617	Reachable Assertion
CWE-674	Uncontrolled Recursion
CWE-697	Incorrect Comparison
CWE-704	Incorrect Type Conversion or Cast
CWE-763	Release of Invalid Pointer or Reference
CWE-770	Allocation of Resources Without Limits or Throttling
CWE-772	Missing Release of Resource after Effective Lifetime
CWE-787	Out-of-bounds Write
CWE-824	Access of Uninitialized Pointer
CWE-835	Loop with Unreachable Exit Condition ('Infinite Loop')
CWE-908	Use of Uninitialized Resource
CWE-94	Improper Control of Generation of Code ('Code Injection')

Table 3.9: Common Weakness Enumeration (from mitre.org)

RealPlayer, and Apple QuickTime show vulnerabilities mainly in the earlier years. Overall, the trend demonstrate a growing issue with specific parsers, highlighting the need for improved security measures and vigilance.

While some parsers from the previous study were retained, others were excluded due to the absence of related vulnerabilities. Additional parsers included in this study aren't strictly stand alone parsers, but are embedded components of the software, such as Google Chrome, Mozilla Firefox and VLC Media Player. These parsers were included in the study because they had documented vulnerabilities, even though these libraries do not provide executable code for direct comparison. This broader approach allowed more comprehensive exploration of vulnerabilities not only in stand alone parsers, but software that integrates parsers as component.

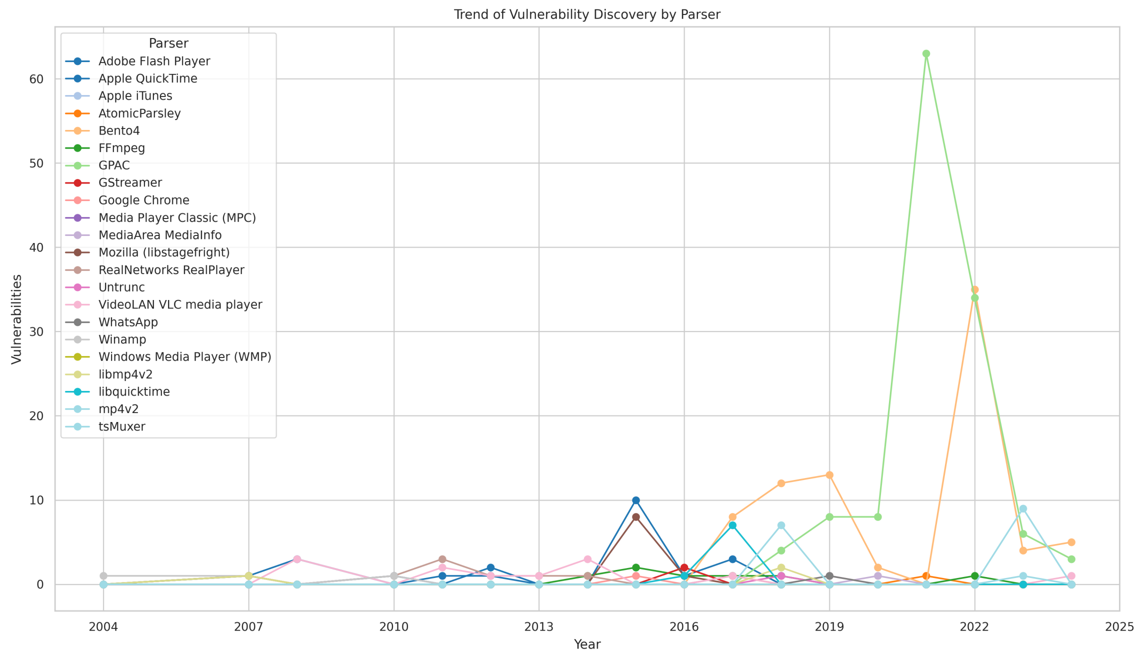


Figure 3.21: Trend of Vulnerability Discovery over Time

Prevalent CWEs in Parsers Figure 3.22 presents the distribution of CWEs across parsers. GPAC stands out as the most vulnerable parser with 126 reported issues, including 32 instances of CWE-476 and 23 of CWE-787. These highlight significant problems with memory handling and pointer management, leading to potential crashes and security exploits. It also has 16 occurrences of CWE-120, emphasizing buffer overflow and memory corruption concerns. Bento4 follows with 79 reported issues, notably 13 instances each of CWE-787 and CWE-401, and 12 of CWE-476, indicating critical memory management flaws. Adobe Flash Player has 17 vulnerabilities, primarily CWE-787 (10 instances), pointing to out-of-bounds write issues.

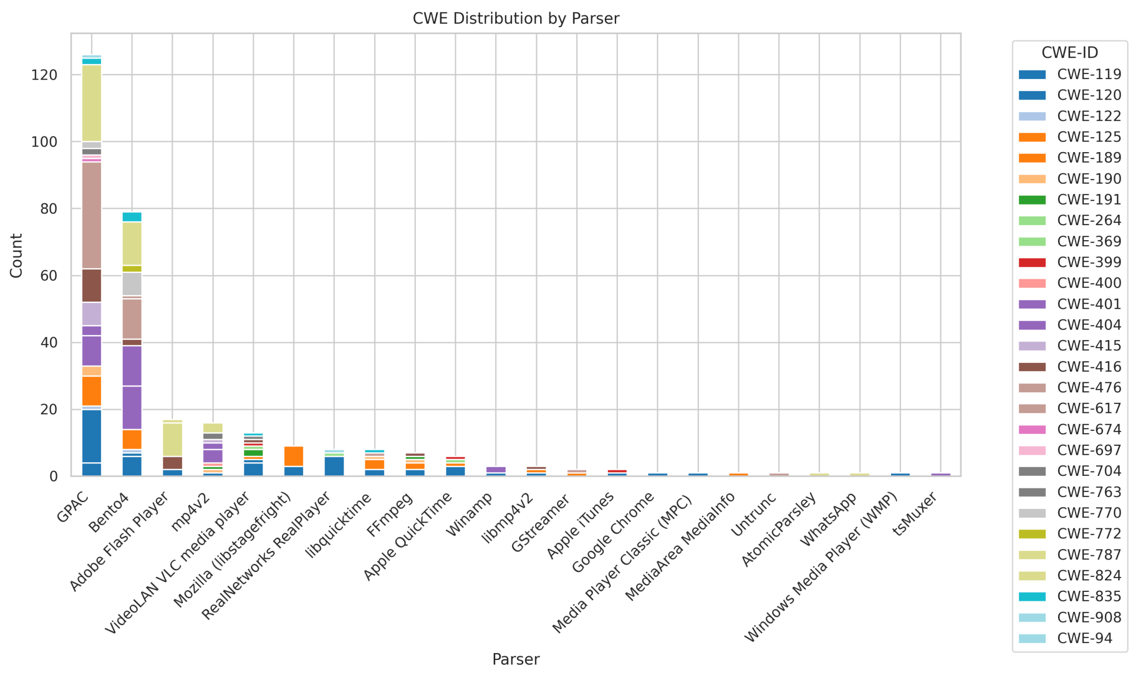


Figure 3.22: CWEDistribution by Parser

Mid-level parsers like mp4v2, VLC media player, and Mozilla's libstagefright also show vulnerabilities. mp4v2 has 16 issues, mainly CWE-401 (4 instances). VLC media player has

13 issues, with CWE-119 (4 instances) related to buffer management. Libstagefright has 9 issues, predominantly CWE-189 (6 instances), indicating data handling problems. Figure 3.23 ranks the top CWE categories by severity.

The trend of vulnerability discovery over time by parser is depicted in Figure 3.23.

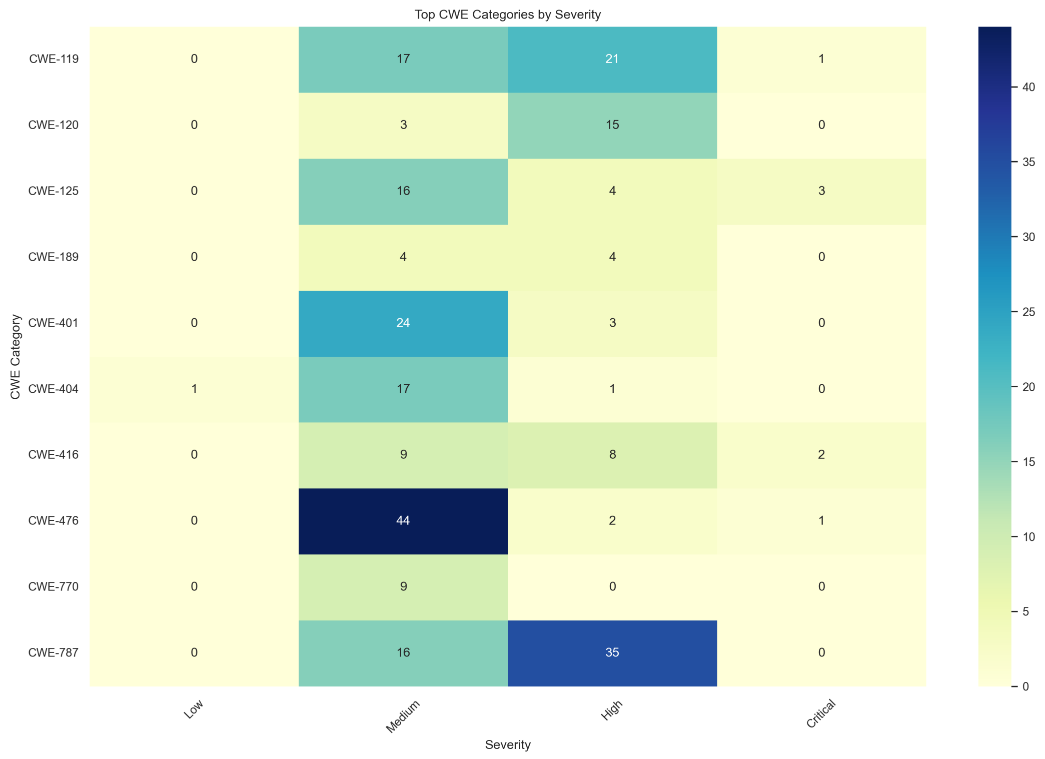


Figure 3.23: Top CWE Categories by Severity

Lower-level parsers, including libmp4v2, Winamp, and Apple iTunes, show fewer vulnerabilities but still pose risks. Libmp4v2 has 3 issues, including one instance of CWE-119. Winamp has 3 vulnerabilities, with 2 instances of CWE-404. Apple iTunes has 2 issues, with CWE-399 being notable. Other parsers like GStreamer, WhatsApp, and AtomicParsley each have a single reported vulnerability, indicating that while they are less prone to issues, they are not entirely risk-free.

In summary, memory management and buffer handling issues are prevalent across many video parsers, emphasizing the need for improved security measures.

Severity Distribution Figure 3.24 illustrates the severity distribution by parser. A total of 307 vulnerabilities were identified, with Medium-level severity vulnerabilities comprising 60.26% of the total. GPAC and Bento4 account for a significant portion of these Medium severity vulnerabilities, with GPAC alone contributing nearly 44%. This indicates a need for focused attention on these parsers to address the numerous medium-level threats.

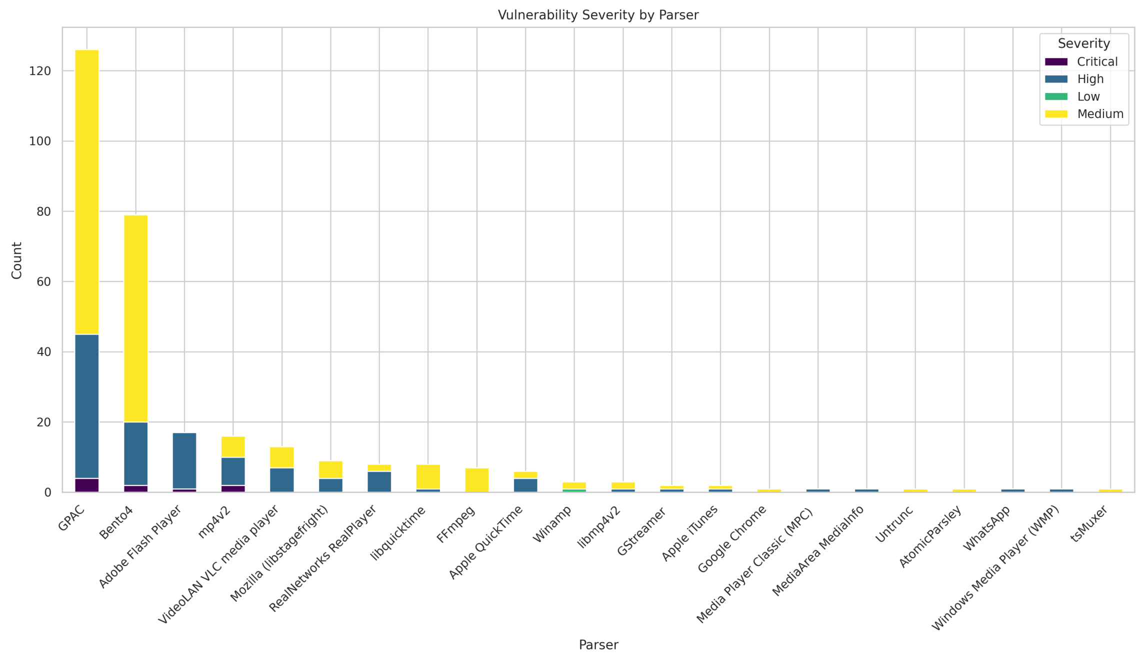


Figure 3.24: Severity Distribution by Parser

High severity vulnerabilities make up 36.49% of the total, again heavily concentrated in GPAC and Bento4. GPAC is particularly notable with 41 High severity vulnerabilities, representing over one-third of the total. Bento4 and Adobe Flash Player also require significant security enhancements due to their high severity vulnerabilities. Critical severity vulnerabilities, though fewer, are mainly found in GPAC, mp4v2, and Bento4, highlighting the urgent need for remediation in these parsers. The single Low severity vulnerability in Winamp indicates the generally high-risk nature of the vulnerabilities.

In conclusion, this analysis demonstrates the urgent need for robust security measures. A comprehensive security assessment and mitigation strategy should be implemented across all parsers to enhance security and protect against potential exploits.

3.4.6.3 Recommendations

The vulnerabilities present in video container parsers pose significant security risks that attackers can exploit for malicious purposes. For example, Denial of Service (DoS) attacks could cause a parser to consume excessive resources or crash. One example is found in CVE-2017-15186, where attackers can leverage specially crafted container files to cause the parser to crash, leading to service disruptions. The double free vulnerability is exploited when a program attempts to free the same memory block twice, leading to undefined behavior, program crashes, or security vulnerabilities. In addition to DoS risks, arbitrary remote code execution vulnerabilities are also critical concerns. An example is CVE-2022-2566, a heap out-of-bounds memory write in FFmpeg. This flaw allows an attacker to execute arbitrary code by crafting a malicious MP4 file that manipulates memory allocation incorrectly. These examples demonstrate that vulnerabilities in parsers are not merely theoretical: they create concrete pathways for attackers to disrupt services or gain unauthorized access to systems, emphasizing the need for robust security measures in media parsing implementations.

The traditional parsers examined in this study were primarily written in C and C++, which are known for their performance but lack memory safety features, making them prone to vulnerabilities such as buffer overflows, use-after-free errors, and null pointer dereferencing. These vulnerabilities, as highlighted by the prevalence of CWEs such as CWE-119 (Improper Restriction of Memory Buffer), CWE-416 (Use After Free), and CWE-476 (Null Pointer Dereference) in our dataset, can lead to severe security exploits. In contrast, our parsers were written in Rust, a language designed to offer memory safety without compromising performance. Rust's ownership model ensures that when using the default safe mode, variables are safely managed, preventing many of the common memory vulnerabilities found in C and C++ parsers. For example, Rust's compiler:

- prevents null pointer dereferencing, avoiding vulnerabilities like CWE-476.
- enforces bounds checking to safeguard against buffer overflows (CWE-787, CWE-119).
- manages memory allocation and de-allocation strictly, preventing use-after-free and double-free errors (CWE-416, CWE-415).
- includes default checks for integer overflow, reducing risks tied to CWE-190.

While Rust significantly reduces the likelihood of memory-related vulnerabilities, it does not completely eliminate all risks. Logical errors or improper handling of data input can still lead to logical vulnerabilities, but by leveraging Rust's memory safety features, developers can greatly reduce the risk of severe memory-related vulnerabilities.

Chapter 4

Full-Motion Metadata Redaction

Metadata plays a crucial role in organizing, managing, and accessing information across various digital file formats. Full-motion video (FMV) is a specialized video container format that combines geospatial metadata with standard audio-visual data streams. Public disclosure of such video without appropriate redaction may allow a skilled analyst with knowledge of FMV structure, to extract the underlying KLV metadata, illustrating details of a confidential flight path during an ISR mission, the drone registration and ownership details, and more. While FMV technology significantly advances the ISR capabilities, the exposure potential for such metadata to expose sensitive information has prompted extensive research into methods for removing, redacting, or anonymizing such data. The redaction of visual information is beyond the scope of this chapter. However, we begin this research by exploring several object detection techniques, as discussed in Chapter 5, which serve as the initial step in the visual redaction process.

4.1 Metadata in Different Media Types

Metadata in media files can be divided into two categories: *fine-grained* and coarse-grained. Fine-grained metadata provides more information about the building blocks that constitute the internal structure of a file, such as the specific frames within a video or individual sections within a document, while coarse-grained metadata offers general information about the file as a whole, such as the file's owner or creation date.

In video files, fine-grained metadata includes frame-specific details such as timestamps, geospatial data, and embedded subtitles. For example, KLV metadata in FMV videos provides detailed information like Unix Timestamps, Longitude, Latitude and unique drone identifiers (e.g., Platform Tail Number). Coarse-grained metadata includes information that applies to the file as a whole, such as video codec, resolution, and frame rate. In consumer-grade drones, examples of coarse-grained metadata include pilot details, serial numbers, and drone IDs.

Fine-grained metadata in documents includes information such as user comments or notes, responses made to the comments, and details of the individuals who made each comment on a particular page. Coarse-grained metadata might include the file's creation date, last modified date, and file ownership details. Although this metadata is generally less sensitive, it can still pose privacy risks in legal or corporate environments.

4.2 Existing Techniques for Metadata Redaction

Various approaches have been developed for the automated redaction of sensitive metadata in different media file formats. This literature review focuses on tools and techniques developed to address the privacy risks associated with sensitive metadata across different file types, including images, documents, and multimedia files. We analyze these tools, with respect to their ability to handle fine versus coarse metadata and compare their effectiveness in different file types.

4.2.1 Metadata Redaction in Images

One of the most prominent areas of concern regarding metadata privacy is digital imaging, where metadata such as EXIF data can include sensitive information like GPS coordinates, ownership details, and timestamps.

A Python-based tool introduced in [74] strips all metadata from images or specifically targets sensitive information, such as GPS data, before sharing images on social media. However, the authors raise concerns about unexplained increases in image size after metadata removal, suggesting the need for further optimization for large datasets.

Another approach discussed in [75] allows users to selectively remove or retain metadata when sharing photos. Users can choose to strip specific metadata elements like location data while preserving others. The flexibility extends to removing metadata from all photos, groups of photos, or individual photos .

ImageMagick is a software suite for creating, editing, and converting images [76]. ImageMagick mainly supports stripping coarse metadata: The "convert" utility with "-strip" flag can be used to remove metadata coarse metadata from single image. For bulk removal, "morgify" utility with "-strip" flag can be used.

exiv2 is a C++ library and command-line utility for managing image metadata [77]. It handles multiple image file types and provides robust metadata read and write capabilities.

We can use exiv2 with "rm" or "delete" flag for general metadata removal, and "del" for removing specific tags.

Pillow is a Python Imaging Library (PIL) fork that enables opening, manipulating, and saving various image file formats [78]. The "PIL.Image.Exif" be used to remove specific exif metadata, provided user is able to supply appropriate tag has number.

ExifTool is a comprehensive Perl library and command-line application for reading, writing, and editing metadata in various file types [79]. The "-all= tag" flag removes all metadata, and "-Tag=" targets specific metadata tags.

A method described in [80] secures image metadata using AES-128 encryption before sharing. The encrypted metadata is embedded back into the image and can later be decrypted without affecting the appearance of the image.

Another technique outlined in [81] focuses on online social networks, encrypting sensitive attributes in tweets, sensitive regions in images and memes, and anonymizing metadata. The approach, while innovative, faces challenges in scaling and applicability to real-world scenarios due to reliance on synthetic data.

4.2.2 Metadata Redaction in Documents

Documents, particularly those created in word processing and PDF formats may contain extensive metadata that can reveal sensitive information.

The Metadata Anonymization Toolkit (MAT) aggressively removes metadata from various file formats, including Microsoft Word, Excel, and PDF, using a whitelist approach that retains only essential metadata [82]. It can either remove all metadata with no additional tags required and provides a -L or –lightweight option for partial metadata removal.

PDF-mangler is a Python library developed to anonymize PDF content and metadata, particularly for submitting reproducible bug reports. It uses the PikePDF library to replace text, blur images, and selectively remove sensitive metadata, though it struggles with embedded JavaScript and complex image formats [83].

AnonymousXL is a tool designed to anonymize spreadsheet metadata. It adjusts numerical metadata to random values, converts dates random dates within the valid range, and replaces text with random characters of the same size, though it may introduce errors due to its adjustment methods [84].

dmeta is a Python package for removing metadata from Microsoft Office .DOCX files [85]. It offers a simple command-line interface for clearing and updating metadata, enhancing privacy and security in document handling. The "clear" flag removes basic metadata, while "clear –all" performs a comprehensive cleanup.

OpenPyXL is a Python library for reading and writing Excel (xlsx or xlsm) files [86]. It supports data manipulation, chart creation, and styling in Excel files. The "properties" module from "openpyxl.workbook" can be used to extract, update, or delete metadata properties. OpenPyXL is effective for document metadata and content redaction.

PDFtk (PDF Toolkit) is a versatile tool for manipulating PDF documents, offering functionalities like merging, splitting, encrypting, and decrypting PDFs [87]. It is useful for batch processing via a command-line interface. A combination of dump_data, and update_info flag from PDFtk, piped into a sed command may allow deleting either complete stream of metadata or specific metadata items.

