Delivering script and database schema
I need to promptly provide a script with database schemas for storing evidence like images, deepfakes, and tool versions, possibly through SQLite or JSON. I'll deliver both, including tables for evidence_files, jpeg_quant_samples, and others. The user also wants to automate image capture and hash generation, likely from a folder.

- Implementing file ingestion and metadata extraction
  I'm working on a tool to ingest files from a directory, preserve them in an evidence store with a unique ID, and compute their sha256 hashes. I'll extract metadata with `ExifTool`, or if that's unavailable, fall back to Pillow for JPEGs. I need to check the existence of specific features in Pillow and test.

</> Generating technical scripts and protocols for image acquisition

```
import PIL, PIL.Image
PIL.__version__
```

</> Checking Pillow installation and version

```
from PIL import Image, ImageFile,
JpegImagePlugin import PIL
PIL.__version__
```

</> Analisado

- Checking Pillow 11.3.0 for subsampling
  I need to check how to get subsampling information in Pillow 11.3.0. Specifically, I'll look at `JpegImagePlugin` to inspect how it handles this. I'll check the available methods and properties related to subsampling to confirm how I can retrieve that data.

Inspecting subsampling methods in JpegImagePlugin

```
import inspect,
PIL.JpegImagePlugin as jip [name
for name in dir(jip) if "sub" in
name.lower() or "sampling" in
name.lower()][:50]
```

Analisado

- Inspecting `get_sampling` method
  I found the `get_sampling` method in Pillow. I need to check its signature to understand exactly how it works. This will help me determine if it provides the subsampling information I need. I'll take a closer look at how it's defined and used.

Inspecting get_sampling function signature

```
import PIL.JpegImagePlugin as jip,
inspect
inspect.getsource(jip.get_sampling)
[:400]
```

Analisado

Printing function code for jip.get_sampling

```
print(inspect.getsource(jip.get_samp
```

Analisado

- Understanding JPEG sampling with Pillow
  For a JPEG opened as `JpegImageFile`, `im.layer` holds the sampling factors. The `get_sampling` method returns a code like 0, 1, 2, etc. I need to check how these codes map to specific sampling factors. I'll examine the mapping of these values to better understand how they work.

Examining JPEG image sampling factors

```
import PIL.JpegImagePlugin as jip
jip.samplings
```

Analisado

- Understanding JPEG encoding details with Pillow
  I've figured out the mapping for sampling factors: 0 = 4:4:4, 1 = 4:2:2, 2 = 4:2:0. For progressive vs baseline, I can check if a JPEG is progressive with `img.info.get("progressive")`. For quantization tables, Pillow exposes `img.quantization`, which returns a dictionary mapping table numbers to 64-value lists. I'll hash these tables and support both "Save As" and "Export As" options.

- Handling "Save As" vs "Export As" in JPEG conversion
  The challenge here is replicating Adobe's "Save As" vs "Export As" behavior in a script. We can approximate this by using different libraries and encoders. For example, Pillow will handle baseline encoding, while we can use `mozjpeg` or `ImageMagick` for the "Export As" mode with better optimizations. Two modes will be implemented: "save" for baseline and "export" for enhanced encoding.

- Replicating export behavior with different modes
  This isn't an exact match for Photoshop's "Save vs Export", but we can try to simulate it. We can implement two modes in the script: "save_as"

using Pillow with default tables, and "export_as" using Pillow with optimizations. This will cover differences in quantization, Huffman tables, and chroma subsampling, though there may still be slight variations.

- Implementing plugin architecture for external encoders
  The script can be designed with a plugin architecture where users can choose and plug in encoder backends. "Mode" will influence which backend and flags are used. For example, `mode="save_as"` triggers Pillow with no optimizations or progressive encoding, while `mode="export_as"` might use mozjpeg with advanced options. Additionally, we're considering integrating Comprint for heatmap generation, but we don't plan to implement the full algorithm. We'll need clean training data for Comprint, as noted in the PDF.

- Avoiding contamination and identity leakage
  To prevent contamination, ensure source images are lossless (raw, PNG, TIFF) with no previous compression, verified by metadata and histograms. Maintain a "pristine" directory, record `source_format`, and verify no prior compression. Regarding deepfakes, define identity IDs and assign groups to splits using a custom method, ensuring that both the source and target identities are grouped to avoid leakage. Group by original video/person and implement an identity ID based on dataset annotations or filenames.

- Planning robust identity and file handling
  In FaceForensics++, each manipulated video corresponds to source-target pairs, so we can approximate identity by video ID or compute face embeddings (e.g., ArcFace) for better accuracy. We'll create an "identity graph," ensuring no leaks by splitting connected components. For MVP, we'll base splits on video. I'll outline scripts for file acquisition and tracking, including integrity checks like SHA256, time stamps, and detailed metadata. The goal is to offer a minimal setup but allow for extension with more rigorous methods.

- Planning CLI arguments and metadata extraction
  The script will use file statistics like size and modification time, extract metadata with tools like python-magic, Pillow, and ffprobe. For JPEG, PNG, or TIFF, we'll check properties like dimensions, EXIF/ICC data, and quantify characteristics like progressive encoding. Then, a manifest will track metadata, file hashes, integrity, and processing steps to ensure a proper chain of custody. Lastly, the script will handle input images, validating their losslessness, implementing various quality settings, and using encoder plugins for transformation.

