Forensic Science International: Digital Investigation 56 (2026) 302061

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS EU 2026 - Selected Papers from the 13th Annual Digital Forensics Research Conference Europe

REPDF: Repairing corrupted PDF files through font mapping and object relationship reconstruction

Seungeun Park, Byeongchan Jeong, Jieon Kim, Jungheum Park *

School of Cybersecurity, Korea University, Seoul, South Korea

ARTICLE INFO

Keywords:
Digital forensics
Corrupted file repair
Electronic document
PDF
Reference data

ABSTRACT

PDF is widely used as the standard for digital records across administrative, legal, academic, and business domains owing to its portability and structural consistency. As the volume of PDF documents continues to grow, they are increasingly collected as evidence in digital forensic investigations. However, the complex and hierarchical structure of the PDF file format poses significant challenges when repairing files that are corrupted. In this study, we propose a novel PDF repair framework that automatically reconstructs object relationships along with a pre-constructed font database, enabling effective repair even when embedded fonts or Unicode mappings are missing. To evaluate its performance, we generate a corpus of 1,000 PDF files covering ten real-world corruption scenarios, multiple languages, and different PDF creation methods. Experimental results show an average text recovery rate of 90.67 %, along with successful image recovery, demonstrating superior performance compared to existing PDF repair tools.

1. Introduction

The Portable Document Format (PDF) is a widely used electronic document format, recognized for its consistent layout and appearance across various operating systems and platforms. It has become the standard for digital records in administrative, legal, academic, and corporate contexts. In the field of digital forensics, PDF files often serve as key evidence, making reliable interpretation and accurate information extraction essential for forensic analysis (Zia and Adedayo, 2025). Accordingly, research on effective processing techniques for PDF files has become an active area within the forensic domain (Trojahn et al., 2013; Liu et al., 2021; Lam et al., 2024). Therefore, repairing corrupted PDF files can serve as a prerequisite for those efforts aimed at improving the accuracy, reliability, and efficiency of PDF file handling.

PDF documents are organized hierarchically into four main components: *header*, *body*, *cross-reference table*, and *trailer* (International Organization for Standardization, 2020). Specifically, the *body* area of a PDF file consists of various individual objects such as pages, images, and fonts, which are directly or indirectly linked to each other to form the overall structure and content of the document. This complex and hierarchical structure provides flexibility and efficient rendering, but it also makes repairing damaged files difficult. As a result, in practical forensic investigations or incident response situations, recovering content such as text and images from damaged files in a human-readable format is a very challenging task.

Previous studies on PDF files from a digital forensics perspective have primarily focused on recovering deleted files using traditional approaches, such as file system metadata-based deleted file recovery and file carving (Chen et al., 2008). However, these methods do not consider repairing damaged files and thus have limitations, such as being unable to interpret parts of the document or recover internal content when structural damage or loss of essential resources occurs.

In particular, when *font*-related resources such as `/FontFile2` and `/ToUnicode` are corrupted, the character codes in a PDF may be displayed with incorrect glyphs instead of the proper Unicode characters, or may not be interpreted as searchable text. Furthermore, text in PDF files is often rendered visually as vector-based *path* objects. As a result, previous techniques based on simple string extraction are insufficient to obtain semantically meaningful content.

This study proposes a novel PDF repair framework designed to handle various corruption scenarios. The proposed methodology extracts valid objects within a corrupted file and then reconstructs the relationships between them. During this process, it utilizes a pre-constructed font database to ensure text within each page is interpreted correctly, even when embedded fonts or Unicode mapping information are missing. Ultimately, the framework can automatically

* Corresponding author.
E-mail addresses: pse0103@korea.ac.kr (S. Park), naaya@korea.ac.kr (B. Jeong), kijie@korea.ac.kr (J. Kim), jungheumpark@korea.ac.kr (J. Park).

https://doi.org/10.1016/j.fsidi.2026.302061

generate visually accurate and renderable PDF files. The main contributions of this paper are as follows:

- We propose a PDF file repair framework capable of handling severe corruption scenarios where object relationships are not fully preserved and embedded fonts or Unicode mapping information are missing.
- We develop a proof-of-concept tool called REPDF to automate the proposed PDF repair framework and release a website with sample files so that anyone can test its effectiveness.
- Through experiments involving six languages, ten types of corruption, and two PDF generation methods, we demonstrate that the proposed method achieves superior repair performance compared to existing tools.

The remainder of this paper is structured as follows. Section 2 reviews how text and images are represented in PDF files and related work. Section 3 describes the proposed PDF repair framework, and Section 4 presents REPDF, a proof-of-concept tool for automating the framework. Section 5 introduces a dataset generated considering ten types of corruption scenarios, and evaluates the proposed method against existing tools. Section 6 discusses the results and limitations of this work. Finally, Section 7 summarizes the contributions and suggests future research directions.

2. Background and related work

2.1. Text and image representation in PDF files

A PDF file is composed of four primary structural components: the *header*, *body*, *cross-reference table*, and *trailer* (International Organization for Standardization, 2020). The header defines the version of the PDF specification used. The body contains a collection of numbered objects, including pages, fonts, and images, each describing a portion of the document's content. The cross-reference table provides byte offsets for locating these objects within the file, enabling random access and efficient parsing. Finally, the trailer stores references to the document catalog and other essential metadata, linking all parts into a coherent structure. This hierarchical organization allows PDF readers to locate, interpret, and render objects independently, even when portions of the file are partially damaged.

A PDF content stream represents text in two main ways: geometric outlines and font-driven glyphs. The first uses 'Path' objects that describe vector shapes of characters through drawing operators, while the second relies on 'Text' objects that reference fonts to render glyphs according to character codes.

Path-based text describes character outlines with operators such as `m` (move to), `l` (line to), `c` (cubic Bézier curve), and `h` (close path). Because only geometric information is stored, leveraging layout information such as character positions, line spacing, character spacing, and baselines allows text to be rendered without storing additional data for specific fonts.

Text-based representation depends on font resources to link codes to glyphs. A Text object begins with `BT` and ends with `ET`; within this scope, `Tf` selects a font key (e.g., `/F1`) and size, and `Tj` renders glyphs according to the corresponding font. In practice, PDF files use two major font families: TrueType (simple) fonts that employ single-byte encodings and are suitable for small character sets, and Type 0 (composite) fonts that combine a CMap (Character Map) with a CIDFont (Character IDentifier Font) to support large sets such as CJK (Chinese, Japanese, and Korean). CIDFonts are commonly based on either Compact Font Format (CIDFontType0) or TrueType (CIDFontType2).

