

Memulihkan Reproduksiabilitas Bit-Eksak pada *Benchmark* Steganalisis di bawah Kompresi JPEG Berantai dan Derau Gauss melalui Pemulihan *Salt* Kondisi dan Audit Artefak Pelatihan

Nurdin Wibisono

Magister Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang, Indonesia
e-mail: avt.deto@gmail.com

Muhammad Faisal, Fresy Nugroho, Yunifa Miftachul Arif

Magister Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang, Indonesia
e-mail: mfaaisal@ti.uin-malang.ac.id , fresy@ti.uin-malang.ac.id , yunif4@ti.uin-malang.ac.id

Imam Ashar, Suko Wiyanto

Telekomunikasi Militer, Politeknik Angkatan Darat (POLTEKAD), Kota Batu, Indonesia
e-mail: imamasharmit@gmail.com , sukowiyanto@poltekad.ac.id

Nursih Dwi Hastuti

Teknik Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang, Indonesia
e-mail: nursih.dwi@gmail.com

Abstrak

Klaim ketahanan untuk *steganalyzer deep learning* hanya dapat dipercaya sejauh citra *benchmark* tempat klaim itu diukur dapat dipercaya, padahal dataset steganalisis jarang diperiksa dari segi regenerasi bit-eksak, konsistensi artefak pelatihan, maupun statistik yang dapat dihitung ulang. Penelitian ini membangun *benchmark* berpasangan dan terpisah per sumber (*source-disjoint*) dengan 13.000 citra (2.500 sumber *Break Our Steganographic System* (BOSSbase), *Adaptive Least Significant Bit-Modified Median Edge Detector* (LSB-MMED) pada 0,2 bit per piksel, sembilan kondisi kompresi *Joint Photographic Experts Group* (JPEG) dan derau Gauss tunggal maupun berantai) beserta manifest *Secure Hash Algorithm 256-bit* (SHA-256), metadata Croissant 1.0 yang tervalidasi, sembilan eksekusi detektor *baseline*, serta protokol reproduksiabilitas empat tingkat yang mencakup integritas berkas, regenerasi bit-eksak, audit artefak pelatihan, dan penghitungan ulang metrik. Kode dan artefak tersebut tersedia di Zenodo (<https://doi.org/10.5281/zenodo.23524043>). Seluruh 12.200 berkas yang diperiksa langsung (subset pelatihan dan uji) cocok dengan manifest; 800 berkas validasi mengacu pada verifikasi terdahulu. Dalam lingkungan audit yang dilaporkan, pembagian data, seluruh 2.100 hasil penyisipan yang tersedia, serta kedua kondisi JPEG teregenerasi identik byte demi byte. Sebaliknya, keenam kondisi derau meregenerasi 0% berkasnya pada setiap pengaturan *interpreter* yang diuji, karena *seed* derau bergantung pada keluaran fungsi *hash* string Python yang kuncinya diacak per proses. Pencarian menyeluruh atas 65.536 kemungkinan *salt* kondisi, yaitu 16 bit terendah keluaran hash(c), mengidentifikasi satu nilai unik per kondisi hanya dari lima sampai tujuh piksel dan memulihkan regenerasi identik byte demi byte atas seluruh 6.000 berkas ber-derau. Kunci *hash* dan nilai PYTHONHASHSEED asli tetap tidak diketahui; tidak ada nilai PYTHONHASHSEED hingga $1,2 \times 10^7$ yang menghasilkan *salt* tersebut. Pada satu detektor dan satu kondisi derau ($\sigma = 2$), empat realisasi derau alternatif mengubah *area under the receiver operating characteristic curve* (AUC) dalam rentang 0,010. Audit artefak tidak menemukan inkonsistensi pada sembilan eksekusi yang dirilis, dengan satu eksekusi berstatus peringatan karena riwayat pelatihannya tidak lengkap, dan mendeteksi keempat kesalahan yang disuntikkan, sedangkan penghitungan ulang metrik dari prediksi per citra mengoreksi kesalahan agregasi satu-*seed* dan uji DeLong yang mengabaikan kovarians; koreksi kedua mengubah satu keputusan signifikansi nominal yang tidak bertahan setelah koreksi perbandingan berganda. Oleh karena itu, *seed* tetap dan kode yang dipublikasikan tidak cukup; *seed* tidak boleh diturunkan dari *hash* berkunci acak, dan regenerasi harus diverifikasi pada tingkat byte.

Kata Kunci: *benchmark* steganalisis, reproduksiabilitas, asal-usul data, pengacakan *hash*, Adaptive LSB-MMED

Abstract

Robustness claims for deep steganalyzers are only as trustworthy as the benchmark images on which they were measured, yet steganalysis datasets are rarely checked for bit-exact regeneration, consistent training artifacts, or recomputable statistics. This study builds a paired, source-disjoint benchmark of 13,000 images

(2,500 *Break Our Steganographic System* (BOSSbase) sources, Adaptive *Least Significant Bit-Modified Median Edge Detector* (LSB-MMED) at 0.2 bits per pixel, nine single and cascaded *Joint Photographic Experts Group* (JPEG) and Gaussian-noise conditions) with a *Secure Hash Algorithm 256-bit* (SHA-256) manifest, validated Croissant 1.0 metadata, nine baseline detector runs, and a four-level reproducibility protocol covering file integrity, bit-exact regeneration, training-artifact audit, and metric recomputation. The code and artifacts are available on Zenodo (<https://doi.org/10.5281/zenodo.23524043>). All 12,200 directly checked files (training and test subsets) matched the manifest; the 800 validation files rely on an earlier verification. In the reported audit environment, the split, all 2,100 available embeddings, and both JPEG conditions regenerated byte-identically. The six noise conditions, in contrast, regenerated 0% of their files under every tested interpreter setting, because the noise seed depended on the output of Python's string hash, whose key is randomized per process. An exhaustive search over the 65,536 possible condition salts, i.e., the low 16 bits of hash(c), identified a unique value per condition from only five to seven pixels and restored byte-identical regeneration of all 6,000 noisy files. The original hash key and PYTHONHASHSEED remain unknown; no PYTHONHASHSEED value up to 1.2×10^7 reproduces these salts. For one detector and one noise condition ($\sigma = 2$), four alternative noise realizations varied the *area under the receiver operating characteristic curve* (AUC) within a range of 0.010. The artifact audit found no inconsistency in the nine released runs, with one run flagged by a warning for an incomplete training history, and detected all four injected faults, and metric recomputation from per-image predictions corrected a single-seed aggregation error and a DeLong test that ignored covariance; the latter correction changed one nominal significance decision that does not survive multiple-comparison correction. Fixed seeds and published code are therefore not sufficient; seeds must not be derived from randomly keyed hashes, and regeneration must be verified at the byte level.

Keywords: steganalysis benchmark, reproducibility, data provenance, hash randomization, Adaptive LSB-MMED

PENDAHULUAN

Steganalisis *deep learning* tidak lagi dinilai semata-mata dari seberapa baik sebuah detektor memisahkan citra *cover* dan *stego* pada kondisi laboratorium, tetapi semakin dari apakah kemampuan itu bertahan setelah citra melewati kanal yang realistis. Detektor seperti *Steganalysis Residual Network* (SRNet) (Boroumand dkk., 2019), WSErNet, jaringan untuk ekstraksi dan penguatan sinyal steganografi lemah (Liang dan Li, 2026), dan HSMNet, jaringan multikanal dengan konvolusi dilatasi hibrida dan *self-attention* (Yang dkk., 2026) kini rutin diuji terhadap kompresi ulang *Joint Photographic Experts Group* (JPEG), derau aditif, dan ketidakcocokan sumber *cover* (*cover-source mismatch*) (Giboulot dkk., 2020; Mallet dkk., 2024; Alrusaini, 2025; Yousfi dan Fridrich, 2020), dan studi eksperimental serta ulasan terbaru melaporkan bahwa pascapemrosesan semacam itu dapat menghapus sebagian besar keunggulan deteksi *steganalyzer* berbasis *convolutional neural network*, sehingga ketahanan tetap menjadi persoalan terbuka (Alrusaini, 2025; Kheddar dkk., 2024). Sulitnya membawa steganalisis dari laboratorium ke dunia nyata telah disadari lebih dari satu dekade lalu (Ker dkk., 2013). Namun, setiap klaim ketahanan tersebut bertumpu pada *benchmark* yang setiap varian citra hasil pascapemrosesannya harus persis sama dengan yang dievaluasi para penulis. Persyaratan ini jauh dari sepele untuk kondisi ber-derau. Realisasi derau yang berbeda mengubah hampir setiap piksel citra uji ber-derau, tetapi tidak mengubah tampilannya secara visual, sehingga *benchmark* yang tidak dapat diregenerasi bit

demi bit tidak dapat diperiksa secara independen, dan dua studi yang mengira memakai set uji yang sama mungkin sebenarnya mengukur citra yang berbeda.