- Building compatibility DB and deepfake sampling pipeline
  I'll create a database mapping qhash values to encoders, modes, quality settings, and more. This will include a table with columns like qhash, software family, and encoder backend. Then, I'll build a script to generate `qdb.sqlite` from the manifest. For deepfake sampling, I'll implement a script to extract frames, crop faces, generate a manifest with video info, and ensure proper group splits using algorithms to prevent identity leakage in training.

- Integrating Comprint and generating heatmaps
  I'll create a Python wrapper for Comprint that processes images, runs inference, & generates heatmaps. The module will work with directories or image paths, load model weights, and save the results as both curated metrics (JSON) and image overlays. For seamless integration, I'll call this within the data pipeline and track the tool's version and environment. We'll also deliver a database schema for the evidence repository in SQLite and JSON format while ensuring proper governance with file citations based on user instructions.

- Preventing data contamination in file processing
  To avoid data contamination, I'll set up a robust procedure ensuring that source files stay

untouched. I'll copy images to a destination while preserving their integrity (e.g., using hashes for verification). I'll detect codecs like JPEG or PNG and gather EXIF/ICC data. For JPEG, I'll analyze subsampling and avoid JPEG-derived PNGs. While metadata such as quantization tables will be extracted, I'll ensure that all inputs are lossless to maintain data integrity throughout.

- Designing a protocol to avoid compressed sources
  To prevent contamination, the protocol involves acquiring sources directly as RAW or PNG/TIFF, ensuring no prior compression. The transformation should be strictly lossless, using specified tools. Any file with JPEG-related metadata will be rejected. The workflow also tracks processing steps to maintain the chain of custody. I'll implement the described scripts.

- Designing forensic acquisition script with metadata extraction
  The script handles forensic image acquisition, ensuring original raw files are preserved. While copying, a SHA-256 hash is computed to prevent file tampering. It extracts important metadata such as codec, dimensions, and more. The script can also support webcam image capture with OpenCV for lab generation. A manifest containing all processed files is updated, stored, and written safely using a manifest writer.

- Implementing JPEG processing and external tool support
  To handle JPEG processing, we'll integrate multiple backends (such as Pillow and MozJPEG) with the ability to export images with varying compression qualities and parameters. We'll also enable users to work with external encoders like GIMP and Photoshop, offering templates for them to provide their own tooling for quantization and metadata extraction. The script will support both creating images from scratch and indexing existing JPEGs, making it versatile for different workflows.

Generating forensic image workflows, quantization, and deepfake processing

- Looking at DeepfakeBench details
  I'm planning to check out the DeepfakeBench README for more information about the dataset format, JSON files, and how the splits are organized. It'll give me a better understanding of how to structure the dataset for training and how things should be set up.

  Opening DeepfakeBench README for JSON details

- Looking for DeepfakeBench dataset JSON
  The current view of DeepfakeBench didn't show the relevant content, so I need to search for the specific "dataset_json" file. I'll try finding it using the search tool on view0 to locate it more directly. That should help me get the right files and information I need.

  Finding dataset_json in DeepfakeBench view0

- Accessing DeepfakeBench README raw content
  It seems like the GitHub HTML is blocking the file content, possibly due to dynamic loading or a cookie issue. To get around this, I'll try using the raw version of the README with a direct endpoint. I'll open it using the URL for the raw content instead.

  Finding link to raw file on GitHub

- Searching for raw GitHub file location
  I'm having trouble retrieving the GitHub content directly due to dynamic issues, but the search summary shows that preprocessing leads to JSON files in `./preprocessing/dataset_json`, which includes the desired splits. I'll try searching specifically for the raw URL to bypass these restrictions and open it directly.

  Searching for raw URL and preprocessing details

- Exploring alternative sources for README content
  I've found some relevant sources, but I'm still missing the full README content. To fill in the gaps, I'll check if Deepwiki pages provide the content we need. I'll specifically look at the '1-overview' page to understand the dataset pipeline and other details.

  Exploring DeepWiki for dataset pipeline content

- Gathering dataset structure and planning integration
  Great, Deepwiki has provided the dataset structure, so I'll be citing that. DeepfakeBench uses text manifest files that list image file paths. I'll implement our custom splitting algorithm to organize by identity and video origin, then adapt it to match the expected format. Also, I'll integrate the Comprint repo, noting its use of TensorFlow 2.9.1 and the notebook for extracting data. I'll likely need to review the notebook to see how to integrate it smoothly.

  Searching for Comprint notebook functions and references

Navigating code repository to open specific notebook file

- Searching for the raw Comprint notebook
  GitHub isn't showing the file list due to an error again. So, I think I'll try searching for the raw URL for that specific file instead. The tool restricts me to search results, but if I can find the right raw URL, I should be able to open it that way. Let's try searching for "get_comprint_heatmap.ipynb raw.githubusercontent.com."

