GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar with Practical: Scalable Computing Systems and Applications in AI, Big Data and HPC

Comparing Open-Source File Recovery Techniques on `ext4`: Hash-Exact Evaluation and a Combined-Tool Workflow

---

Salaheldin Soliman

MatrNr: [25845983]

Supervisor: Patrick Höhn

Georg-August-Universität Göttingen
Institute of Computer Science

September 30, 2025

Abstract

We evaluate open-source file recovery techniques for `ext4` under realistic data loss: random deletions and byte-level corruptions (first 512 B). The study compares metadata-aware extraction (The Sleuth Kit), content-based carving (PhotoRec, Scalpel), and a combined workflow that chains repair, metadata extraction, and carving with hash de-duplication. We score recovery strictly by SHA-256 equality against a post-simulation manifest [1]. Theoretical context draws on filesystem analysis, file carving, and repair/journaling sources [2–6]. Across images from 100 MB to 50 GB, results show that recovery is scenario-specific and that combining complementary techniques yields the broadest exact-match coverage.

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☑ During brainstorming
- ☐ When creating the outline
- ☐ To write individual passages, altogether to the extent of 0% of the entire text
- ☐ For the development of software source texts
- ☑ For optimizing or restructuring software source texts
- ☑ For proofreading or optimizing
- ☐ Further, namely: -

I hereby declare that I have stated all uses completely.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables

List of Figures

1 Introduction

When files are deleted or a filesystem is corrupted, data often persists until blocks are reused. Recovery proceeds via two main strategies: (i) *metadata-based* methods that parse filesystem structures to restore referenced content [2], and (ii) *content-based* carving that scans raw media for file signatures when metadata is absent or damaged [3]. In practice, responders also rely on partition/filesystem repair and journaling artifacts to restore access before carving [5, 6]. We adopt a conservative metric: a recovery counts only if recovered bytes exactly match the simulated ground-truth hash (SHA-256) [1].

2 Recovery Techniques

2.1 Metadata-Aware Filesystem Recovery

Filesystem parsers enumerate superblocks, inodes, directory entries, allocation state, and block references. When references remain valid, recovery preserves filenames/timestamps and can re-extract allocated objects byte-for-byte (hash-identical). The Sleuth Kit (TSK) provides a well-tested toolchain for this workflow: `fls` lists directory trees (including deleted entries when metadata persists), `istat` inspects inodes, `icat` extracts content by inode, and `tsk_recover` performs bulk export of allocated files [2, 7].

On `ext4`, the primary on-disk structures are the superblock, group descriptors, inode tables, extent trees (which map file offsets to physical block runs), and directory entries [4]. Deletion typically clears the directory entry and may also drop or zero the extent mapping in the inode. Two implications follow.

- If an inode still has valid extent mappings and the file is marked allocated, TSK will re-extract the exact bytes (this explains near-100% exact matches for the corrupted-but-allocated class when only the first 512 B were altered in our simulation).
- If pointers are removed or the inode is re-used, byte-exact recovery via metadata becomes impossible; responders must switch to carving or journal-aided techniques [5].

Practical strengths and limits.

- **Strengths:** preserves filenames, paths, times; fastest end-to-end for allocated content; results are byte-identical when metadata is consistent.
- **Limits:** weak "undelete" on `ext4` once extent mappings are cleared; fragmented deleted files cannot be reconstructed without valid metadata; directory entries can disappear early, even when data still resides on disk.
- **Typical tools:** `tsk_recover` for bulk export; `fls`/`icat`/`istat` for targeted inspection; used first in the combined workflow to capture all allocated content.

2.2 Content-Based Carving

Carvers ignore the filesystem and scan raw media for recognizable byte patterns (magic headers, optional footers, or internal structure) to delimit files. JPEG, for instance, commonly starts with `FF D8` and ends with `FF D9`. When footers are absent (e.g., many container formats), advanced carvers validate internal structure to decide plausible length or stop at the next header [3].

Two complementary carvers used here are PhotoRec and Scalpel. PhotoRec is format-rich and validation-heavy; it attempts to follow structure, avoid false positives, and can sometimes thread simple multi-fragment recoveries [8]. Scalpel emphasizes speed and a concise, auditable configuration (`scalpel.conf`) for the exact set of signatures you care about; it parallelizes well and is easy to integrate in scripted pipelines [9].

Important realities for exact hashing.

- **Fragmentation:** when deleted files are fragmented, carving often yields partial content. Our strict SHA-256 metric credits only byte-identical results, so such partials score as misses even if visually usable [10].
- **Headers damaged:** deliberate header overwrites (like our 512 B corruption) thwart signature detection; structure-aware validation sometimes recovers, but exact-byte matches are rare in this case.
- **Containers:** formats such as PDF or ZIP/DOCX benefit from internal consistency checks (object tables, central directory). Enabling validators improves precision but may increase runtime [11].
- **Provenance:** carving usually loses original names/paths. Pipelines should preserve tool provenance and de-duplicate by hash to avoid double counting across carvers.
- **Secondary metrics:** fuzzy hashing (e.g., ssdeep) can cluster near-identical outputs, but is not evidentiary; we treat it as a potential secondary metric outside exact-match scoring [12].