Masalah ini merupakan bagian dari kekhawatiran reproduksibilitas yang lebih luas dalam *machine learning*. Laporan dari program reproduksibilitas berskala besar, survei atas karya yang telah terbit, dan ulasan terbaru berulang kali mengidentifikasi kode yang hilang, keacakan yang tidak terdokumentasi, kebocoran data, dan pelaporan selektif sebagai penyebab berulang hasil yang tidak dapat direproduksi (Pineau dkk., 2021; Gundersen dkk., 2022; Gundersen dan Kjensmo, 2018; Kapoor dan Narayanan, 2023; Semmelrock dkk., 2025). Namun, sebagian besar studi tentang nondeterminisme meninjau proses pelatihan, tempat inisialisasi acak, urutan data, dan *kernel graphics processing unit* (GPU) yang nondeterministik mengubah model yang dihasilkan dan skor yang dilaporkan (Zhuang dkk., 2022; Pham dkk., 2020; Bouthillier dkk., 2021). Perhatian terhadap nondeterminisme dalam pembangkitan data jauh lebih sedikit, padahal sama mampunya mengubah set evaluasi secara diam-diam dan, tidak seperti varians pelatihan, tidak dirata-ratakan oleh pengulangan eksekusi. Rekayasa perangkat lunak menangani masalah yang sama melalui *reproducible builds*, yang menganggap sebuah artefak telah direproduksi hanya jika pembangunan ulang secara independen menghasilkan keluaran yang identik bit demi bit (Lamb dan Zacchiroli, 2022). Dokumentasi dataset telah berkembang seiring dengan *datasheet* (Gebru dkk., 2021), prinsip *Findable, Accessible, Interoperable, Reusable* (FAIR) (Wilkinson dkk., 2016), format metadata

Croissant untuk dataset *machine learning* (Akhtar dkk., 2024), dan audit skala besar atas asal-usul dataset (Longpre dkk., 2024). Upaya-upaya ini menjelaskan isi sebuah dataset dan asalnya, tetapi tidak membuktikan bahwa berkas-berkasnya dapat dibangun ulang dari prosedur yang dipublikasikan.

Benchmark steganalisis menggambarkan celah ini dengan jelas. *Break Our Steganographic System* (BOSSbase) 1.01 menyediakan 10.000 citra *cover* dan menyerahkan penyisipan serta seluruh pemrosesan selanjutnya kepada masing-masing pengguna (Bas dkk., 2011). ALASKA2 (Cogranne dkk., 2020), IStego100K (Yang dkk., 2020), dan REVEAL (Kombrink dkk., 2025) memperbesar jumlah citra, kamera, dan algoritma penyisipan, yang menjawab persoalan keragaman dan ketidakcocokan sumber *cover*. Namun, tidak satu pun dirancang di sekitar tiga pertanyaan yang harus dijawab oleh studi ketahanan sebelum kesimpulannya dapat dipercaya: (i) dapatkah setiap varian hasil pascapemrosesan diregenerasi secara bit-eksak dari kode yang dirilis, (ii) apakah artefak pelatihan yang dirilis, seperti *checkpoint*, ambang keputusan, dan riwayat pelatihan, konsisten satu sama lain, dan (iii) dapatkah setiap statistik yang dilaporkan dihitung ulang dari prediksi per citra? Tanpa pemeriksaan ini, kesalahan dalam penentuan *seed*, eksekusi pelatihan yang dilanjutkan dari *checkpoint* perantara, atau jalan pintas dalam cara hasil diagregasi dapat merambat tanpa terdeteksi ke dalam kesimpulan yang dipublikasikan.

Sepengetahuan kami, tidak ada *benchmark* steganalisis yang telah dipublikasikan melaporkan uji regenerasi tingkat byte untuk citra hasil pascapemrosesannya, dan tidak ada studi yang mengukur seberapa besar metrik deteksi yang dilaporkan bergantung pada realisasi derau tertentu yang kebetulan terbangkit. Studi ketahanan terhadap derau aditif sangat rentan terhadap kelemahan ini, karena medan derau berasal dari pembangkit bilangan semu-acak yang *seed*-nya biasanya diturunkan di dalam skrip pembangkitan dan jarang didokumentasikan. Oleh karena itu, penelitian ini memperlakukan kemampuan regenerasi, konsistensi artefak, dan kemampuan penghitungan ulang sebagai sifat *benchmark* yang dapat diukur, bukan sebagai asumsi, dan menguji masing-masing dengan pemeriksaan yang dapat dieksekusi alih-alih deskripsi tertulis.

Artikel ini menjawab celah tersebut dengan sebuah *benchmark* untuk mengevaluasi *steganalyzer deep learning* terhadap penyisipan Adaptive Least Significant Bit-Modified Median Edge Detector (LSB-MMED) (Chakraborty dkk., 2017) pada kompresi JPEG dan derau Gauss tunggal maupun berantai, yang dirilis bersama protokol reproduksibilitas empat tingkat. Ketika protokol ini diterapkan pada *benchmark* kami sendiri, ia

menunjukkan bahwa pengaman yang lazim, yaitu kode yang dipublikasikan, *seed* global tetap, dan manifes *digest*, tidak cukup untuk meregenerasi enam dari sembilan kondisi uji, dan ia memungkinkan informasi yang hilang dipulihkan dari berkas-berkas yang dirilis itu sendiri. Kontribusinya adalah sebagai berikut.

(1) *Benchmark* berpasangan dan terpisah per sumber yang terdiri atas 13.000 berkas (2.500 sumber BOSSbase, sembilan kondisi uji) yang manifesnya mengunci setiap berkas dengan *Secure Hash Algorithm 256-bit* (SHA-256) dan dipublikasikan sebagai metadata Croissant 1.0 yang tervalidasi, beserta *checkpoint* dan prediksi per citra dari sembilan eksekusi detektor *baseline*. (2) Protokol audit empat tingkat (R1 integritas berkas, R2 regenerasi bit-eksak, R3 audit artefak pelatihan, R4 penghitungan ulang metrik) yang diimplementasikan sebagai peralatan yang dirilis dan berjalan pada *central processing unit* (CPU) biasa. (3) Penemuan nondeterminisme tersembunyi dalam pembangkitan data: *seed* derau bergantung pada keluaran fungsi *hash* string Python yang kuncinya diacak per proses, sehingga enam dari sembilan kondisi uji tidak dapat diregenerasi pada pengaturan *interpreter* yang diuji. Kami memulihkan *salt* kondisi 16-bit yang hilang (16 bit terendah keluaran *hash()*, bukan kunci *hash* maupun PYTHONHASHSEED) melalui pencarian menyeluruh, menjelaskan jumlah piksel yang dibutuhkan pencarian dengan argumen probabilistik sederhana, dan memulihkan regenerasi bit-eksak 100% dalam lingkungan audit tanpa mengubah satu pun berkas yang dirilis. (4) Demonstrasi kuantitatif bahwa audit menangkap cacat yang nyata maupun yang disuntikkan: audit artefak pelatihan menandai keempat kesalahan yang disuntikkan, dan penghitungan ulang metrik mengoreksi kesalahan agregasi serta kesalahan uji statistik pada keluaran *pipeline* asli.