  Searching for raw GitHub file URLs

- Trying search with raw URL for notebook
  The search didn't return the raw URL for the notebook. Maybe the tool restricts access to cert websites like raw.githubusercontent.com for safe reasons since it flagged it as unsafe earlier. I'll try searching for the exact raw URL directly to see if returns a list, allowing me to open it safely. Let's querying for `https://raw.githubusercontent.com/IDLabM a/comprint/main/code/get_comprint_heatmap.ip b`.

  Searching for raw GitHub notebook URL

Exploring alternatives for accessing comprint module files

- Trying workaround with file lines
  GitHub pages aren't loading the file content due to an error. Earlier, the Comprint repo worked, but something's different now. Maybe it's because GitHub uses a script to load files, and our tool can't render those properly. The file could also have more than 200 lines. I'll try opening around line 150 to see if that works.

  Investigating GitHub file content loading issues

- Focusing on pipeline and future proposals
  It seems like the GitHub file content isn't working, so I'll stick to the README for Comprint integration. Their repo provides clear installation instructions and usage. The dataset does mention biological inconsistencies, but the current tasks don't include biological extraction. I'll suggest future work on capturing physiological signals, but for now, I'll focus on the requested scripts and pipeline with the three layers.

- Details for MVP and database proposal
  To align with the MVP in 90 days, we're focusing on four main steps: acquisition, fingerprint quantization, deepfake detection pipeline, and localization with Comprint—matching what's in the PDF. For the evidence repository, I'll go with SQLite for queries and indexing. I've outlined a database structure and will also include a JSON schema for the manifest.

- Final answer setup and structured delivery
  We'll follow specific protocols for security, focusing on ISO 27037, like write protection, logging, and hash checks. Also, consider acquisition device and camera setup. For the final answer, I'll provide a detailed step-by-step guide in Portuguese with separate sections: preventing data contamination, folder structure, SQLite schema, and scripts for each component. This ensures clarity, including a minimal script for script 3 with pipeline splits and DeepfakeBench manifest generation. I'll ensure reproducibility while minimizing external dependencies like Pillow and numpy.

- Plan for identity mapping and dataset handling
  For deepfake splitting, I'll implement a light dependency solution using DSU or networkx. The dataset will be generated from folder structure, using `dataset_index.csv` for mapping sample paths and identity IDs. For FF++ videos, we'll map identities using either `video_to_identity.json` or infer based on video names for real and fake videos, ensuring correct splits and no identity leakage.

- Deepfake dataset splitting and Comprint integration
  For CelebDF, we'll designate identity patterns like `id0`. The script `deepfake_identity_split.py` will scan the dataset to extract video names, determine labels (real or fake), and assign identity IDs based on filename patterns. Then, an identity graph will be built for proportional splits. For Comprint integration, we'll create a lightweight wrapper that runs inference locally, possibly using `nbconvert` to execute notebook code efficiently.

  Integrating Comprint and preparing dataset repository

  Opening file for function details in `turn13search13`

- Comprint integration with simplified wrapper
  We can't rely on the open view, so we'll use a high-level integration for Comprint. I'll implement a wrapper with a `papermill` template notebook and allow users to run inference, using Comprint's trained models. We'll simplify the inference pipeline but need to understand how to load models first.

- Addressing issues with accessing Comprint's notebook
  It seems accessing Comprint's notebook might be problematic due to restrictions. I can't rely

on fetching raw content directly. However, I can still provide an integration code that runs Comprint's notebook with `papermill` and passes the `IMAGE_PATH` parameter. I'll create a simplified approach for running the notebook.

- Considering alternatives to Comprint integration
  I could try creating a "comprint-like" heatmap based on JPEG grid features, but that's not exactly what they want. They'll likely prefer genuine Comprint integration. One possible option: cloning the Comprint repo and using a wrapper to execute their notebook with `papermill`, passing the `image_path` parameter.

  Searching for Comprint notebook access alternatives

  Exploring options for accessing notebook or code snippet

- Planning Comprint integration module
  To integrate Comprint, I can create a module that either runs Comprint code if installed or executes their notebook using `nbconvert` to generate the heatmap. The module would check if the Comprint repo exists, and if not, show an error. Once they clone it, we could provide a `comprint_runner.py` with these features.

- Creating a wrapper for Comprint notebook
  Instead of building our own full inference system, we could create a wrapper that uses Comprint's repo notebook directly. The notebook can be modified programmatically to set file paths or use environment variables. We'll propose a protocol where users store the path in an environment variable, and the wrapper runs the notebook.

- Delivering a module for Comprint integration

Instead of modifying the notebook directly, we can create a module that expects Comprint to be installed and uses `tf.saved_model` to load the model from their local path. We can guide them to set up the environment while avoiding GitHub restrictions or needing direct access to the repo.

- Testing container.download for notebook access
  It seems the python environment doesn't have internet access, but the container tool might with its download function. We can try using `container.download` to fetch the notebook directly from GitHub via a raw URL. I'll test this to see if it's allowed for downloading the file.

Analisado

- Figuring out how to get the raw URL
  It looks like I can't directly use the `container.download` tool without having the raw URL in the conversation. I can't access it via `web.run` because of safety restrictions, but maybe I can work with the GitHub file page instead. If we open the file page first, I can get the raw URL from there.