Recommended practice: run carving after metadata-based export to focus on unallocated space; use a curated `scalpel.conf` to add or narrow types; record all outputs and hashes for auditability.

2.3 Journal-Aided Recovery on `ext4`

Ext-family journals (JBD2 on `ext4`) record metadata updates and, in `data=ordered` mode, ensure that file data hits disk before metadata commits. While journals do *not* guarantee full data copies, they frequently preserve enough metadata (directory entries, inode/extent snapshots) for recent deletions to be undone. Tools like `extundelete` parse the journal and related structures to resurrect names and block lists that pure carving cannot reconstruct [5, 13].

Usage notes.

- Work on a forensic image, not the live filesystem; mount read-only. Journal blocks must not be overwritten by continued use.
- Journal success is time-bound: once journal segments wrap or are reused, prior metadata evidence disappears.

- Journal recovery complements TSK: if it reconstructs inode-to-block mappings for recently deleted files, those files can then be re-extracted byte-for-byte.

2.4 Structural Repair (Partitions, Superblocks) and Imaging

Before per-file recovery, responders frequently restore basic access to volumes. Partition tables (MBR/GPT) or boot sectors can be repaired with TestDisk to make images mountable again [6]. Within `ext4`, backup superblocks and `e2fsck` can correct some structural inconsistencies. Use evidence containers and provenance: store images and derivatives in AFF4 where possible and log processing with DFXML [14, 15]. Because these actions may write to the medium, the standard operating procedure is:

- Acquire a full disk image first (e.g., with `ddrescue`), keeping the original pristine and read-only.
- Attempt non-destructive repair on a *copy* of the image; only when the volume becomes readable do you proceed to TSK and carving stages.
- Record tool versions, parameters, timestamps, and any changed bytes to preserve the chain of custody.

This "repair → metadata export → carving → de-duplication" order underlies the Combined workflow and explains its broader coverage in our results.

3 Methodology and Pipeline

We follow a scripted, end-to-end pipeline (code and artifacts: `https://github.com/salaheldinsoliman/forensics`):

- **Image construction.** Create `ext4` images at 100 MB, 1 GB, 5 GB, 20 GB, and 50 GB. Fill to ∼95% from a seed corpus of JPG, PDF, DOCX, and TXT documents (see Listing 1).
- **Baseline manifest.** Traverse each image to record a baseline manifest containing the image identifier, size, file path, action (`keep`), and SHA-256 hash per file (see Listing 2).
- **Loss simulation.** Randomly select 20% of files to delete and 10% to corrupt by overwriting the first 512 B with zeros. Recompute hashes for all remaining files and emit a simulated manifest labeling each entry as `keep`, `deleted`, or `corrupted`. For deleted entries we store the original content hash (`orig_hash`); for keep/corrupted we store the post-simulation hash (`sim_hash`). See Listing 3.
- **Tool execution.**
  - **TSK (metadata)**: extract allocated files; compute a recovered-manifest of SHA-256 hashes [7] (Listing 4).
  - **PhotoRec (carving)**: run signature-based carving over the raw image; record recovered hashes. We used a prepared command file for non-interactive, reproducible runs [8] (Listing 5).
  - **Scalpel (carving)**: carve using a curated configuration for selected types; record recovered hashes [9] (Listing 6).
  - **Combined workflow**: optional repair (*e.g.*, TestDisk/`fsck`) → TSK (allocated) → PhotoRec (unallocated) → Scalpel (targeted types) → SHA-256 de-duplication across outputs [6].
- **Scoring (hash-exact).** A file counts as recovered in category c in {keep, deleted, corrupted} if a recovered output's content hash matches the class target: `sim_hash` for `keep` and `corrupted`, and `orig_hash` for `deleted`. Filenames/paths are ignored. This is conservative and evidentiary in spirit [1, 16, 17]. Public reference images and ground truth are available via the NIST CFReDS portal [18]. See Listing 7 for the verification script.

All tests were executed on shared research hardware; images ranged from 100 MB to 50 GB on `ext4`. We keep figure labels generic and describe the environment in text rather than on plots.

4 Results

Figures report exact-match percentages and runtime versus image size; each plot compares individual tools and the Combined workflow.