Bagian selanjutnya dari artikel ini disusun sebagai berikut. Bagian Metode menjelaskan *benchmark*, model determinisme yang mendasari audit, keempat tingkat audit, dan prosedur pemulihan *salt*. Bagian Hasil dan Pembahasan melaporkan hasil setiap tingkat audit, mengukur pengaruh realisasi derau terhadap sebuah metrik deteksi, dan membahas implikasi praktisnya, dan bagian Penutup menyimpulkan artikel ini.

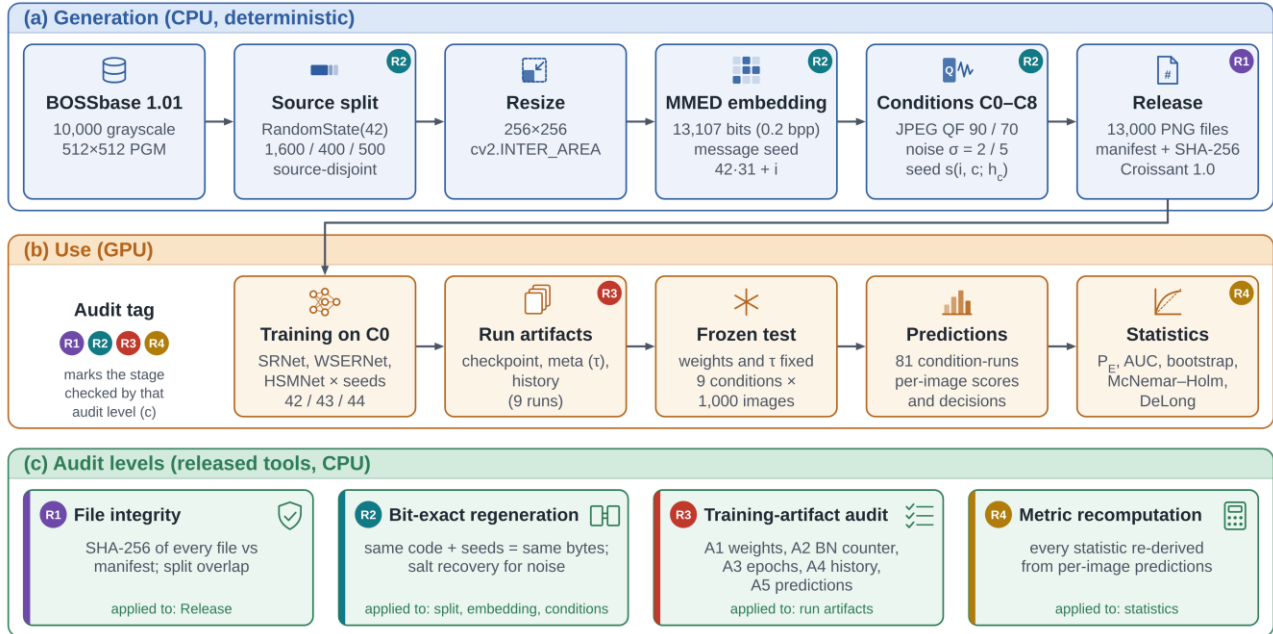
METODE

Desain Benchmark

Gambar 1 merangkum *pipeline* pembangkitan, artefak yang dirilis, dan titik penerapan masing-masing dari keempat tingkat audit. Sumbernya adalah 10.000 citra skala keabuan 512×512 dari BOSSbase 1.01 (Bas dkk., 2011). RandomState(42) milik NumPy mengundi 2.500 indeks tanpa pengembalian dan membaginya menurut urutan undian menjadi 1.600 sumber pelatihan, 400

sumber validasi, dan 500 sumber uji. Karena pembagian dilakukan pada tingkat citra sumber, tidak ada identitas sumber (indeks BOSSbase) yang muncul pada dua subset. Jaminan ini terbatas pada identitas sumber: pembagian tersebut tidak menguji apakah dua citra BOSSbase dengan indeks berbeda memiliki konten yang mirip, misalnya adegan yang serupa, dan pemeriksaan kemiripan konten

(*near-duplicate*) tidak dilakukan dalam penelitian ini. Setiap citra diubah ukurannya menjadi 256×256 dengan cv2.INTER_AREA sebelum penyisipan; citra hasil perubahan ukuran itu dengan demikian adalah *cover* itu sendiri, bukan kondisi uji, dan setiap artefak perubahan ukuran dimiliki bersama oleh *cover* dan versi *stego*-nya.



Gambar 1. Pipeline pembangkitan, artefak yang dirilis, dan keempat tingkat audit pada *benchmark*

Citra *stego* dihasilkan dengan Adaptive LSB-MMED (Chakraborty dkk., 2017), skema penggantian LSB adaptif-tepi dan non-butu (*non-blind*). Skema ini mengurutkan piksel berdasarkan galat prediksi dari templat kausal, sehingga piksel pada daerah bertekstur dan tepi dipakai lebih dahulu, dan menyisipkan 3, 2, atau 1 bit per piksel pada tiga zona (G3, G2, G1) yang masing-masing menerima 60%, 30%, dan 10% dari *payload*. *Payload*-nya sebesar 13.107 bit, setara dengan 0,2 bit per piksel untuk citra 256×256. Pesan untuk sumber i adalah untai bit semu-acak dari pembangkit yang di-*seed* dengan

$$s_{\text{msg}}(i) = (42 \cdot 31 + i) \bmod (2^{31} - 1). \quad (1)$$

Pelatihan dan validasi hanya memuat kondisi C0 yang tidak ditransformasi. Setiap sumber uji direplikasi menjadi sembilan kondisi (Tabel 1): *baseline* tanpa transformasi, faktor kualitas (*quality factor*, QF) JPEG 90 dan 70, derau Gauss aditif dengan $\sigma = 2$ dan $\sigma = 5$, serta keempat kaskade yang kompresi JPEG-nya diikuti derau. *Cover* dan *stego* dari sumber yang sama selalu menerima transformasi dan medan derau yang sama. Pemasangan ini membuat selisih *cover*–*stego* dapat dikaitkan dengan penyisipan, bukan dengan dua pengundian derau yang independen, dan memungkinkan uji statistik berpasangan. Citra ber-derau

mengikuti $Y = \text{clip}(\text{round}(X + N), 0, 255)$ dengan $N \sim \mathcal{N}(0, \sigma^2)$ yang diundi dari pembangkit yang di-*seed* dengan

$$s_{\text{noise}}(i, c) = (42 \cdot 1000003 + 1009i + h_c) \bmod (2^{31} - 1), \quad (2)$$

dengan $h_c = \text{hash}(c) \& 0xFFFF$, yaitu 16 bit terendah dari keluaran fungsi hash() Python untuk pengenalan string c milik kondisi tersebut. Dalam artikel ini, h_c disebut *salt* kondisi. Besaran ini perlu dibedakan dari dua besaran lain: kunci internal SipHash 128-bit yang dipakai hash() untuk menghitung keluaran tersebut, dan nilai variabel lingkungan PYTHONHASHSEED yang, jika ditetapkan, menentukan kunci itu. Pengacakan per proses terjadi pada kunci; h_c hanyalah akibatnya. Persamaan (2) diambil apa adanya dari *pipeline* asli. Sekilas persamaan ini tampak deterministik, tetapi bagian Hasil dan Pembahasan menunjukkan bahwa h_c adalah satu-satunya titik tempat regenerasi gagal.