- Exploring how to access the raw URL
  The page isn't loading, but the raw link might still be in the HTML. I could try appending `?plain=1` to the URL to see if that reveals the content along with the raw link. Let's first search for the file without the plain parameter and see if that works.

  Searching for GitHub file page URLs

- Trying a different search method
  The notebook page didn’t show up in the search results—probably because the search engine doesn’t index ipynb pages. Let’s try searching with different terms, like "get_comprint_heatmap.ipynb IDLabMedia/comprint blob" or "get_comprint_heatmap.ipynb

site:raw.githubusercontent.com IDLabMedia comprint". Hopefully, this will turn up the result.

Searching for "get_comprint_heatmap.ipynb" on GitHub

- Relying on Comprint or an alternative approach
  Since I haven't found the raw link, maybe a practical solution is to rely on the arXiv paper that explains the inference steps. We could implement a simplified version using JPEG quantization for heatmap generation. This could satisfy the user's needs, with the option to upgrade to full Comprint later. I'll call this fallback "comprint-lite" to stay transparent.

- Searching for Comprint file structure
  I'll try to implement code that runs the official Comprint repo if the user has it. For users without it, I'll design a fallback. Searching for Comprint files, like 'get_comprint_heatmap.ipynb', might give us useful leads. I'll search for 'get_comprint_heatmap.ipynb IDLabMedia/comprint code' and look for relevant results.

  Searching for Comprint heatmap implementation and resources

- Reviewing deepfake dataset and PDF alignment
  Since we still don't have the notebook, I'll deliver a wrapper with a fallback solution. The user wants dataset inconsistencies and biological signals, like rPPG, for deepfake detection, but hasn't specifically asked to include them. I'll mention this as part of the protocol, and I'll review the uploaded PDF to align everything.

- Citing PDF and relevant sources
  We got results from file_search, and we'll cite the relevant lines, like those about controlled variables and errors. Since the user didn't ask

for the latest status, we don't need to start the response with "'According to a document from [DATE]." I'll focus on citing the necessary document lines and minimize web citations, as the response is mainly about code.

- Structuring response on dataset design
  I'll structure the response in sections, beginning with dataset design principles based on forensic needs. We'll emphasize maintaining chain-of-custody, hash verification, and data contamination prevention. I'll mention lab-controlled image generation and highlight classic errors like mixing lossy formats. To support, I'll cite relevant sections from the PDF regarding dataset contamination.

- Dataset protocol and code overview
  First, we'll ensure dataset integrity by using a "Pristine Vault" with only lossless sources, capture acquisition details, and hash verification in the source manifest. Each file will also track its processing steps with `parent_sha256`. I'll emphasize maintaining proper export mode separation to avoid contamination. Now, I'll structure the code accordingly.

- Providing code and documentation for forensic dataset
  With the dataset in mind, we'll first verify the ingestion process using `validate_pristine.py`, checking file formats, hashes, and export settings like progressive modes. For the evidence repository, I'll provide SQL and JSON schema. Scripts for acquisition, quantization, and identity splitting will follow, including integration with DeepfakeBench.

  Let's also ensure the necessary sources are aligned for accurate citation.

  Designing and coding forensic dataset protocols

- Designing identity-split pipeline with DeepfakeBench

It looks like Deepwiki doesn't specify the exact JSON format, but that's okay; we don't necessarily need it. By adapting DeepfakeBench preprocessing, we can leverage its output and modify the splits with our `enforce_identity_split.py` script, ensuring identity separation. We don't need to extract frames — just align with DeepfakeBench's output.

- Preparing the enforce_identity_split script
  We'll make the script versatile by allowing it to handle different datasets (e.g., FF++, CelebDF, DFDC). It will accept a `video_to_identity.json` mapping or infer identities. I'll check DeepfakeBench's preprocessing notes to confirm the output paths, ensuring everything's in line. The script will parse the dataset accordingly.

- Versatile data input for pipeline integration
  We'll offer two options for customization: one for a unified `all.json` file, and another if samples are in separate files like `train.json`. The solution will be flexible to accept `samples.jsonl` or `samples.csv`, allowing adaptation. Our code will ensure robustness, with a snippet for DeepfakeBench integration as requested.

- Providing tools for identity splits and DeepfakeBench integration
  I'll provide a script, `deepfake_identity_split.py`, that processes videos or frames along with a mapping file to generate splits like `train.txt`. For DeepfakeBench, I plan to offer guidelines to generate the dataset JSON, though I'll look up examples of `dataset_json` content to be sure.

  Searching for DeepfakeBench dataset JSON format examples

- Looking into DeepfakeBench's dataset format

We still don’t know the exact JSON format for DeepfakeBench, but our script can work at the file list level after face extraction. We can make our own `dataset_json` fitting DeepfakeBench’s expected format. I’ll check the Deepwiki “Data Rearrangement” page or deepwiki snippets for more details.

Searching for DeepfakeBench Data Rearrangement details

- Exploring DeepfakeBench's rearrange.py and data details
  We still don’t know the exact JSON structure, but we could create our own dataset list. Maybe DeepfakeBench's `rearrange.py` file on Deepwiki could help, since it might have relevant code. I’ll check the Deepwiki page for "Data Rearrangement" to find specific details.