In a PDF file, each font object contains two essential keys: `/ToUnicode` and `/FontFile2`. The `/ToUnicode` key stores mappings from character codes in the content stream to their corresponding Unicode values. The `/FontFile2` key refers to the embedded font file stream, which not only includes Unicode mapping information, but also contains the mappings to glyphs, which are the visual shapes representing characters. It also holds all the elements required to describe these glyphs on the page.

In addition, images in a PDF file are represented in two forms: Image XObjects and Inline Images. An Image XObject is an independent stream object that appears under the `/XObject` key of the page's `/Resources` dictionary and is rendered using the `Do` operator in the content stream. Their dictionaries typically include `/Subtype /Image`, `/Width`, `/Height`, `/BitsPerComponent`, `/ColorSpace`, and filters such as `/FlateDecode`, `/DCTDecode`, or `/JPXDecode`. Placement and scaling follow the Current Transformation Matrix (CTM) set by the `cm` operator. On the other hand, Inline Images are small images embedded directly in a content stream. They begin with `BI`, specify attributes such as `/W`, `/H`, `/BPC`, and `/CS`, include raw image data after the `ID` operator, and end with `EI`.

In this study, we consider this background knowledge to repair the corrupted internal structure of PDF files, in order to preserve the original layout and readability while recovering both path-based and font-based text as well as embedded images.

2.2. Existing studies on forensic analysis of PDF documents

With the rapid increase in the use of digital documents, digital forensic investigations frequently face situations in which a massive number of files on a single computer system must be examined (Joun et al., 2023). In such environments, continuous research has been conducted to efficiently identify and analyze files that possess evidentiary value. In the case of document files, studies have been proposed that utilize textual content to identify and classify files that may serve as evidence, thereby reducing the analytical burden on investigators (Nassif and Hruschka, 2013).

Among these, PDF files are one of the most widely used document formats due to their high compatibility, and they are frequently employed as evidence in forensic analysis. Previous studies have demonstrated that residual information remaining in PDF files, such as pre-modification data, can be analyzed to trace document-related user activities and identify user traces (Chung et al., 2011). In addition, carving PDF files remaining in RAM has been shown to provide evidence for cybercrimes such as illegal downloading (Al-Saleh and Al-Sharif, 2013). These studies suggest that, in the field of digital forensics, PDF files can serve as key digital evidence linking user actions to criminal activities.

Studies aimed at carving various document files, including PDF files, from memory or file systems have been continuously conducted (Yoo et al., 2012; Ravi et al., 2016; Ali et al., 2022). For example, it has been reported that PDF file data can be recovered by carving data remaining in RAM after a PDF file has been viewed using a document viewer (Al-Sharif et al., 2024), or by carving data left in the file system after a PDF file has been deleted. However, data obtained through such approaches often do not preserve the complete file structure and may exist only as partial data or in a fragmented form across multiple locations. Accordingly, in addition to studies on effective file repair methodologies for damaged documents, research has also been continuously conducted to extract and analyze meaningful data from PDF files with incomplete structures residing in memory or file systems.

Previous studies on PDF repair or carving techniques have focused on extracting available streams that contain meaningful information such as text or reconstructing content using the tags located at the beginning and end of objects or streams.

Al-Sharif et al. (2015) utilized indicators that denote the beginning and end of objects to detect and extract data related to previously opened PDF files from RAM. The study presents a carving procedure based on the structure of PDF files and demonstrates the feasibility of carving through experiments conducted under various RAM memory dump scenarios. However, because the analysis relies solely on object

indicators, the approach is limited to structurally intact objects. In contrast, our study performs recovery not only at the object level but also at the lower stream level, thereby enabling the extraction of textual data even when object tags are damaged.

Taşdelen and Süzen (2022) proposed a method that carves PDF files from RAM images using header and footer signatures, extracts their streams, and inserts them into a blank PDF. While effective for filter-based decompression, it cannot repair cases requiring font mappings for Unicode characters. In contrast, our approach improves the accuracy of text object-based recovery by pre-constructing a font database and generating a template file in advance.

Vol et al. (2018) introduced a method for repairing PDF documents that appear normal when rendered on the page but produce corrupted text when copied or extracted. The proposed approach classifies all rendered glyphs extracted from the document into homoglyph groups using a Convolutional Neural Network (CNN) trained on diverse fonts. Based on the resulting probability distributions, the method derives an optimal glyph-to-character mapping. Unlike this approach that restores Unicode mappings based on rendered glyphs, our study focuses on recovering visually corrupted glyphs even in the absence of font resources.

As mentioned above, existing studies mainly rely on residual streams or glyphs but rarely consider the connection between a document's logical structure and its visual layer. A PDF file, in particular, is not just a simple container; Objects such as pages, fonts, and images interact through cross-references and store raw data using various encoding and compression algorithms. Moreover, text is represented indirectly through glyphs, fonts, and mapping tables. Therefore, effectively repairing a corrupted PDF file requires interpreting its complex hierarchical structure and reconstructing cross-references between objects. To address this challenge, we propose an integrated method aligned with the PDF's multilayer architecture, enabling Unicode mapping and visual reconstruction from residual stream data.

3. Methodology

The proposed PDF repair methodology consists of seven steps, as illustrated in Fig. 1. The process involves ❶ generating template PDF files to serve as a font database, ❷ parsing objects and streams through byte-level scanning, ❸ extracting page information, ❹ decompressing and classifying stream data, ❺ analyzing resource data, ❻ reconstructing font mappings, and finally ❼ generating repaired PDF files. The following subsections describe each step in detail.

3.1. Constructing template PDF files with reference fonts

When a PDF is corrupted and loses its font or Unicode mapping information, existing repair methods are of limited effectiveness. To address this, we first construct a font database by collecting a set of reference fonts. Next, a document is created using Microsoft Word (version 2509) containing at least one Unicode character from each font, and it is exported to PDF (see Fig. 2). To diversify font objects, word file includes representative characters from every language supported by each font. For example, if font A supports six languages, at least one character from each language is typed. This procedure helps that the template PDF file contains the corresponding font types for the selected fonts.

PDF file generation typically performs subset embedding, in which only the glyphs that appear in the document are embedded. In this case, characters that do not occur in the template PDF file cannot be rendered. To remove this constraint, we replace each font object's `/FontFile2` stream with the full font file. We note that full-font embedding is

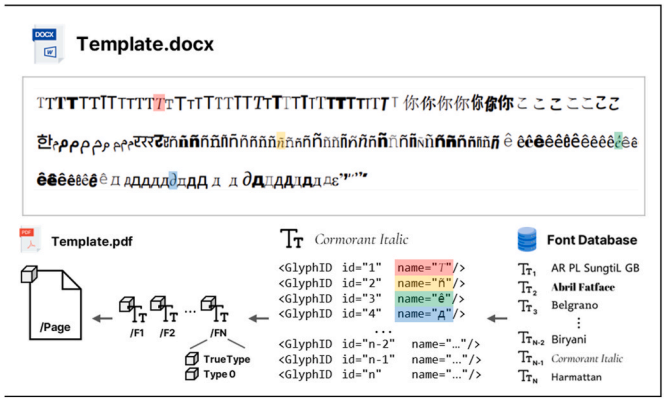

Fig. 2. Excerpt from an MS Word file used to construct a template PDF file containing reference fonts.