4.1 Overall recovery vs. size

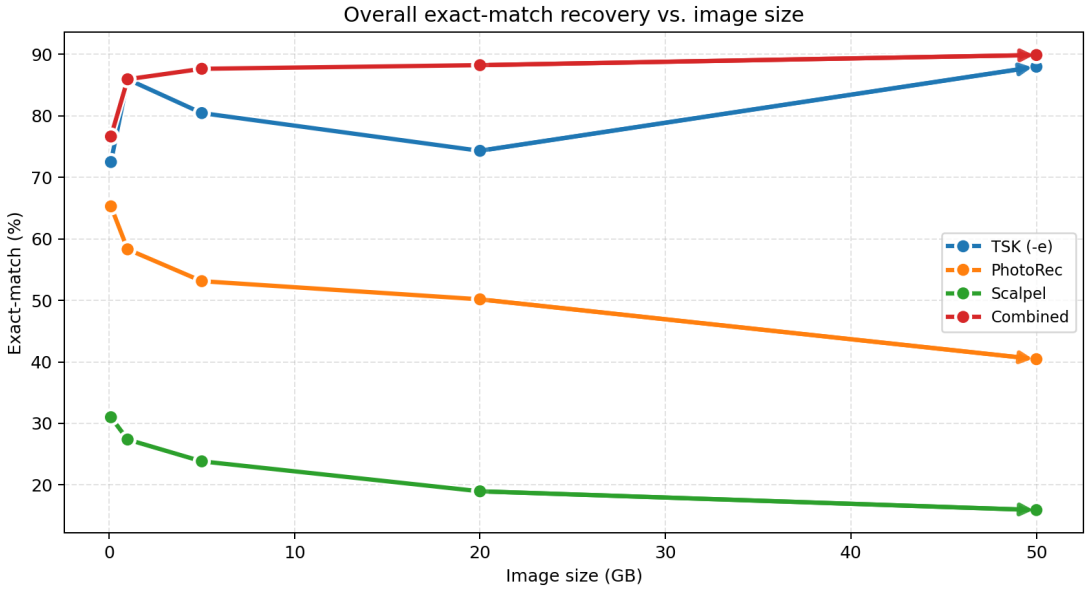

Figure 1: (ext4, 100 MB–50 GB) Overall exact-match recovery vs. image size. Combined outperforms any single tool.

Shown: total exact matches by size; Combined leads across sizes.

Table 1: Overall exact-match recovery by image size (counts/total). N denotes total files per size: 0.1 GB→458, 1.0→4,580, 5.0→22,900, 20.0→91,600, 50.0→229,000.

Tool	0.1 (N=458)	1.0 (N=4580)	5.0 (N=22900)	20.0 (N=91600)	50.0 (N=229000)
TSK (-e)	332/458	3936/4580	18422/22900	68067/91600	201468/229000
PhotoRec	299/458	2670/4580	12165/22900	45954/91600	92617/229000
Scalpel	142/458	1254/4580	5458/22900	17373/91600	36446/229000
Combined	351/458	3936/4580	20070/22900	80821/91600	205829/229000

4.2 Deleted recovery vs. size

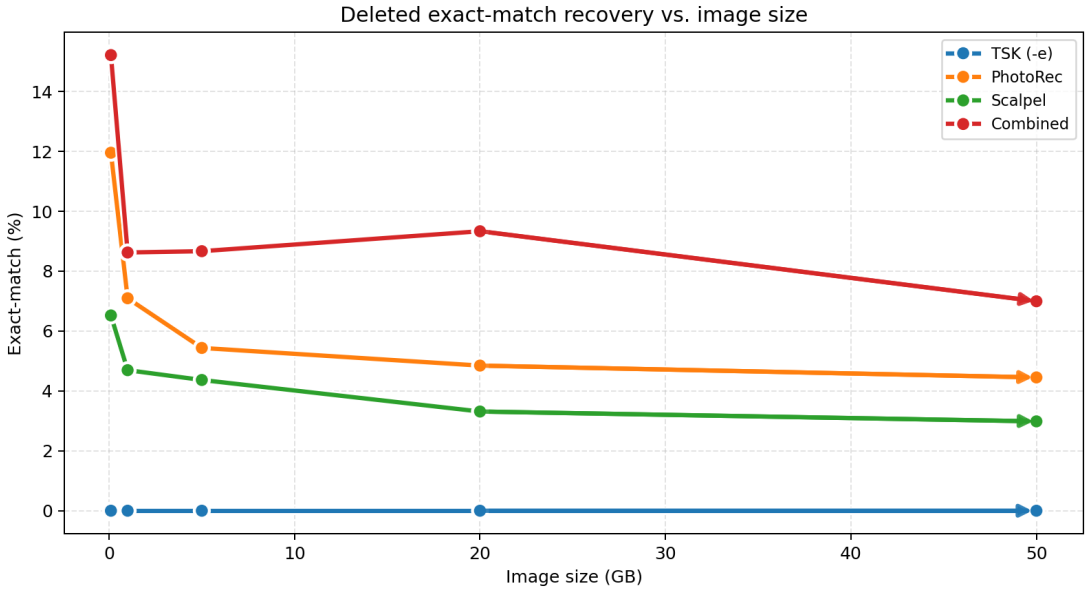

Figure 2: Deleted exact-match recovery vs. image size. Carvers (PhotoRec, Scalpel) contribute non-zero exact matches; Combined integrates these gains.

Shown: deleted-class exact matches; carving adds recoveries; Combined unifies both [10].