  Searching for DeepWiki Data Rearrangement Info

- Handling identity split and integration with DeepfakeBench
  I still don’t have the exact JSON format, but we can proceed by creating independent identity splits. Our script could generate group-split mapping files for DeepfakeBench's `train_config.yaml`. The solution might be to create “split file lists” and let DeepfakeBench handle the dataset JSON. We'll provide the algorithm, code, and instructions for integrating it.

- Handling identity splits within DeepfakeBench's directory structure
  From the info on the website, it seems DeepfakeBench groups videos by labels and data splits within a directory structure. If that’s the case, we could enforce identity splits by moving the videos into split directories based on their identity. Our script can read the videos,

compute their identity ID, and reorganize the splits accordingly. This should work with DeepfakeBench's preprocessed datasets.

- Defining identity split script workflow
  Our script will take the root path to the `dataset_split` directory, scanning for specific splits and labels. It will gather video directories, determine unique video and identity IDs using dataset parsers, then assign them to appropriate splits. Rather than moving large files, we'll create a manifest file mapping splits and updating the dataset accordingly, producing a `identity_split.json` and lists. We'll include code, split assignment, and possibly a test harness. We'll also consider the splicing/copy-move tasks as requested.

- Describing qhash-based quantization matching process
  I'll describe the algorithm for database matching with qhash. For JPEG queries, extract quantization tables and qhash, then compare with the database to find matching hashes. To refine, we can use additional characteristics like markers or huffman optimization. I'll also mention best practices on file handling, file hashes, and evidence preservation without overwriting files, while ensuring proper read-only settings for evidence storage. The final code may be lengthy but will provide the full solution.

- Crafting forensic scripts for evidence acquisition and quantization
  I'll create separate scripts as requested, while avoiding duplication.
  For `forensic_acquire.py`, I'll define a `manifest.json` structure to log acquisition details and minimal metadata.
  For `quantization_extend.py`, I'll maintain a separate manifest or update the existing one, logging processing steps. Additionally, I'll include optional integration for SQLite DB operations through a minimal `EvidenceDB` class.

- Integrating Comprint and clarifying key risks

I'll include both a JSON manifest for truth and an optional database index. For Comprint, I'll create `run_comprint.py` to handle backends like `comprint_repo` for external use or a fallback to compute block-wise quality factor estimates. I'll also provide necessary instructions for integrating Comprint, highlighting the key risk around terminology confusion, such as “compatible with” vs “proof of.” Plus, I'll mention protocols for capturing dataset.

- Outlining data contamination prevention workflow
  We're structuring datasets for quantization and deepfakes. For quantization, we'll use different software baselines (e.g., Photoshop, GIMP) and record their settings, quality, and version. For deepfakes, sources like CelebDF will be used, tracking identity splits.

  For data contamination prevention, we'll keep "pristine" sources isolated and validate lossless formats, ensuring transformations like JPEG conversions are properly documented with unique hashes and metadata like tool versions, flags, and processes.

- Handling identity split and image processing details
  Start by building an identity mapping for dataset splitting, ensuring proper tracking in both manifests and DB. For face swaps, I'll account for combined identities. Regarding image handling, I'll use optional libraries like Pillow and piexif and ensure proper video codec handling with ffprobe when applicable. For formats like PNG with alpha, I'll composite them over a background to preserve information, including necessary parameters and conversion. For TIFF files, 16-bit images will be converted to 8-bit and recorded.

- Implementing splitting and external command wrappers
  To assign datasets to splits, I’ll use a greedy algorithm that approximates the target ratios for each split. I'll ensure no identity leakage by linking related samples in a data structure. We’ll

generate the necessary output files and integrate with DeepfakeBench.

Additionally, I'm building a wrapper for running Comprint, storing heatmaps, and capturing command details – all while following preservation guidelines like chain of custody. The final output will include PNG overlays and a JSON file as requested.

- Finalizing answer with citations
  We need to finalize by crafting the response, adding disclaimers, and including citations. I'll reference lines from the PDF, making sure to properly cite the relevant content. For example, we'll cite for contamination discussions, quality control variables, and the identity split rule. I'll also include citations from web sources for Comprint and DeepfakeBench based on the descriptions.

Thought for 11m 59s
Feito

**Fontes** · 96

my_files

**Detecção Deepfakes Forense.pdf**

GitHub

**GitHub - SCLBD/DeepfakeBench: A comprehensive benchmark of deepfake ...**
We have conducted more comprehensive evaluations using the DeepfakeBench-v2, with mo…

DeepWiki

**Dataset Preparation | yermandy/deepfake-detection | DeepWiki**

GitHub

**GitHub - IDLabMedia/comprint: Comprint: Image Forgery Detection and Localization using…**

GitHub

**comprint/code/splicebuster/noiseprint/noiseprint_blind_concat ... - GitHub**

Comprint: Image Forgery Detection and Localization using Compression Fingerprints -...

papercodex.com

**DeepfakeBench: The First Standardized Benchmark for Fair, Reproducible ...**

12 de janeiro de 2026 — DeepfakeBench solves this by offering a standardized data pipeline that...