4.2.3 Video Metadata Redaction

Multimedia files, including videos recordings, often contain embedded metadata that can reveal sensitive information, such as recording locations, timestamps, and device identifiers. Majority of literature for "metadata in videos" is centered around forensic applications, where metadata is used to track alterations in videos. Although we could not find any relevant tools or techniques that offer fine-grained metadata removal or anonymization in MPEG-TS videos, much less KLV metadata, there are tools available to redact coarse metadata from MPEG-TS videos.

FFmpeg is a cross-platform solution for recording, converting, and streaming audio and video [88]. It supports numerous codecs and formats, making it essential for multimedia processing. FFmpeg includes capabilities to strip metadata from image, audio and video files. Users can utilize -metadata tag="value" to set specific metadata and -map_metadata for stripping the complete stream of metadata.

In [89], an approach for video source identification such as camera device or smartphone using metadata in MP4 files is presented. The method employs a 'block list' to filter out common metadata atoms that do not contribute to source identification, introduces a new *path type* data structure representing the positioning boxes from root to leaf atoms, and uses a matching algorithm to evaluate the distinctiveness of the *path types* against known *paths type* data structures. The approach leveraging existing knowledge of tree structures from various manufactures. Despite its high accuracy, the method has limitations. Refining the block list is a complex task requiring extensive knowledge of the MP4/QuickTime specification.

A novel tamper-protection system for MP4 files is proposed in [90] where perceptual hashes of I-frames are embedded into the audio stream. The hash of the audio stream is then further embedded into the synchronization information, specifically the STSS atom, of the MP4 file. They use FFmpeg to extract I-frames, pHash library for hashing, and Mp4Box from GPAC for embedding and multiplexing. However, the authors recognize that the system is vulnerable to transmission errors, as even a single bit flip can lead to authentication failure.

Finally, [91] introduces a method to verify the integrity of AVI files and identify the editing software used by analyzing the unique field data structures left by various video editing tools. The authors first manipulate several AVI video files using different editing software, then employ a custom parser to extract a comprehensive list of metadata, including artifacts from the editing process, which is stored in a signature database. The method automatically detects manipulations in a given video by generating a new signature for the input video and comparing it with the database, allowing for the categorization of the video based on the editing software used. However, the authors note that the method may be less effective for videos re-rendered by online platforms and is currently confined to the AVI format, with further research required to extend its application to other video formats.

4.2.4 Metadata Redaction in other file formats

Mutagen is a Python module for handling audio metadata. It allows for reading and writing metadata, providing a pure Python solution for managing audio files [92]. Typically, metadata is removed using functions like audio.delete() in Python scripts.

StripZIP is a command-line tool for removing metadata from ZIP archives, enhancing privacy and security [93]. It focuses on stripping unnecessary metadata, making it useful for secure file distribution. The -X or –no-extra flags ensure complete metadata removal, while standard execution performs normal removal.

4.3 Comparative Analysis of Metadata Redaction Tools

Table 4.1 categorizes existing tools based on their ability to redact content and metadata tags, both fine-grained and coarse-grained across different file types. This work is the only known redaction tool with the unique capability to redact both coarse and fine metadata from full-motion videos, a feature not offered by any other tools on the list.

Type	Format	Content Redaction	Metadata Redaction	
			Coarse	Fine
Image	JPEG	[88, 76, 78]	[79, 77, 88, 76, 94, 78]	—
	PNG	[88, 76, 78]	[79, 77, 88, 76, 94, 78]	—
	BMP	[88, 76, 78]	[88, 76, 94, 78]	—
	WebP	[88, 76, 78]	[79, 77, 88, 76, 78]	—
Video	MP4	[88]	[88, 94], This Work	[88], This Work
	AVI	[88]	[88, 94]	[88]
	MPEG-TS	[88]	[88], This Work	[88], This Work
	FMV	n/a	This Work	This Work
Audio	FLAC	[88, 92]	[88, 94, 92]	[88]
	MP3	[88, 92]	[88, 94, 92]	[88]
	AAC	[88, 92]	[88, 92]	[88]
	ID3	[92]	[92]	[92]
Document	PDF	[87]	[79, 94]	[79, 87]
	DOCX	—	[94, 85]	—
	XLSX	[86]	[94]	[86]
	PPTX	—	[94]	—
ZIP	TAR	—	[94]	—
	TAR.GZ	—	[94]	—
	XZ	—	[94]	—
	ZIP	—	[94, 93]	—

Table 4.1: Comparison of Existing Parsers with Proposed Redaction Tool

4.4 Challenges in Metadata Redaction

Automated metadata redaction faces significant challenges, mainly in maintaining the usefulness of the file while protecting privacy. Many existing tools fall short in fine-grained redaction, often removing all metadata and compromising the usability of the file. For example, if a tool removes all metadata from a image to protect privacy, it also deletes valuable information like the camera model and date, reducing the photo's usefulness.

Another challenge arises when redaction results in the metadata growing beyond its intended size, creating extra space in the file. Fixing this requires resizing and rewriting the entire file, which can be time-consuming, especially for large files.

Most widely used redaction tools do not support a broad range of file types and metadata formats. Fine-grained redaction needs a deep understanding of different container formats, requiring high technical expertise.

Lastly, there are no standardized metrics to evaluate the effectiveness of metadata redaction tools. Future research should focus on developing clear benchmarks to measure how well these tools balance privacy protection with maintaining file utility.

4.5 Proposed Redaction Approach for KLV Metadata

The Motion Imagery Standards Board, abbreviated as MISB, is an organization that develops standards for motion imagery systems within the U.S. Department of Defense (DoD) and the National Geospatial-Intelligence Agency (NSG) [95]. The MISB ST 0601.17 and STANAG 4906 are two important standards utilized in the US defense sector for remote sensing applications [50, 96]. To be compliant with the MISB ST 0601.17 specification, a full-motion video imagery must be digitally produced using a combination of MPEG-TS container format for storage, either H.264 or H.265 as video codec and hold KLV metadata. Unmanned aircraft systems (UAS) produce full-motion video by integrating metadata captured through sensor systems at regular intervals of one second. Each FMV frame is given a universal precision timestamp to synchronize the audio-visual streams with the geospatial metadata. We used MISB-compliant FMV samples, some provided by ArcGIS website and some extracted from the footage captured by our DJI mini 2 drone, produced via telemetry logs, and the ArcGIS FMV multiplexer.

4.5.1 Architecture

The proposed metadata redaction technique is illustrated in figure 4.1 and consists of four key components: FMV ingestion, FMV parsing, KLV de-muxing, KLV redaction, and KLV re-muxing. The arrows in figure indicate the path across the processing pipeline the input video goes throughout the metadata redaction process.

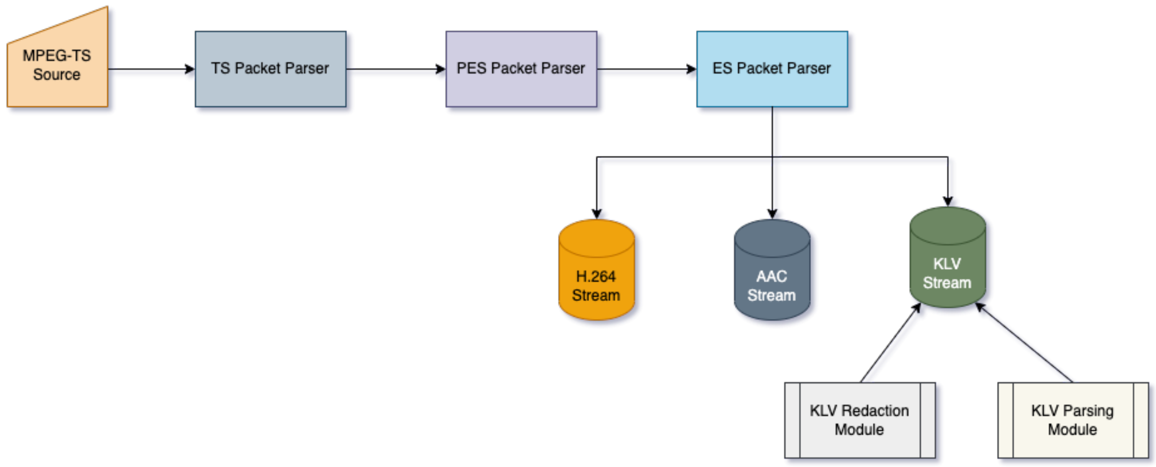


Figure 4.1: KLV redaction approach

The pre-processing of the video file for parsing is part of the KLV ingestion procedure, which entails determining if the input file has embedded geospatial metadata. It achieves this by partial parsing to verify the existence of the MPEG-TS container and associated KLV metadata stream. Upon ingestion of FMV Video into the redaction system, the parser retrieves the complete internal structure of the MPEG-TS container, including the addresses of audio, video, and KLV stream packets.

De-muxing includes unpacking audio, video, and metadata streams utilizing the knowledge of offset addresses of matching packets. Geospatial metadata stream is encoded using the BER encoding standard. The redaction module conducts byte-level redaction to *value* bytes, while other header data, such as the key and length bytes, are kept to preserve the TLV format. For string values, we substitute, the *value* with 'X' characters of the same length. Integer values are replaced with zeros. The re-muxing stage repackages data using the redacted values. The result of this redaction process can be observed in the hex dump presented in Figure 4.2. The Universal Local Set Key, which is a fixed-size key, marks the beginning of a KLV packet, with the start-code prefix 0x000001FC signaling its initiation. The KLV packet shown includes a value field of 241 bytes, clearly identifiable at offset 0xF1.

Listing 4.1 shows the pseudocode for parsing and redaction.The implementation was developed in Rust without relying on any third-party crates or libraries. Rust was chosen for its performance and safety features, which are vital in video processing systems due to the complexity of video rendering and the security requirements of defense applications.

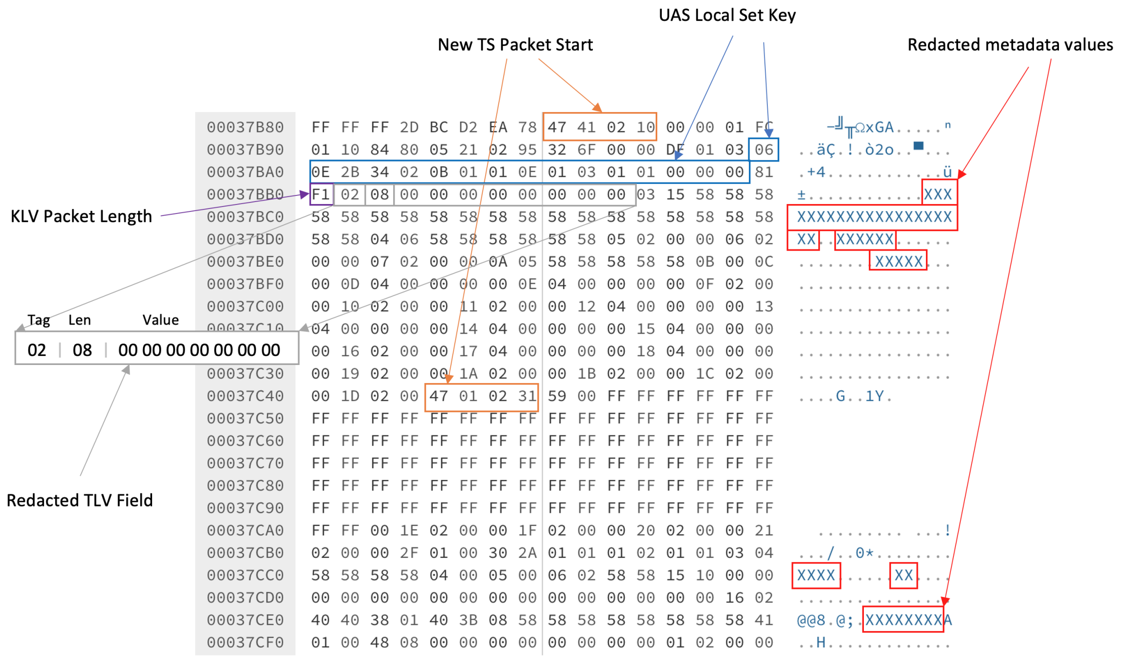
```
For index=0 To Len(FMVfile) Do
  TSBuf = FMVfile.read(index + TS_HEADER_LENGTH)
  While TSBuf[0] != SYNC_BYTE Do
    TSHeader, TSPayload = ParseTS(TSBuf)
    IF TSHeader.AFC == AFC_ONLY || AFC_WITH_PAYLOAD Then
      HandleAFC(TSPayload)
    Endif

    //De-muxing-Stage
    IF TSHeader.PUSI == 1 Then
      PESHeader, PESPayload = ParsePES(TSPayload)
        IF PESHeader.PID == VIDEO_PID Then
          ESVideoPacket = ParseES(PESPayload)
        ElseIf PESHeader.PID == KLV_PID Then
          ESKLVPacket = ParseES(PESPayload)
        ElseIf PESHeader.PID == AUDIO_PID Then
          ESAudioPacket = ParseES(PESPayload)
        Else print error
    Endif

    //Accessing, redacting and re-muxing KLV Packet
    IF ESKLVPacket [0..KLV_KEY_LENGTH] = ULKey Then
      Key, KLength, KValue = KLVParser (ESKLVPacket)
      IF Value[1] = FIRST_TAG Then
        Tag, TLength, TValue = TLVParser (Value)
        While Tag != NULL Do
          KValue[TLength] = TLVRedact(TValue)
        Endwhile
      Endif
    Endif

```

Listing 4.1: Pseudo code for KLV Redaction Algorithm

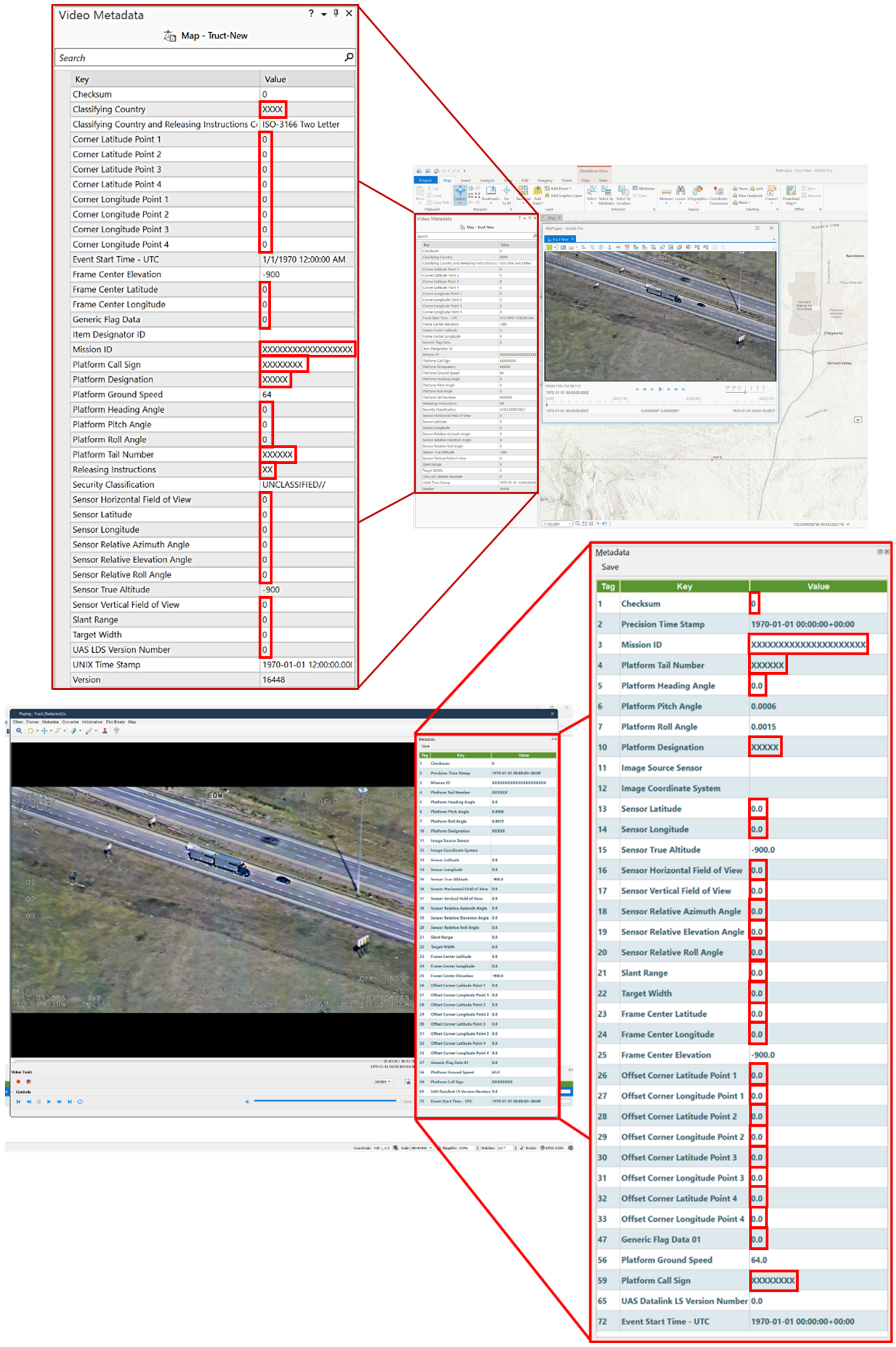


```
00037B80  FF FF FF 2D BC D2 EA 78 47 41 02 10 00 00 01 FC   ─╜╥ΩxGA.....ⁿ
00037B90  01 10 84 80 05 21 02 95 32 6F 00 00 DF 01 03 06  ..äÇ.!.ò2o..▀...
00037BA0  0E 2B 34 02 0B 01 01 0E 01 03 01 01 00 00 00 81  .+4............ü
00037BB0  F1 02 08 00 00 00 00 00 00 00 00 03 15 58 58 58  ±............XXX
00037BC0  58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
00037BD0  58 58 04 06 58 58 58 58 58 58 05 02 00 00 06 02  XX..XXXXXX......
00037BE0  00 00 07 02 00 00 0A 05 58 58 58 58 58 0B 00 0C  ........XXXXX...
00037BF0  00 0D 04 00 00 00 00 0E 04 00 00 00 00 0F 02 00  ................
00037C00  00 10 02 00 00 11 02 00 00 12 04 00 00 00 00 13  ................
00037C10  04 00 00 00 00 14 04 00 00 00 00 15 04 00 00 00  ................
00037C20  00 16 02 00 00 17 04 00 00 00 00 18 04 00 00 00  ................
00037C30  00 19 02 00 00 1A 02 00 00 1B 02 00 00 1C 02 00  ................
00037C40  00 1D 02 00 47 01 02 31 59 00 FF FF FF FF FF FF  ....G..1Y.
00037C50  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00037C60  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00037C70  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00037C80  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00037C90  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00037CA0  FF FF 00 1E 02 00 00 1F 02 00 00 20 02 00 00 21    ......... ...!
00037CB0  02 00 00 2F 01 00 30 2A 01 01 01 02 01 01 03 04  .../..0*........
00037CC0  58 58 58 58 04 00 05 00 06 02 58 58 15 10 00 00  XXXX......XX....
00037CD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 16 02  ................
00037CE0  40 40 38 01 40 3B 08 58 58 58 58 58 58 58 58 41  @@8.@;.XXXXXXXXA
00037CF0  01 00 48 08 00 00 00 00 00 00 00 00 01 02 00 00  ..H.............
```

Figure 4.2: Details of redacted KLV packet

4.5.2 Results

ArcGIS Pro and QGIS are the industry, standard tools for playing MISB-compliant full-motion videos. Figure 4.3 showcase the compatibility of redacted version FMV with these tools. As seen in the metadata panel, integer values are replaced with zeros, and string metadata values are substituted with a series of 'X' characters to indicate redaction. In this case, the footprint of geo-spatial metadata cannot be seen on the map view, due the GIS metadata being redacted.