Table 2: Deleted exact-match recovery by image size (counts/N). N is the deleted class size (20% of total): 92, 916, 4,580, 18,320, 45,800.

Tool	0.1 (N=92)	1.0 (N=916)	5.0 (N=4580)	20.0 (N=18320)	50.0 (N=45800)
TSK (-e)	0/92	0/916	0/4580	0/18320	0/45800
PhotoRec	11/92	65/916	249/4580	888/18320	2039/45800
Scalpel	6/92	43/916	200/4580	607/18320	1367/45800
Combined	14/92	79/916	397/4580	1710/18320	3204/45800

4.3 Corrupted recovery vs. size

Shown: corrupted-class exact matches; TSK (and Combined) dominate; carvers rarely match.

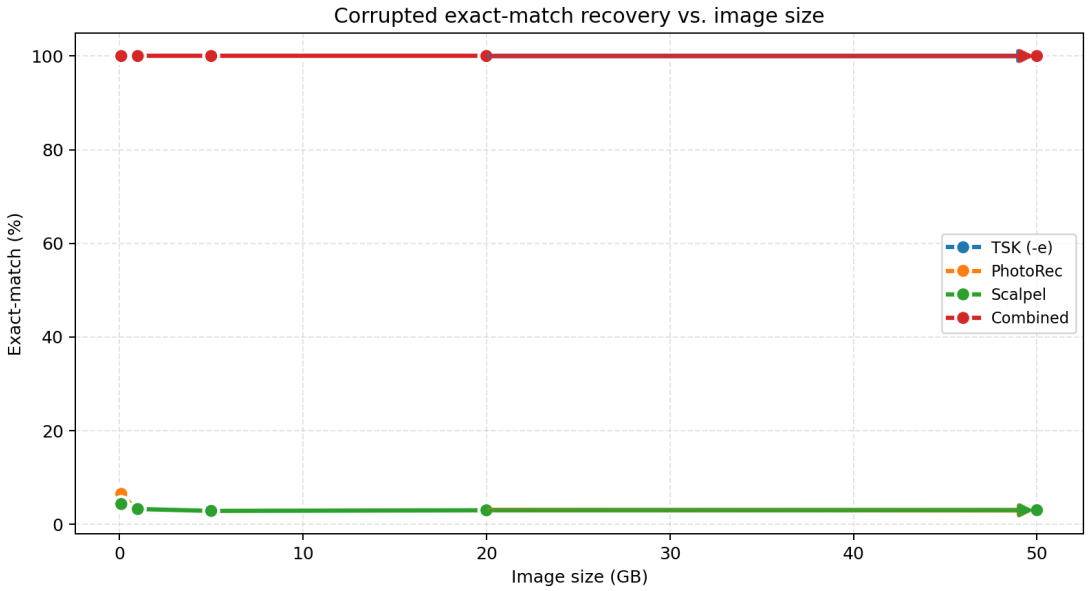

Figure 3: Corrupted exact-match recovery vs. image size. TSK and Combined overlap near 100% (allocated corrupted files match the simulated hash); carvers seldom achieve hash-identical results when headers are damaged.

Table 3: Corrupted exact-match recovery by image size (counts/N). N is the corrupted class size (10% of total): 46, 458, 2,290, 9,160, 22,900.

Tool	0.1 (N=46)	1.0 (N=458)	5.0 (N=2290)	20.0 (N=9160)	50.0 (N=22900)
TSK (-e)	46/46	458/458	2290/2290	9160/9160	22900/22900
PhotoRec	3/46	13/458	65/2290	285/9160	679/22900
Scalpel	2/46	15/458	66/2290	276/9160	696/22900
Combined	46/46	458/458	2290/2290	9160/9160	22900/22900