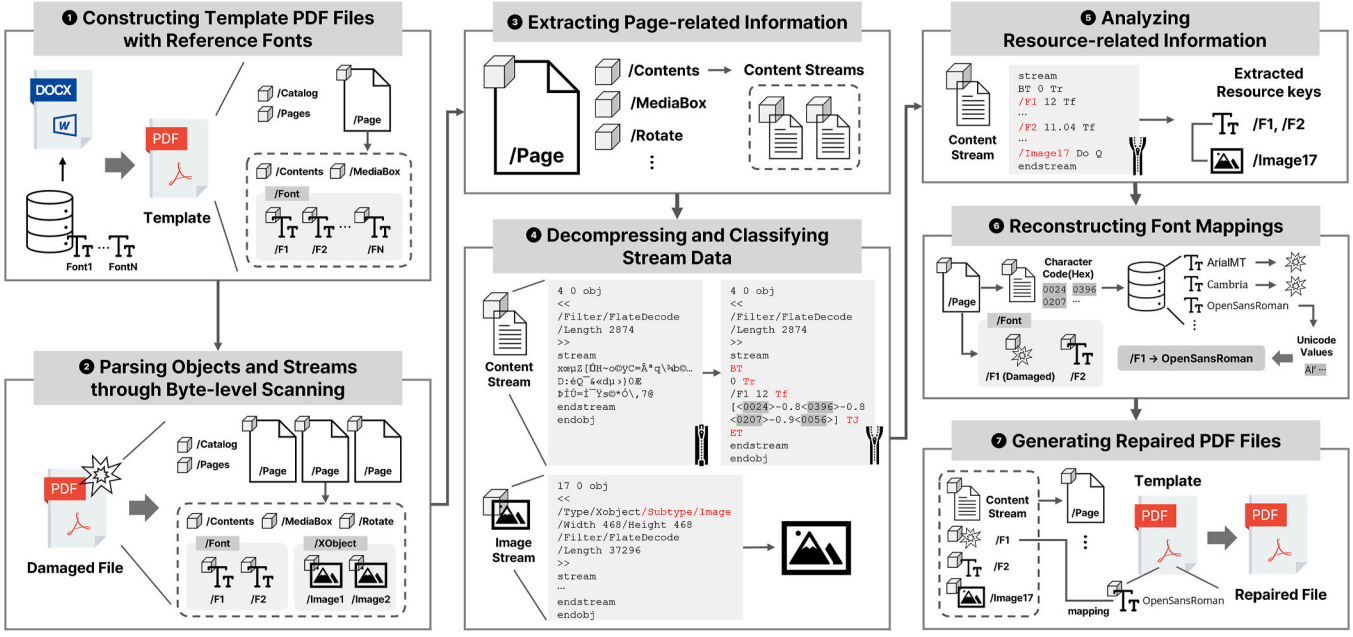

Fig. 1. Overall workflow for repairing corrupted PDF files through font mapping and object relationship reconstruction.

performed only when font resources are corrupted. This selective strategy helps optimize the size of the repaired file.

In addition, each font defines its own glyph scale (`hmtx`) and, accordingly, provides either a `W` key or a `Widths` key. For example, in Font A the width of the letter "A" is 1239. We extract these glyph-width metrics and embeds the corresponding `W/Widths` entries in each font object of the template PDF file. This improves PDF repair performance by preventing character overlap or layout distortion.

3.2. Parsing objects and streams through byte-level scanning

This step extracts valid objects from the damaged PDF file by identifying object blocks defined by the `obj` and `endobj` keywords, with object number and a generation number.

In the case of a normal file, a parser can automatically interpret these object structures through syntactic analysis. However, when the file is damaged, some object start or end keywords may be missing or corrupted. To address this, our method performs byte-level keyword scanning to search for structural indicators such as `obj`, `endobj`, `stream`, and `endstream`, thereby recovering potential object blocks.

From those extracted objects, `/Type` keys such as `/Page`, `/Pages`, `/Font`, `/XObject`, and `/Catalog` are identified. Depending on the identified keys, each object will be used to analyze page structures, resource mapping, content stream interpretation and etc. We also extract metadata, font definitions, and image streams to reference resources during the repair process.

3.3. Extracting page-related information

In the third step, we identify all objects whose `/Type` key is `/Page`, and then values of `/Contents` and `/MediaBox` keys. `/Contents` value references one or more content streams such as body text, images, and vector graphics on the page. `/MediaBox` key specifies the page's bounding box as four coordinates (xmin, ymin, xmax, ymax). With this information, we reconstruct damaged page objects and place the restored content streams in their correct positions. The `/Contents` and `/MediaBox` keys and their values for each page play a crucial role in reconstructing the spatial layout of textual and graphical content.

3.4. Decompressing and classifying stream data

When an object within a PDF file contains a large amount of data, the data is typically stored in a compressed form, referred to as a content stream. A stream object resides within `obj` and `endobj` tags and contains a compressed data section enclosed by the `stream` and `endstream` tags. Typical examples include the content stream referenced by the `/Contents` key of a page object, image data referenced by the `/XObject` key, and font file data referenced by the `/FontFile2` key. In this step, the data enclosed between the `stream` and `endstream` tags is decompressed to extract essential information used in the repair process.

When a page object is not corrupted, the stream objects referenced by keys such as `/Contents`, `/XObject`, and `/FontFile2` remain intact, allowing the decompressed streams to be clearly identified by their roles. However, when those keys or the stream object numbers are corrupted, it becomes difficult to determine the role of each stream. To address this issue, the proposed method classifies streams using the `/SubType` key or the operators contained within the stream. For example, a stream whose `/SubType` is specified as `/Image` is identified as an image stream, while the presence of text-related operators (e.g., `Tf`, `Tj`, `BT/ET`) or an image rendering operator (e.g., `Do`) indicates a content stream. This classification enables identification of stream types even when reference information is damaged, allowing the recovered streams to be effectively utilized in the repair process.

3.5. Analyzing resource-related information

This step analyzes resource keys and values, content streams, and resource data. Resource keys/values provide references to the actual data (image or font file). The content stream is the primary stream containing text and graphic operators to be rendered on a page, and it uses the keys/values to specify the visual composition. When both the resource keys/values and the content stream are available, this step inserts the references and the corresponding data into the template PDF file so that the page is rendered correctly.