Video Metadata

Map - Truct-New

Key	Value
Checksum	0
Classifying Country	XXXX
Classifying Country and Releasing Instructions C	ISO-3166 Two Letter
Corner Latitude Point 1	0
Corner Latitude Point 2	0
Corner Latitude Point 3	0
Corner Latitude Point 4	0
Corner Longitude Point 1	0
Corner Longitude Point 2	0
Corner Longitude Point 3	0
Corner Longitude Point 4	0
Event Start Time - UTC	1/1/1970 12:00:00 AM
Frame Center Elevation	-900
Frame Center Latitude	0
Frame Center Longitude	0
Generic Flag Data	0
Item Designator ID	
Mission ID	XXXXXXXXXXXXXXXXXX
Platform Call Sign	XXXXXXXX
Platform Designation	XXXXX
Platform Ground Speed	64
Platform Heading Angle	0
Platform Pitch Angle	0
Platform Roll Angle	0
Platform Tail Number	XXXXXX
Releasing Instructions	XX
Security Classification	UNCLASSIFIED//
Sensor Horizontal Field of View	0
Sensor Latitude	0
Sensor Longitude	0
Sensor Relative Azimuth Angle	0
Sensor Relative Elevation Angle	0
Sensor Relative Roll Angle	0
Sensor True Altitude	-900
Sensor Vertical Field of View	0
Slant Range	0
Target Width	0
UAS LDS Version Number	0
UNIX Time Stamp	1970-01-01 12:00:00.00
Version	16448

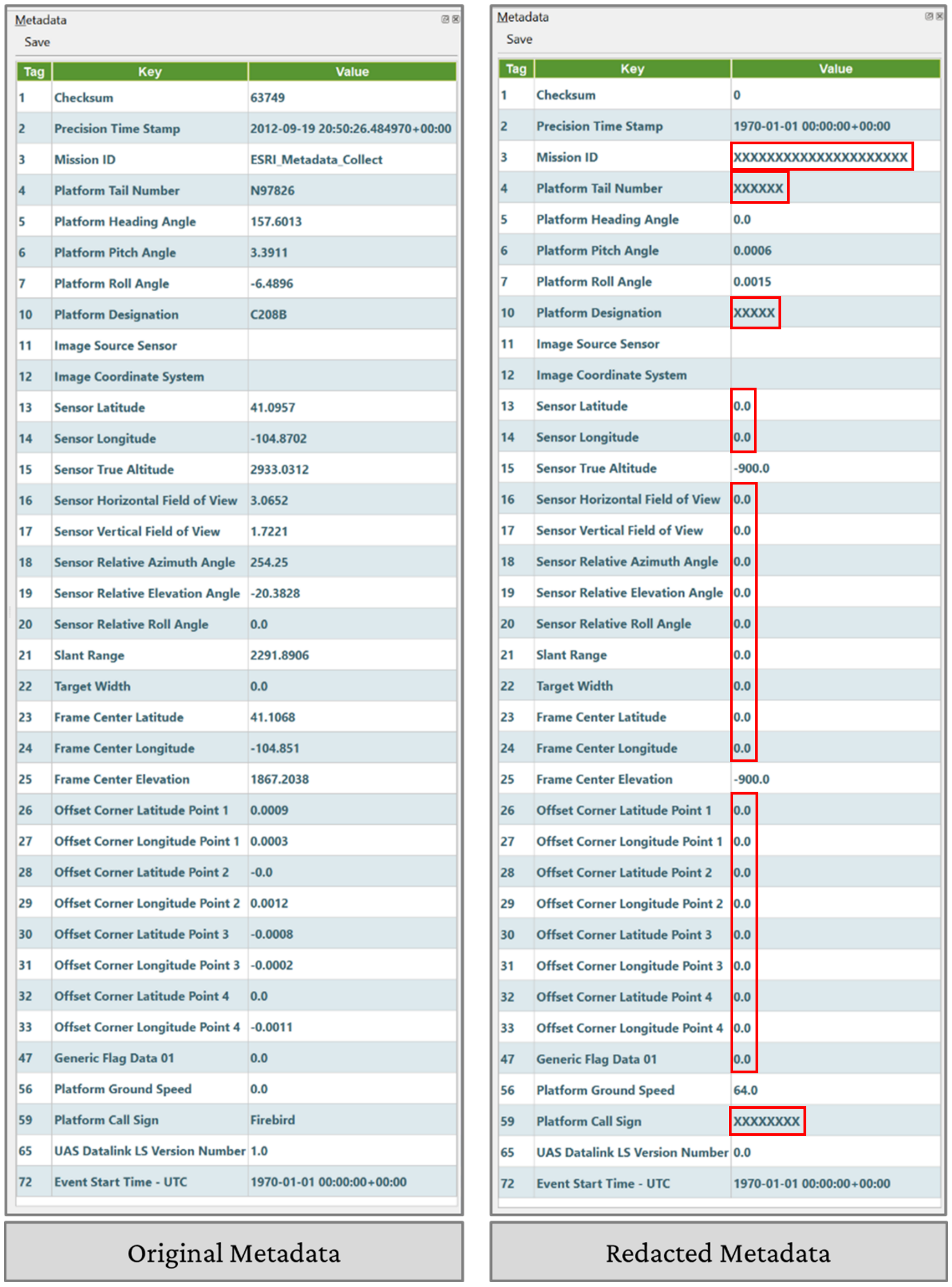
Metadata

Save

Tag	Key	Value
1	Checksum	0
2	Precision Time Stamp	1970-01-01 00:00:00+00:00
3	Mission ID	XXXXXXXXXXXXXXXXXXXX
4	Platform Tail Number	XXXXXX
5	Platform Heading Angle	0.0
6	Platform Pitch Angle	0.0006
7	Platform Roll Angle	0.0015
10	Platform Designation	XXXXX
11	Image Source Sensor	
12	Image Coordinate System	
13	Sensor Latitude	0.0
14	Sensor Longitude	0.0
15	Sensor True Altitude	-900.0
16	Sensor Horizontal Field of View	0.0
17	Sensor Vertical Field of View	0.0
18	Sensor Relative Azimuth Angle	0.0
19	Sensor Relative Elevation Angle	0.0
20	Sensor Relative Roll Angle	0.0
21	Slant Range	0.0
22	Target Width	0.0
23	Frame Center Latitude	0.0
24	Frame Center Longitude	0.0
25	Frame Center Elevation	-900.0
26	Offset Corner Latitude Point 1	0.0
27	Offset Corner Longitude Point 1	0.0
28	Offset Corner Latitude Point 2	0.0
29	Offset Corner Longitude Point 2	0.0
30	Offset Corner Latitude Point 3	0.0
31	Offset Corner Longitude Point 3	0.0
32	Offset Corner Latitude Point 4	0.0
33	Offset Corner Longitude Point 4	0.0
47	Generic Flag Data 01	0.0
56	Platform Ground Speed	64.0
59	Platform Call Sign	XXXXXXXX
65	UAS Datalink LS Version Number	0.0
72	Event Start Time - UTC	1970-01-01 00:00:00+00:00

Figure 4.3: ArcGIS and QGIS Desktop displaying redacted KLV metadata

Figure 4.4 compares the redacted metadata frame with the un-redacted one, with each column representing Tag, Key, and Value metadata items in accordance with the MISB standard.



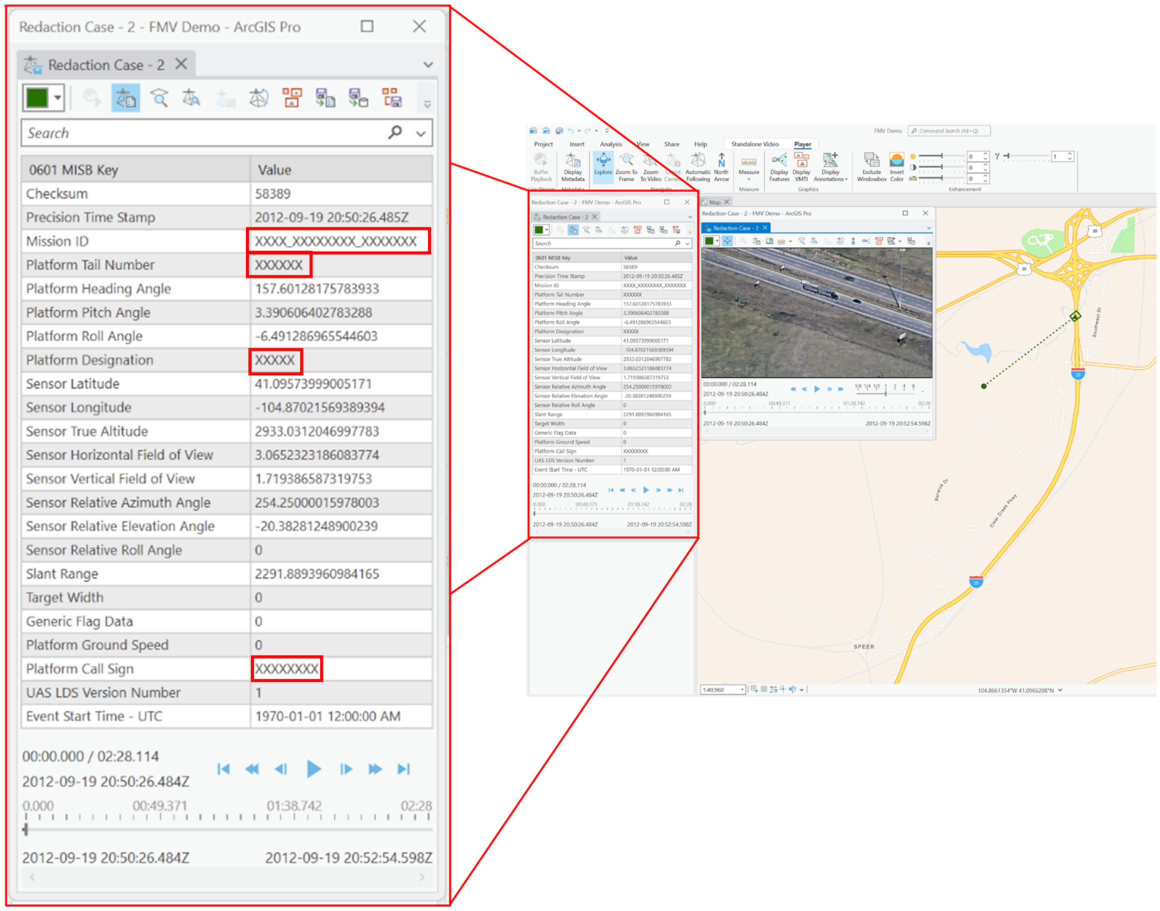
Original Metadata

Tag	Key	Value
1	Checksum	63749
2	Precision Time Stamp	2012-09-19 20:50:26.484970+00:00
3	Mission ID	ESRI_Metadata_Collect
4	Platform Tail Number	N97826
5	Platform Heading Angle	157.6013
6	Platform Pitch Angle	3.3911
7	Platform Roll Angle	-6.4896
10	Platform Designation	C208B
11	Image Source Sensor	
12	Image Coordinate System	
13	Sensor Latitude	41.0957
14	Sensor Longitude	-104.8702
15	Sensor True Altitude	2933.0312
16	Sensor Horizontal Field of View	3.0652
17	Sensor Vertical Field of View	1.7221
18	Sensor Relative Azimuth Angle	254.25
19	Sensor Relative Elevation Angle	-20.3828
20	Sensor Relative Roll Angle	0.0
21	Slant Range	2291.8906
22	Target Width	0.0
23	Frame Center Latitude	41.1068
24	Frame Center Longitude	-104.851
25	Frame Center Elevation	1867.2038
26	Offset Corner Latitude Point 1	0.0009
27	Offset Corner Longitude Point 1	0.0003
28	Offset Corner Latitude Point 2	-0.0
29	Offset Corner Longitude Point 2	0.0012
30	Offset Corner Latitude Point 3	-0.0008
31	Offset Corner Longitude Point 3	-0.0002
32	Offset Corner Latitude Point 4	0.0
33	Offset Corner Longitude Point 4	-0.0011
47	Generic Flag Data 01	0.0
56	Platform Ground Speed	0.0
59	Platform Call Sign	Firebird
65	UAS Datalink LS Version Number	1.0
72	Event Start Time - UTC	1970-01-01 00:00:00+00:00

Redacted Metadata

Tag	Key	Value
1	Checksum	0
2	Precision Time Stamp	1970-01-01 00:00:00+00:00
3	Mission ID	XXXXXXXXXXXXXXXXXXXXX
4	Platform Tail Number	XXXXXX
5	Platform Heading Angle	0.0
6	Platform Pitch Angle	0.0006
7	Platform Roll Angle	0.0015
10	Platform Designation	XXXXX
11	Image Source Sensor	
12	Image Coordinate System	
13	Sensor Latitude	0.0
14	Sensor Longitude	0.0
15	Sensor True Altitude	-900.0
16	Sensor Horizontal Field of View	0.0
17	Sensor Vertical Field of View	0.0
18	Sensor Relative Azimuth Angle	0.0
19	Sensor Relative Elevation Angle	0.0
20	Sensor Relative Roll Angle	0.0
21	Slant Range	0.0
22	Target Width	0.0
23	Frame Center Latitude	0.0
24	Frame Center Longitude	0.0
25	Frame Center Elevation	-900.0
26	Offset Corner Latitude Point 1	0.0
27	Offset Corner Longitude Point 1	0.0
28	Offset Corner Latitude Point 2	0.0
29	Offset Corner Longitude Point 2	0.0
30	Offset Corner Latitude Point 3	0.0
31	Offset Corner Longitude Point 3	0.0
32	Offset Corner Latitude Point 4	0.0
33	Offset Corner Longitude Point 4	0.0
47	Generic Flag Data 01	0.0
56	Platform Ground Speed	64.0
59	Platform Call Sign	XXXXXXXX
65	UAS Datalink LS Version Number	0.0
72	Event Start Time - UTC	1970-01-01 00:00:00+00:00

Figure 4.4: Redacted vs Un-Redacted Packet



0601 MISB Key	Value
Checksum	58389
Precision Time Stamp	2012-09-19 20:50:26.485Z
Mission ID	XXXX_XXXXXXXX_XXXXXXX
Platform Tail Number	XXXXXX
Platform Heading Angle	157.60128175783933
Platform Pitch Angle	3.390606402783288
Platform Roll Angle	-6.491286965544603
Platform Designation	XXXXX
Sensor Latitude	41.09573999005171
Sensor Longitude	-104.87021569389394
Sensor True Altitude	2933.0312046997783
Sensor Horizontal Field of View	3.0652323186083774
Sensor Vertical Field of View	1.719386587319753
Sensor Relative Azimuth Angle	254.25000015978003
Sensor Relative Elevation Angle	-20.38281248900239
Sensor Relative Roll Angle	0
Slant Range	2291.8893960984165
Target Width	0
Generic Flag Data	0
Platform Ground Speed	0
Platform Call Sign	XXXXXXXX
UAS LDS Version Number	1
Event Start Time - UTC	1970-01-01 12:00:00 AM

Figure 4.5: Selective Metadata Redaction Use Case

Figure 4.5 illustrates a type of redaction use-case where we apply selective metadata redaction on metadata items, such as Mission ID, Platform Tail Number, Platform Designation, and Platform Call Sign are selectively redacted, while the geo-spatial metadata remains intact, allowing ArcGIS to draw the geographical footprint of the video. Such selective redaction may be useful when securing identity-revealing metadata, such as Platform Tail Number or Drone Serial Number.

Chapter 5

Utility-aware Video Redaction

In Canada, the Access to Information Act (ATIA) allows citizens to request access to government records, including video footage and photographic evidence, such as those captured by drones [97]. In certain situations, like legal proceedings, these records may be disclosed [98]. To comply with these requests, government and law enforcement agencies often use redaction techniques to protect the identities of uninvolved individuals, including victims, bystanders, and police officers[99].

Traditional methods for redaction require manual intervention by privacy analysts to ensure the video footage is correctly redacted, leaving no personally identifiable information exposed, a process that can be labor-intensive. There is a growing need for automated systems that can redact sensitive information from videos, minimizing the need for manual editing and inspection [100]. Object redaction aims to identify and obscure pixels that reveal sensitive or personally identifiable features within a Region of Interest (ROI). For instance, modern human redaction techniques focus on features such as faces, biometric data, birthmarks, and identifiable clothing details. The process typically involves three stages: object detection, object tracking, and object obfuscation. Object detection identifies the ROI to determine relevant pixels, tracking follows the movement of the object across frames, and obfuscation modifies these pixels to prevent human interpretation. Tracking is only required for video redaction, not for still images.

There are several approaches to evaluate of object detection models today, including Intersection Over Union (IoU), F-1 Score, Average Precision (AP), and Average Recall (AR), among others. Our analysis of these metrics reveals a significant limitation: most existing metrics are single-dimensional, evaluating the performance of an algorithm based on a single object by comparing a detected bounding box with its ground truth. This approach can be inadequate for privacy-focused applications. In privacy-centric contexts, both over-redaction and under-redaction of objects are crucial considerations. Over-redaction may unintentionally obscure public objects, whereas under-redaction could lead to the exposure of sensitive information. Moreover, the effectiveness of an evaluation metric can vary depending on the preferences of different users. For example, a privacy assurance officer may prioritize the maximal redaction of a victim's face, while a law enforcement officer might require the visibility of an offender's face.

Acknowledging these conflicting interests among diverse evaluators, it becomes clear that a privacy-centric object redaction metric should not be constrained to a single dimension. Instead, it should account for both public and private objects for detection and redaction when assessing the effectiveness of an algorithm. To address this issue, we propose a metric called Privacy-Utility Redaction Score, abbreviated PURS, that incorporates regions from both private and public object bounding boxes. In this approach, a single redaction bounding box is compared against two distinct object bounding boxes, each representing a private and a public object region, respectively.

In this section, we have: (1) presented a taxonomy of current state-of-the-art object detection models; (2) introduced a novel approach for privacy-utility aware evaluation; and (3) assessed the studied detection models using traditional methods, such as mean Average Precision (mAP), as well as through a privacy-utility aware approach, employing the proposed Privacy-Utility Redaction Score.

5.1 Foundational Concepts

To comprehend the evaluation metrics outlined in Sections 5.3 and 5.4, it is essential to first understand several key concepts, including the various types of bounding boxes and the distinctions between image classification, detection, and segmentation. Furthermore, learning about Convolutional Neural Networks (CNNs) is important, as they form the foundation of many modern object detection algorithm used today.

5.1.1 Bounding Boxes

Object detection involves object *localization* and bounding box *regression*. A bounding box is used to localize a given object in an image. Most object detection algorithms output a bounding box in terms of a linear vector containing (a) x and y coordinates, (b) the height and width of the bounding box and (c) area of the bounding box. Depending on the convention used, the coordinates may either represent a top left corner or center of a predicted bounding box. For instance, YOLO algorithm outputs detection coordinates that represent the center of the bounding box[101]. Similarly, algorithms from R-CNN family use coordinates that denote the top left corner of a bounding box. Figure 5.1 illustrates a sample bounding box where the axes origin starts from the top left corner. X values increase as we move horizontally on the right, while y values increase as we move down vertically.

Concerning object detection, two different kinds of bounding boxes are considered, i.e., *ground truth* bounding boxes and *predicted* bounding boxes. A *ground truth* bounding box tightly encloses the given object representing its exact location. On the other hand, a *predicted bounding* box is the output vector generated from an object detection model representing a predicted bounding box. A predicted bounding box may not be tightly enclosed as the target bounding box and might contain redundant *False Negative* pixels.

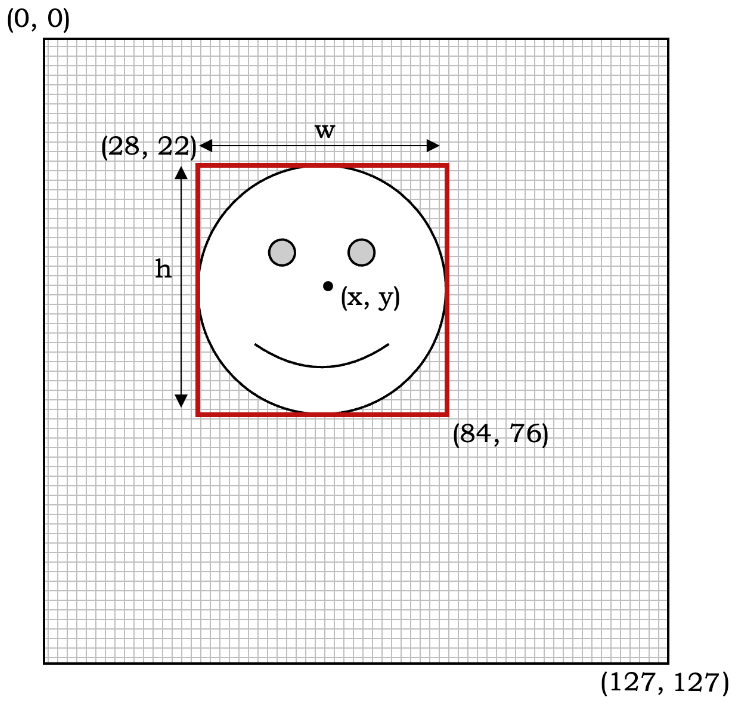


Figure 5.1: An example bounding box

5.1.2 Classification vs Detection vs Segmentation

In computer vision, a large proportion of literature focuses on solving problem of *image classification*, *object detection* and *image segmentation*.

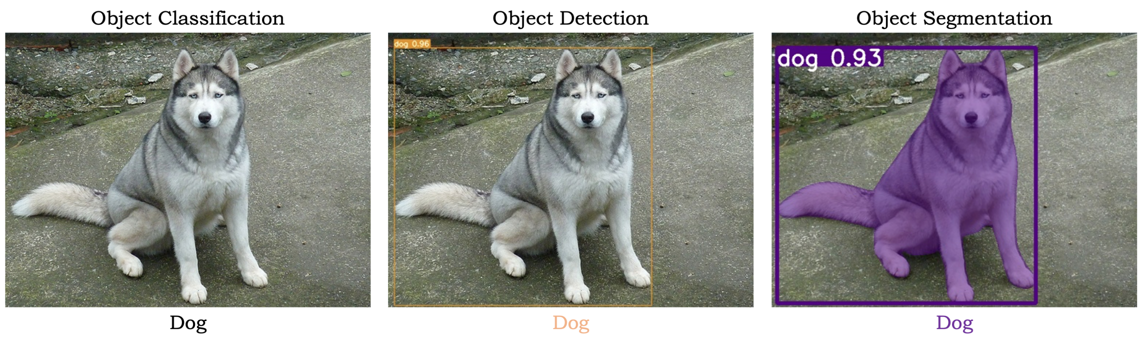


Figure 5.2: Classification vs Detection vs Segmentation

Assigning an entire image to one of the image categorization classes is called *image classification*. For instance, determining the presence of a dog in Figure 5.2 is image classification. The process of classifying an image as belonging to one of the classes while simultaneously locating the position of an object inside the picture is known as image *classification with localization*, also known as *object detection*. For instance, if an image contains several objects, we may construct bounding boxes that indicate where each class is located. These bounding boxes are commonly square or rectangular forms. We may elevate the classification one step further by attributing an object class to each pixel. *Image segmentation* is a fine-grained categorization in which we describe the pixels corresponding to each class item. Figure 5.2 showcases an output from *yolov7* [102] algorithm for Object Detection and Instance Segmentation for a sample image file taken from *ImageNet* dataset [103, 104].

5.1.3 Convolutional Neural Network

Convolutional Neural Networks (CNNs) form the basis of deep learning, a subset of machine learning inspired by the structure of the human brain. CNNs are composed of layers of neurons, starting with an input layer where data is introduced into the system, followed by one or more hidden layers where most computation occurs, and ending with an output layer that produces predictions, as shown in Figure 5.3.