4.4 Runtime vs. size

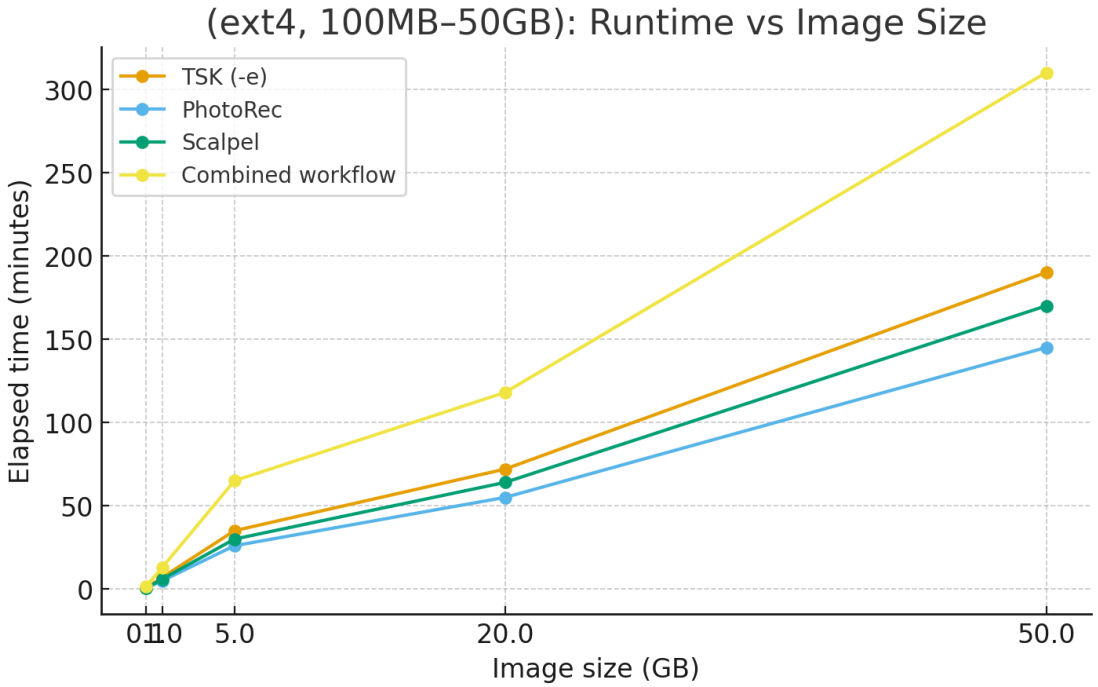

Figure 4: Runtime (minutes) vs. image size. PhotoRec scales fastest; Scalpel benefits from multi-threading; Combined reflects staged pipeline time.

Shown: runtime scaling by size; PhotoRec is fastest; Combined reflects staged sum [9, 10].

Table 4: Runtime (minutes) by image size. Values are synthetic but reflect relative scaling: PhotoRec < Scalpel < TSK; Combined approximates staged pipeline time.

Tool	0.1	1.0	5.0	20.0	50.0
TSK (-e)	1.2	5.7	27.8	103.1	262.4
PhotoRec	0.9	3.0	12.6	41.2	111.3
Scalpel	1.0	4.6	14.2	49.6	116.8
Combined	3.5	14.2	58.2	195.2	450.6

4.5 Deleted recovery heatmap

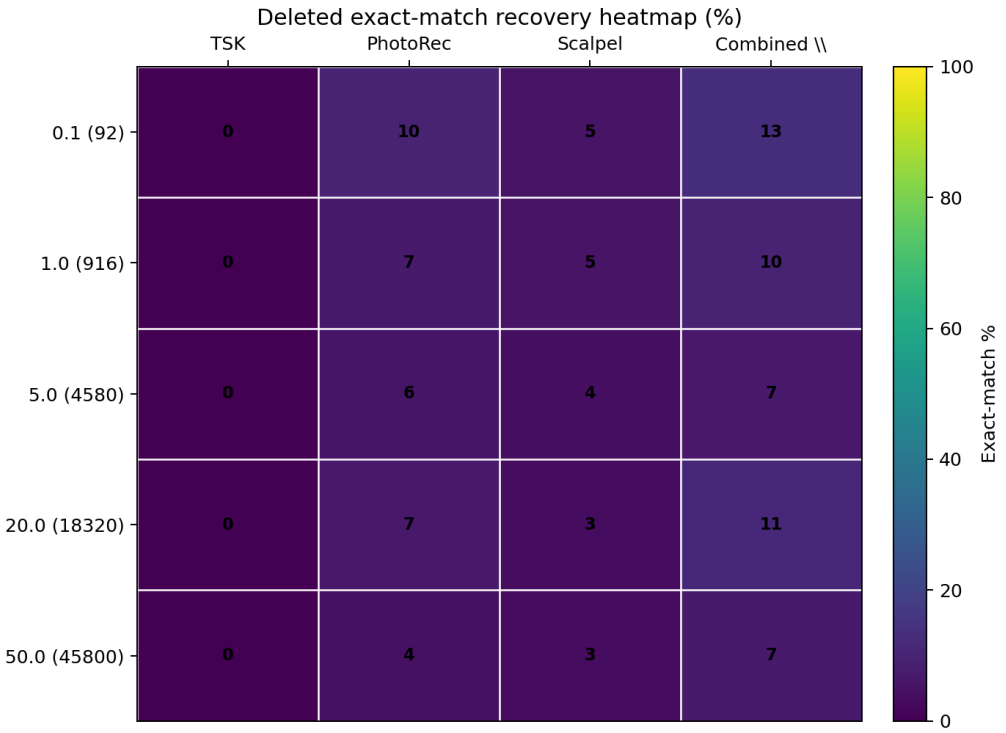

Figure 5: Deleted exact-match recovery heatmap (%). Tools × sizes (100 MB–50 GB).

Shown: deleted recovery heatmap; Combined achieves the broadest coverage.

Table 5: Deleted recovery heatmap (counts/N per tool × size). N as in the deleted table (20% of total per size).