Tabel 1. Kondisi uji pada *benchmark* (500 sumber, 1.000 citra per kondisi)

ID	Transformasi	Seed derau ber-salt
C0	tanpa (<i>baseline</i>)	—
C1	JPEG, QF = 90	—
C2	JPEG, QF = 70	—
C3	derau Gauss, $\sigma = 2$	ya
C4	derau Gauss, $\sigma = 5$	ya
C5	JPEG QF 90 $\rightarrow$ derau $\sigma = 2$	ya
C6	JPEG QF 90 $\rightarrow$ derau $\sigma = 5$	ya
C7	JPEG QF 70 $\rightarrow$ derau $\sigma = 2$	ya
C8	JPEG QF 70 $\rightarrow$ derau $\sigma = 5$	ya

Manifes dan Metadata

Setiap berkas didaftar dalam `manifest.csv` (13.000 baris) dengan kolom pada Tabel 2, antara lain jalur berkas *Portable Network Graphics* (PNG) serta ukuran *peak signal-to-noise ratio* (PSNR) dan *structural similarity index measure* (SSIM). Digest SHA-256 menjadi acuan untuk integritas maupun regenerasi, karena dua berkas dengan digest yang sama dapat dianggap identik byte demi byte. Karena citra BOSSbase tidak boleh didistribusikan ulang, rilisan terdiri atas kode, manifes, dan metadata; pengguna membangun ulang citra secara lokal dan memverifikasinya terhadap digest. Manifes tersebut juga dideskripsikan dalam Croissant 1.0 dalam format *JavaScript Object Notation for Linked Data* (JSON-LD) (Akhtar dkk., 2024), yang mendeklarasikan objek berkas, digest-nya, dan sebuah record set dengan kolom bertipe, sehingga loader standar dapat membaca benchmark tanpa parsing khusus. Metadata divalidasi dengan memuat seluruh 13.000 record menggunakan pustaka acuan `mlcroissant`.

Tabel 2. Skema manifes (satu baris per berkas)

Kolom	Tipe	Isi
source_id	bilangan bulat	indeks BOSSbase; satuan pembagian dan <i>resampling</i> klaster
subset	teks	train, val, atau test
condition_id	teks	C0–C8
label	bilangan bulat	0 = <i>cover</i> , 1 = <i>stego</i>
relative_path	teks	lokasi PNG 256×256
psnr, ssim	float	kualitas citra <i>stego</i> C0
changed_pixels	bilangan bulat	jumlah piksel yang berubah akibat penyisipan
sha256	teks	<i>digest</i> berkas PNG

Detektor *Baseline* dan Artefak Pelatihan yang Dirilis

Tiga detektor dilatih pada C0 dengan masing-masing tiga *seed* (42, 43, 44): SRNet (2.416.354 parameter, *batch* 16) (Boroumand dkk., 2019), WSERNet (3.572.600 parameter, *batch* 8) (Liang dan Li, 2026), dan HSMNet (1.437.714 parameter, *batch* 8) (Yang dkk., 2026). Semua eksekusi memakai pengoptimal *adaptive moment estimation* (Adam; laju pembelajaran 10^{-3} , *weight decay* 10^{-4}), paling banyak 100 *epoch*, *early stopping* dengan *patience* 15 *epoch*, dan jadwal ReduceLROnPlateau.

Untuk setiap eksekusi, rilisan memuat *checkpoint* terbaik, berkas metadata dengan ambang keputusan τ yang dipilih pada set validasi, riwayat per-*epoch*, dan berkas prediksi per citra untuk 9.000 citra uji. Bobot dan τ dibekukan sebelum pengujian, sehingga setiap kondisi dievaluasi dengan titik operasi yang dipelajari pada C0, sebagaimana detektor yang diterapkan di lapangan. Tabel 3 mencantumkan eksekusi, biayanya, dan *area under the receiver operating characteristic curve* (AUC) uji pada C0. Hanya HSMNet dengan *seed* 43 yang mendeteksi penyisipan secara jelas di atas tingkat acak, dan ambang identik sebesar 0,990 yang dimiliki ketiga eksekusi SRNet mencerminkan skor keluaran yang hampir konstan. Oleh karena itu, *baseline* dirilis sebagai artefak acuan untuk audit, bukan sebagai detektor yang kompetitif.

Tabel 3. Eksekusi *baseline* yang dirilis bersama *benchmark* (AUC uji pada C0 dihitung ulang dari prediksi per citra; t.t. = tidak tercatat)

Eksekusi	Epoch	Epoch terbaik	Jam GPU	τ	AUC uji (C0)
SRNet-42	t.t.	37	t.t.	0,990	0,500
SRNet-43	39	24	0,81	0,990	0,500
SRNet-44	56	41	1,04	0,990	0,500
WSERNet-42	100	95	3,57	0,493	0,500
WSERNet-43	83	68	4,88	0,510	0,500
WSERNet-44	85	70	3,60	0,499	0,500
HSMNet-42	100	95	4,92	0,497	0,527
HSMNet-43	100	100	4,31	0,517	0,875
HSMNet-44	100	100	4,41	0,495	0,537

Variasi antar-eksekusi pada Tabel 3 perlu dibaca bersama informasi pelatihannya. Ketiga eksekusi HSMNet memakai arsitektur, data, dan hiperparameter yang sama dan hanya berbeda pada *seed*, tetapi AUC uji C0-nya 0,527; 0,875; dan 0,537. Tabel 3 menunjukkan bahwa ketiganya dihentikan oleh batas anggaran 100 *epoch*, bukan oleh *early stopping*: *epoch* terbaik HSMNet-43 dan HSMNet-44 adalah *epoch* terakhir yang diizinkan, dan HSMNet-42 mencapai *epoch* terbaik pada *epoch* 95. Ketiganya dengan demikian dihentikan oleh anggaran, bukan oleh konvergensi. Riwayat yang dirilis menunjukkan bahwa *loss* validasi HSMNet-43 bertahan dekat $\ln 2 \approx 0,693$, yaitu nilai entropi silang untuk tebakan acak, selama sekitar 55 *epoch* pertama dan baru turun di bawah 0,690 pada *epoch* 56, lalu turun tajam hingga 0,504 pada *epoch* 100, yang sekaligus menjadi *epoch* terbaiknya; pada sepuluh *epoch* terakhir nilainya masih berfluktuasi antara 0,504 dan 0,653, sehingga eksekusi ini pun belum menunjukkan tanda konvergen. Sebaliknya, *loss* validasi HSMNet-42 dan HSMNet-44 tidak pernah keluar dari dataran tersebut: sejak *epoch* 20 nilainya hanya berkisar 0,693–0,694 (HSMNet-42) dan 0,693 (HSMNet-44), *loss* pelatihan keduanya pada *epoch* 100 adalah 0,693, dan akurasi validasi akhirnya hanya 0,511 dan 0,515 (0,756 pada HSMNet-43). Karena selisih *loss* antar-*epoch* pada

kedua eksekusi itu kurang dari 0,0015, *epoch* terbaik mereka (95 dan 100) lebih mencerminkan fluktuasi kecil di sekitar $\ln 2$ daripada kemajuan belajar. Selisih 0,875 terhadap 0,527–0,537 karena itu paling konsisten dengan sensitivitas terhadap *seed*, khususnya kapan pelatihan keluar dari dataran (*plateau*) *loss* awal di bawah anggaran yang terbatas, bukan dengan perbedaan konfigurasi. Pola SRNet berbeda: SRNet-43 dan SRNet-44 berhenti melalui *early stopping* pada *epoch* 39 dan 56, tepat 15 *epoch* setelah *epoch* terbaiknya, dan ketiga eksekusi SRNet memiliki $\tau = 0,990$ yang identik serta skor keluaran yang hampir konstan, yang menunjukkan bahwa pelatihan tidak keluar dari solusi trivial. Ketiga eksekusi WSERNet berakhir dengan AUC 0,500, dua di antaranya melalui *early stopping*. Penelitian ini tidak menguji penyebab variasi tersebut secara terkendali, dan tiga *seed* per detektor tidak cukup untuk menaksir sebaran antar-*seed* secara andal. Karena itu, angka pada Tabel 3 diperlakukan sebagai artefak acuan untuk audit, bukan sebagai estimasi kinerja detektor.