However, complete page rendering may be difficult in the following cases: (1) the page object is corrupted and the resource keys/values cannot be identified; (2) the content stream is corrupted. For text, the available resource keys/values and content stream are used to restore the textual content of the page. For images, because precise placement is not possible, we extract the image data and store it as a separate file. For fonts, when the resource keys/values are damaged or font resources in the content stream is corrupted, proper font mapping and text rendering are not possible. In such cases, we perform the font-repair procedure in the next step.

3.6. Reconstructing font mappings

This step deals with reconstructing font resources. When the font resources are not corrupted, they are parsed then inserted in the template PDF files. However, if font resources are corrupted, the font mapping information and font file should be reconstructed. This process is divided into two cases.

If font object has font name in the `/BaseFont` key, it is possible to find out corresponding font object in the template PDF files. Then, font file from pre-constructed font database will be inserted so that it supports rendering all the characters. It is often applied to the PDF files with the Save As creation method.

Second, when the font resource keys/values or the font name cannot be identified, we infer the font used by analyzing the character-code sequences in the content stream, as illustrated in Fig. 3. After the content stream is decompressed, we extract all hexadecimal character sequences enclosed in angle brackets ('<' and '>'). We then utilize `glyphorder` and `cmap` tables from pre-built font database to determine which Unicode character each code maps to (e.g., ⟨0024⟩ → 0x41 → A). For each candidate font, we generate a Unicode string and compare the results, prioritizing outputs that form meaningful words. We score the generated strings using our own word dictionaries covering six

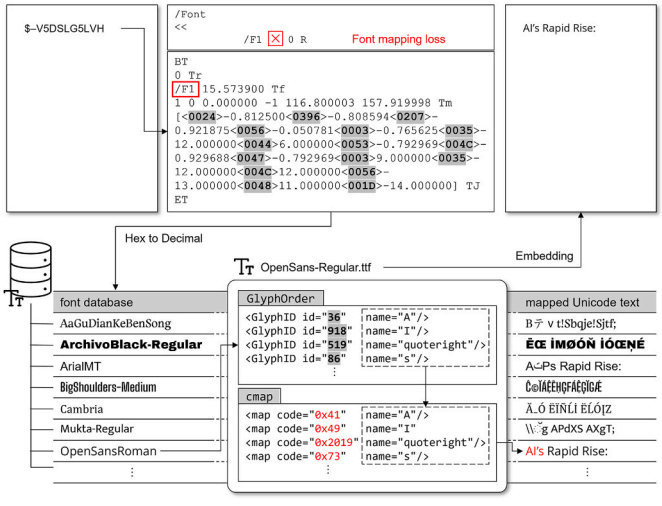

Fig. 3. Example of reconstructing font mappings using a template PDF file; this step enables a corrupted PDF file to be repaired even when the object information corresponding to the font key named `/F1` is missing.

languages and select the font with the highest contextual plausibility. This process usually applies to Print to PDF files or the files with page object corrupted.

3.7. Generating repaired PDF files

Final step integrates the recovered content streams, font objects, and image objects into the template PDF files to produce the repaired document. Each page includes a `/Contents` entry referencing the content streams, a `/MediaBox` entry defining the page boundaries, and a `/Resources` entry referencing the associated image and font resources. If resource data are not corrupted, they are recovered from the corrupted PDF files and then inserted to the repaired PDF files.

Once all content and resource objects have been successfully reconstructed in accordance with the defined relationships, a single, complete PDF file is produced. The repaired file renders correctly in standard PDF readers and preserves the original document to the greatest extent possible.

4. Implementation

This section describes the implementation of REPDF, which automates the sequence of procedures in the proposed PDF repair framework.

4.1. Design of REPDF

To demonstrate the practical effectiveness of our proposal, we developed a proof-of-concept tool named REPDF using Python (v3.12.8). Specifically, fonttools (v4.55.8) is utilized to process font files and extract necessary information.

In addition, pikepdf (v9.5.1) is employed to create and manipulate objects in the template PDF files used for repairing font mappings. The tool is deployed as a web application with a Python backend. The web interface accepts a corrupted PDF as input and provides a repaired PDF as output.

To automate the selection from the candidate fonts mentioned in Section 3.6, a score-based matching system was implemented. It utilizes language-specific word dictionaries to award points for matching words, while applying penalties for factors like mixed languages and uncommon special characters. In particular, for cases where `/ToUnicode` is intact, the system extracts the body text using the Unicode mapping information and uses extracted text as a custom word dictionary to achieve high accuracy.

A separate Python script was also implemented to fully automate the construction of template PDF files, enabling the automatic addition of new font files. This script handles the entire process: creating or opening an MS Word (.docx) file, inserting multilingual text with the new fonts, saving it as a PDF, extracting the newly generated font-related object information, and updating the JSON file that contains all font details.

Availability. REPDF is available at a website, https://repdf.site, with sample files so that anyone can test its effectiveness.

4.2. Usage and outputs

Fig. 4 shows the usage flow and outputs of REPDF: (a) initial interface with access to the upload area; (b) selection and upload of a corrupted PDF via drag and drop or the `Select File` button; (c) automated repair that reconstructs cross-reference tables, reassembles object streams, and remaps fonts and resources; (d) completion view with a downloadable repaired file and a brief summary of file size and page count.

5. Evaluation

This section outlines the evaluation strategy and presents results that demonstrate the effectiveness of our approach. We first define a set of PDF-corruption scenarios to quantitatively assess performance, and then compare our results with those of existing tools.

5.1. Defining PDF corruption scenarios

To evaluate performance, we define ten real-world PDF corruption scenarios, including structural alterations, header removal, loss of font streams, removal of Unicode mappings, and partial file truncation. Common causes include partial data loss on file systems, encoding or decoding errors during transmission, and interrupted or incomplete downloads. The scenarios are defined as follows:

5.1.1. Header corruption (C1)

Header damage prevents standard viewers from opening the document, even when the body content remains largely intact. It is implemented by overwriting up to 12 bytes of the PDF header with random values.

5.1.2. XRef table removal (C2)

In this scenario, the cross-reference (XRef) table has been removed, so the starting byte offsets of indirect objects are missing. Nevertheless, object positions can still be identified using the `obj` and `endobj` tags, allowing the content of each object to be parsed.

5.1.3. Trailer section damage (C3)

This type corresponds to damage to the trailer section located at the end of the file. The trailer provides offsets of the XRef table and certain objects. Our proposed method can repair it by utilizing the `obj` and `endobj` tags to reconstruct the associated objects.