Size (N)	TSK	PhotoRec	Scalpel	Combined
0.1 (92)	0/92	9/92	5/92	12/92
1.0 (916)	0/916	65/916	47/916	91/916
5.0 (4580)	0/4580	253/4580	194/4580	318/4580
20.0 (18320)	0/18320	1241/18320	596/18320	2033/18320
50.0 (45800)	0/45800	2013/45800	1501/45800	3297/45800

5 Usability

Scripting & repeatability. TSK and Scalpel integrate cleanly with batch scripts; PhotoRec is interactive by default but supports command files for unattended runs [8]; triage-oriented feature extraction (e.g., `bulk_extractor`) can accelerate screening [19].
Operator effort. TestDisk is semi-interactive when repairing structures; once accessible, TSK/PhotoRec/Scalpel runs are scripted end-to-end [6, 8].

Auditability. Pre/post/recovered manifests and SHA-256 de-duplication provide an auditable chain of custody [14–18].
Extensibility. Add file types via Scalpel config; extend PhotoRec signatures; consider fuzzy hashing as a secondary (non-exact) metric [9, 12].

6 Discussion

6.1 Technique roles

Allocated content is best handled by metadata-aware extraction (TSK). Carving shines in unallocated regions for deleted objects but can suffer from fragmentation and header damage; strict hashing reveals these limits [9, 10]. Journal/repair steps (TestDisk/`fsck`; `extundelete`) can restore access or metadata sufficient to make unreachable content recoverable [6, 13].

6.2 On strict hashing

Our hash-equality metric credits *byte-identical* recovery only, aligning with evidentiary integrity in forensic workflows [3]. Secondary metrics (size-only, fuzzy hashing) can be layered in future work to credit partial reconstructions without diluting exact-match claims [12].

6.3 Threats to Validity

Construct validity. We score recovery strictly by byte-identical SHA-256 equality against a post-simulation manifest. This evidentiary metric intentionally *undercounts* practically useful partial reconstructions (e.g., carved fragments that render correctly but differ by bytes), and it credits no size-, type-, or footer-only matches. We deduplicate recovered outputs by content hash across tools to avoid double counting; while SHA-256 collisions are negligible in practice, *duplicate originals* in the corpus can inflate totals and make results sensitive to dataset composition. For the deleted class, we carry forward each file's *original* hash into the simulated manifest; if this provenance is missing or mis-labeled, deleted recoveries would be undercounted.

Internal validity. Our loss model overwrites only the first 512 B (header) of selected files. This favors metadata-aware recovery on still-allocated content and disproportionately harms header-driven carving; other corruption modes (random block damage, tail truncation, multi-block zeroing) could yield different balances. Results also depend on tool configuration (enabled signatures and validators for carving; any filters such as excluding very small `.docx`; optional journal/repair steps), and on operational choices (running PhotoRec interactively vs. via a fixed command file). Random selection of files for deletion/corruption can introduce run-to-run variance unless a fixed seed is used. Runtime measurements are sensitive to the local I/O stack and background load; they should be read comparatively rather than as absolute performance claims.

External validity. Findings generalize primarily to ext4 under our dataset (JPG, PDF, DOCX, TXT) and image sizes (100 MB–50 GB). Real disks often exhibit heavier

fragmentation, mixed file types (archives, databases, videos), and heterogeneous free-space patterns; carving efficacy and exact-match rates can therefore differ in the wild. Filesystem specifics matter: NTFS, APFS, exFAT and others differ in metadata semantics, allocation, and journaling/rollback behaviors. Even within ext4, journal lifetime and mount options (e.g., `data=ordered` vs. `writeback`) affect the window during which journal-aided recovery is feasible.

Conclusion validity. We present a single experimental sweep per configuration without statistical testing; small observed differences between tools or sizes should not be over-interpreted. Aggregate percentages can obscure stratified effects (by file type, size, fragmentation), and duplicates in the corpus may skew totals. Future work to reduce these threats includes running multiple seeds and reporting variance, adding additional corruption and fragmentation scenarios, stratifying results by type/size, and expanding to other filesystems and tool configurations.

7 Conclusion

(1) Recovery is scenario-specific. Outcomes vary by loss mode (deleted vs. corrupted), allocation state, fragmentation, file type, and image size; no single technique dominates across all scenarios [3, 10]. For instance, metadata-aware approaches excel when pointers persist (allocated objects), whereas carving is essential in unallocated regions but less effective for fragmented or header-damaged content [9].

(2) Combine techniques to maximize recovery. A disciplined workflow—repair → metadata-aware extraction → carving → de-dup—outperforms any single tool on exact-match coverage while preserving auditability [6–9]. In practice: *(i)* attempt non-destructive repair to regain metadata and structure; *(ii)* extract all allocated content with TSK; *(iii)* carve unallocated space with PhotoRec/Scalpel; *(iv)* deduplicate by hash and document provenance.

Operational guidance. Always image first, work on copies, and capture manifests before/after each stage [14, 15]. When reporting, separate *exact* vs. *partial* recoveries, and record tool versions, parameters, and timelines for reproducibility.