deepwiki.com

**Dataset Preparation | yermandy/deepfake-detection | DeepWiki**

15 de dezembro de 2025 — It covers downloading source data, preprocessing using DeepfakeBench,...

deepwiki.com

**Datasets | yermandy/deepfake-detection | DeepWiki**

15 de dezembro de 2025 — This script creates the text files in config/datasets/FF/ by traversing the...

deepwiki.com

**Data Pipeline | yermandy/deepfake-detection | DeepWiki**

15 de dezembro de 2025 — This document describes the data pipeline architecture for loadin...

github.com

**GitHub - yermandy/deepfake-detection: Official implementation for the ...**

2 de novembro de 2025 — We leverage the Contrastive Language-Image Pre-training (CLIP)...

openreview.net

**DeepfakeBench-MM: A Comprehensive Benchmark for Multimodal Deepfake ...**

16 de setembro de 2025 — The paper proposes a large scale diverse dataset called Mega-MMDF an...

deepwiki.com

**Getting Started | grip-unina/noiseprint | DeepWiki**

22 de maio de 2025 — It covers the essential workflows for extracting camera model fingerprint...

deepwiki.com

**Demo Scripts | grip-unina/noiseprint | DeepWiki**

22 de maio de 2025 — This document covers the automated demo workflows provided in the...

deepwiki.com

**grip-unina/noiseprint | DeepWiki**

22 de maio de 2025 — This document provides a high-level overview of the Noiseprint system, a...

deepwiki.com

**Command Line Tools | grip-unina/noiseprint | DeepWiki**

22 de maio de 2025 — This document provides a comprehensive guide to all command-line scripts...

deepwiki.com

**SCLBD/DeepfakeBench | DeepWiki**

24 de abril de 2025 — DeepfakeBench is a comprehensive benchmark platform for deepfake...

deepwiki.com

**System Architecture | SCLBD/DeepfakeBench | DeepWiki**

24 de abril de 2025 — The DeepfakeBench system consists of four main components: Preprocessing...

github.com

**Why “The real dataset is duplicated and split into train, test, and ...**

17 de março de 2025 — Each subset corresponds to a combination of deepfake and real videos fro...

pytutorial.com

**Python Seaborn Heatmap Tutorial: Data Visualization**

17 de dezembro de 2024 — Learn how to create stunning heatmaps using Python Seaborn. Master...

github.com

**Why do you discard the remaining data of DeepFakeDetection?**

7 de outubro de 2024 — Why do you discard the remaining data of DeepFakeDetection? · Issue #11...

github.com

**GitHub - AmjadOsman/Deepfake-Detection-Pipeline**

11 de setembro de 2024 — The script is designed to handle several preprocessing and training tasks...

github.com

**Classifying a new dataset · Issue #90 · SCLBD/DeepfakeBench - GitHub**

4 de julho de 2024 — By following these steps, you can seamlessly incorporate your own dataset into...

github.com

**Key Error in test.py · Issue #78 · SCLBD/DeepfakeBench - GitHub**

28 de maio de 2024 — I have run the scripts in the README.md. the error are follows: File...

github.com

**Daisy-Zhang/Awesome-Deepfakes-Detection - GitHub**

10 de maio de 2024 — A collection list of Deepfakes Detection related datasets, tools, paper...

arxiv.org

**Comprint: Image Forgery Detection and Localization using Compression ...**

13 de março de 2024 — This paper presents Comprint, a novel forgery detection and localizati...

github.com

**GitHub - IDLabMedia/fusion-idlab: Official implementation of ...**

4 de março de 2024 — The Jupyter notebook run_forgery_detection_fusion.ipynb gives an...

github.com

**massgravel/Microsoft-Activation-Scripts - GitHub**

29 de janeiro de 2024 — How to Activate Windows / Office / Extended Security Updates (ESU)? Click...

springer.com

**Image Forgery Detection Using Comprint: A Comprehensive Study**

14 de novembro de 2023 — In light of this, we present Comprint, an image forgery detection and...

github.com

**Releases: IDLabMedia/comprint - GitHub**

29 de outubro de 2023 — Comprint: Image Forgery Detection and Localization using Compression...

springer.com

**Comprint: Image Forgery Detection and Localization Using ... - Springer**

1 de agosto de 2023 — As such, the deep-learning method can extract a compression fingerprint or...

github.com

**comprint/code/train_network_siamese.py at main · IDLabMedia ... - GitHub**

21 de julho de 2023 — Comprint: Image Forgery Detection and Localization using Compression...

github.com

**Preprocessing · Issue #3 · SCLBD/DeepfakeBench - GitHub**

11 de julho de 2023 — However, it's possible that you may have missed specifying the specific datas...

github.com

**Noiseprint: a CNN-based camera model fingerprint - GitHub**

6 de julho de 2023 — Noiseprint, a CNN-based camera model fingerprint. Contribute to grip-...

arxiv.org

**DeepfakeBench: A Comprehensive Benchmark of Deepfake Detection**

4 de julho de 2023 — Featuring an extensible, modular-based codebase, DeepfakeBench contain...

huggingface.co

**DeepfakeBench: A Comprehensive Benchmark of Deepfake Detection**

3 de julho de 2023 — Featuring an extensible, modular-based codebase, DeepfakeBench contain...