5.1.4. Page tree broken (C4)

The page tree indicates the relationships among page objects. When it is broken, pages may render incorrectly or out of order, while the content of each page remains intact. It is implemented by corrupting the page tree through deleting a random-length span around the `/Pages`, `/Kids`, and `/Count` entries.

5.1.5. Object tag removal (C5)

When the object tag is removed, the corresponding object number is lost. If the affected object is referenced indirectly by other objects, the

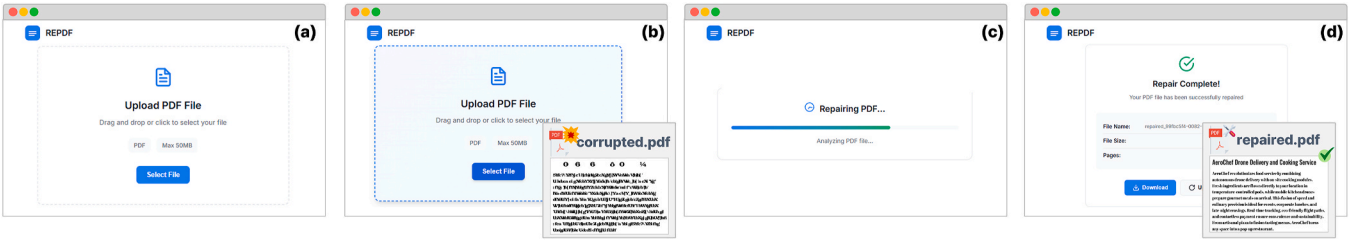

Fig. 4. Workflow for repairing a corrupted PDF file using the developed REPDF web application (https://repdf.site).

Table 1
Summary of the dataset by PDF corruption type; the number of files indicates the count for each type, with a total of 1,000 files (500 created using 'Save As' and 500 using 'Print to PDF').

Type	Creation Method	# of Files (Text only)	# of Files (Text + Image)	Language
C1–C10	Save As	40	10	en, fr, es, ar, hi, zh
	Print to PDF	40	10	

PDF reader may fail to render partial data. This scenario is realized by randomly selecting an object and removing its header, including the object number and the `obj` tag.

5.1.6. Font mapping loss (C6)

This scenario occurs when the reference to a font object is corrupted. Although both text content and font data are present, the PDF reader may fail to render the text correctly. It is implemented by deleting the `/Font` object inside a page's `/Resources`, thereby breaking the mapping between the `/Page` object and the referenced `/Font` objects.

5.1.7. Font stream removal (C7)

When the embedded font stream is deleted, the glyph data required for rendering are lost, and thus the text may not be rendered correctly. It is carried out by deleting the `/FontFile2` stream data.

5.1.8. Font resources removal (C8)

This scenario considers cases where embedded font streams and `/ToUnicode` objects are deleted. Once most text-rendering information is removed, the likelihood of correct rendering decreases significantly. It is implemented by deleting both the `/FontFile2` stream and the `/ToUnicode` stream.

5.1.9. Zlib tampering (C9)

This scenario involves byte-level modifications within zlib-compressed streams. Upon decompression, such corruption causes decoding errors and often results in partial or complete data loss. It is realized by modifying a random byte within a zlib-compressed stream.

5.1.10. Tail truncation (C10)

In this scenario, the beginning of the file remains while the latter part is missing, typically due to downloads interrupted by network errors or unstable connections. It is implemented by truncating the file, cutting off its tail and retaining approximately 70 % of the original bytes.

5.2. Dataset creation

According to the above-mentioned corruption scenarios, 1,000 PDF files (50 base files × 10 scenarios × 2 creation methods) were created and used in the experiments. The dataset is publicly available online[1] and is summarized in Table 1.

The dataset consists of PDF files containing multilingual text to evaluate repairing performance considering diverse character systems. It includes six languages: English (en), French (fr), Spanish (es), Arabic (ar), Hindi (hi), and Chinese (zh), that represent alphabetic, syllabic, and logographic writing systems with varying character encodings. While different fonts generally produce distinct Unicode mappings, it is also possible for two fonts within the same language to share identical mappings. To account for such cases, the dataset incorporates multilingual fonts with diverse mapping configurations, including five representative fonts for each language. When selecting fonts, we ensured that both text-based and path-based representations, as introduced in Section 2, were included.

The content for each language was automatically generated using the GPT-5 model (OpenAI, 2025) to ensure grammatical accuracy and a natural tone. To support cross-lingual evaluation, each document comprises six pages, with identical content presented in a different language on each page. Two of the six pages include simple images to assess image recovery performance. All images were generated using the same GPT model and saved in diverse formats (e.g., .jfif, .jpeg, .png, .svg, .bmp), then randomly assigned within each six-page document. Leveraging GPT model for text and image generation enabled the creation of a PDF document dataset encompassing diverse layouts and content structures, such as multiple image formats and tabular elements.

To create sample files with AI-generated content, we used Microsoft Word to save them as individual PDF files using two methods, 'Save As' and 'Print to PDF'. These PDF creation methods reflect document creation workflows commonly used in real-world environments.

5.3. Evaluation metrics

This study defines separate evaluation metrics for text and image recovery rates to quantitatively assess the performance of the proposed methodological framework. Each metric is measured under various conditions such as corruption type, language, and creation method, and is designed to provide quantitative outputs for comparative analysis.

5.3.1. Text recovery rate

The text recovery rate is calculated as the percentage of words in the repaired document that exactly match those in the original document, based on the total number of words in the original. Matching is determined on a word-by-word basis; any differences in spelling or phrasing are considered recovery failures. The formula is as follows:

$$\text{Text Recovery Rate} = \frac{\#\ \text{of correctly recovered words}}{\#\ \text{of words in original document}} \times 100$$

This metric is measured according to language and corruption type, and is applied consistently across tool performance comparisons. A Python script was developed to automate the evaluation process by using the OCR engine of the Google Cloud Document AI API (Google Cloud, 2025) through extracting words from repaired PDF files and comparing them with the originals. In other words, the OCR results of the original PDF files were used as ground truth and compared with those of the recovered PDF files to evaluate recovery performance. Applying the same OCR-based character extraction method to both the original and recovered documents ensured that glyphs were extracted and compared using a consistent criterion.

5.3.2. Image recovery rate

For image recovery, evaluation is based on (a) whether the image data was fully and correctly extracted, and (b) whether it was properly rendered within the PDF document. For both 'Save As' and 'Print to PDF', the proportions of successfully extracted and saved images and correctly rendered images are measured for 10 files per corruption type. For (a), any image with even slight damage was excluded from the count, while for (b), images were counted only if they were both undamaged and correctly rendered in their expected positions.