Model Determinisme dan Protokol Audit

Sebuah berkas pada *benchmark* adalah fungsi deterministik $f = G(x_i; \theta, s)$ dari sumber BOSSbase x_i , parameter tetap θ (QF, σ , *payload*), dan *seed* s dari (1)–(2) serta dari pembangkit pembagian data. Oleh karena itu, reproduksibilitas bergantung pada dua asumsi: bahwa setiap *seed* hanya merupakan fungsi dari besaran yang tercatat, dan bahwa setiap pemanggilan pustaka mengembalikan keluaran yang sama untuk masukan yang sama. Protokol menguji kedua asumsi tersebut melalui empat tingkat, yang masing-masing menjawab pertanyaan tersendiri dan dapat gagal secara independen dari tingkat lainnya.

R1 Integritas berkas. *Digest* SHA-256 setiap berkas yang tersedia dibandingkan dengan manifes, dan ketiga subset diuji terhadap tumpang tindih sumber. Tingkat ini menjawab apakah berkas yang ada di tangan adalah berkas yang dipublikasikan.

R2 Regenerasi bit-eksak. Pembagian data diturunkan kembali dari daftar indeks BOSSbase; setiap *cover* C0 yang tersedia disisipi ulang dengan kode asli dan *seed* pesan (1); dan C1–C8 diregenerasi dari pasangan C0 yang tersimpan dengan kode transformasi asli. Sebuah berkas hasil regenerasi dinyatakan lolos hanya jika pengodean PNG-nya memiliki *digest* SHA-256 yang sama dengan berkas yang dirilis (identik byte); identitas piksel dicatat terpisah, sehingga perbedaan pengodean dapat dibedakan dari perbedaan konten. R2 dijalankan pada empat pengaturan *interpreter*: PYTHONHASHSEED=0 pada CPython 3.13, PYTHONHASHSEED=0 dengan *salt* yang dihasilkan CPython ≤ 3.10 , PYTHONHASHSEED tidak diatur, dan tabel *salt* kondisi yang dibekukan. Seluruh

pemeriksaan R2 dijalankan dalam lingkungan audit pada Tabel 4. Versi pustaka pada pembangkitan asli tidak tercatat, kecuali syarat $\text{OpenCV} \geq 4.8$, sehingga hasil R2 menyatakan keberhasilan regenerasi dalam lingkungan tersebut, bukan jaminan untuk lingkungan lain.

R3 Audit artefak pelatihan. Lima pemeriksaan diterapkan pada setiap eksekusi. A1 mensyaratkan *digest* SHA-256 atas seluruh tensor bobot berbeda dari *digest* setiap eksekusi lain. A2 mensyaratkan penghitung *batch normalization* (BatchNorm) sama dengan *epoch checkpoint* dikalikan jumlah iterasi per *epoch*,

$$n_{bt} = e^* \cdot [N_{\text{train}}/B], \quad (3)$$

dengan $N_{\text{train}} = 3.200$ citra dan ukuran *batch* B . Relasi (3) bukan aturan umum; ia berlaku di bawah empat asumsi tentang cara pelatihan dijalankan: (a) penghitung `num_batches_tracked` milik PyTorch bertambah satu kali untuk setiap *forward pass* dalam mode pelatihan dan tidak bertambah dalam mode evaluasi, sehingga validasi tidak memengaruhinya; (b) setiap *epoch* melewati seluruh N_{train} citra tepat satu kali, tanpa akumulasi gradien, *forward pass* tambahan dalam mode pelatihan, atau pelatihan paralel yang memperbarui salinan *buffer* secara terpisah; (c) *checkpoint* disimpan pada akhir *epoch* e^* , dan *buffer BatchNorm* ikut tersimpan bersama *checkpoint*; skrip pelatihan yang dirilis tidak memiliki mekanisme melanjutkan pelatihan; dan (d) pembulatan ke atas berlaku bila *batch* terakhir yang tidak penuh tidak dibuang. Pada penelitian ini N_{train} habis dibagi kedua ukuran *batch*, sehingga asumsi (d) tidak berpengaruh. Pelanggaran salah satu asumsi akan menghasilkan FAIL pada A2 tanpa adanya kesalahan, sehingga A2 perlu disesuaikan dengan kode pelatihan sebelum diterapkan pada *pipeline* lain. A3 mensyaratkan *epoch checkpoint*, *epoch* terbaik pada metadata, dan minimum riwayat *loss* validasi saling cocok. A4 mensyaratkan riwayat itu ada, mencakup *epoch* terbaik, dan tidak identik dengan riwayat eksekusi lain. A5 mensyaratkan berkas prediksi memuat grid lengkap $500 \times 9 \times 2$ dengan satu τ yang dibekukan dan keputusan sama dengan $[\text{prob} \geq \tau]$. Setiap pemeriksaan menghasilkan salah satu dari tiga status: PASS berarti tidak ditemukan inkonsistensi pada artefak yang tersedia, FAIL berarti ditemukan inkonsistensi, dan WARN berarti pemeriksaan tidak dapat dijalankan sepenuhnya karena artefak yang diperlukan tidak ada. Ketika riwayat tidak ada, A4 melaporkan WARN dan A3 hanya memeriksa kecocokan *epoch checkpoint* dengan metadata, karena catatan yang hilang merupakan celah dokumentasi, bukan bukti adanya kesalahan. Dengan demikian, PASS menyatakan konsistensi, bukan kelengkapan dokumentasi: sebuah eksekusi dapat lolos semua pemeriksaan yang dapat dijalankan sementara catatan pelatihannya tetap tidak lengkap. Untuk menguji audit itu sendiri, empat kesalahan

disuntikkan ke salinan artefak: F1 *checkpoint* yang diganti dengan bobot eksekusi lain, F2 kolom *epoch* yang disunting, F3 riwayat yang disalin dari eksekusi lain, dan F4 berkas prediksi yang τ -nya diubah pada separuh barisnya.

R4 Penghitungan ulang metrik. Semua statistik diturunkan kembali dari prediksi per citra: $P_E = (P_{FA} + P_{MD})/2$ pada τ yang dibekukan, dengan P_E probabilitas galat, P_{FA} probabilitas alarm palsu (*false alarm*), dan P_{MD} probabilitas deteksi terlewat (*missed detection*), ROC-AUC dengan rumus peringkat, interval *cluster-bootstrap* 95% (5.000 *resample* atas 500 sumber) (Efron dan Tibshirani, 1993), uji McNemar (McNemar, 1947) dengan koreksi Holm (Holm, 1979) atas delapan perbandingan C1–C8 terhadap C0 dalam satu eksekusi, dan uji DeLong untuk kurva ROC berkorelasi yang memasukkan kovarians antara kedua AUC (DeLong dkk., 1988); nilai-*p* DeLong dilaporkan tanpa penyesuaian perbandingan berganda kecuali disebutkan lain. *Me-resample* sumber, bukan citra tunggal, menjaga *cover*, *stego*, dan semua kondisi dari satu sumber tetap bersama, yang menghormati rancangan berpasangan. Nilai hasil penghitungan ulang kemudian dibandingkan, nilai demi nilai, dengan tabel yang dihasilkan *pipeline* asli.