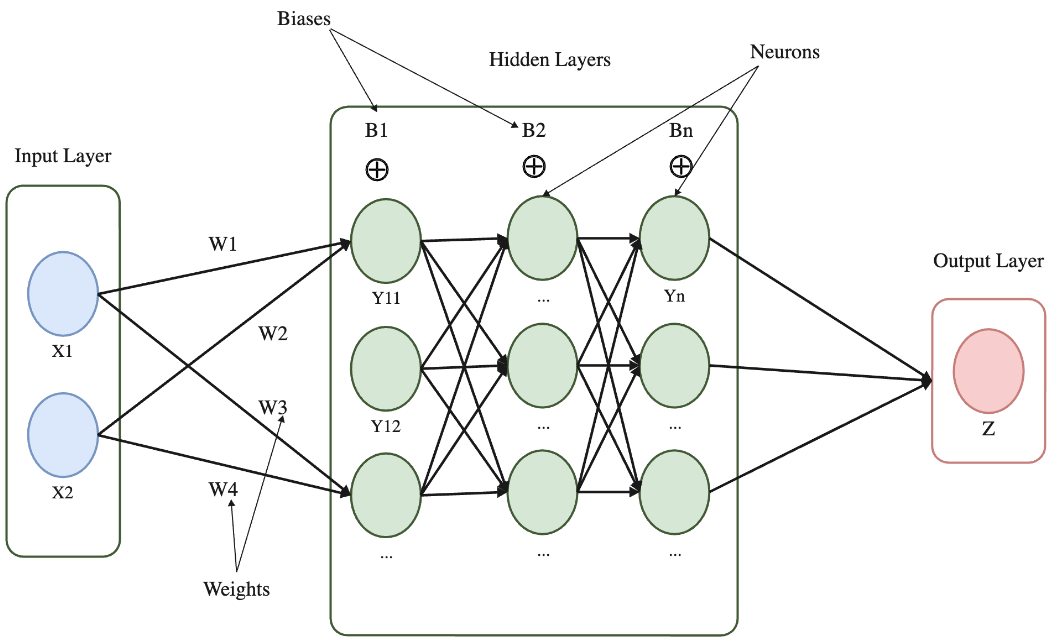


Figure 5.3: Typical CNN Architecture [105]

Neurons in one layer are connected to the next through channels, each assigned a weight. Inputs are multiplied by these weights, summed, and combined with a bias value before passing through an activation function, which determines if the neuron is activated [106, 107]. For instance, the value of neuron Y_{11} is computed by combining weights and biases, as shown in Equation 5.1. Activated neurons transmit data forward through the network, a process called forward propagation. In the output layer, the neuron with the highest value represents the final prediction. During training, the network compares its predictions with known outputs, calculating an error or loss. This error informs how the weights should be adjusted during backpropagation. This cycle of forward and backpropagation repeats across many training epochs, refining the weights until the network accurately predicts objects in most cases.

$$Y_{11} = X_1 \times W_1 + X_2 \times W_2 + B_1 \tag{5.1}$$

5.2 Object Detection Approaches

A crucial step that comes before any type of redaction is identifying the region of interest (RoI). Once determined, techniques for tracking the region of interest can be employed to monitor the movement of the object in the video. Following this stage, a type of obfuscation can be used on specific portions of the moving object. This section explores various object detection techniques and their significance. Investigating the relevant literature on object detection reveals that initial approaches relied on traditional image processing. However, more recent literature indicate a transition towards the adoption of deep learning by many researchers to tackle challenges in object detection.

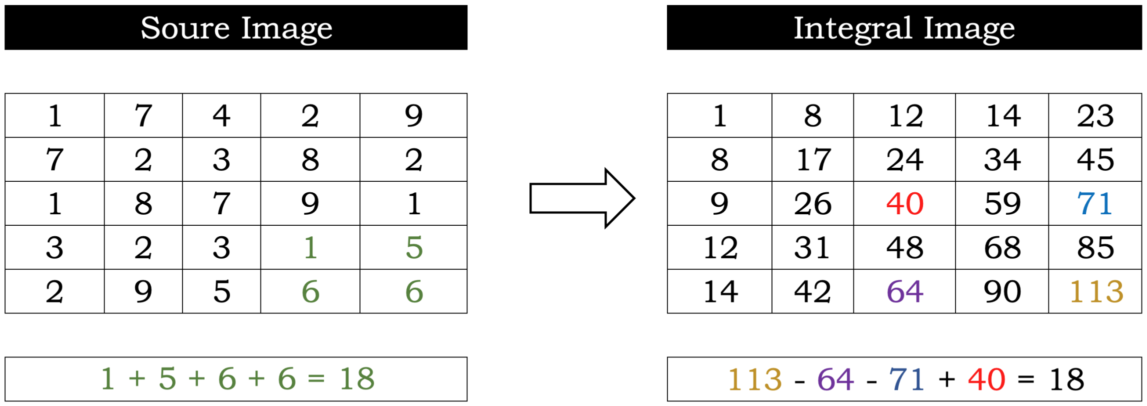
5.2.1 Traditional Image Processing techniques

5.2.1.1 Viola-Jones Detector

The face detection approach proposed by *Viola-Jones* in 2001 serves as the foundation for several recent object detection systems [108]. Despite being regarded as an obsolete algorithm, it remains a feasible approach for systems with limited computing capabilities.

The technique employs a *sliding-window technique* with each block containing around 24x24 pixels. Moving the window across different image regions from left to right, top to bottom retrieves several human-face-like elements. These elements are known as *Haar-like features*, named after the mathematician *Alfred Haar*, who introduced the notion of Haar wavelets [109]. A Haar-like feature is a number deduced by subtracting the sum of pixels from one part of a rectangular image region from the other part within a rectangular Haar block. For example, a human face will have the area near the eyes colored darker than the region surrounding the cheekbones or forehead, which contains much shinier pixels. Thus, two adjacent rectangular boxes can make up a Haar block to symbolize the eyes. The top rectangular box will include a collection of darker pixels representing eyes, while the bottom rectangular box will contain brighter pixels depicting cheeks.

A 24x24 pixel sliding window yields over 160,000 Haar-like features, each of which involves computing delta values for adjacent rectangular Haar blocks and summing pixels inside them. Performing addition operations for each pixel value could be time-consuming, especially for Haar blocks with much bigger features. A more efficient form of image representation called Integral Image is used to overcome this challenge. An integral image is a precomputed image matrix that eliminates the need for several pixel value additions in favor of a few subtractions. An element in the integral image matrix would be computed by summing the pixel values from the above and left sides of the matrix element. The transition from the source to the integral image matrix is depicted in the figure 5.4.


Soure Image

1	7	4	2	9
7	2	3	8	2
1	8	7	9	1
3	2	3	1	5
2	9	5	6	6

1 + 5 + 6 + 6 = 18

Integral Image

1	8	12	14	23
8	17	24	34	45
9	26	40	59	71
12	31	48	68	85
14	42	64	90	113

113 - 64 - 71 + 40 = 18

Figure 5.4: Converting Source Image to Integral Image using Viola-Jones Technique (example image matrix reproduced from [110])

In *Viola-Jones* technique, the following stage is to train a *classifier* for each Haar-like feature using a version of the *AdaBoost* classification algorithm [111]. Each classifier is initially considered a *"weak classifier"* since it lacks enough information to predict a face. However, they may be coupled to form a "cascade of strong classifiers," which the authors refer to as *Attentional Cascades*. Finally, the source image is processed through one of the classifiers in the Cascade for face detection. Although this approach detected faces rapidly, the training procedure was time-consuming, and the system struggled to detect faces with occlusions [112].

5.2.1.2 Histogram of Oriented Gradients (HOG)

HOG, an abbreviation for Histogram of Oriented Gradients, is a prominent feature extraction approach in computer vision [113]. The algorithm captures the distribution of gradient magnitudes and orientations within various regions of an image. Consider Figure 5.5, where different regions of the image have varying pixel intensities. Region A, characterized by continuous black pixels, has a pixel intensity value of 0. Region B, containing gray pixels, has a pixel intensity of approximately 63. Regions C and E, composed of white pixels, have the highest intensity value of 255, while region D, with intermediate pixel intensity, is at 127. These varying intensity levels are critical for understanding the gradients formed at the boundaries between different regions.

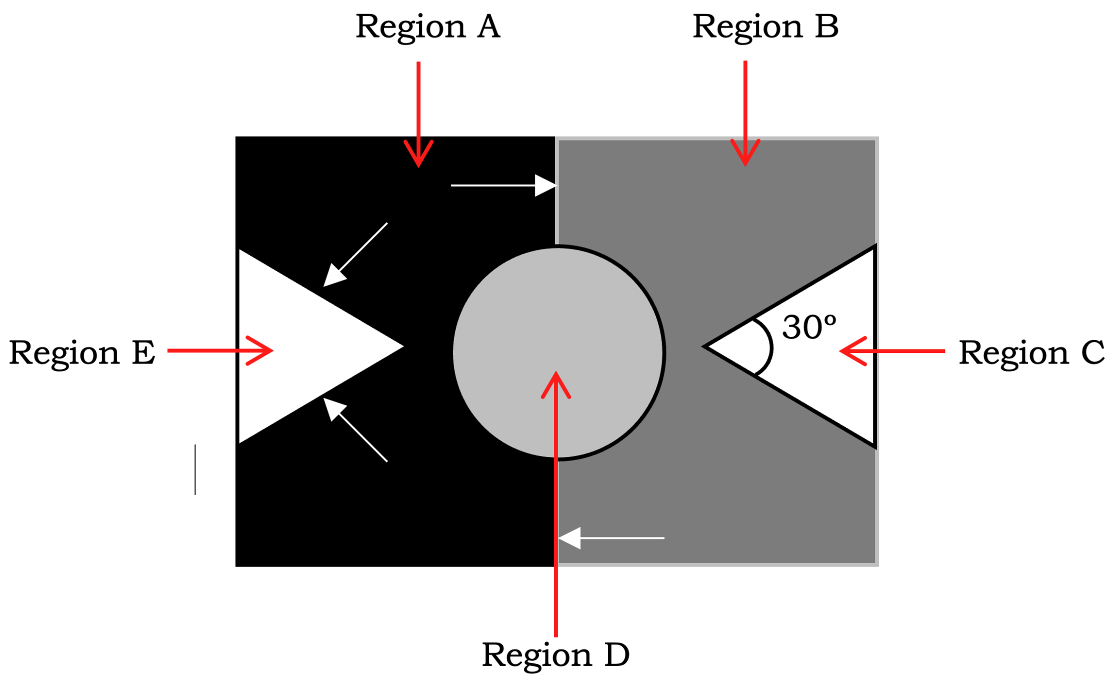


Figure 5.5: Image sample demonstrating HOG features

The *gradient magnitude* represents the absolute value of the difference in pixel intensity between adjacent pixels along a specific direction, such as horizontally (X-axis) or vertically (Y-axis). Areas where pixel intensity changes rapidly across neighboring pixels exhibit high gradient magnitudes, while regions with uniform pixel intensities show low gradient magnitudes. The *gradient direction* or orientation, indicates the angle of this intensity change relative to the image axes. For instance, a sharp transition from dark to bright pixels results in a high gradient magnitude, with the gradient direction pointing from the darker to the brighter region. When switching between picture areas A and B, the gradient magnitude varies in the X-axis direction, as shown by horizontal arrows. Similarly, the gradient magnitude shifts by 30 degrees while moving from region B to region C and vice versa.

A HOG feature descriptor is typically represented as a histogram that captures the distribution of gradient magnitudes over different orientations within a localized region of the image. For example, in a commonly used configuration, the histogram may consist of nine bins corresponding to different orientation ranges, each accumulating the weighted sum of gradient magnitudes falling within that range. This histogram forms the basis of the feature vector used for classification. In the HOG framework, the input image is typically resized to a standard aspect ratio, often 1:2, such as a 64x128 pixel frame. The image is then divided into an 8x16 block grid, where each block contains 8x8 pixel cells.

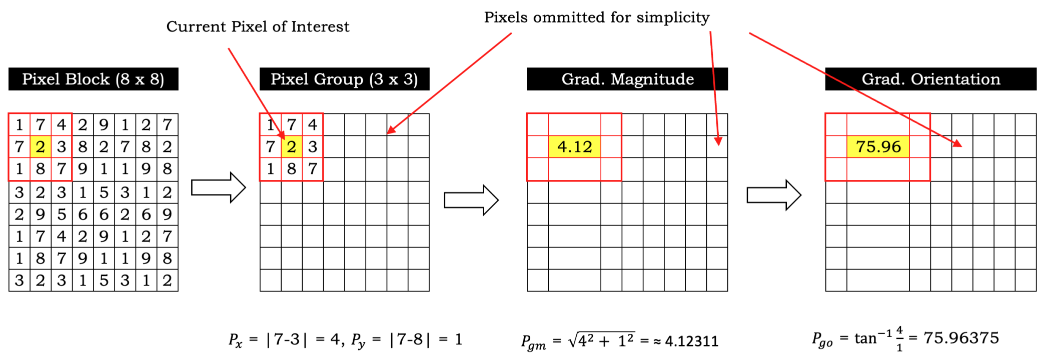

Current Pixel of Interest
Pixels ommitted for simplicity
Pixel Block (8 x 8)
Pixel Group (3 x 3)
Grad. Magnitude
Grad. Orientation
4.12
75.96
P_x = |7-3| = 4, P_y = |7-8| = 1
$P_{gm} = \sqrt{4^2 + 1^2} = \approx 4.12311$
$P_{go} = \tan^{-1}\frac{4}{1} = 75.96375$

Figure 5.6: Calculating HOG vectors

Figure 5.6 demonstrates the computation of gradient magnitudes and orientations for a specific pixel located at column-2 and row-2 within a block. Here, P_x and P_y represent the gradient components along the X and Y axes, respectively. The gradient magnitude (P_{gm}) is derived from the difference in pixel values across the X-axis, while the gradient orientation (P_{go}) is calculated as the angle of the gradient vector. These values contribute to the formation of a histogram, where each bin corresponds to a specific orientation range (e.g., 0-20 degrees, 20-40 degrees, etc.). The magnitude value at a given orientation is distributed across the corresponding bins, forming the *feature vector* for the block. Once the histogram for each pixel value is completed, the feature vector for the block is formed based on these histogram values. This feature vector can then be used with classification techniques to recognize objects.

5.2.2 Deep Learning based models

The last decade of object detection literature demonstrates application of deep learning to object detection problem. In this section, we discuss some popular deep learning based object detection models which are then evaluated in section 5.4.

5.2.2.1 R-CNN

The R-CNN family of algorithms is the state-of-the-art approach for two-staged object detection. The R-CNN stands for region-based convolutional neural networks and consists of three significant steps: (1) region proposal, (2) feature extraction and (3) classification [114].

The *region proposal* stage selects a small set of regions in an image to run the image classifier on, unlike the sliding-window-based approach where a classifier is executed N number of times for each image block. R-CNN uses a *selective search* algorithm to extract about 2000 category-independent region proposals. The selective search involves segmenting the given image and grouping adjacent regions based on color, texture and size similarity. The small regions in the segmented image are iteratively merged with adjacent small regions until no more change is observed between the two adjacent regions [115].The underlying CNN architecture is a forked version of the *Caffe* library, prior work by the authors of R-CNN in collaboration with the AI (Artificial Intelligence) research group at the University of California, Berkeley [116]. It consists of 5 convolution layers and two fully connected layers pre-trained with the ImageNet dataset [117]. Although various frameworks such as PyTorch and TensorFlow now offer implementations of R-CNN models, the authors of R-CNN opted to use the existing Caffe library at that time. In addition to R-CNN, subsequent versions such as Fast R-CNN and Faster R-CNN utilized the Caffe library for their implementation.

The region proposals from the initial stage must be compatible with the CNN architecture, which accepts fix sized input image blocks. Some pre-processing steps, such as dilation, warping and mean subtraction, are applied before passing the proposed regions through CNN. Before *warping* is applied, *dilation* produces a tight bounding box around the proposed region. The *warping* step involves resizing the region proposal to a fixed size that the underlying CNN expects. *Mean subtraction* involves subtracting the mean of all pixels in an image to have pixels centered around zero. The pre-processing of proposed regions also helps improve the mAP or *mean average precision* as results discussed in Section 5.3.1.5. The pre-processed region proposals forward propagated through CNN for *feature extraction*. The output of CNN is 2000 $\times$ 4096 feature vectors, each of which is passed through a class-specific linear SVM - a commonly used supervised learning algorithm for Classification, which scores each feature vector with a confidence score. The classifier is pre-trained on a large-scale image dataset to distinguish an object from its background and assign a confidence score. Followed by *classification*, greedy NMS is applied to discard detections with lower scores having an IoU (discussed in Section 5.3.1.4) overlap of greater than 0.5 with high-scoring detections to suppress redundant detections and retain the most confident ones.

5.2.2.2 Fast R-CNN

Unlike R-CNN, where 2000 individual *region proposals* are supplied to the CNN for generating feature vectors one after another, in *Fast R-CNN*, the entire image is supplied to a shared CNN to extract a single set of features called a 2D high-level *feature map*. Alongside, the algorithm also uses selective search to propose regions. Another step, RoI or *Region of Interest Pooling*, takes the feature map and proposed regions as input. RoI pooling aims to map or align the proposed regions from selective search to a fixed-size grid on the feature map, which allows extracting feature vectors by *flattening* pooled features from the feature map. Flattening is translating a 2D array of pixels into a linear vector. This reduces pre-processing steps for each region proposal, such as *warping* or *dilation* is needed, unlike R-CNN [118].

The *feature vectors* obtained from RoI pooling steps are then forward propagated through a series of fully connected layers of a backbone CNN, which simultaneously performs two tasks: *Classification* that offers the confidence level or probability of the presence of an object in an image and assign an object class label to it. *Regression* provides the exact coordinates for the bounding box and localizes the object of interest. One of the limitations of Fast R-CNN is that it still uses the selective search for proposing regions.

5.2.2.3 Faster R-CNN

Faster R-CNN uses a shared convolutional neural network consisting of initial convolutional layers responsible for accepting the given image as input and producing a feature map, similar to the Fast R-CNN. The *region-proposal network* is integrated within the shared CNN to take these feature maps to generate region proposals directly. The region proposal network (RPN) step eliminates the selective search approach, which is computationally expensive [119].

The RPN uses an *anchor generation* method where pre-defined anchor boxes of different sizes and aspect ratios are aligned across the feature map in a sliding window fashion. The region proposal network then applies *localization* and *Classification* steps responsible for adjusting the anchor boxes to fit the object of interest. An *objectiveness score* is assigned for each localized anchor box, defining the probability of the refined box containing an object vs background. The proposed regions from RPN are then passed through RoI pooling layers, followed by forward propagation through fully connected layers for Classification and bounding box regression, similar to Fast R-CNN. One of the limitations of RPN is that even though it does not use the slow selective search approach, it still relies on the anchor generation method for proposing regions.

5.2.2.4 YOLO: You Only Look Once

YOLO is one of the fastest and most accurate algorithms used for single-stage object detection today. It is one of the first algorithms to support real-time object detection in videos [120]. The YOLO model requires a single pass for the classification and localization of objects in the given image, unlike Faster R-CNN. It divides the input image into an $S \times S$ sized grid where each cell is responsible for detecting the presence of an object inside it. The backbone CNN architecture of YOLO algorithm consists of 24 convolutional layers and two fully connected layers. The output of the algorithm is of the form $S \times S \times (B \times 5 + C)$ where $S \times S$ is the size of the image grid, B is the number of bounding boxes that each cell should compute, and C is the number of object classes. Out of the 24 layers, the first 20 layers of models are trained using the *ImageNet* dataset, and each of these layers performs a set of actions on the input image to extract and learn high-level features and semantic information. This follows 4 *max-pooling* layers responsible for suppressing the size of feature maps and reducing the overall computational load on the network. Finally, the output of max pooling layers is processed through fully connected layers responsible for performing classification and bounding box regression. There exists a lighter version of YOLO capable of achieving 150 fps for real time object detection. This is due to the reduction in number of layers responsible for training, the lighter version only has about 9 convolution layers instead of 20 layers in base model.

5.2.2.5 SSD: Single Shot Multibox Detector

SSD stands for *Single Shot Multibox Detector*. The term *Single Shot* in the title refers to the fact that it is a single-stage detector where object detection is performed in just one forward pass [121]. The underlying network architecture of SSD consists of a backbone *VGG-16* network without fully connected layers pre-trained on *ImageNet* responsible for extracting feature maps from the input image. Each grid cell on the feature map is assigned about six *default boxes* of different sizes and aspect ratios, similar to the concept of *anchor boxes* in Faster R-CNN. When training, these default boxes are matched with ground truth bounding boxes, and the bounding box with the highest IoU overlap is selected as responsible for predicting the object. This helps SSD predict an offset vector for each default box for representing the modifications needed in the default box to make it match more accurately to a tight bounding box around the object of interest. The offset vector consists of four values, i.e. x-coordinate, y-coordinate, height and width concerning center coordinates. Finally, the predicted boxes are processed through subsequent layers in the network to perform classification and apply a class label.

5.3 Evaluation Metrics

This section examines both traditional evaluation metrics used to assess the performance of object detection models and introduces a novel metric for privacy-centric, utility-aware redaction. While established metrics such as Precision, Recall, Intersection Over Union (IoU), and mean Average Precision (mAP) are widely used, they focus primarily on detection performance without addressing the need to balance privacy and utility in redacted content.

Inspired by these conventional metrics, this section introduces a new approach that shifts the focus toward optimizing both privacy protection and the preservation of useful information in redacted material.

5.3.1 Traditional Evaluation Approaches

5.3.1.1 Confusion Matrix

Figure 5.7 represents the *confusion matrix* representing prediction outcomes. The confusion matrix counts all the probable outputs of an object detection approach. It is a two-by-two matrix with each field representing a count of True Positive, False Positive, False Negative, and True Negative.

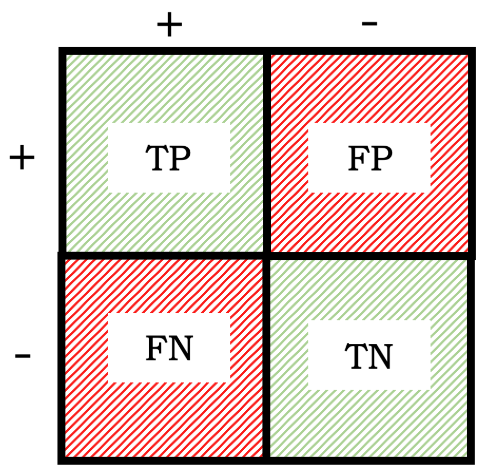


Figure 5.7: Confusion Matrix

A *true positive* instance occurs when an object detection model accurately predicts the presence of an object when there is an object. When an object detection model predicts the presence of an object when there is none, the result is a *false positive*. When an object detection model fails to detect the existence of an object when there is one, the result is a *false negative*. A *true negative* outcome occurs when an object detection model accurately predicts the absence of an object where there is no object.