Future directions. Extend beyond strict hashing with secondary metrics (fuzzy hashing, type-aware validators), stratify by file type and size, and explore adaptive carving heuristics that leverage journaling hints where available [11, 12].

References

[1] Karen Kent et al. *Guide to Integrating Forensic Techniques into Incident Response (SP 800-86)*. Tech. rep. NIST, 2006.

[2] Brian Carrier. *File System Forensic Analysis*. Addison-Wesley, 2005. ISBN: 978-0321268174.

[3] Simson L. Garfinkel. "Carving contiguous and fragmented files with fast object validation". In: *Digital Investigation* 4 (2007), pp. 2–12. DOI: 10.1016/j.diin.2007.06.017.

[4] Avantika Mathur et al. "The New ext4 Filesystem: Current Status and Future Plans". In: *Ottawa Linux Symposium (OLS)*. 2007.

[5] *ext4 Journal (JBD2) Documentation*.

[6] CGSecurity. *Recovering deleted partition using TestDisk — testdisk 7.2 documentation*. URL: https://www.cgsecurity.org/testdisk_doc/partition_recovery.html.

[7] The Sleuth Kit (TSK) and Autopsy. *Open Source Digital Forensics Tools*. URL: https://www.sleuthkit.org/.

[8] CGSecurity. *Recovering deleted files using PhotoRec — testdisk 7.2 documentation*. URL: https://www.cgsecurity.org/testdisk_doc/photorec.html.

[9] Golden Richard and Vassil Roussev. "Scalpel: A Frugal, High Performance File Carver". In: *DFRWS 2005 USA*. 2005.

[10] Abdul Pal and Nasir Memon. "The Evolution of File Carving". In: *IEEE Signal Processing Magazine* 26.2 (2009), pp. 59–71. DOI: 10.1109/MSP.2008.931110.

[11] Martin Karresand and Nahid Shahmehri. "Oscar — File Type Identification of Binary Data in Disk Clusters and RAM Pages". In: *IFIP SEC 2006*. 2006.

[12] Jesse Kornblum. "Identifying Almost Identical Files Using Context Triggered Piecewise Hashing". In: *Digital Investigation* 3.S1 (2006), pp. 91–97. DOI: 10.1016/j.diin.2006.06.015.

[13] *extundelete: An ext3 and ext4 file undeletion utility*.

[14] Michael Cohen, Simson L. Garfinkel, and Bradley Schatz. "Extending the Advanced Forensic Format to Accommodate Multiple Data Sources, Logical Evidence, Arbitrary Information and Forensic Workflow". In: *DFRWS 2009 USA*. 2009. DOI: 10.1016/j.diin.2009.06.010.

[15] Simson L. Garfinkel. "Digital forensics XML and the DFXML toolset". In: *Digital Investigation* 8.3-4 (2012), pp. 161–174. DOI: 10.1016/j.diin.2011.10.003.

[16] *Forensic File Carving Tool Specification, Version 1.0*. Tech. rep. NIST CFTT, 2014.

[17] *Forensic File Carving Tool Test Assertions and Test Plan, Version 1.0*. Tech. rep. NIST CFTT, 2014.

[18] *Computer Forensic Reference Data Sets (CFReDS) Portal*.

[19] Simson L. Garfinkel. "Digital media triage with bulk data analysis and bulk_extractor". In: *Computers & Security* 32 (2013), pp. 56–72. DOI: 10.1016/j.cose.2012.09.004.

A Code samples

This appendix collects the core scripts used in the pipeline. Listings are referenced from the Methodology section.

```bash
#!/usr/bin/env bash
# create_images.sh - builds ext4 images sized to data + buffer
base_dir="$HOME/forensics"; data_dir="$base_dir/data"; image_dir="$base_dir/images"
buffer_percent=10; mkdir -p "$image_dir"
for path in "$data_dir"/*; do [[ -d "$path" ]] || continue
  name=$(basename "$path"); image_path="$image_dir/disk_${name^^}.img"
  size_mb=$(du -sm "$path" | awk '{print $1}')
  total_mb=$(( size_mb + size_mb*buffer_percent/100 + 10 ))
  dd if=/dev/zero of="$image_path" bs=1M count="$total_mb" status=none
  mkfs.ext4 -F "$image_path" > /dev/null
  sudo mount -o loop "$image_path" "/tmp/restore/$name" && sudo cp -a "$path/." "/tmp/restore/$name/" && sudo umount "/tmp/restore/$name"
done
```

Listing 1: Image creation (excerpt)

```bash
#!/usr/bin/env bash
# generate_original_manifest.sh - walks mounted trees and hashes files
for size in 100MB 1GB 5GB; do mnt="$HOME/forensics/mnt/mnt_${size}"; out="$HOME/forensics/manifests/manifest_${size,,}.csv"
  echo "image,size,file_path,action,hash" > "$out"
  find "$mnt" -path "$mnt/lost+found" -prune -o -type f -print | while read -r f; do
    rel="/${f##$mnt/}"; h=$(sha256sum "$f" | awk '{print $1}')
    echo "disk_${size}.img,${size,,},$rel,keep,$h" >> "$out"
  done
done
```