github.com

**Speedup Comprint & Noiseprint Process · Issue #2 · IDLabMedia/comprint**

10 de maio de 2023 — I am trying to compute the Comprint and Noiseprint maps for a large director...

ugent.be

**IEEE ICCE - Comprint - Image Forgery Localization and Detection**

30 de novembro de 2022 — For more detailed insights on Comprint, check out our previous wor...

arxiv.org

**Comprint: Image Forgery Detection and Localization using Compression ...**

5 de outubro de 2022 — This paper presents Comprint, a novel forgery detection and localizati...

github.com

**comprint/README.md at main · IDLabMedia/comprint · GitHub**

28 de agosto de 2022 — The Jupyter notebook code/get_comprint_heatmap.ipynb gives an...

ugent.be

**MMForWild ICPR - Comprint - Image Forgery Localization and Detection**

16 de agosto de 2022 — Our paper presents Comprint, a novel forgery detection and localizati...

github.com

**IDLabMedia repositories · GitHub**

26 de fevereiro de 2022 — Short overview of useful NumPy functions for in the lab sessions of...

github.com

**deepfake-detection · GitHub Topics · GitHub**

19 de novembro de 2019 — Here are 454 public repositories matching this topic... A list of tools,...

stackoverflow.com

**heatmap - How to digitize (extract data from) a heat map image using ...**

24 de março de 2018 — There are multiple ways to do it, many Machine Learning libraries offering...

github.com

**Deepfake-Bench/README.md at main - GitHub**

17 de março de 2014 — DeepfakeBench will be continuously updated to track the latest advances...

github.com

**DeepFake/README.md at main · DRG31/DeepFake · GitHub**

17 de março de 2014 — DF40 is designed for seamless integration into the workflow of...

github.com

**DeepfakeBench/README.md at main · SCLBD/DeepfakeBench · GitHub**

We propose a highly generalizable and efficient detection method that can be used to detect both...

github.com

**GitHub - IDLabMedia/comprint: Comprint: Image Forgery Detection and ...**

The Jupyter notebook code/get_comprint_heatmap.ipynb gives an...

github.com

**DeepfakeBench/README.md at main - GitHub**

After running the above line, you will obtain the JSON files for each dataset in the...

github.com

**DeepfakeBench: A Comprehensive Benchmark of Deepfake Detection - GitHub**

All datasets used in DeepfakeBench can be downloaded from their own websites or...

github.com

**GitHub - YZY-stack/DF40: Official repository for the next-generation ...**

In this work, we propose: (1) a diverse deepfake dataset with 40 distinct generations methods; and...

github.com

**GitHub - DRG31/Deepfake: Deep fake Detection Project research**

DeepfakeBench follows a sequential workflow for face detection, alignment, and cropping. The...

github.com

**comprint/code/splicebuster/noiseprint/noiseprint.py at main ... - GitHub**

Comprint: Image Forgery Detection and Localization using Compression Fingerprints -...

github.com

**IDLab Media - GitHub**

IDLab Media has 19 repositories available. Follow their code on GitHub.

github.com

**GitHub - tufo830/DeepfakeBench: A comprehensive benchmark of deepfake ...**

Unified Platform: DeepfakeBench presents the first comprehensive benchmark for deepfake detection...

github.com

**GitHub - supriyosadhya/DeepfakeBench_extended: A...**

All datasets used in DeepfakeBench can be downloaded from their own websites or...

github.com

**Issues · SCLBD/DeepfakeBench · GitHub**

Can pre-training weights of UIA-ViT be provided? A comprehensive benchmark of deepfake detection....

github.com

**Activity · IDLabMedia/comprint · GitHub**

Comprint: Image Forgery Detection and Localization using Compression Fingerprints -...

github.com

**comprint/run-training.sh at main · IDLabMedia/comprint · GitHub**

Comprint: Image Forgery Detection and Localization using Compression Fingerprints -...

github.com

**GitHub**

\n Dependencies \n The code requires Python 3.X and was built with Tensorflow 2.15. Additionally,...

github.com

**yfiles-jupyter-graphs/examples/29_heat_mapping.ipynb at mai...**

The home of the Jupyter notebook graph visualization widget powered by yFiles for HTML -...

github.com

**heatmap/code22.py at main · vbertotto/heatmap · GitHub**

import streamlit as st import cv2 import numpy as np from ultralytics import YOLO, solutions import...

github.com

**StrokeViT/heatmap.py at main · phymucs/StrokeViT · GitHub**

Contribute to phymucs/StrokeViT development by creating an account on GitHub.

github.com

**colabtools/notebooks/colab-github-demo.ipynb at main · googlecolab ...**

Any time you open a GitHub hosted notebook in Colab, it opens a new editable view of the...

github.com

**folium/examples/Heatmap.ipynb at main - GitHub**

Contribute to python-visualization/folium development by creating an account on GitHub.

github.com

**noiseprint/noiseprint/noiseprint_blind.py at master · grip-unina ...**

Noiseprint, a CNN-based camera model fingerprint. Contribute to grip-unina/noiseprint development...