5.3.1.2 Precision

Precision is a numeric value that decides how many true positive predictions out of all the predictions that an object detection model can predict accurately, which is summarized in Equation 5.2. For example, consider a *human detector* model given 15 images. The model was able to make 7 correct predictions (TP), while 3 predictions were incorrectly predicted (FP) and 5 predictions were missed (FN). In this example, the precision is 0.7 as the number of *true positive* predictions were 7, while *false positive* predictions were 3. Thus, *precision* is the ratio of TP predictions over a sum of TP and FP predictions.

$$\text{Precision} = \left[\frac{\text{TP}}{\text{TP + FP}}\right] \tag{5.2}$$

5.3.1.3 Recall

Recall is a numeric value that decides how many true positive predictions out of all the positive predictions (or outcomes) that an object detection model can predict accurately, which is summarized in Equation 5.3. In the above example of *human detector*, the *recall* is 0.58 as the number of *true positive* predictions were 7, while *false negative* predictions were 5. Thus, *recall* is the ratio of TP predictions over a sum of TP and FN predictions.

$$\text{Recall} = \left[\frac{\text{TP}}{\text{TP + FN}}\right] \tag{5.3}$$

5.3.1.4 Intersection Over Union

Intersection Over Union (IoU) is a numerical value between 0 and 1 which represents how close a predicted bounding box is to the target or ground-truth bounding box. Given a ground truth and predicted bounding box, the *intersection over union* is the ratio of (a) the region where both the boxes intersect each other and (b) the overall surface area covered when combining both the boxes. For example, if PD and GT are predicted and ground truth boxes respectively, their IoU can be computed using Equation 5.4.

$$\text{IoU} = \frac{PD \cap GT}{PD \cup GT} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{5.4}$$

Figure 5.8 provides a diagrammatic representation of IoU, including both predicted and ground truth bounding boxes. The region shaded as *true positive* indicates pixels our model has correctly detected pixels belonging to the object of interest. The region shaded as *false positive* illustrates that our model has incorrectly detected pixels belonging to the object of interest. The region shaded as *false negative* indicated that our model had missed these pixels to be detected as belonging to the object of interest. The region shaded as *true negative* indicates pixels that have been correctly not detected as belonging to the object of interest in our model.

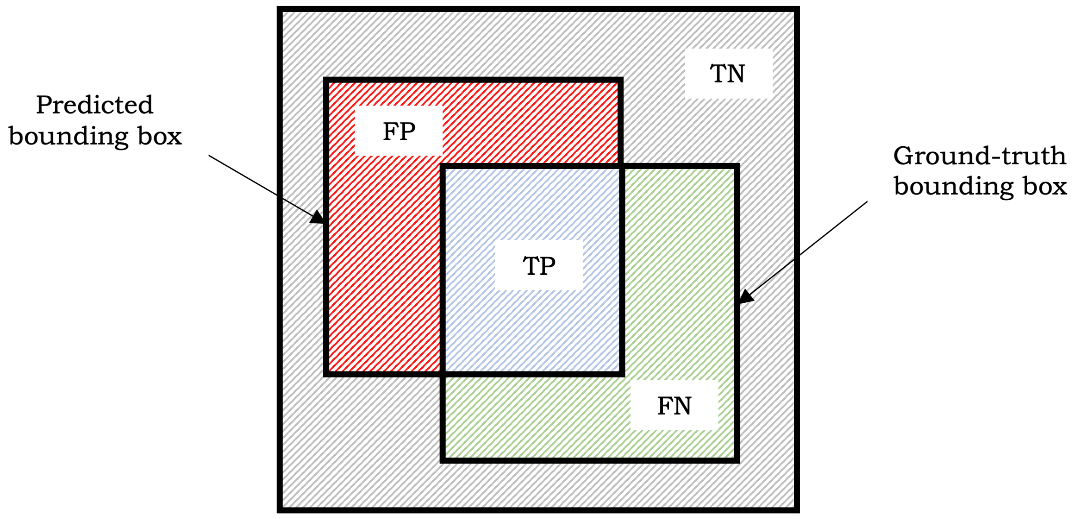


Figure 5.8: Intersection over Union (IoU)

Precision values can be calculated for several IoU thresholds to apply further granularity. Since the IoU threshold is directly proportional to the precision value, the higher the IoU threshold, the higher should be the precision value observed. To calculate the Average Precision (AP), the precision values computed at different IoU thresholds are averaged. The value of intersection over union may range between 0 and 1. An IoU value of 1 the predicted and ground truth bounding boxes are precisely overlapping. Similarly, an IoU value of 0 indicates no intersection between ground truth and the predicted bounding box. Most conventions used an IoU threshold greater than or equal to 0.5 to be the minimum required IoU to validate the object to be correctly detected as *true positive*. In a scenario where a picture reveals personally identifiable information (PII) about a person, it would be desirable to expect the object detection model to have a higher IoU threshold, such that appropriate obfuscation can be applied to all the pixels representing PII data.

5.3.1.5 Mean Average Precision

Intersection Over Union (IoU) may introduce a threshold bias influencing the evaluation results leading to different conclusions about the effectiveness of an algorithm. The IoU threshold used to establish acceptable overlap may impact the number of true positives, false positives, and false negatives. A higher IoU threshold will improve accuracy while increasing false positives and decreasing recall. Similarly, lowering the IoU threshold increases recall while decreasing the number of false positives. Setting a higher IoU threshold rewards algorithms with precise localization accuracy while punishing algorithms that may still deliver accurate outcomes with a more modest IoU threshold. Similarly, a lower IoU threshold will provide an algorithm with lower localization accuracy with the same accuracy weightage as those exhibiting greater localization accuracy, causing a bias.

Another important consideration is the size of an object of interest; certain algorithms may be built to identify smaller objects than others. A higher IoU threshold encourages algorithms that identify more oversized objects rather than small ones. Similarly, a lower IoU threshold benefits algorithms that excel at identifying small objects but struggle with large ones. Selecting an IoU threshold based on image dataset features may also generate bias, as it would reward algorithms developed with the attributes of a particular dataset in mind while punishing those not. As a result, selecting a more generalized evaluation approach that does not favor one algorithm over another is essential. This may be accomplished by including randomness and using varying IoU threshold values. A good example of such evaluation metric is *mean average precision* (mAP) [122].

If we have multiple classes of objects to be predicted by the detector, average precision is computed for each class object, and a mean is taken for all the AP values to deduce the *mean average precision*. Thus, mean average precision can be considered a mean of average precision values computed over varying IoU thresholds for multiple object classes. Equation 5.5 summarizes the mean average precision, where n and c represent an object class and the total count of object classes respectively.

$$\text{mAP} = \frac{1}{\text{c}} \sum_{n=1}^{c} AP_c = \frac{1}{\text{c}} \sum_{n=1}^{c} \frac{\text{TP}}{\text{TP + FP}} \tag{5.5}$$

5.3.2 Privacy-Centric Redaction Metric

Considering an image containing both private and public object classes, when an object is to be redacted, there are six distinct possible outcomes for image regions: redacted public, un-redacted public, redacted private, and un-redacted private, redacted free and un-redacted free. The area within the redaction bounding box that does not overlap with the public or private bounding boxes is considered redacted free. The area outside the union of public, private, or redaction bounding boxes is regarded un-redacted free. Given that many metrics used today are essentially ratios, we leveraged the fundamental principle of ratios: a higher result is achieved when the numerator is larger, and a lower result when the denominator is larger. With this insight, we structured the proposed metric by placing desirable image regions in the numerator and elements we seek to discourage in the denominator.

The base metric rewards an algorithm for accurately redacting a private object while revealing a public object. Conversely, it penalizes an algorithm for under-redacting a private object, redacting a public object, or redacting a free object region due to over-redaction. Furthermore, the metric comes with weight factors, ϵ and δ to adjust the focus of the evaluation between privacy versus utility. Figure 5.9 illustrates several regions that PURS takes into consideration during evaluation.

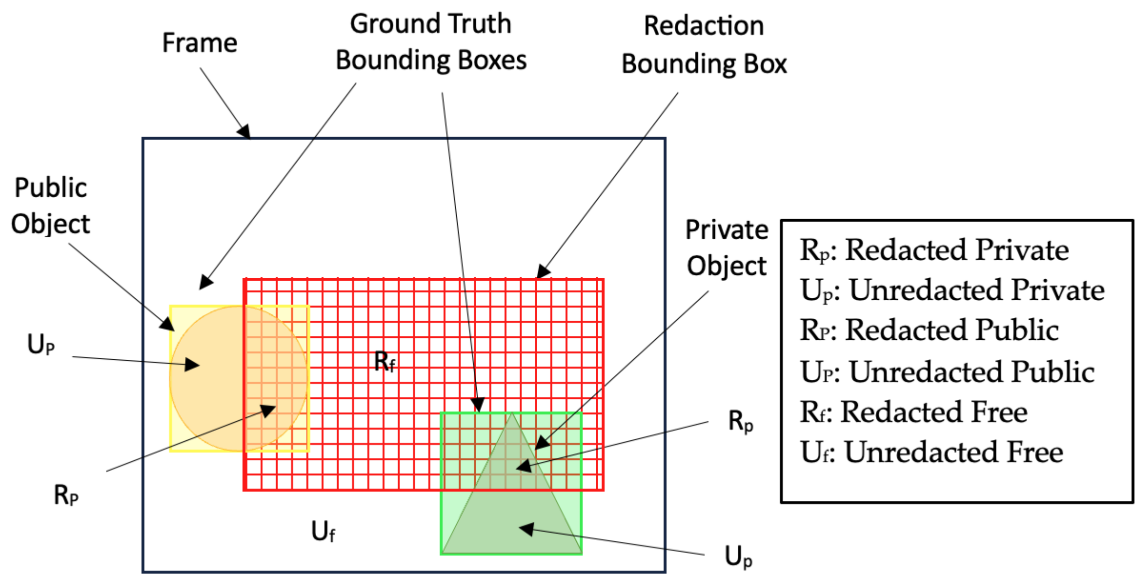


Figure 5.9: Components of PURS

5.3.2.1 Use Case for Redaction

To evaluate the impact of varying levels of redaction, we explored a series of scenarios that represent common redaction outcomes that are shown below. Each of these metrics were executed against 9 possible outcome uses cases that are possible in prospect of object detection. Figure 5.10 summarizes these use cases for ease of comparison.

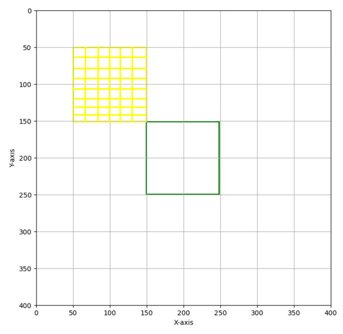
Case 1 – The entire private object is fully redacted, ensuring that no private information is exposed, while the public object is entirely visible without any redaction.

Case 2 – The private object is redacted beyond its boundaries, possibly affecting adjacent regions or the public object, while the public object remains fully visible.

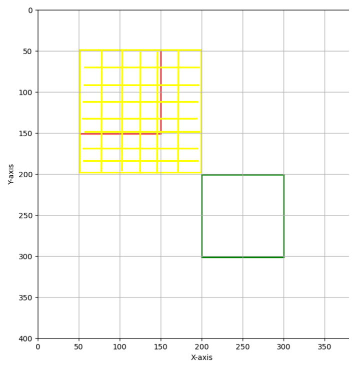
Case 3 – Both the private and public objects are fully redacted, ensuring maximum privacy but completely obscuring the public information as well.

Case 4 – Only 25% of the private object is redacted, leaving the remaining 75% visible, while the public object is fully revealed without any redaction.

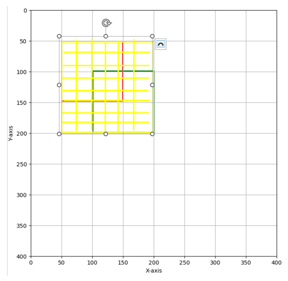
Case 5 – Both the private and public objects are partially redacted, with only 25% of each object being redacted, leaving the majority of both objects visible.



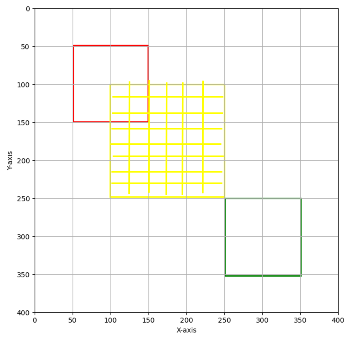
(a) Use Case - 1



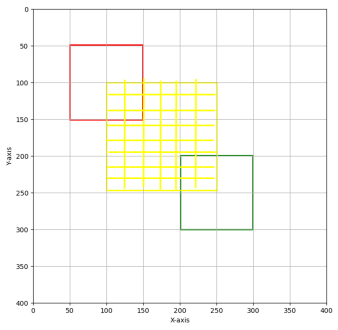
(b) Use Case - 2



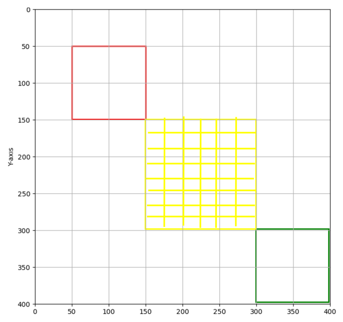
(c) Use Case - 3



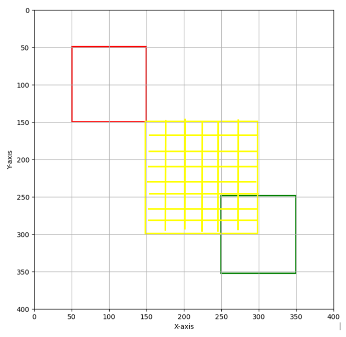
(d) Use Case - 4



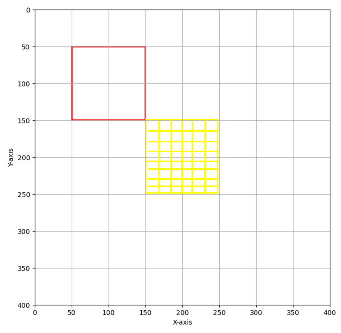
(e) Use Case - 5



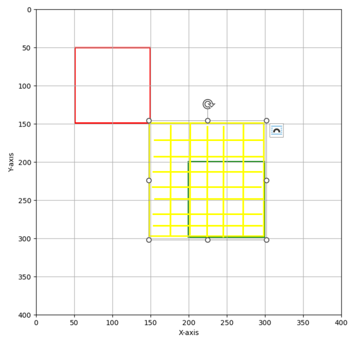
(f) Use Case - 6



(g) Use Case - 7



(h) Use Case - 8



(i) Use Case - 9

Figure 5.10: Use Case chosen for Redaction

Case 6 – Both the private and public objects are fully revealed, with no redaction applied to either, exposing all information.

Case 7 – The private object is completely revealed without any redaction, while 25% of the public object is redacted.

Case 8 – The private object is fully revealed, while the public object is entirely redacted, protecting public information while exposing the private object.

Case 9 – The private object is fully revealed without redaction, but the public object is redacted beyond its boundaries, potentially obscuring adjacent areas or additional information.

5.3.2.2 Preliminary Metrics for Redaction

In the process of developing metrics to evaluate the effectiveness of redaction algorithms, we explored various formulas that combine different regions of an image containing one public object, one private object, and one redaction bounding box. The inspiration for these metrics was drawn from existing evaluation methods, such as the Intersection over Union (IoU), which is commonly used in object detection tasks. Essentially, these metrics are ratios that compare various combinations of different regions within an image, each representing distinct meanings and considerations in the context of redaction and privacy. Below is a set of nine different metrics, denoted as M_1 through M_9, that were proposed during the development phase:

$$M_1(R, U) = \frac{R_p + U_P}{U_p + R_P} \tag{5.6}$$

$$M_2(R, U) = \frac{R_p + U_P}{U_p + R_f + R_P} \tag{5.7}$$

$$M_3(R, U) = \frac{R_p + U_P}{R_p + U_p + R_P + U_P} \tag{5.8}$$

$$M_4(R,U) = \frac{R_p + U_P}{R_p + U_p + R_f + R_P + U_P} \tag{5.9}$$

$$M_5(R,U) = \frac{R_p + R_P}{R_p + U_p + R_P + U_P} \tag{5.10}$$

$$M_6(R,U) = \frac{R_p + R_P}{R_p + U_p + R_f + R_P + U_P} \tag{5.11}$$

$$M_7(R,U) = \frac{U_p + U_P}{R_p + U_p + R_P + U_P} \tag{5.12}$$

$$M_8(R,U) = \frac{U_p + U_P}{R_p + U_p + R_f + R_P + U_P} \tag{5.13}$$

$$M_9(R,U) = \frac{R_p + U_f + U_P}{U_p + R_f + R_P} \tag{5.14}$$

where:

$R_p =$ Redacted Private Object Area,

$U_p =$ Un-Redacted Private Object Area,

$R_P =$ Redacted Public Object Area,

$U_P =$ Un-Redacted Public Object Area,

$R_f =$ Redacted Free Area,

$U_f =$ Un-Redacted Free Area

Essentially, these metrics are ratios that compare various combinations of different regions within an image, each representing distinct meanings and considerations in the context of redaction and privacy.

5.3.2.3 Metric Selection

The following analysis delves into the weaknesses of each proposed metric, ultimately guiding the selection of the most appropriate metric for our purposes. Table 5.1 represents evaluation results for tentative evaluation metrics considered for comparison. The initial column lists the redaction use cases outlined in Section 5.3.2.1, with subsequent columns displaying preliminary metrics as defined in Section 5.3.2.2. The results for each metric can be visualized through a color gradient ranging from 0 to 1 – values approaching 0 are represented in red, signifying lower performance, while values nearing 1 are depicted in green, reflecting higher performance. Intermediate results are represented in various shades of yellow, with yellow indicating a midpoint score of 0.5.

Use case #	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5	Metric 6	Metric 7	Metric 8	Metric 9
1	1.00000	1.00000	1.00000	1.00000	0.50000	0.59793	0.50000	1.00000	1.00000
2	1.00000	0.18794	1.00000	0.61538	0.50000	0.35944	0.50000	0.55404	0.16342
3	0.11750	0.10144	0.50000	0.46498	1.00000	1.00000	0.00000	0.00000	0.16342
4	0.18161	0.05698	0.62500	0.30118	0.12500	0.07421	0.87500	0.80075	0.01857
5	0.11750	0.04596	0.50000	0.25812	0.25000	0.15757	0.75000	0.72865	0.01857
6	0.11750	0.03787	0.50000	0.21943	0.00000	0.00000	1.00000	0.86494	0.00000
7	0.07398	0.02840	0.37500	0.17413	0.12500	0.07421	0.87500	0.80075	0.00000
8	0.00000	0.00630	0.00000	0.05121	0.50000	0.59793	0.50000	1.00000	0.06383
9	0.00000	0.00000	0.00000	0.00000	0.50000	0.35944	0.50000	0.55404	0.00000

Table 5.1: Test Results for Different Evaluation Metrics

Metric-1 fails to differentiate between scenarios where over-redaction occurs. For instance, it rates both Use Case 1 and 2 with the highest score of "1", neglecting over-redaction of private object. This indicates an inability to recognize over-redaction, rendering this metric ineffective in cases where over-redaction is a concern.

Metric-2 produces highly skewed results. This is evident from the fluctuating scores across different use cases. For example, the score drastically drops from 1.00000 in Case 1 to 0.18794 in Case 2, and further to 0.00000 in Case 9. Such inconsistency limits its reliability as a measure.

Metric-3 also fails to recognize over-redaction, like Metric-1. For example, it rates Case-

8 and Case-9 with a score of "0", neglecting over-redaction of public object. The metric provides the same score for vastly different scenarios, as seen in its identical ratings for Case 3, Case 5, and Case 6. This questions its ability to accurately assess redaction effectiveness.

Metric-4 was chosen for its relatively balanced performance across different use cases, starting to best to worst case. While it still displays some variation, it provides a more consistent scores compared to other metrics, making it a potentially useful measure.

Metric 5 struggles with distinguishing between different levels of redaction. It rates the algorithm highest when both private and public objects are completely redacted and lowest when both are completely revealed. This indicates that the metric might be biased towards extreme cases of redaction, without adequately recognizing subtler differences.

Metric-6 shows a similar pattern to Metric 5. It scores two opposing use cases equally. For example, it rates Case 1 and Case 8, also, the best- and worst-case scenarios, equally. This again suggests a bias toward extreme scenarios, making it less useful for nuanced analysis.

Metric-7 fails to correctly evaluate cases where no redaction occurs. For example, it rates Case 6 the highest, which should not be the case in a scenario prioritizing privacy. Additionally, it struggles to distinguish between over redaction, neither in private, nor public objects. For example, it scored the algorithms equally for Case 1 and 2, which confirms its inability to identify over-redaction in private objects. Similarly, in rates an algorithm equally for Case 8 and Case 9, again reflecting its inability to identify over-redaction in public objects.

Metric-8 provides the lowest score when both public and private objects are completely redacted. It also struggles with consistency, as it rates the algorithm equally high for opposite scenarios, such as completely redacting a private object while leaving the public object untouched and vice versa. This suggests an inability to effectively balance the competing needs of privacy and public access.

Metric-9 could not distinguish between several cases. For instance, it fails to differentiate between Case 2 and Case 3, Case 4 and Case 5, and Case 6, Case 7 and Case 9. This lack of judgement indicates that Metric-9 may not be suitable for use cases requiring precise redaction evaluations.

5.3.2.4 Final Metric Formulation

Among the proposed metrics, we found that **Metric 4** (M4) was the most suitable for our purposes. Metric 4 effectively balances the concerns of redacting private information while also considering the areas occupied by public objects. We call the final metric the ***PURS***, short for Privacy-Utility Redaction Score, and denoted as $PURS(R, U, \epsilon, \delta)$, is expressed as follows:

$$PURS(R, U, \epsilon, \delta) = \frac{(1-\epsilon)R_p + \epsilon U_P}{(1-\epsilon)R_p + (1-\epsilon)U_p + \epsilon(1-\delta)R_f + \epsilon R_P + \epsilon U_P} \tag{5.15}$$

Interpretation:

Redacted (R) **and Un-Redacted** (U): The variables *R* and *U* represent the areas that are redacted (hidden) and un-redacted (revealed) in the context of private and public objects. The subscript *p* refers to private objects, while *P* refers to public objects.

Privacy Parameter (ϵ): ϵ is a crucial parameter that controls the emphasis on privacy. When ϵ is close to 1, the equation prioritizes privacy, giving more weight to the redacted areas of private objects. When ϵ is closer to 0, the equation prioritizes revealing public objects over preserving privacy.

Utility Parameter (δ): δ is a secondary parameter that allows for adjustments based on the significance of redacting free regions within an image. Higher δ values penalize algorithms for redacting free regions, while lower values diminish the importance of redacting such regions.