Pemulihan Salt

Ketika *salt* h_c pada (2) tidak diketahui, *salt* tersebut tetap dapat dipulihkan dari berkas yang dirilis, karena hanya ada $2^{16} = 65.536$ nilai yang mungkin dan masing-masing dapat diuji secara langsung. Untuk sumber i dan kondisi c , kandidat k dinyatakan konsisten jika K piksel pertama yang diregenerasi dengan *seed* $s_{\text{noise}}(i, c; h_c = k)$ sama dengan piksel yang dirilis:

$$\begin{aligned} \mathcal{K}_K(c) = \{k: \\ \text{clip}(\text{round}(x_{1:K} + N_{1:K}(k)), 0, 255) = y_{1:K}\}. \end{aligned} \quad (4)$$

Kandidat yang salah bertahan pada satu piksel hanya jika nilai derau yang dibulatkan kebetulan sama dengan nilai yang sebenarnya. Karena piksel masukan berupa bilangan bulat, hal ini terjadi dengan probabilitas $\rho = \sum_j p_j^2$, dengan p_j adalah probabilitas bahwa derau Gauss yang dibulatkan sama dengan j ; jauh dari batas *clipping*, $\rho \approx 1/(2\sigma\sqrt{\pi})$, yaitu sekitar 0,140 untuk $\sigma = 2$ dan 0,056 untuk $\sigma = 5$. Jumlah kandidat salah yang diharapkan bertahan setelah K piksel karena itu sekitar $65.535 \rho^K$, yang turun di bawah satu setelah enam piksel untuk $\sigma = 2$ dan empat piksel untuk $\sigma = 5$, dan di bawah 0,1 setelah masing-masing tujuh dan lima piksel. Pencarian dijalankan dengan $K = 64$ pada satu sumber per kondisi ber-derau, yang menyisakan margin keamanan lebar, dan setiap *salt* yang bertahan kemudian diverifikasi pada seluruh 1.000 citra kondisi tersebut. Untuk menentukan apakah *salt* yang dipulihkan sesuai dengan pengaturan *interpreter* mana pun

yang dapat direproduksi, *hash* string SipHash-1-3 dan SipHash-2-4 (Aumasson dan Bernstein, 2012) yang dipakai CPython ≥ 3.11 dan ≤ 3.10 (Heimes, 2013; Python Software Foundation, 2022) diimplementasikan, dicocokkan dengan *interpreter* yang berjalan, dan dievaluasi untuk setiap nilai PYTHONHASHSEED dari 0 hingga $1,2 \times 10^7$.

Prosedur ini perlu dibaca secara tepat. Yang dipulihkan hanyalah enam nilai h_c 16-bit, satu untuk setiap kondisi ber-derau; nilai-nilai ini cukup untuk menghitung ulang (2) dan karenanya cukup untuk regenerasi. Yang tetap tidak diketahui adalah kunci SipHash 128-bit, nilai PYTHONHASHSEED (jika ada), versi *interpreter* pada pembangkitan asli, dan 48 bit atas keluaran hash(c). Keenam *salt* kondisi secara bersama-sama hanya memuat 96 bit informasi, lebih sedikit daripada 128 bit kunci, sehingga kunci tersebut tidak dapat ditentukan secara unik dari *salt* yang dipulihkan.

Sensitivitas pada Tingkat Metrik

Untuk mengukur dampak realisasi derau saja terhadap hasil yang dilaporkan, model HSMNet *seed*-43 yang dibekukan dievaluasi pada CPU dengan presisi *floating point* 32-bit terhadap citra C3 yang dirilis dan terhadap C3 yang diregenerasi dengan empat *salt* alternatif yang diundi secara acak. Alternatif-alternatif ini persis merupakan realisasi derau yang akan dihasilkan kode asli di bawah *hash seed* lain, sehingga masing-masing adalah set uji yang secara masuk akal dapat diperoleh pengguna lain dari kode yang sama. AUC dan P_E pada τ yang dibekukan dibandingkan dengan interval *cluster-bootstrap* hasil C3 yang dirilis. HSMNet-43 dipilih berdasarkan kriteria kelayakan uji, bukan karena kinerjanya paling tinggi. Pengaruh realisasi derau terhadap AUC hanya dapat diamati pada detektor yang skornya memisahkan *cover* dan *stego*; pada eksekusi dengan AUC sekitar 0,5, pergeseran AUC akibat realisasi derau tidak dapat dibedakan dari fluktuasi di sekitar tingkat acak. Di antara sembilan eksekusi, hanya HSMNet-43 yang memenuhi kriteria ini. Konsekuensinya, analisis ini mengukur sensitivitas satu detektor yang berhasil belajar, yang menurut Tabel 3 justru tidak khas di antara ketiga *seed* HSMNet, sehingga hasilnya tidak mewakili eksekusi atau detektor lain. Semua audit dijalankan pada CPU dalam lingkungan yang tercantum pada Tabel 4.

Tabel 4. Lingkungan audit (lingkungan pembangkitan asli tidak tercatat, kecuali syarat OpenCV ≥ 4.8)

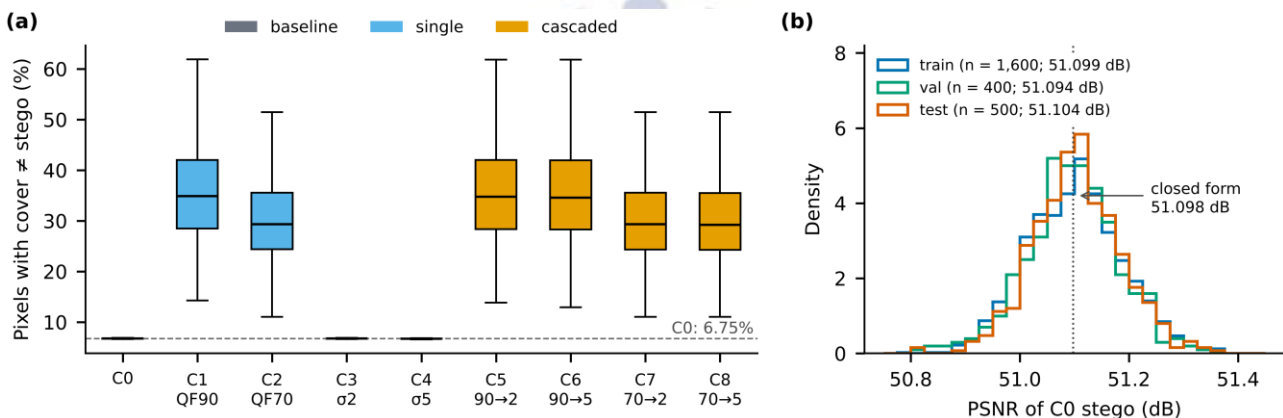
Komponen	Versi
Python	3.13.16
NumPy / SciPy / pandas	2.5.3 / 1.18.1 / 3.0.5
OpenCV / Pillow	5.0.0 / 12.3.0
PyTorch (CPU)	2.14.1
mlcroissant	1.x (validasi metadata)

HASIL DAN PEMBAHASAN

Karakterisasi *Benchmark*

Ke-2.500 citra *stego* memiliki PSNR sebesar $51,10 \pm 0,09$ dB dan SSIM (Wang dkk., 2004) sebesar 0,9989, dengan 4.423 ± 31 piksel berubah (6,75%) per citra. Nilai-nilai ini mengikuti langsung dari aturan penyisipan. Dengan 13.107 bit yang dibagi 60/30/10 pada ketiga zona, sekitar 2.621; 1.966; dan 1.311 piksel membawa masing-masing 3, 2, dan 1 bit, dan karena penggantian k -bit membiarkan piksel tidak berubah dengan probabilitas 2^{-k} , jumlah piksel berubah yang diharapkan adalah 4.423,4, sedangkan galat kuadrat rata-rata yang diharapkan memberikan PSNR bentuk-tertutup sebesar 51,098 dB. Distribusi PSNR

ketiga subset hampir sepenuhnya tumpang tindih (Gambar 2b), dan PSNR rata-rata sesuai dengan prediksi bentuk-tertutup. Gambar 2a memperlihatkan bagaimana kondisi-kondisi tersebut mengubah selisih *cover-stego* yang harus ditemukan detektor. Karena *cover* dan *stego* berbagi medan derau yang sama, derau hampir tidak mengubah selisih tersebut (6,7% piksel), sedangkan JPEG menyebarkan ke median 29–35% piksel melalui kuantisasi ulang koefisien *discrete cosine transform* kedua citra secara berbeda. Dengan demikian *benchmark* ini memuat dua jenis pergeseran yang berbeda secara kualitatif: derau yang menutupi jejak penyisipan yang tetap, dan kompresi yang menulis ulanginya.