github.com

**noiseprint/main_blind.py at master · grip-unina/noiseprint**

Noiseprint, a CNN-based camera model fingerprint. Contribute to grip-unina/noiseprint development...

github.com

**fusion-idlab/forgery_detection.py at main · IDLabMedia/fusion ... - GitHub**

Official implementation of "Harmonizing Image Forgery Detection & Localization: Fusion of...

github.com

**ComfyUI_Noise/examples/example_unsample.png at master · BlenderNeko ...**

There was an error while loading. Please reload this page.

github.com

**comfyui_controlnet_aux/examples/comfyui-controlnet-aux-logo.png at main ...**

ComfyUI's ControlNet Auxiliary Preprocessors. Contribute to Fannovel16/comfyui_controlnet_aux...

github.com

**imaging/testdata/out_example.jpg at master · disintegration/imaging ...**

Uh oh! There was an error while loading. Please reload this page.

github.com

**BlindEditions/BlindEditions.lua at main · Mathguy23/BlindEditions**

Blinds may now have Editions. Contribute to Mathguy23/BlindEditions development by creatin...

github.com

**Pi-Somfy/documentation/p1.png at master · Nickduino/Pi-Somfy**

A script to open and close your Somfy (and SIMU) blinds with a Raspberry Pi and an RF emitter. - Pi-...

github.com

**Curie/benchmark/mle_bench/aptos2019-blindness-detection/report-fig ...**

Uh oh! There was an error while loading. Please reload this page.

github.com

**blindboss/blindboss.lua at main · Minirebel/blindboss · GitHub**

Contribute to Minirebel/blindboss development by creating an account on GitHub.

github.com

**bgf/images/code_struct.png at master · imanlab/bgf · GitHub**

There was an error while loading. Please reload this page.

github.com

**SegLossOdyssey/test/LossOverview.PNG at master · JunMa11/SegLossOdyssey ...**

Uh oh! There was an error while loading. Please reload this page.

github.com

**basic-android-kotlin-compose-training-practice-problems/Unit 1/Pathway ...**

Contribute to google-developer-training/basic-android-kotlin-compose-training-practice-...

github.com

**GitHub - vardhan1379/Deepfake-Bench**

DeepfakeBench follows a sequential workflow for face detection, alignment, and cropping. The...

github.com

**DeepfakeBench/training/detectors/meso4_detector.py at main · SCLBD ...**

A comprehensive benchmark of deepfake detection. Contribute to SCLBD/DeepfakeBench...

deepwiki.com

**Face Extraction and Alignment | SCLBD/DeepfakeBench | DeepWiki**

This page documents the face extraction and alignment component of DeepfakeBench's...

deepwiki.com

**Evaluation System | SCLBD/DeepfakeBench | DeepWiki**

This document explains the architecture, workflow, and components of the evaluation system, which...

deepwiki.com

**Data Loading and Augmentation | SCLBD/DeepfakeBench | DeepWiki**

This page documents the data loading and augmentation system in DeepfakeBench, explainin...

deepwiki.com

**SCLBD/DeepfakeBench | DeepWiki**

DeepfakeBench is a comprehensive benchmark platform for deepfake detection research. This...

deepwiki.com

**Metrics and Analysis | SCLBD/DeepfakeBench | DeepWiki**

This document provides a detailed explanation of the metrics and analysis system used in...

springer.com

**Comprint: Image Forgery Detection and Localization Using Compression**

Therefore, we propose (b) Comprint, a forgery detection and localization method based on...

springer.com

**Image Forgery Detection Using Comprint: A Comprehensive Study - Springer**

This section gives an overview of the Comprint method, which combines two main research...

arxiv.org

**Comprint: Image Forgery Detection and Localization using Compression ...**

exploit compression artifacts and extract a different finger-print, called Comprint. It utilizes compressi...

arxiv.org

**Training Data Improvement for Image Forgery Detection using Comprint**

To detect forgeries, the Comprint-algorithm transforms an image to a compression fingerprint...

arxiv.org

**Abstract - arXiv.org**

ibility. Featuring an extensible, modular-based codebase, DeepfakeBench contains 15 state-of-th...

google.com

**29_heat_mapping.ipynb - Colab**

We use a graph representing a production chain and add our heatmap according to the expected...

google.com

**heatmaps.ipynb - Colab**

A heatmap generated with Ultralytics YOLO26 transforms complex data into a vibrant, color-cod...

ugent.be

**Comprint: Image Forgery Detection and Localization using Compression ...**

As such, Comprint combines the exploitation of compression artifacts with one-class deep learnin...

neurips.cc

**DeepfakeBench: A Comprehensive Benchmark of Deepfake Detection - NeurIPS**

Featuring an extensible, modular-based codebase, \textit {DeepfakeBench} contains 15 state-of-the-a...

python-graph-gallery.com

**Heatmap - Python Graph Gallery**

A heatmap is a graphical representation of data where each value of a matrix is represented as a...

pydata.org

**seaborn.heatmap — seaborn 0.13.2 documentation**

This is an Axes-level function and will draw the heatmap into the currently-active Axes if none is...

imec.be

**Methods - COM-PRESS**

By combining these methods into a single heatmap, it may be easier to make conclusions. It learned th...