This metric can be viewed as a modified version of the Intersection over Union (IoU), but tailored specifically for the problem of redaction. A ratio of areas in image that prioritize from a redaction perspective to the total area occupied by all three bounding boxes i.e., public, private, and redaction. Moreover, the concept of *differential privacy* inspires the incorporation weights into the metric. Differential privacy is a framework that seeks to provide guarantees about the privacy of individuals in a dataset, typically by adding noise to the data in a controlled manner [123].

The balance between anonymizing private information and revealing public information can change depending on the target audience. For instance, given a crime scene, a privacy officer may set ϵ to a low value, while increasing the value for δ, focusing on redacting non-relevant objects, such as the police officer, bystanders, etc. On the other hand, a police officer, focused on ensuring the visibility of an offender's face in video footage, may prioritize reducing the δ parameter, while setting a higher value for ϵ.

5.4 Evaluation

5.4.1 mean Average Precision (mAP)

A previous survey on object detection evaluated algorithms available up to 2016, using a relatively small AFLW (Annotated Facial Landmarks in the Wild) dataset [124] with 25,000 images of human faces [125]. In contrast, this study incorporates the latest advancements in object detection, including algorithms such as SSD, DETR, EfficientDet, and YOLOv8. We utilize weights trained on the more extensive Microsoft COCO dataset, which consists of approximately 328,000 images across 90 object classes [41].

For this evaluation, we used a MacBook Pro running on Apple M1 Max with 10 CPU cores, 24 core Apple Metal 3 GPU and 32 GB of LPDDR5 physical memory. The PyCocoTools API (Application Programming Interface), from Microsoft COCO object detection challenge was employed to measure average precision, average recall, and mean average precision across 10 different IoU thresholds ranging from 0.50 to 0.95, with increments of 0.05, for 80 different object classes. Wherever possible, we used the original code provided by the original authors of detection models. For models with no official code base available, implementations from the TensorFlow 2.0 deep learning framework were used.

The evaluation results are summarized in Table 5.2, with first column listing the candidate object detection models along with their corresponding backbone CNNs. The second column presents the mean average precision (mAP) computed over 10 different IoU thresholds, with two of these thresholds detailed in columns 3 and 4. Columns 5 through 7 show varying mAP values according to object size.

Our findings reveal that most object detection models have difficulty achieving high mAP for smaller objects, but exhibit improved mAP for medium and large objects. Additionally, mAP varies depending on the backbone used. The value of mAP also differs based on the underlying backbone used. For instance, the SSD model with MobileNet as its backbone CNN generally produces lower mAP compared to ResNet-50. This is due to greater number of convolutional layers in ResNet-50, which enhance precision, though at the cost of slower detection speed. In contrast, combination of SSD and MobileNet offers faster detection speeds over the combination of SSD with ResNet-50. Similarly, using the CenterNet architecture with the HourGlass backbone yields significantly faster detection speeds compared to using it with ResNet101. Figure 5.11 illustrates a comparison of various object detection algorithms based on inference time, using the Microsoft COCO validation dataset.

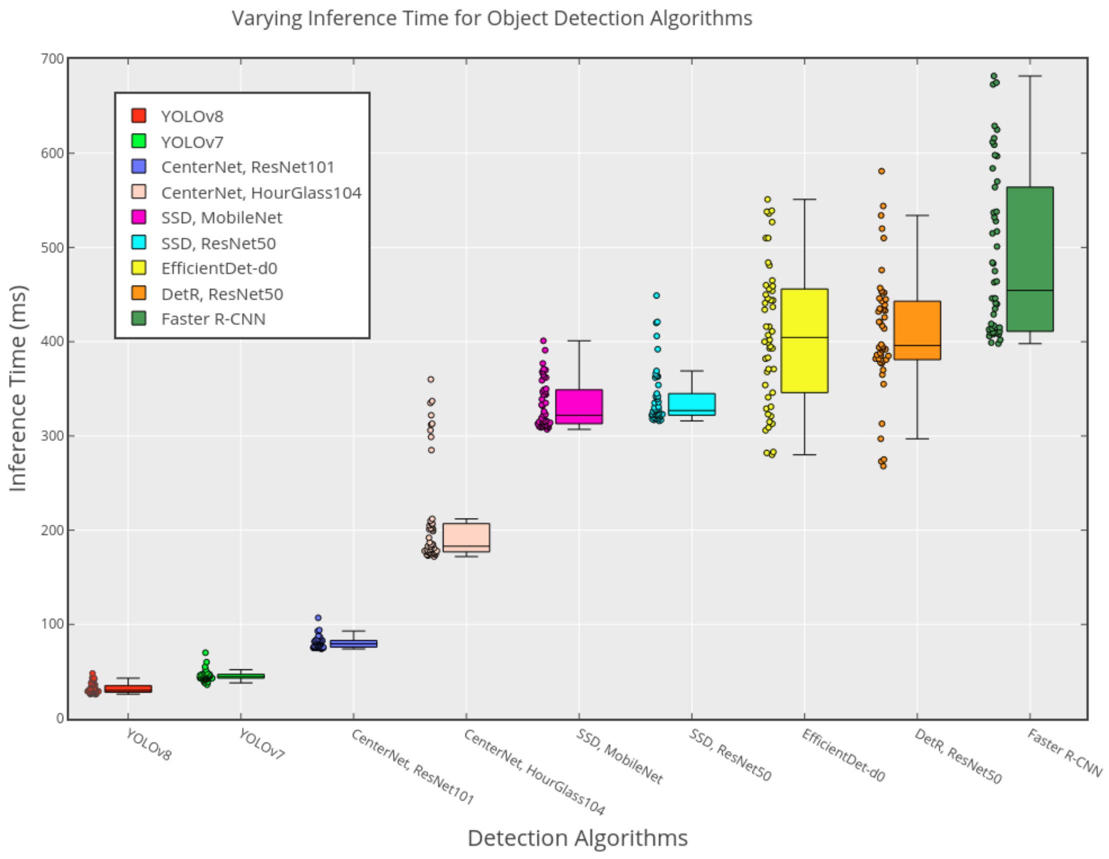


Figure 5.11: Inference Time

Figure 5.12 illustrates the comparison of several object detection algorithms for varying mAP values with respect to Microsoft COCO dataset.

Model	mAP	mAP @.50IOU	mAP @.75IOU	mAP (small)	mAP (medium)	mAP (large)
Faster R-CNN, ResNet101	0.317991	0.499855	0.333913	0.063305	0.273727	0.495995
CenterNet, HG104	0.418513	0.596293	0.450574	0.176202	0.396971	0.567429
CenterNet, Resnet101	0.341795	0.51551	0.360114	0.106329	0.30578	0.506753
SSD, MobileNet	0.291359	0.462906	0.311403	0.086949	0.26647	0.411738
SSD, ResNet50	0.343024	0.51974	0.374497	0.103556	0.319488	0.487745
DetR, ResNet50	0.42	0.624	0.442	0.205	0.458	0.611
EfficientDet-D0	0.334914	0.5154238	0.352957	0.124756	0.387916	0.526467
YOLOv7	0.512	0.697	0.556	0.353	0.56	0.667
YOLOv8	0.540	0.710	0.588	0.360	0.594	0.707

Table 5.2: Microsoft COCO mAP results

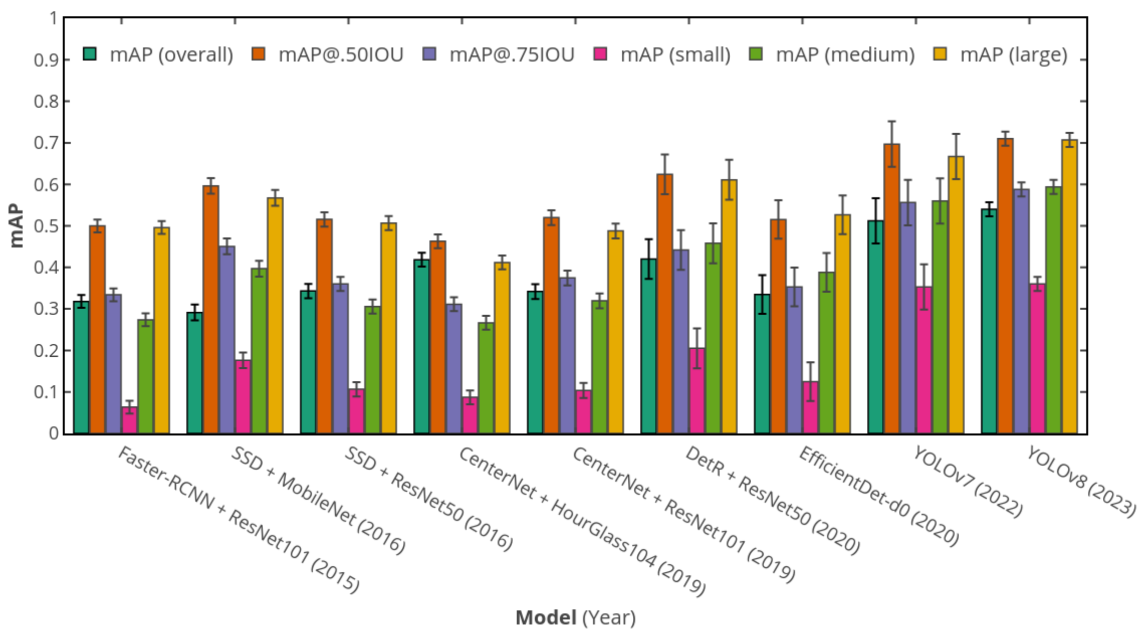


Figure 5.12: Mean Average Precision for Object Detection Algorithms

Figure 5.13 and 5.14 show the comparison of several object detection algorithms for varying average recall values with respect to Microsoft COCO dataset.

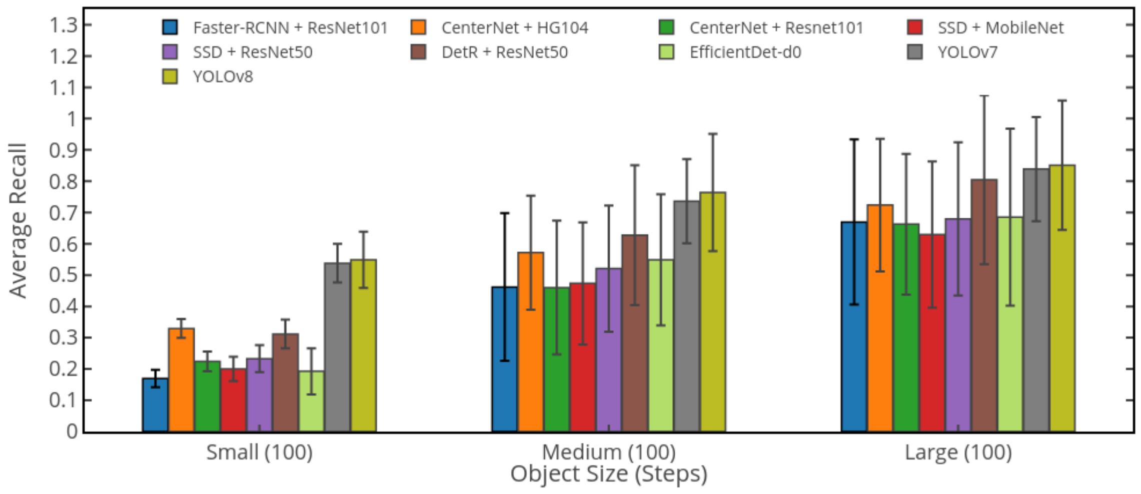


Figure 5.13: Average Recall based on Object Size

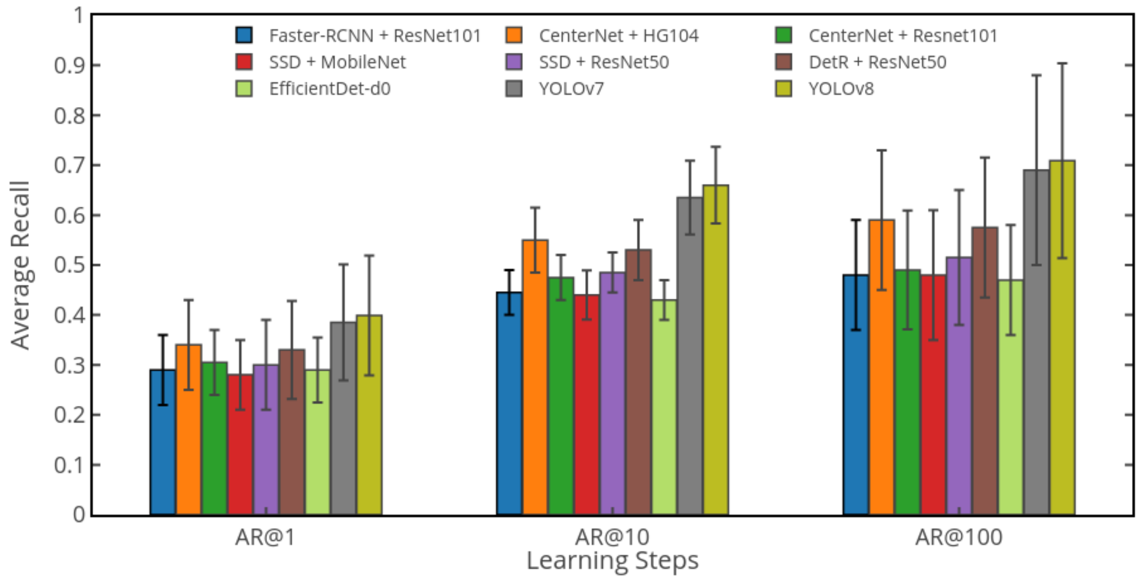
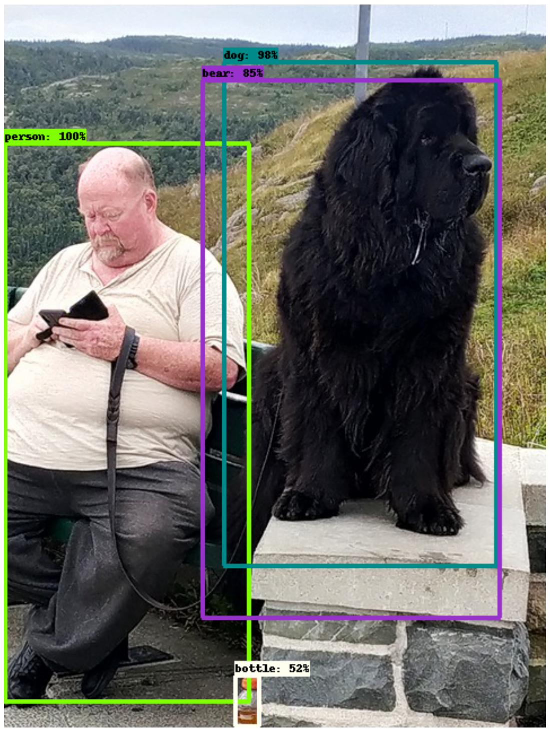
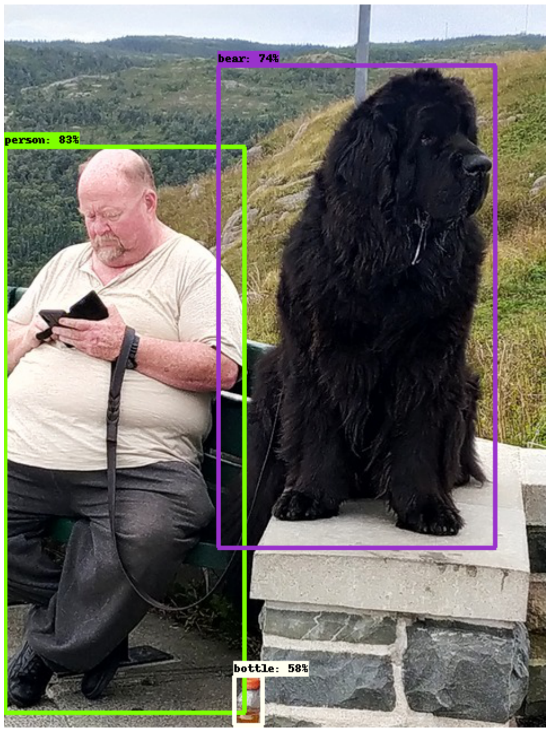


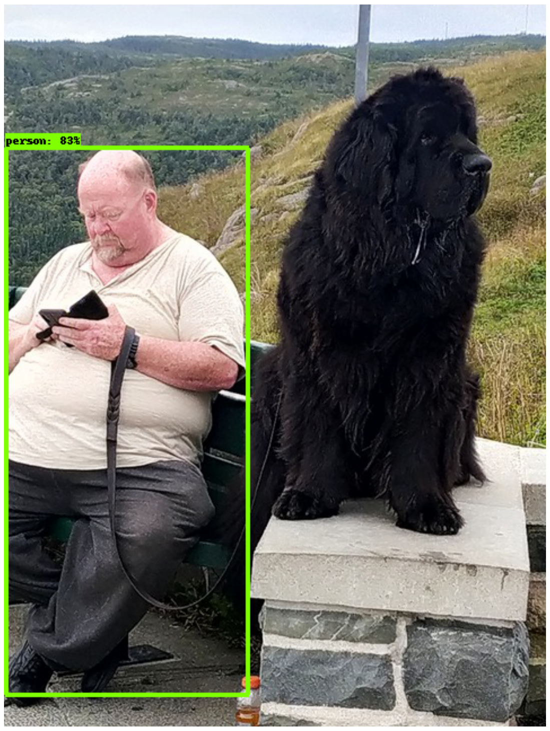
Figure 5.14: Average Recall based on Learning Steps



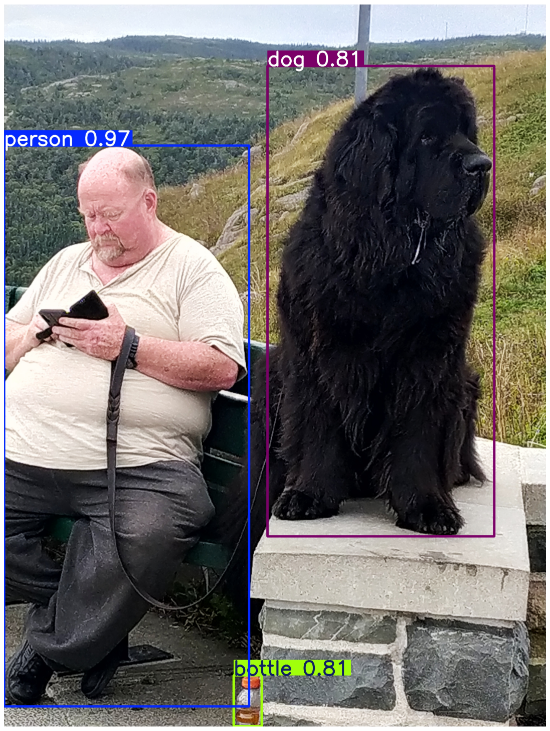
(a) Faster R-CNN



(b) CenterNet + ResNet



(c) SSD + MobileNet



(d) YOLOv8

Figure 5.15: Inference Example for Different Object Detection Algorithms

Figure 5.15 shows misclassifications and missed classifications (e.g., bear) for various object detection methods explored in this study. We can note that some algorithms accurately identified the presence of a dog in the sample image, while others failed to do so or incorrectly suggested the presence of a bear. Despite being trained on the same Microsoft COCO dataset, variations in performance arise from differences in model architectures and their abilities to extract, learn, and distinguish similar-looking objects. As a result, some detection models could be better at identifying dogs versus bears, while others might not be as effective.

The results indicate that YOLOv8 is the fastest algorithm tested, whereas Faster R-CNN is the slowest. When paired with HourGlass104 and ResNet101 backbones, the CenterNet method shows nearly identical performance. Using the SSD technique with the ResNet50 architecture, as opposed to MobileNet, significantly improves inference speed. YOLOv8 achieved the highest mean average precision (mAP) of 0.54, outperforming all other algorithms in terms of precision. The combination of SSD and MobileNet yielded the lowest mAP, which aligns with its design for mobile applications. We discover that the ResNet50 backbone is more precise when combined with SSD.

Overall, we conclude that the choice of detection approach and backbone design profoundly impacts model performance. The acceptable number of false positives may vary depending on the intended use of the detection model. For example, in a human face redaction application, a high rate of false positives could inadvertently reveal individual identities, undermining the purpose of redaction. There are currently no evaluation metrics available that consider the risks associated with inaccurate detections.

5.4.2 Privacy-Utility Redaction Score (PURS)

This comparison highlights the trade-off between simple object redaction and the more complex task of selectively redacting specific categories while leaving others un-redacted. Figures 5.16 and 5.17 illustrates the results for PURS across different object detection models, in terms of weighted and unweighted forms. In this study, we selected images specifically featuring dogs and people from the Microsoft COCO dataset.

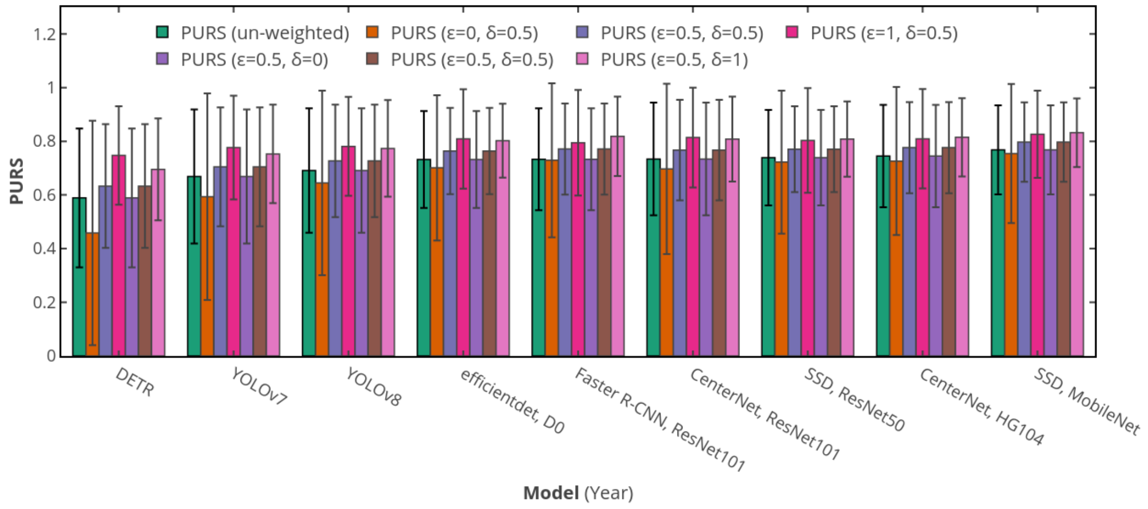


Figure 5.16: PURS by Model (Private: Person, Public: Dog)

In the unweighted PURS metric, CenterNet with ResNet101 (0.9277) and CenterNet with HG104 (0.9271) achieve the highest scores, effectively redacting the private object (dog) while revealing the public object (person). Conversely, YOLOv7 (0.8944) and YOLOv8 (0.9074) score lower, indicating they either inadequately redact the private object (dog) or fail to properly reveal the public object (person).