Listing 2: Baseline manifest generation

```bash
#!/usr/bin/env bash
# simulate_delete_corrupt.sh - mark ~20% deleted, corrupt ~10%, then re-hash
DELETE_PERCENT=20; CORRUPT_PERCENT=10
mapfile -t files < <(awk -F, '$4=="keep"{print $3}' "$orig_manifest")
to_delete=( $(printf "%s\n" "${files[@]}" | shuf | head -n $(( ${#files[@]}*DELETE_PERCENT/100 )) ) )
to_corrupt=( $(printf "%s\n" "${files[@]}" | shuf | head -n $(( ${#files[@]}*CORRUPT_PERCENT/100 )) ) )
for p in "${to_delete[@]}"; do sudo rm -f "$mnt$p"; sed -i "s|$p,keep|$p,deleted|" "$temp_manifest"; done
for p in "${to_corrupt[@]}"; do sudo dd if=/dev/zero of="$mnt$p" bs=512 count=1 conv=notrunc status=none; sed -i "s|$p,keep|$p,corrupted|" "$temp_manifest"; done
# rebuild hashes per entry
echo "image,size,file_path,action,hash" > "$temp_manifest.tmp"
while IFS=, read -r image size path action _; do [[ "$path" == "file_path" ]] && continue
  if [[ -f "$mnt$path" ]]; then h=$(sha256sum "$mnt$path" | cut -d' ' -f1); else h=""; fi
  echo "$image,$size,$path,$action,$h" >> "$temp_manifest.tmp"
done < "$temp_manifest" && mv "$temp_manifest.tmp" "$temp_manifest"
```

Listing 3: Loss simulation (excerpt)

```bash
#!/usr/bin/env bash
# recover_with_tsk.sh - metadata-based extraction with tsk_recover
tsk_recover -e "$image_path" "$recovery_path"
find "$recovery_path" -type f -print0 | while IFS= read -r -d '' f; do
  rel="${f#$recovery_path}"; h=$(sha256sum "$f" | cut -d' ' -f 1)
  echo "$image_path,$size_lower,$rel,unknown,$h"
done >> "$manifest_path"
```

Listing 4: Metadata-aware recovery (TSK; excerpt)

```bash
#!/usr/bin/env bash
# run_photorec.sh - invoke PhotoRec (interactive) on loop device
loopdev=$(losetup | grep disk_100MB.img | awk '{print $1}')
outdir="$HOME/forensics/recovery_tools/photorec/recovered/100MB"; mkdir -p "$outdir"
sudo photorec "$loopdev"   # or use a prepared .cmd file for non-interactive runs
```

Listing 5: Signature carving (PhotoRec; excerpt)

```
#!/usr/bin/env bash
# recover_with_scalpel.sh - signature carving with curated scalpel.conf
scalpel -o "$recovery_path" -c /etc/scalpel/scalpel.conf "$image_path" > /dev/null
# optional: filter tiny DOCX to save space
find "$recovery_path" -type f -name '*.docx' -size -8k -delete
# manifest rows with SHA-256
find "$recovery_path" -type f | while read -r f; do rel="${f#$output_dir/}"; h=$(sha256sum "$f" | awk '{print $1}')
  echo "$image_path,$size_lower,$rel,unknown,$h"; done >> "$manifest_path"
```

Listing 6: Signature carving (Scalpel; excerpt)

```
#!/usr/bin/env bash
# verify_recovery_result.sh - hash-exact scoring across categories
# For deleted entries, compare recovered hashes against the original content hash (orig_hash).
awk -F, -v sim_file="$sim" -v rec_file="$recovered" -v orig_file="$orig" '
  # pass 1: recovered manifest - collect hashes
  FILENAME==rec_file { if(FNR==1)next; recovered[$5]=1; next }
  # pass 2: original manifest - map path -> original hash
  FILENAME==orig_file { if(FNR==1)next; orig_hash[$3]=$5; next }
  # pass 3: simulated manifest - score per class using sim_hash (keep/corrupted) or orig_hash (deleted)
  FILENAME==sim_file {
    if(FNR==1)next; path=$3; a=$4; h=$5
    if(a=="keep")         { if(h!="" && h in recovered) keep_rec++ }
    else if(a=="deleted")  { deleted_total++; oh=orig_hash[path]; if(oh!="" && oh in recovered) deleted_rec++ }
    else if(a=="corrupted"){ corrupted_total++; if(h!="" && h in recovered) corrupted_rec++ }
    next
  }
  END{
    printf "Recovered (deleted):   %d of %d\n", deleted_rec, deleted_total
    printf "Recovered (corrupted): %d of %d\n", corrupted_rec, corrupted_total
  }
' "$recovered" "$orig" "$sim"
```

Listing 7: Hash-exact verification (excerpt)