Since the tool developed in this study includes a step to extract all images embedded in the target corrupted PDF file during the repair process, the extracted images were saved to a specified path and evaluated according to criterion (a). Since other tools do not provide this functionality, the comparative evaluation was conducted based solely on criterion (b). In addition, the metric for image recovery is assessed

[1] https://github.com/dfrc-korea/REPDF.

Table 2
Text recovery rates of the proposed method by corruption type and language (%).

Corruption Type	Save As						Print to PDF					
	English	Chinese	Hindi	Spanish	French	Arabic	English	Chinese	Hindi	Spanish	French	Arabic
C1: Header corruption	100	100	100	100	100	100	100	100	100	100	100	100
C2: XRef table removal	100	100	100	100	100	100	100	100	100	100	100	100
C3: Trailer section damage	100	100	100	100	100	100	100	100	100	100	100	100
C4: Page tree broken	98.71	99.84	99.98	99.90	99.81	99.83	100	100	100	100	100	100
C5: Object tag removal	100	99.89	100	100	100	99.98	100	100	100	100	100	100
C6: Font mapping loss	97.14	94.05	97.94	97.78	93.41	33.46	99.41	95.83	98.13	93.75	89.35	39.94
C7: Font stream removal	100	99.87	99.92	98.81	98.51	99.85	99.44	97.52	99.68	98.97	98.08	99.78
C8: Font resources removal	100	99.87	99.92	98.81	98.51	99.85	99.41	95.83	97.98	92.45	88.76	38.51
C9: Zlib tampering	64.52	66.59	56.74	65.41	52.98	54.43	65.77	49.94	67.64	71.69	52.92	52.71
C10: Tail truncation	100	99.95	99.98	99.04	98.59	99.85	60.30	79.68	38.12	32.38	1.58	0.05

Table 3
Image recovery and rendering rates of the proposed method by corruption type (%).

Corruption Type	Save As		Print to PDF	
	Recovery	Rendering	Recovery	Rendering
C1: Header corruption	100	100	100	100
C2: XRef table removal	100	100	100	100
C3: Trailer section damage	100	100	100	100
C4: Page tree broken	100	90	100	100
C5: Object tag removal	100	100	100	100
C6: Font mapping loss	100	100	100	100
C7: Font stream removal	100	100	100	100
C8: Font resources removal	100	100	100	100
C9: Zlib tampering	80	60	60	40
C10: Tail truncation	100	100	55	50

individually for each image object contained within each document (two images in total per document), allowing for a quantitative evaluation of visual content recovery rather than relying on qualitative judgment.

Both metrics for text and image data are analyzed based on the experimental results summarized in Tables 2 and 3, and the same criteria are applied in the comparative analysis between the proposed method and existing tools.

5.4. Results in recovering text data

Table 2 summarizes the text recovery rates (%) for ten types of PDF file corruption (C1–C10). The results show clear variation by corruption type and creation method and demonstrate the effectiveness of the proposed method.

For C1, header damage prevents viewing but leaves body content intact, achieving in 100 % recovery across all languages. C2 and C3 affected only structural metadata, so recovery is likewise complete (100 %). Even with such structural damage for C4, nearly complete recovery was possible across all languages under both 'Save As' and 'Print to PDF' methods. We observed a small reduction in recovery rate ($\leq$ 2 %) from those files where a `/Pages` object and a `/Page` object were stored adjacent to each other. This occurred because corrupting `/Pages` inadvertently removed font and image-mapping entries from the adjacent `/Page` object.

For C5, our method achieved near-100 % recovery across languages (10/12 at 100 %; 99.89 % and 99.98 % otherwise). Its byte-level analysis and stream-section scanning enable successful recovery from object tag removal.

C6–C8 are all corruption cases related to font data. With mapping information missing from the page object (C6), average recovery rate achieved above 90 % by utilizing a font database and automatic font mapping. However, this font mapping system showed limited accuracy for Arabic, as its inflectional nature causes word forms to change depending on grammatical context, making dictionary-based word matching unreliable. While PDF files in C7 do not have `/FontFile2` stream, the overall rate achieved nearly 100 %. With `/ToUnicode` remained, the text could be decoded to actual characters.

The files in C8, where even `/ToUnicode` map is deleted, showed distinct recovery rate depending on the PDF creation method. Because PDF files created using the 'Save As' method contains the actual font name, it allowed REPDF to map a proper font file using a font database. However, in files generated via 'Print to PDF', font names keep changing (e.g., `CIDFont+F1`), it showed a similar recovery rate with C6 that applies automatic font mapping process.

For C9, the average rate is 60.11 %, lower than in other scenarios. This reflects corruption of compressed streams containing the actual text, which creates unrecoverable segments depending on the location of the corruption. Nevertheless, REPDF's byte-level decompression, checking for errors byte-by-byte to prevent total data loss, enables a substantial recovery rate.

For C10, recovery rates differed markedly by creation method: 'Save As' averaged 99.57 %, whereas 'Print to PDF' averaged 35.35 %. We hypothesize that this gap arises from layout differences: 'Print to PDF' tends to place text streams and font resources near the end of the file.

In summary, text recovery was near-complete (100 %) for structural corruption scenarios (C1–C5), and even under content corruption scenarios (C6–C10), stream analysis and font inference achieved substantial results.

5.5. Results in recovering image data

REPDF achieved average recovery rate of 94.75 % and average rendering rate of 92 %, as summarized in Table 3. For images, corruption falls into two broad categories: (1) damage to the image XObject stream that contains the image data, and (2) damage within the page content stream that controls image rendering (e.g., broken `/Resources` entries or `Do` operations). In the former case, the image cannot be extracted or only a corrupted file is obtained; in the latter, the image file can be extracted correctly but is not rendered in the PDF.

For C1–C5, where only structural metadata was damaged and the image data remained intact, REPDF achieved 100 % image recovery. In C4, page-tree corruption removed certain image references, resulting in image-rendering failures in some files. By contrast, C6–C8, where corruption affects only fonts or text-related page resources, showed complete image extraction and rendering across all datasets.

C9 affects the image data or the content stream that renders images. As a result, the number of recovered images exceeds the number of rendered images. This indicates that the image data often remain intact, whereas the rendering operators in the content stream are corrupted. In some files, images remained structurally intact and visually recognizable despite substantial color corruption. For C10, image recovery succeeded for 'Save As' with 100 % recovery rate. However, 'Print to PDF' showed fewer extracted or rendered images, because its main content and image streams are typically located near the end of the file.

In summary, REPDF shows strong image-recovery robustness under structural corruption (C1–C5) and font-loss cases (C6–C8). Even with

image related data corruption (C9–C10), it extracts partially distorted or discolored images and renders them at their original positions, thereby preserving layout information.