Gambar 2. Karakterisasi *benchmark*: (a) persentase piksel yang berbeda antara *cover* dan *stego* pada setiap kondisi uji (500 pasang; kotak = kuartil, *whisker* = rentang); (b) PSNR citra *stego* C0 per subset dibandingkan nilai bentuk-tertutup

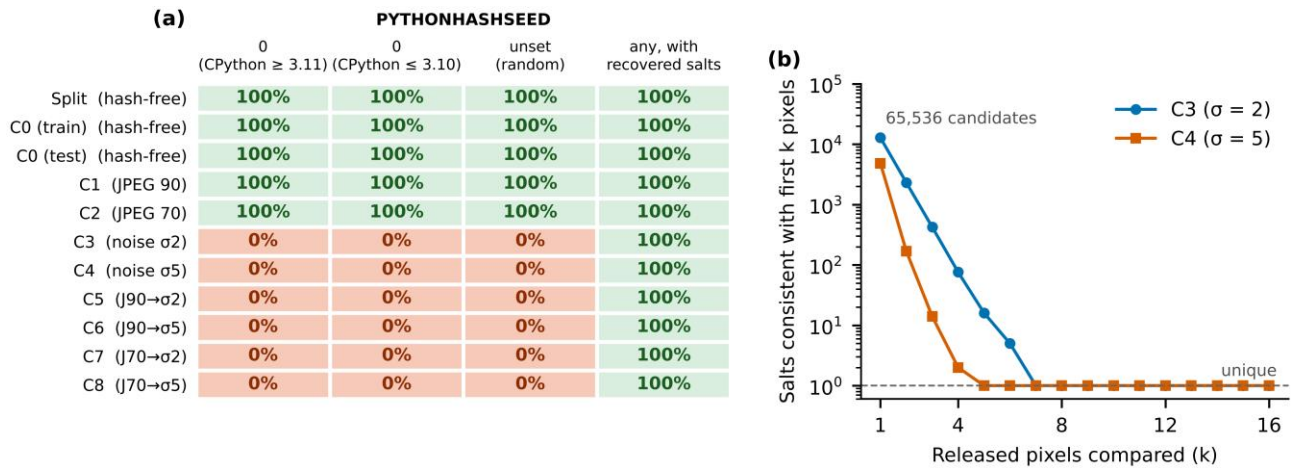
R1: Integritas Berkas

Seluruh 12.200 berkas yang tersedia untuk audit (subset pelatihan dan uji) diperiksa langsung dan cocok dengan *digest* manifestnya, dan seluruh 13.000 *digest* dalam manifest berbeda satu sama lain, yang menyingkirkan kemungkinan berkas ganda. Ketiga subset tidak berbagi identitas sumber. Ke-800 berkas validasi bukan bagian dari salinan yang diaudit; *digest*-nya telah diverifikasi pada studi awal, dan berkas-berkas itu dihasilkan oleh jalur kode yang sama dengan subset pelatihan. Dengan demikian, cakupan pemeriksaan langsung R1 pada audit ini adalah 12.200 dari 13.000 berkas (93,8%); 800 berkas validasi tidak diperiksa ulang, dan statusnya bersandar pada verifikasi terdahulu. Perbedaan ini dipertahankan pada setiap ringkasan hasil dalam artikel ini.

R2: Regenerasi Bit-Eksak dan *Salt* Kondisi yang Tersembunyi

Gambar 3a melaporkan hasil regenerasi untuk setiap kondisi dan pengaturan *interpreter*. Pembagian data direproduksi secara persis, termasuk urutan sumbernya. Seluruh 2.100 *cover* C0 yang tersedia disisipi ulang

menjadi berkas *stego* identik byte (1.600 pelatihan, 500 uji), dengan ekstraksi bebas galat atas seluruh 13.107 bit dan selisih PSNR dan SSIM tercatat di bawah 10^{-13} . Kondisi JPEG C1 dan C2 juga identik byte, meskipun audit memakai OpenCV 5.0 sedangkan *pipeline* asli hanya mensyaratkan OpenCV ≥ 4.8 dan tidak mencatat versinya. Hasil ini menunjukkan bahwa pengodean JPEG dan PNG oleh OpenCV 5.0.0 dalam lingkungan audit mereproduksi berkas yang dirilis, tetapi tidak membuktikan bahwa versi atau *build* OpenCV lain, dengan pustaka *codec* yang berbeda, akan menghasilkan byte yang sama. Namun, keenam kondisi yang melibatkan derau gagal total: 0% dari 6.000 berkas hasil regenerasi cocok, dan 86–94% pikselnya berbeda, pada PYTHONHASHSEED=0, pada *salt* CPython ≤ 3.10 , dan dengan *hash seed* tidak diatur. Proporsi ini sesuai dengan probabilitas $1 - p$ bahwa dua nilai derau terbulatkan yang independen berbeda, yaitu 0,860 untuk $\sigma = 2$ dan 0,944 untuk $\sigma = 5$ (lihat subbagian Pemulihan *Salt*), sehingga derau hasil regenerasi independen dari derau yang dirilis, bukan sekadar terganggu sedikit.



Gambar 3. Audit regenerasi: (a) persentase berkas identik byte per kondisi dan pengaturan *hash seed* (1.000 berkas per kondisi ber-derau); (b) jumlah dari 65.536 kandidat *salt* yang tetap konsisten dengan *k* piksel pertama yang dirilis

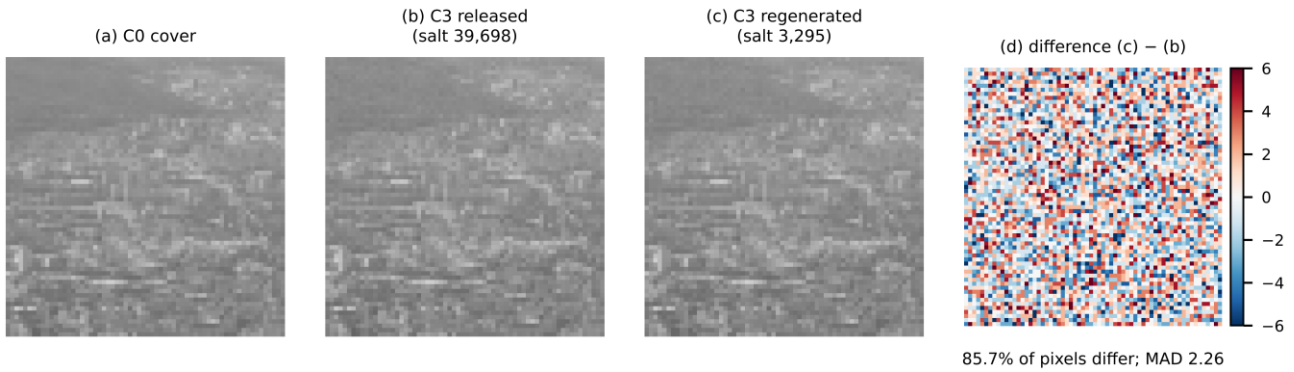
Penyebabnya adalah *salt* h_c pada (2). Fungsi *hash()* Python untuk string adalah SipHash berkunci (Aumasson dan Bernstein, 2012) yang kuncinya diacak per proses kecuali PYTHONHASHSEED ditetapkan, sebagaimana diatur dalam *Python Enhancement Proposal* (PEP) 456 (Heimes, 2013), dan variannya berubah dari SipHash-2-4 menjadi SipHash-1-3 pada CPython 3.11 (Python Software Foundation, 2022). Eksekusi asli tidak mencatat *hash seed* maupun versi *interpreter*, sehingga medan derau terikat pada keadaan proses yang tidak ada lagi. Pencarian menyeluruh pada (4) menyelesaikan hal ini. Untuk setiap kondisi ber-derau, tepat satu dari 65.536 kandidat bertahan setelah 64 piksel pertama; seperti ditunjukkan Gambar 3b, keunikan dicapai setelah 7 piksel untuk $\sigma = 2$ dan setelah 5 piksel untuk $\sigma = 5$, sesuai dengan ekspektasi yang diturunkan pada subbagian Pemulihan *Salt*. *Salt* yang dipulihkan (Tabel 5) tidak cocok dengan kedua varian SipHash dengan PYTHONHASHSEED=0, maupun dengan nilai *seed* mana pun dari 0 hingga $1,2 \times 10^7$. Ketika PYTHONHASHSEED tidak diatur, CPython mengambil kunci *hash* 128-bit dari sumber acak sistem operasi (Heimes, 2013), ruang yang tidak dapat dicari secara menyeluruh, sehingga citra yang dirilis kemungkinan besar dibangkitkan dalam mode bawaan ini. Namun, asal-usul *salt* tidak berpengaruh pada perbaikan, karena tabel *salt* yang dibekukan memulihkan regenerasi bagaimanapun *salt* itu muncul; batas $1,2 \times 10^7$ dipilih