When controlling privacy with the parameter ϵ, a lower value places greater emphasis on privacy. We observe a gradual decline in algorithm performance as ϵ decreases, with δ values held constant. For instance, the PURS score for the R-CNN model drops from 0.9398 to 0.9165 as δ values are tightened. This suggests that the algorithm is more strictly penalized for either inadequately revealing private object (dog) or failing to expose public object (person).

When adjusting utility with the δ parameter, a lower value indicates a greater focus on utility. Similar to adjustments for the privacy parameter, a decrease in δ results in reduced algorithm performance. For example, the PURS score for the SSD ResNet model decreases from 0.9443 to 0.9205 as δ values are lowered. This trend indicates that the algorithm is penalized for not preserving utility.

A similar pattern results with different ranking for models is observed in an alternative use case where the human is treated as the private object and the dog as the public object. Figure 5.18 showcases Utility-aware redaction using object detection approaches, particularly instances where objects fails to redact private objects revealing sensitive PII, with cases where over-redaction of private objects ends up completely redacting private object.

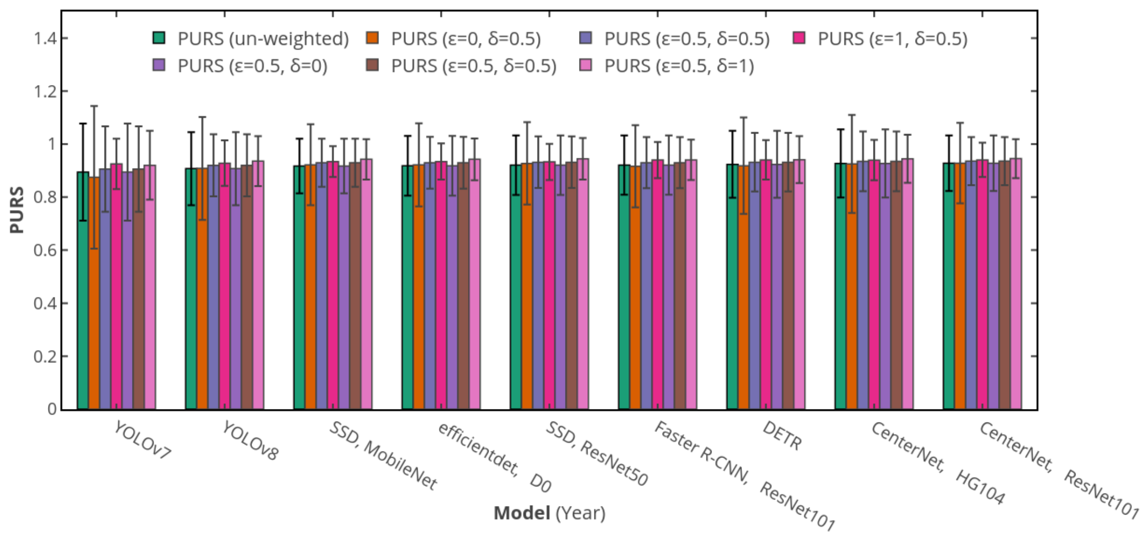


Figure 5.17: PURS by Model (Private: Dog, Public: Person)

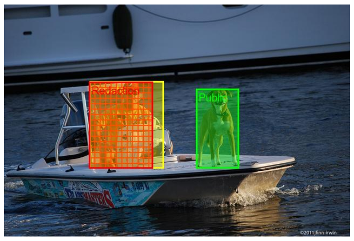
We captured results by changing value of ϵ and δ. Upon capturing the un-weighted metric results, next three results are captured by changing the values of ϵ, while keeping the δ consistent, followed by varying the δ values, while keeping the ϵ values consistent to capture next three results. This demonstrates the effect of adjusting the Privacy Parameter (ϵ) and Utility Parameter (δ). The results of both weighted and un-weighted metric are summarized in Table 5.3 and 5.4.

Model	PURS (unweighted)	PURS1 (ϵ=0, δ=0.5)	PURS2 (ϵ=0.5, δ=0.5)	PURS3 (ϵ=1, δ=0.5)	PURS4 (ϵ=0.5, δ=0)	PURS5 (ϵ=0.5, δ=0.5)	PURS6 (ϵ=0.5, δ=1)
YOLOv7	0.8944	0.8747	0.9056	0.9255	0.8944	0.9056	0.9202
YOLOv8	0.9074	0.9081	0.9200	0.9283	0.9074	0.9200	0.9361
SSD, MobileNet	0.9173	0.9218	0.9293	0.9339	0.9173	0.9293	0.9427
EfficientDetD0	0.9183	0.9215	0.9296	0.9343	0.9183	0.9296	0.942614523
SSD, ResNet50	0.9206	0.9270	0.9316	0.9329	0.9206	0.9316	0.9443
Faster R-CNN, ResNet101	0.9209	0.9165	0.9300	0.9398	0.9209	0.9300	0.9406
DETR	0.9237	0.9184	0.9316	0.9402	0.9237	0.9316	0.9413
CenterNet, HG104	0.9271	0.9252	0.9348	0.9397	0.9271	0.9348	0.9443
CenterNet, ResNet101	0.9277	0.9282	0.9359	0.9405	0.9277	0.9359	0.9453

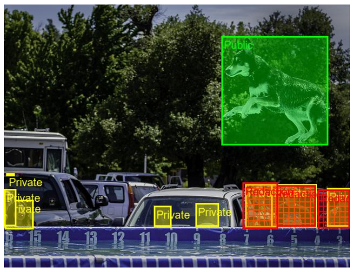
Table 5.3: PURS results for Private Object: Dog and Public Object: Person

Model	PURS (unweighted)	PURS1 (ϵ=0, δ=0.5)	PURS2 (ϵ=0.5, δ=0.5)	PURS3 (ϵ=1, δ=0.5)	PURS4 (ϵ=0.5, δ=0)	PURS5 (ϵ=0.5, δ=0.5)	PURS6 (ϵ=0.5, δ=1)
DETR	0.5889	0.4583	0.6337	0.7471	0.5889	0.6332	0.6956
YOLOv7	0.6688	0.594	0.7046	0.7767	0.6688	0.7046	0.7530
YOLOv8	0.6910	0.6449	0.7270	0.7811	0.6910	0.7270	0.7735
EfficientDet D0	0.7323	0.70112	0.7638	0.8088	0.7323	0.7638	0.8021
Faster R-CNN, ResNet101	0.7328	0.7293	0.7710	0.7946	0.7328	0.7710	0.8186
CenterNet, ResNet101	0.7340	0.6969	0.7669	0.8138	0.7340	0.7669	0.8082
SSD, ResNet50	0.7393	0.7223	0.7706	0.8029	0.7393	0.7706	0.8083
CenterNet, HG104	0.7447	0.7263	0.7762	0.8094	0.7447	0.7762	0.8146
SSD, MobileNet	0.7680	0.7543	0.7971	0.8263	0.7680	0.7971	0.8318

Table 5.4: PURS results for Private Object: Person, Public Object: Dog

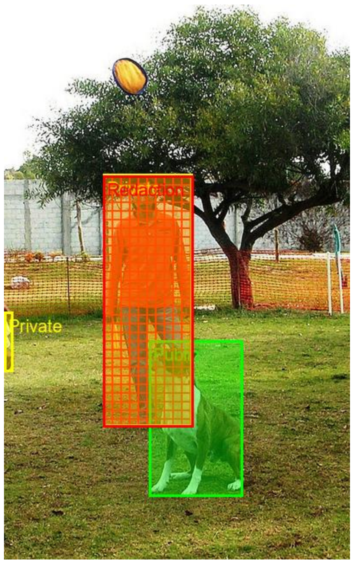


(a) CenterNet, ResNet101

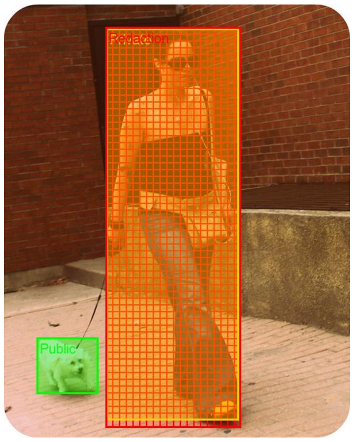


(b) EfficientDet

(c) Faster R-CNN

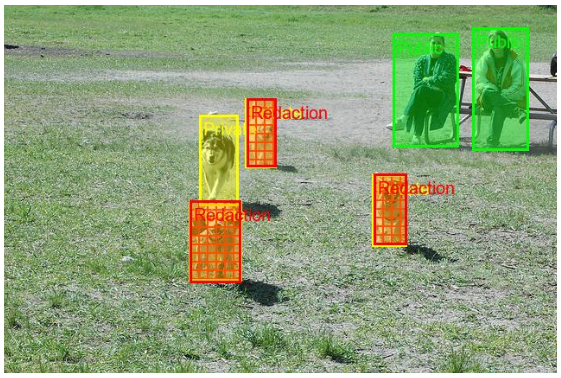


(d) DetR

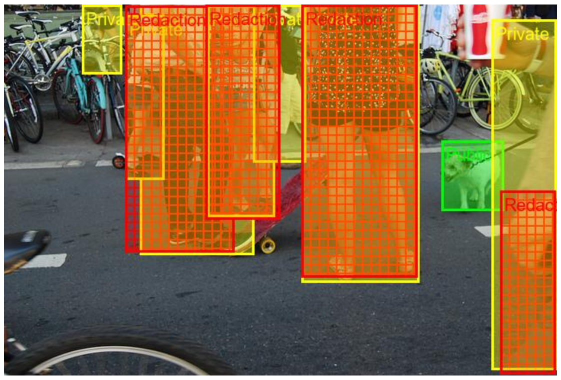


(e) SSD, ResNet50

(f) YOLOv7



(g) YOLOv8



(h) SSD, MobileNet

Figure 5.18: PURS in Action

When comparing the PURS (with human as redaction object) and mAP (Mean Average Precision), we observe an inverse relationship between these metrics. For instance, best performing algorithms such as DetR, YOLOv7 and YOLOv8, with some of highest mAP scores ranging from 0.42 to 0.540, yield one of the lowest PURS at between 0.58 to 0.69. This indicates that while these algorithm excel at the task of single object detection, they struggle with the task of privacy centric selective redaction. Conversely, SSD with MobileNet, despite its lowest mAP score of 0.2914, achieves the highest PURS at 0.768, indicating it handles privacy centric redaction better despite weaker detection capabilities. Other algorithms like CenterNet with HourGlass and ResNet101, along with SSD + ResNet50, have mid-range mAP scores, perform relatively well in redaction, with PURS scores between 0.7343 and 0.745. EfficientDet D0 and Faster R-CNN with ResNet101 follow this trend, with lower mAP scores but solid PURS results.

The comparison of mAP and PURS result is summarized in Figure 5.19.

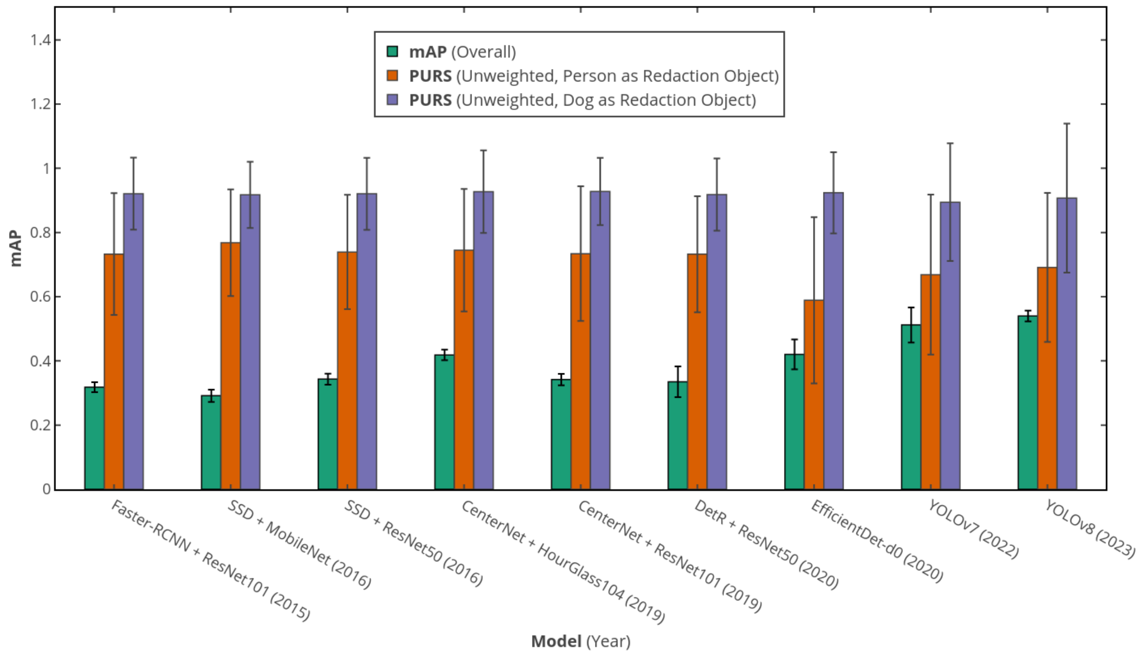


Figure 5.19: Comparison of Mean Average Precision with PURS

These results suggest that algorithms with better mAP results, indicating strong single category object detection capabilities, may still struggle with the task of multi-dimensional privacy-centric redaction, where they must distinguish between private and public objects. This highlights that excelling in object detection does not necessarily translate to effective privacy-focused redaction. This demonstrates the importance of developing algorithms that can handle the problem of multi-category detection for purposes of redaction.

Chapter 6

Conclusions

This dissertation systematically investigates redaction in full-motion videos. In doing so, we devised tools for researchers to work with FMV content, such as parsers capable of analyzing and reactor for manipulating full-motion videos, as well as a taxonomy of visual object detection approaches and privacy-centric evaluation metric.

Tools for FMV

The MP4 parser library imports consumer drone footage and generates atom-specific metadata followed by the offset address of each atom, allowing us to narrow in on bytes of interest for visual and metadata redaction. The MPEG-TS parser library allows us to investigate a MISB-compliant full-motion video by presenting different TS, PES, and KLV packets. For example, the parsers help us identify the bytes that include geographical metadata that has to be redacted.

KLV metadata stream redaction

The KLV redaction module receives a KLV packet and obfuscates sensitive metadata fields to maintaining confidentiality. The censored version of geospatial metadata prevents reverse engineering techniques from examining and extracting sensitive drone flight information, such as the location of the drone or its pilot.

Object detection in video streams

In the context of metadata stream redaction, a pre-defined structure with fixed-sized metadata components simplifies the processing and redaction of geographical information. Decoding the codec-specific video stream data (e.g., in recognizing a human face for redaction), on the other hand, necessitates using complex image processing and deep learning algorithms. We also investigated the redaction of visual information in videos by exploring well-known object detection models ranging from traditional image processing to the deep learning era.

In conclusion, the accuracy of a detection model improves with an increased number of processing layers within the model architecture. On the other hand, as we increase the number of processing layers, the detection rate slows down. Therefore, it is crucial to develop techniques that accelerate the processing of video streams by employing a single-stage pipeline with improved methods for region proposals to enable more seamless redaction. The evaluation of detection models from privacy prospective using the PURS metric reveals that having a higher mAP score does not always mean that the detection model can perform well when challenged with privacy-centric, multi-object categorical scenarios.

The proposed tools for FMV video enable opportunities for integration with other privacy preserving systems. The unique approach to metadata redaction in full-motion videos addresses a privacy risk in consumer drones, adding to the broader effort of enabling the safe use of drones.

Chapter 7

Future Work

Expanding upon the findings and studies presented in this research, several promising directions for future research emerge in the area of drone-based data processing and privacy protection. Given the challenges linked with post-processing geospatial metadata and footage from consumer-grade drones, there lies an opportunity for the development of a FMV multiplexing approach. Such an approach could seamlessly integrate drone specific metadata streams and audio-visual streams, enabling offline processing of drone data without reliance on third-party parsing websites.

Current techniques for object redaction primarily concentrate on identifying and obscuring sensitive visual information within videos. However, a newer approach to redaction could utilize parsed metadata to selectively redact items within a video frame. For instance, this could involve redaction based on the precise GPS coordinates and camera angle of the drone. Such an approach would offer geo-specific visual privacy protection customized to the location and viewpoint of the drone.

Traditional object detection algorithms often rely on large-scale datasets, including object classes that may not be relevant to privacy-centric applications. However, training algorithms on non-sensitive classes does not contribute to effectively detecting private information in images. Introducing a smaller, more focused dataset tailored specifically to privacy concerns could significantly enhance the accuracy and efficiency of object detection algorithms for privacy-centric applications.

As consumer drones become integral to surveillance and public safety, ensuring real-time data protection becomes increasingly important. The proposed metadata redaction method operates on pre-captured drone footage through post-processing procedures. It assumes that the drone footage remains unaltered during transmission. However, there is potential to extend this approach to offer real-time redaction from the moment the footage is generated.

References

[1] *Airborne Intelligence Surveillance and Reconnaissance (ISR) - Global Market Trajectory & Analytics. Research & Markets Ltd.* https://www.researchandmarkets.com/reports/4845770/airborne-intelligence-surveillance-and. (Visited on 11/30/2022) (cit. on p. 2).

[2] *Drone Safety Day | Federal Aviation Administration.* https://www.faa.gov/uas/events/drone_safety_day. (Visited on 06/20/2023) (cit. on p. 2).

[3] *Transport Canada's RPAS Task Force Newsletter May 2023/Bulletin de l'équipe Des SATP de Transports Canada Mai 2023.* https://us4.campaign-archive.com/?u=83e8d9d2228adc4b6b2b696e9&id=f9b25f2823. (Visited on 06/20/2023) (cit. on p. 2).

[4] *FarmBeats: AI, Edge & IoT for Agriculture. Microsoft Research.* https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/. (Visited on 06/20/2023) (cit. on p. 2).

[5] S. Paulissen. *Delivering Vital Medicines via Drone. DHL Express.* https://www.dhlexpress.be/en/news/we-care/dhl-parcelcopter/. May 2020. (Visited on 06/20/2023) (cit. on p. 3).

[6] *Amazon Prime Air Prepares for Drone Deliveries. US About Amazon.* https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries. June 2022. (Visited on 06/20/2023) (cit. on p. 3).

[7] *Using Unmanned Aircraft System Tech - Bechtel. Bechtel Corporate.* https://www.bechtel.com/newsroom/releases/2015/04/unmanned-aircraft-system-technology-construction/. (Visited on 06/20/2023) (cit. on pp. 3, 58).

[8] N. Shor. *SKYLOCK - Narcodrones Are Changing the Smuggling Game -. SKYLOCK.* https://www.skylock1.com/blog/narcodrones-are-changing-the-smuggling-game/. Feb. 2023. (Visited on 06/20/2023) (cit. on p. 3).

[9] A. Press. "Escaped South Carolina Inmate May Have Used Drone-Delivered Wire Cutters". In: *The Guardian* (July 2017). ISSN: 0261-3077. (Visited on 06/20/2023) (cit. on p. 3).

[10] *Gatwick Airport Forced to Shut Runway for Almost an Hour over 'Suspected Drone'. Sky News.* https://news.sky.com/story/gatwick-airport-force-to-shut-runway-for-almost-an-hour-over-suspected-drone-12880784. (Visited on 06/20/2023) (cit. on p. 3).

[11] R. Bunker. "Terrorist and Insurgent Unmanned Aerial Vehicles: Use, Potentials, and Military Implications". In: *CGU Faculty Books* (Jan. 2015) (cit. on p. 3).

[12] *Russia and Ukraine Are Fighting the First Full-Scale Drone War. Washington Post*. https://www.washingtonpost.com/world/2022/12/02/drones-russia-ukraine-air-war/. Dec. 2022. (Visited on 06/20/2023) (cit. on p. 4).

[13] *Security Vulnerabilities Detected in Drones Made by DJI - Newsportal - Ruhr-Universität Bochum*. https://news.rub.de/english/press-releases/2023-03-02-it-security-security-vulnerabilities-detected-drones-made-dji. (Visited on 06/20/2023) (cit. on p. 4).

[14] *Xxd(1): Make Hexdump/Do Reverse. Linux Man Page*. https://linux.die.net/man/1/xxd. (Visited on 11/30/2022) (cit. on p. 5).

[15] P. Read, M.-P. Meyer, and G. Group, eds. *Restoration of Motion Picture Film*. Butterworth-Heinemann Series in Conservation and Museology. Oxford ; Boston: Butterworth-Heinemann, 2000. ISBN: 978-0-7506-2793-1 (cit. on p. 7).

[16] M. Lyra et al. *MATLAB as a Tool in Nuclear Medicine Image Processing*. IntechOpen, Oct. 2011. ISBN: 978-953-307-907-3. DOI: 10.5772/19999. (Visited on 12/06/2022) (cit. on p. 7).

[17] J. D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. The Systems Programming Series. Reading, Mass: Addison-Wesley Pub. Co, 1982. ISBN: 978-0-201-14468-0 (cit. on p. 8).

[18] *A Beginner's Guide to Video Resolution. Adobe*. https://www.adobe.com/ca/creativecloud/video/discover/video-resolution.html. (Visited on 12/02/2022) (cit. on p. 9).

[19] R. Rahaman. *Why Do TV Broadcasts Still Use Interlaced Video?* en. Section: Hardware. Aug. 2023. URL: https://www.howtogeek.com/why-do-tv-broadcasts-still-use-interlaced-video/ (visited on 08/24/2024) (cit. on p. 9).

[20] C. Poynton. *Digital Video and HD: Algorithms and Interfaces*. Morgan Kaufmann, 2003. ISBN: 978-1-55860-792-7 (cit. on p. 9).

[21] *BT.601 : Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide Screen 16:9 Aspect Ratios*. https://www.itu.int/rec/R-REC-BT.601/. (Visited on 04/05/2023) (cit. on p. 11).