5.6. Comparison with existing tools

In our study, we compared the performance of the developed tool against existing PDF repair tools, including iLovePDF Desktop (iLovePDF, 2025), Repairit (Wondershare, 2025), and PDF24 Creator (Geek Software GmbH, 2025), as listed in Table 4. As shown in Table 5, REPDF achieved the highest recovery rates in most corruption types among the evaluated tools.

iLovePDF demonstrated effective file structure repair, but it showed clear limitations regarding font resource-related damages. Specifically, it achieved a high recovery rate for file structure damages (C1–C5), excluding page tree corruption (C4), and also demonstrated performance comparable to our tool for types C9 and C10. On the other hand, its recovery rate for font resource damages (C6–C8) only reached a maximum of 56.86 %, and the performance dropped to less than 5 % for files created by 'Print to PDF' that had lost both their embedded fonts and Unicode mapping information.

Wondershare showed an overall performance pattern similar to iLovePDF. While it was effective against file structure damage, it revealed the same limitation where the recovery rate significantly decreased for font resource-related damages. A unique limitation of this tool was observed in C10 for files created by 'Save As', where the repaired output itself contained errors, making them unprocessable by the OCR API and resulting in no valid recovery rate being recorded.

PDF24 showed the lowest performance among the tools compared. For some corruption types, such as C4, C5, and C10, repair was impossible as the tool failed to read the corrupted files. Similar to the other tools, PDF24 was also most vulnerable to font resource-related damages. However, it showed an exceptional strength specifically for C9, where it recorded higher performance than the other two tools.

REPDF, the tool we developed, not only maintains high performance for structure damages (C1-C5) but also shows superior performance on font-related resource damages (C6-C8), which other tools struggle to handle. Specifically, for 'Print to PDF' files that do not record actual font names, our tool achieved an average recovery rate of over 80 % by attempting automatic font mapping reconstruction.

Overall, for some corruption types, other tools produced repaired files with no normal text or images across the entire 1,000 files, or failed to read the damaged PDF entirely. These tools also showed significant performance drops on specific corruption types. In contrast, our tool successfully repaired files across all corruption types and consistently demonstrated superior performance by leveraging the available residual data, confirming its broader coverage compared to the other tools.

6. Discussion

In this study, we introduced a novel framework that supports structure-based PDF repair and implemented a proof-of-concept tool, REPDF. To evaluate the tool, we designed possible corruption scenarios. However, for certain scenarios in which object tags (C5) or font-related streams (C7, C8) are damaged, the assumption of damage to a single component makes it unclear how frequently such damage patterns occur in real-world environments. Using corrupted PDF files collected from real-world cases as datasets, or constructing datasets that reflect realistic damage patterns derived from analysis of such files, could further enhance evaluation reliability. Additionally, the PDF files used in this study were generated using only Microsoft Word. However, PDF files can also be produced through various other methods, such as the use of different PDF creation tools, or the conversion of scanned documents into PDF format. Evaluating the proposed framework on PDF files produced through such diverse methods would help validate its broader applicability and further improve the framework.

In cases involving font-related corruption (C6–C8), existing tools are unable to recovery text because they lack the necessary font resources. In contrast, the proposed tool generates a template PDF that includes all font objects in the font database and embeds complete glyph and

Table 4
Existing PDF repair tools used for performance comparison.

Vendor	License	Tool	Version
iLovePDF	Commercial	iLovePDF Desktop	2.1.28.0
Wondershare	Commercial	Repairit	6.5.15
Geek Software GmbH	Free	PDF24 Creator	11.28.2

Table 5
Comparison of text and image recovery rates (%) with existing tools.

Corruption Type	Data Type	Save As				Print to PDF			
		Ours	iLovePDF	Wondershare	PDF24	Ours	iLovePDF	Wondershare	PDF24
C1: Header corruption	Text	100	99.99	99.90	99.90	100	100	99.90	99.90
	Image	100	100	100	100	100	100	100	100
C2: XRef table removal	Text	100	99.99	100	99.90	100	100	100	99.90
	Image	100	100	100	100	100	100	100	100
C3: Trailer section damage	Text	100	99.99	100	99.90	100	100	100	99.90
	Image	100	100	100	100	100	100	100	100
C4: Page tree broken	Text	99.74	96.25	92.43	0	100	100	99.90	0
	Image	90.00	90.00	90.00	0	100	100	100	0
C5: Object tag removal	Text	99.99	99.99	99.90	0	100	100	98.00	99.90
	Image	100	100	100	100	100	100	100	100
C6: Font mapping loss	Text	88.12	40.90	40.90	37.17	88.13	4.87	4.87	5.00
	Image	100	100	100	100	100	100	100	100
C7: Font stream removal	Text	99.51	56.86	56.86	42.42	99.21	34.32	34.32	16.00
	Image	100	100	100	100	100	100	100	100
C8: Font resources removal	Text	99.51	41.91	41.91	42.42	87.57	4.38	4.38	16.00
	Image	100	100	100	100	100	100	100	100
C9: Zlib tampering	Text	58.77	53.09	54.25	58.66	62.70	59.68	57.08	60.43
	Image	60.00	60.00	40.00	60.00	40.00	35.00	35.00	40.00
C10: Tail truncation	Text	99.58	69.92	0	70.53	30.49	31.47	31.52	0
	Image	100	100	90.00	100	50.00	45.00	50.00	0

Unicode mapping information. As a result, while other tools achieved an average recovery rate of up to 56 %, our tool consistently reached nearly 90 %. This highlights the critical importance of building and utilizing a pre-constructed font database to improve recovery performance. However, our approach is limited when the damaged font file does not exist in the pre-built font database. Accordingly, constructing a more systematic and extensible font database to broaden recovery coverage across diverse languages and fonts remains an important challenge.

Accurate font mapping requires an appropriate font selection process. The currently implemented automatic font mapping has proven effective not only for Latin-based languages but also for CJK languages that contain a large number of characters. However, dictionary-based font mapping becomes challenging in languages like Arabic, where word forms change depending on the grammatical context. Therefore, for accurate recovery, an interactive step is required in which the user reviews the recovered text and selects the most contextually appropriate font. In the future, if this step can be replaced with AI-based contextual understanding (Kim et al., 2025), both usability and recovery efficiency are expected to be improved.

7. Conclusion and future directions

This study proposed a new PDF repair framework through analysis and extraction of key objects. The framework was evaluated using a dataset comprising six multinational fonts with diverse Unicode mapping information. A total of 1,000 corrupted files were created by considering ten types of corruption scenarios and two document creation methods, and text recovery rates by language as well as image recovery and rendering rates were measured. The proposed method outperformed three existing tools by achieving the highest recovery rates across the majority of corruption types.