[22] *RGB: Full vs. Limited - ReferenceHT*. https://referencehometheater.com/2014/commentary/rgb-full-vs-limited/. June 2014. (Visited on 04/05/2023) (cit. on p. 11).

[23] G. J. Sullivan et al. “Overview of the High Efficiency Video Coding (HEVC) Standard”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1649–1668. ISSN: 1558-2205. DOI: 10.1109/TCSVT.2012.2221191 (cit. on p. 11).

[24] J.-R. Ohm et al. “Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC)”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1669–1684. ISSN: 1558-2205. DOI: 10.1109/TCSVT.2012.2221192 (cit. on p. 11).

[25] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Pearson Education, Nov. 2011. ISBN: 978-0-13-300228-7 (cit. on p. 13).

[26] *Sampling Theorem - an Overview | ScienceDirect Topics*. https://www.sciencedirect.com/topics/computer-science/sampling-theorem. (Visited on 12/06/2022) (cit. on p. 13).

[27] D. Purves et al. “The Audible Spectrum”. In: *Neuroscience. 2nd edition* (2001). (Visited on 12/06/2022) (cit. on p. 13).

[28] *Sampling Rate*. https://www.phonetik.uni-muenchen.de/forschung/BITS/TP1/Cookbook/node61.html. (Visited on 12/07/2022) (cit. on p. 13).

[29] H. Schulzrinne. *Explanation of 44.1 kHz CD Sampling Rate*. https://www1.cs.columbia.edu/~hgs/audio/44.1.html. (Visited on 12/07/2022) (cit. on p. 13).

[30] *Cinema Studies Technology Resources | Audio Recording Standards*. https://blogs.uoregon.edu/uocinetech/recording-and-exporting-standards/audio-recording-standards/. (Visited on 12/07/2022) (cit. on p. 13).

[31] *What Is an Audio Sample Rate and What Sample Rate Should I Record at? - CrumplePop*. https://crumplepop.com/what-sample-rate-should-i-record-at/. (Visited on 12/07/2022) (cit. on p. 13).

[32] *Kaggle Cats and Dogs Dataset. Microsoft Download Center*. https://www.microsoft.com/en-us/download/details.aspx?id=54765. (Visited on 04/08/2023) (cit. on p. 15).

[33] *T.45 : Run-length Colour Encoding*. https://www.itu.int/rec/T-REC-T.45. (Visited on 01/20/2023) (cit. on p. 16).

[34] G. Wallace. “The JPEG Still Picture Compression Standard”. In: *IEEE Transactions on Consumer Electronics* 38.1 (Feb. 1992), pp. xviii–xxxiv. ISSN: 1558-4127. DOI: 10.1109/30.125072. (Visited on 01/13/2024) (cit. on p. 18).

[35] T. Rabie and I. Kamel. “On the Embedding Limits of the Discrete Cosine Transform”. In: *Multimedia Tools and Applications* 75.10 (May 2016), pp. 5939–5957. ISSN: 1573-7721. DOI: 10.1007/s11042-015-2557-x. (Visited on 01/13/2024) (cit. on p. 19).

[36] 14:00-17:00. *ISO/IEC 10918-7:2021. ISO*. https://www.iso.org/standard/80896.html. (Visited on 01/20/2023) (cit. on p. 20).

[37] *Delta Encoding - Amazon Redshift*. https://docs.aws.amazon.com/redshift/latest/dg/c_Delta_encoding.html. (Visited on 12/08/2022) (cit. on p. 21).

[38] *Embedded Video Metadata - GroundSDK Tools 7.0*. https://developer.parrot.com/docs/pdraw/video-metadata.html. (Visited on 06/09/2023) (cit. on p. 22).

[39] I. E. G. Richardson. *Video Codec Design: Developing Image and Video Compression Systems*. Chichester: Wiley, 2002. ISBN: 978-0-471-48553-7 (cit. on p. 22).

[40] *Global Share of Online Video Codecs and Containers 2018. Statista*. https://www.statista.com/statistics/710673/worldwide-video-codecs-containers-share-online/. (Visited on 11/30/2022) (cit. on p. 26).

[41] *COCO - Common Objects in Context*. https://cocodataset.org/#home. (Visited on 01/16/2023) (cit. on pp. 26, 27, 132).

[42] *ISO/IEC 14496-10:2020. ISO*. https://www.iso.org/standard/75400.html. (Visited on 11/30/2022) (cit. on p. 27).

[43] *H.264: Advanced Video Coding for Generic Audiovisual Services*. https://www.itu.int/rec/T-REC-H.264-202108-I/en. (Visited on 11/30/2022) (cit. on pp. 30, 52).

[44] *Compilers: Principles, Techniques, and Tools (Dragon Book)*. URL: `https://suif.stanford.edu/dragonbook/` (visited on 08/16/2024) (cit. on p. 32).

[45] *ISO/IEC 14496-14:2020. ISO*. https://www.iso.org/standard/79110.html. (Visited on 11/30/2022) (cit. on p. 34).

[46] *Criterion - Rust*. https://docs.rs/criterion/latest/criterion/. (Visited on 05/03/2023) (cit. on p. 42).

[47] *ISO/IEC 13818-1:2018. ISO*. https://www.iso.org/standard/74427.html. (Visited on 11/30/2022) (cit. on p. 44).

[48] *H.222.0: Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: Systems*. https://www.itu.int/rec/T-REC-H.222.0-202106-I/en. (Visited on 11/30/2022) (cit. on p. 44).

[49] *Unmanned Aerial Systems for Intelligence, Surveillance, Reconnaissance - DSIAC*. https://dsiac.org/state-of-the-art-reports/unmanned-aerial-systems-for-intelligence-surveillance-reconnaissance/. (Visited on 06/13/2023) (cit. on p. 49).

[50] *MISB Standard 0601.17: UAS Datalink Local Set*. 2020. (Visited on 09/10/2022) (cit. on pp. 51, 93).

[51] *Bitvec - A Crate for Managing Memory Bit by Bit*. Rusty Bit-Sequences. Nov. 2022. (Visited on 11/30/2022) (cit. on p. 52).

[52] *ISO/IEC 8824-1:2021. ISO*. https://www.iso.org/standard/81416.html. (Visited on 11/30/2022) (cit. on p. 53).

[53] *X.690: Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. https://www.itu.int/rec/T-REC-X.690. (Visited on 11/30/2022) (cit. on p. 53).

[54] *wez/atomicparsley: AtomicParsley is a lightweight command line program for reading, parsing and setting metadata into MPEG-4 files, in particular, iTunes-style metadata*. en. URL: https://github.com/wez/atomicparsley (visited on 08/02/2024) (cit. on p. 57).

[55] *axiomatic-systems/Bento4: Full-featured MP4 format, MPEG DASH, HLS, CMAF SDK and tools*. en. URL: https://github.com/axiomatic-systems/Bento4 (visited on 08/02/2024) (cit. on p. 57).

[56] *danielgtaylor/qtfaststart: Quicktime atom positioning in Python for fast streaming*. en. URL: https://github.com/danielgtaylor/qtfaststart (visited on 08/02/2024) (cit. on p. 57).

[57] *gpac/gpac: GPAC Ultramedia OSS for Video Streaming & Next-Gen Multimedia Transcoding, Packaging & Delivery*. en. URL: https://github.com/gpac/gpac (visited on 08/02/2024) (cit. on p. 58).

[58] *MediaArea/MediaInfo: Convenient unified display of the most relevant technical and tag data for video and audio files*. en.

URL: https://github.com/MediaArea/MediaInfo (visited on 08/02/2024) (cit. on p. 58).

[59] *exiftool/exiftool: ExifTool meta information reader/writer*. en. URL: https://github.com/exiftool/exiftool (visited on 08/02/2024) (cit. on p. 58).

[60] *FFmpeg/FFmpeg: Mirror of https://git.ffmpeg.org/ffmpeg.git*. URL: https://github.com/FFmpeg/FFmpeg (visited on 08/02/2024) (cit. on p. 58).

[61] *chemag/m2pb: m2pb: an mpeg2ts parsing tool*. en. URL: https://github.com/chemag/m2pb (visited on 08/02/2024) (cit. on p. 58).

[62] *lmshao/mpeg-ts-media: MPEG-TS (.ts) file format parsing tool*. en. URL: https://github.com/lmshao/mpeg-ts-media (visited on 08/02/2024) (cit. on p. 58).

[63] *sayanna/mpeg2tsparser: mpeg2 transport stream parser*. en. URL: https://github.com/sayanna/mpeg2tsparser (visited on 08/02/2024) (cit. on p. 58).

[64] *jyotidivya/mpeg_parser: Parse mpeg transport stream and write to xml file*. en. URL: https://github.com/jyotidivya/mpeg_parser (visited on 08/02/2024) (cit. on p. 58).

[65] *jeoliva/mpegts-basic-parser: Basic MPEG-TS parser written in C language*. en. URL: https://github.com/jeoliva/mpegts-basic-parser (visited on 08/02/2024) (cit. on p. 59).

[66] *afilipkowski/TS_Parser: MPEG transport stream parser*. en.
URL: https://github.com/afilipkowski/TS_Parser (visited on 08/02/2024) (cit. on p. 59).

[67] *olghfwqgkerd/mpeg-ts-parser*. en.
URL: https://github.com/olghfwqgkerd/mpeg-ts-parser (visited on 08/02/2024) (cit. on p. 59).

[68] D. Peter. *hyperfine*. original-date: 2018-01-13T15:49:54Z. Mar. 2023.
URL: https://github.com/sharkdp/hyperfine (visited on 08/02/2024) (cit. on p. 64).

[69] *Valgrind Home*. URL: https://valgrind.org/ (visited on 08/02/2024) (cit. on p. 64).

[70] *KDE/heaptrack*. original-date: 2015-09-18T14:46:29Z. Aug. 2024.
URL: https://github.com/KDE/heaptrack (visited on 08/02/2024) (cit. on p. 64).

[71] *perf-stat(1) - Linux manual page*.
URL: https://man7.org/linux/man-pages/man1/perf-stat.1.html (visited on 08/02/2024) (cit. on p. 64).

[72] *Solving Problems for a Safer World | MITRE*. en. July 2024.
URL: https://www.mitre.org/ (visited on 08/02/2024) (cit. on p. 77).

[73] *National Institute of Standards and Technology*. en.
text. Last Modified: 2024-07-31T08:16-04:00. July 2024.
URL: https://www.nist.gov/ (visited on 08/02/2024) (cit. on p. 77).

[74] S. Tayeb et al. “Toward metadata removal to preserve privacy of social media users”. In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. Jan. 2018, pp. 287–293. DOI: 10.1109/CCWC.2018.8301741. URL: https://ieeexplore.ieee.org/document/8301741 (visited on 08/10/2024) (cit. on p. 87).

[75] B. Azose and M. Clancy. “User-controlled Selective Metadata Removal Prior to Content Sharing”. In: *Defensive Publications Series* (Dec. 2022). URL: https://www.tdcommons.org/dpubs_series/5571 (cit. on p. 87).

[76] *ImageMagick – Download*. en. URL: https://imagemagick.org/ (visited on 08/10/2024) (cit. on pp. 87, 92).

[77] *Exiv2 - Image metadata library and tools*. URL: https://exiv2.org/getting-started.html (visited on 08/10/2024) (cit. on pp. 87, 92).

[78] *Pillow*. en. URL: https://pillow.readthedocs.io/en/stable/index.html (visited on 08/10/2024) (cit. on pp. 88, 92).

[79] *ExifTool by Phil Harvey*. URL: https://exiftool.org/ (visited on 08/10/2024) (cit. on pp. 88, 92).

[80] R. Bhangale. “Securing Image Metadata using Advanced Encryption Standard”. en. MA thesis. Dublin, National College of Ireland, Feb. 2020. URL: https://norma.ncirl.ie/4149/ (visited on 08/10/2024) (cit. on p. 88).

[81] N. P. Shetty et al. "Protecting Your Online Persona: A Preferential Selective Encryption Approach for Enhanced Privacy in Tweets, Images, Memes, and Metadata". In: *IEEE Access* 12 (2024). Conference Name: IEEE Access, pp. 86403–86424. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3415663. URL: https://ieeexplore.ieee.org/document/10559831 (visited on 08/10/2024) (cit. on p. 88).

[82] J. Voisin, C. Guyeux, and J. M. Bahi. *The Metadata Anonymization Toolkit*. arXiv:1212.3648 [cs]. May 2013. DOI: 10.48550/arXiv.1212.3648. URL: http://arxiv.org/abs/1212.3648 (visited on 08/10/2024) (cit. on p. 88).

[83] C. Curtis. "Anonymizing and obfuscating PDF content while preserving document structure". In: *Proceedings of the 22nd ACM Symposium on Document Engineering*. DocEng '22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 1–4. ISBN: 978-1-4503-9544-1. DOI: 10.1145/3558100.3563849. URL: https://doi.org/10.1145/3558100.3563849 (visited on 08/10/2024) (cit. on p. 89).

[84] J. Van Veen and F. Hermans. "Anonymizing spreadsheet data and metadata with anonymousXL". In: *CEUR Workshop Proceedings* 1209 (2014), pp. 45–46. ISSN: 1613-0073. URL: http://www.scopus.com/inward/record.url?scp=84925238081&partnerID=8YFLogxK (visited on 08/10/2024) (cit. on p. 89).

[85] *openscilab/dmeta*. original-date: 2023-09-23T17:12:13Z. Aug. 2024. URL: https://github.com/openscilab/dmeta (visited on 08/10/2024) (cit. on pp. 89, 92).

[86] *openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files — openpyxl 3.1.3 documentation*. URL: https://openpyxl.readthedocs.io/en/stable/# (visited on 08/10/2024) (cit. on pp. 89, 92).

[87] *PDFtk - The PDF Toolkit*. URL: https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/ (visited on 08/10/2024) (cit. on pp. 89, 92).

[88] *FFmpeg/FFmpeg*. original-date: 2011-04-14T14:12:38Z. Aug. 2024. URL: https://github.com/FFmpeg/FFmpeg (visited on 08/10/2024) (cit. on pp. 90, 92).

[89] E. Gelbing et al. "Video Source Identification from MP4 Data Based on Field Values in Atom/Box Attributes". en. In: *Electronic Imaging* 33 (Jan. 2021). Publisher: Society for Imaging Science and Technology, pp. 1–7. ISSN: 2470-1173. DOI: 10.2352/ISSN.2470-1173.2021.4.MWSF-337. URL: https://library.imaging.org/ei/articles/33/4/art00013 (visited on 08/10/2024) (cit. on p. 90).

[90] A. P. M. Maung, Y. Tew, and K. Wong. "AUTHENTICATION OF MP4 FILE BY PERCEPTUAL HASH AND DATA HIDING". en. In: *Malaysian Journal of Computer Science* 32.4 (Oct. 2019). Number: 4, pp. 304–314. ISSN: 0127-9084. DOI: 10.22452/mjcs.vol32no4.4.

URL: https://ejournal.um.edu.my/index.php/MJCS/article/view/20408 (visited on 08/10/2024) (cit. on p. 90).

[91] J. Song et al. "Integrity verification of the ordered data structures in manipulated video content". In: *Digital Investigation* 18 (Sept. 2016), pp. 1–7. ISSN: 1742-2876. DOI: 10.1016/j.diin.2016.06.001. URL: https://www.sciencedirect.com/science/article/pii/S1742287616300627 (visited on 08/10/2024) (cit. on p. 91).

[92] *Overview — mutagen*. URL: https://mutagen.readthedocs.io/en/latest/index.html (visited on 08/10/2024) (cit. on pp. 91, 92).

[93] *KittyHawkCorp/stripzip*. original-date: 2016-02-11T01:00:47Z. Mar. 2024. URL: https://github.com/KittyHawkCorp/stripzip (visited on 08/10/2024) (cit. on pp. 91, 92).

[94] *jvoisin / mat2 · GitLab*. en. July 2024. URL: https://0xacab.org/jvoisin/mat2 (visited on 08/10/2024) (cit. on p. 92).

[95] *Motion Imagery Standards Board (MISB) | Geospatial-Intelligence Standards Working Group*. URL: https://gwg.nga.mil/gwg/focus-groups/Motion_Imagery_Standards_Board_(MISB).html (visited on 04/18/2024) (cit. on p. 93).

[96] *STANAG-4609. Standards Technology Group - Standards Central.* https://publishers.standardstech.com/content/military-dod-stanag-4609. (Visited on 12/25/2022) (cit. on p. 93).

[97] O. o. t. P. C. of Canada. *Public Interest Disclosures by Federal Institutions under the Privacy Act.* https://www.priv.gc.ca/en/privacy-topics/surveillance/police-and-public-safety/02_05_d_29/. Apr. 2006. (Visited on 06/26/2023) (cit. on p. 101).

[98] P. S.

bibinitperiod P. C. Government of Canada. *Understanding Your Right to Obtain Information – Access to Information and Privacy at Public Services and Procurement Canada - PSPC.* https://www.tpsgc-pwgsc.gc.ca/aiprp-atip/comprendredroit-understandingright-eng.html. Dec. 2016. (Visited on 06/26/2023) (cit. on p. 101).

[99] *Best Practices for Video Redaction.* https://www.justice.gov/archives/oip/best-practices-video-redaction. Mar. 2022. (Visited on 03/31/2023) (cit. on p. 101).

[100] I. Government of Canada. *Automated Redaction of Video Recordings for the Purposes of Access to Information Requests.* https://ised-isde.canada.ca/site/innovative-solutions-canada/en/automated-redaction-video-recordings-purposes-access-information-requests. Nov. 2020. (Visited on 06/26/2023) (cit. on p. 101).

[101] Ultralytics. *Oriented Bounding Box (OBB) Datasets Overview.* https://docs.ultralytics.com/datasets/obb. (Visited on 01/14/2024) (cit. on p. 103).

[102] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors*. July 2022. DOI: 10.48550/arXiv.2207.02696. arXiv: 2207.02696 [cs]. (Visited on 02/04/2023) (cit. on p. 105).

[103] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. (Visited on 11/30/2022) (cit. on p. 105).

[104] O. Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. (Visited on 11/30/2022) (cit. on p. 105).

[105] *Neural Network Weights and Biases*. URL: https://www.renom.jp/notebooks/tutorial/basic_algorithm/neural_network_general/notebook.html (visited on 01/14/2024) (cit. on p. 106).

[106] V. Rastogi. *What do neurons in a CNN learn?* en. Sept. 2023. URL: https://medium.com/@vaibhav1403/what-do-neurons-in-a-cnn-learn-bcfedba6c000 (visited on 04/19/2024) (cit. on p. 106).

[107] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/convolutional-networks/ (visited on 04/19/2024) (cit. on p. 106).

[108] P. Viola and M. Jones. "Rapid Object Detection Using a Boosted Cascade of Simple Features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. Vol. 1. Dec. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517 (cit. on p. 108).

[109] A. Haar. "Zur Theorie der orthogonalen Funktionensysteme". In: *Mathematische Annalen* 69.3 (Sept. 1910), pp. 331–371. ISSN: 1432-1807. DOI: 10.1007/BF01456326. (Visited on 02/06/2023) (cit. on p. 108).

[110] Computerphile. *Detecting Faces (Viola Jones Algorithm) - Computerphile*. Oct. 2018. (Visited on 02/13/2023) (cit. on p. 109).

[111] Y. Freund and R. E. Schapire. "A Desicion-Theoretic Generalization of on-Line Learning and an Application to Boosting". In: *Computational Learning Theory*. Ed. by P. Vitányi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 23–37. ISBN: 978-3-540-49195-8. DOI: 10.1007/3-540-59119-2_166 (cit. on p. 109).

[112] K. Cen. "Study of Viola-Jones Real Time Face Detector". In: () (cit. on p. 109).

[113] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). Vol. 1. June 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177 (cit. on p. 110).

[114] R. Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014 IEEE Conference on Computer Vision and Pattern Recognition. June 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81 (cit. on p. 113).

[115] J. R. R. Uijlings et al. “Selective Search for Object Recognition”. In: *International Journal of Computer Vision* 104.2 (Sept. 2013), pp. 154–171. ISSN: 1573-1405. DOI: 10.1007/s11263-013-0620-5. (Visited on 06/15/2023) (cit. on p. 113).

[116] Y. Jia et al. *Caffe: Convolutional Architecture for Fast Feature Embedding*. June 2014. DOI: 10.48550/arXiv.1408.5093. arXiv: 1408.5093 [cs]. (Visited on 06/26/2023) (cit. on p. 113).

[117] *Caffe | Deep Learning Framework*. https://caffe.berkeleyvision.org/. (Visited on 06/15/2023) (cit. on p. 113).

[118] R. Girshick. *Fast R-CNN*. Sept. 2015. DOI: 10.48550/arXiv.1504.08083. arXiv: 1504.08083 [cs]. (Visited on 11/30/2022) (cit. on p. 115).

[119] S. Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (June 2017), pp. 1137–1149. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2016.2577031 (cit. on p. 115).

[120] J. Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91 (cit. on p. 116).

[121] W. Liu et al. “SSD: Single Shot MultiBox Detector”. In: *Computer Vision - ECCV 2016*. Ed. by B. Leibe et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 21–37. ISBN: 978-3-319-46448-0. DOI: 10.1007/978-3-319-46448-0_2 (cit. on p. 117).

[122] K. Kishida. “Property of Average Precision and its Generalization: An Examination of Evaluation Indicator for Information Retrieval Experiments”. en. In: () (cit. on p. 122).

[123] C. Dwork et al. “Calibrating noise to sensitivity in private data analysis”. In: *Proceedings of the Third conference on Theory of Cryptography*. TCC’06. Berlin, Heidelberg: Springer-Verlag, Mar. 2006, pp. 265–284. ISBN: 978-3-540-32731-8. DOI: 10.1007/11681878_14. URL: https://doi.org/10.1007/11681878_14 (visited on 08/16/2024) (cit. on p. 131).

[124] M. Köstinger et al. “Annotated Facial Landmarks in the Wild: A Large-Scale, Real-World Database for Facial Landmark Localization”. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops). Nov. 2011, pp. 2144–2151. DOI: 10.1109/ICCVW.2011.6130513 (cit. on p. 131).

[125] S. Sah et al. “Video Redaction: A Survey and Comparison of Enabling Technologies”. In: *Journal of Electronic Imaging* 26.5 (July 2017), p. 051406.

ISSN: 1017-9909, 1560-229X. DOI: 10.1117/1.JEI.26.5.051406.
(Visited on 03/30/2023) (cit. on p. 131).