Our developed tool reconstructs the internal structure by leveraging the maximum number of remaining objects extracted using the `objstream` patterns, achieving high recovery rates even when critical metadata such as the *header*, *XRef table*, or *trailer* are corrupted. Experiments on datasets with structural damage confirmed that all text and image data could be successfully recovered. In particular, the tool achieved high recovery rates even when font information such as `/ToUnicode` or `/FontFile2` is missing, by utilizing a pre-constructed font database. Font recovery is performed by generating text based on the Unicode mapping information of each font in the database, followed by automatic or manual comparison to select the font that yields the most meaningful result.

In the field of digital forensics, large numbers of PDF documents are collected and examined as digital evidence. The proposed repair framework contributes to the extraction of meaningful data by salvaging the maximum remaining information from corrupted PDF files. Future work will focus on expanding the font database and incorporating methods such as AI, which can accurately interpret language-specific features and contextual information, with the goal of further increasing recovery rates and reducing processing time.

Acknowledgments

This work was supported by the Commercializations Promotion Agency for R&D Outcomes (COMPA) grant funded by the Ministry of Science and ICT (MSIT, Korea) & Korean National Police Agency (KNPA, Korea) (No.RS-2025-02653744).

References

Al-Saleh, M., Al-Sharif, Z., 2013. Ram forensics against cyber crimes involving files. In: The Second International Conference on Cyber Security, Cyber Peacefare and Digital Forensic (Cybersec2013). pp. 189–197.

Al-Sharif, Z.A., Al-Senjalawi, R., Alzoubi, O.A., 2024. The effects of document's format, size, and storage media on memory forensics. Forensic Sci. Int.: Digit. Invest. 48, 301692. https://doi.org/10.1016/j.fsidi.2024.301692.

Ali, N.U.A., Iqbal, W., Afzal, H., 2022. Carving of the OOXML document from volatile memory using unsupervised learning techniques. J. Inf. Secur. Appl. 65, 103096. https://doi.org/10.1016/j.jisa.2021.103096.

Al-Sharif, Z.A., Odeh, D.N., Al-Saleh, M.I., 2015. Towards carving pdf files in the main memory. In: The International Technology Management Conference (ITMC2015). pp. 24–31.

Chen, M., Zheng, N., Xu, M., Lou, Y., Wang, X., 2008. Validation algorithms based on content characters and internal structure: the PDF file carving method. In: 2008 International Symposium on Information Science and Engineering, pp. 168–172. https://doi.org/10.1109/ISISE.2008.209.

Chung, H., Park, J., Lee, S., 2011. Forensic analysis of residual information in adobe PDF files. In: Future Information Technology (FutureTech 2011), pp. 100–109. https://doi.org/10.1007/978-3-642-22309-9_12.

Geek Software GmbH, 2025. PDF24 Tools: Free PDF Solutions for all PDF Problems. URL: https://tools.pdf24.org. (Accessed 10 October 2025).

Google Cloud, 2025. Document AI. URL: https://cloud.google.com/document-ai. (Accessed 10 October 2025).

iLovePDF, 2025. iLovePDF—Online PDF Tools for PDF Lovers. URL: https://www.ilovepdf.com. (Accessed 10 October 2025).

International Organization for Standardization, 2020. Document Management — Portable Document Format — Part 2: PDF 2.0. URL: https://pdfa.org/resource/iso-32000-2/.

Joun, J., Lee, S., Park, J., 2023. Data remnants analysis of document files in windows: Microsoft 365 as a case study. Forensic Sci. Int.: Digit. Invest. 46, 301612. https://doi.org/10.1016/j.fsidi.2023.301612.

Kim, J., Jeong, B., Park, S., Lee, S., Park, J., 2025. Your forensic AI-assistant, SERENA: systematic extraction and reconstruction for enhanced A2P message forensics. Forensic Sci. Int.: Digit. Invest. 53, 301931. https://doi.org/10.1016/j.fsidi.2025.301931.

Lam, D., Li, L., Anderson, C., 2024. PDF investigation with parser differentials and ontology. IEEE Trans. Inf. Forensics Secur. 19, 7774–7782. https://doi.org/10.1109/TIFS.2024.3445733.

Liu, C., Lou, C., Yu, M., Yiu, S., Chow, K., Li, G., Jiang, J., Huang, W., 2021. A novel adversarial example detection method for malicious PDFs using multiple mutated classifiers. Forensic Sci. Int.: Digit. Invest. 38, 301124. https://doi.org/10.1016/j.fsidi.2021.301124.

Nassif, L.F.d.C., Hruschka, E.R., 2013. Document clustering for forensic analysis: an approach for improving computer inspection. IEEE Trans. Inf. Forensics Secur. 8, 46–54. https://doi.org/10.1109/TIFS.2012.2223679.

OpenAI, 2025. ChatGPT. URL: https://chatgpt.com. (Accessed 10 October 2025).

Ravi, A., Kumar, T.R., Mathew, A.R., 2016. A method for carving fragmented document and image files. In: 2016 International Conference on Advances in Human Machine Interaction (HMI), pp. 1–6. https://doi.org/10.1109/HMI.2016.7449170.

Taşdelen, K., Süzen, A., 2022. Analysing and carving MS word and PDF files from RAM images on windows. Tehnički vjesnik - Technical Gazette 29, 1714–1720. https://doi.org/10.17559/TV-20210218122046.

Trojahn, M., Pan, L., Schmidt, F., 2013. Developing a cloud computing based approach for forensic analysis using OCR. In: 2013 Seventh International Conference on IT Security Incident Management and IT Forensics, pp. 59–68. https://doi.org/10.1109/IMF.2013.11.

Vol, M., Krutko, A., Stefanovitch, N., Postanogov, D., 2018. Automatic recovery of corrupted font encoding in PDF documents using CNN-Based symbol recognition with language model. In: 2018 13th IAPR International Workshop on Document Analysis Systems (DAS), pp. 121–126. https://doi.org/10.1109/DAS.2018.64.

Wondershare, 2025. Wondershare Official Website: Creativity, Productivity, Utility. URL: https://www.wondershare.com. (Accessed 10 October 2025).

Yoo, B., Park, J., Lim, S., Bang, J., Lee, S., 2012. A study on multimedia file carving method. Multimed. Tool. Appl. 61, 243–261. https://doi.org/10.1007/s11042-010-0704-y.

Zia, M.A.M., Adedayo, O.M., 2025. Tool type identification for forensic digital document examination. Forensic Sci. Int.: Digit. Invest. 54, 301972. https://doi.org/10.1016/j.fsidi.2025.301972.