untuk mencakup nilai kecil yang ditetapkan manual seperti 0, 42, atau 1234 yang lazim dalam praktik. Dengan *salt* yang dipulihkan dibekukan ke dalam *lookup table*, seluruh 8.000 berkas C1–C8 teregenerasi identik byte dalam lingkungan audit pada *hash seed* sembarang (Gambar 3a, kolom terakhir).

Tabel 5. *Salt* kondisi $h_c = \text{hash}(c) \ \& \ 0\text{xFFFF}$ yang dipulihkan dari berkas yang dirilis dan yang dihasilkan dengan PYTHONHASHSEED = 0

Kondisi	Dipulihkan	CPython $\geq$ 3.11	CPython $\leq$ 3.10
C3	39.698	3.295	35.963
C4	46.624	14.388	13.573
C5	57.111	13.905	44.133
C6	3.712	143	53.103
C7	14.343	43.968	17.229
C8	20.750	23.473	29.384

Gambar 4 menunjukkan apa yang dilakukan *salt* yang salah pada satu citra. Citra C3 hasil regenerasi secara visual tidak dapat dibedakan dari yang dirilis, namun 85,7% pikselnya berbeda hingga ± 6 tingkat keabuan, mendekati 86% yang diharapkan untuk $\sigma = 2$. Pengguna yang meregenerasi *benchmark* dengan kode yang dipublikasikan dengan demikian akan memperoleh set uji yang tampak masuk akal tetapi berbeda tanpa peringatan apa pun, dan hanya perbandingan tingkat byte dengan manifest yang mengungkap perbedaannya.

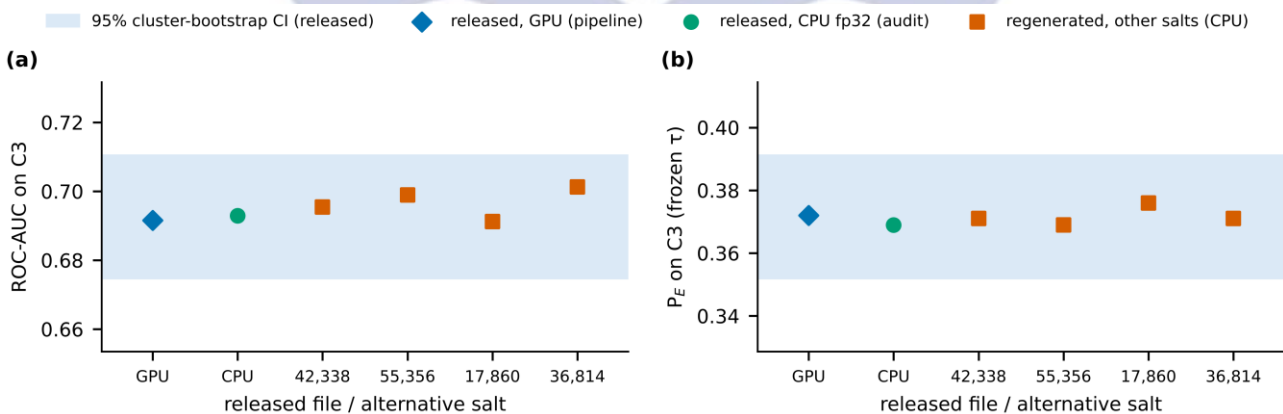


Gambar 4. Pengaruh *salt* yang salah pada citra uji C3 (potongan 64×64): (a) *cover*, (b) berkas yang dirilis, (c) berkas hasil regenerasi dengan *salt* yang dihasilkan PYTHONHASHSEED = 0 pada CPython ≥ 3.11, (d) selisih piksel

Dampak Realisasi Derau pada Tingkat Metrik

Gambar 5 menunjukkan bahwa, pada konfigurasi yang diuji, realisasi derau saja sudah cukup untuk menggeser hasil yang dilaporkan. Penilaian ulang citra C3 yang dirilis pada CPU mereproduksi hasil GPU *pipeline* dengan dekat tetapi tidak persis: skor per citra berbeda hingga 0,020, yang kami kaitkan dengan inferensi GPU presisi campuran milik *pipeline* asli, dan AUC berubah dari 0,692 menjadi 0,693. Pada keempat realisasi derau alternatif, AUC berkisar dari 0,691 hingga 0,701 dan P_E dari 0,369 hingga 0,376. Rentang 0,010 pada AUC ini setara dengan 28% dari lebar interval *cluster-bootstrap* 95% hasil yang dirilis [0,675; 0,710]. Seluruh nilai tetap berada di dalam interval tersebut, sehingga kesimpulan ilmiahnya, bahwa derau dengan $\sigma = 2$ mempertahankan sebagian keunggulan

deteksi, kokoh terhadap realisasi pada detektor ini. Temuan ini berlaku untuk konfigurasi yang diuji, yaitu satu detektor (HSMNet-43), satu kondisi (C3, $\sigma = 2$), dan empat realisasi alternatif: pada konfigurasi tersebut, desimal ketiga AUC tidak dapat direproduksi tanpa *salt* yang persis. Implikasinya bagi penelitian lain bersifat hipotesis. Jika pengaruh dengan besaran serupa juga terjadi pada detektor dan kondisi lain, selisih AUC sekitar 0,01 antartstudi pada kondisi ber-derau dapat sebagian berasal dari realisasi derau. Besaran pengaruh pada σ yang lebih besar, pada kaskade, atau pada detektor dengan daya pisah berbeda belum diukur dan dapat lebih kecil maupun lebih besar, dan empat realisasi hanya memberi batas kasar, bukan distribusi pengaruh tersebut.



Gambar 5. Pengaruh realisasi derau pada tingkat metrik untuk HSMNet *seed* 43 yang dibekukan pada C3: (a) ROC-AUC dan (b) P_E untuk berkas yang dirilis (*pipeline* GPU dan audit CPU) dan untuk empat realisasi derau hasil regenerasi; pita = interval *cluster-bootstrap* 95% hasil yang dirilis

R3: Audit Artefak Pelatihan

Gambar 6a menunjukkan bahwa tidak ada inkonsistensi pada kesembilan eksekusi yang dirilis: semuanya berstatus PASS pada A1, A2, A3, dan A5. *Digest* bobot semuanya berbeda, penghitung *BatchNorm* sama dengan $e^* \times 200$ untuk SRNet dan $e^* \times 400$ untuk WSERNet dan HSMNet

seperti diprediksi oleh (3) untuk ukuran *batch* 16 dan 8, *epoch checkpoint* sesuai dengan metadata dan riwayat, dan setiap berkas prediksi lengkap serta konsisten dengan τ -nya. Satu-satunya peringatan menyangkut SRNet *seed* 42, yang riwayatnya kosong karena skrip pelatihan menemukan *checkpoint* yang sudah ada dan melewati loop

pelatihan (riwayat hanya ditulis setelah loop selesai, sehingga berkas yang tertulis kosong dan *total_epochs_trained* = 0 pada metadata); cara *checkpoint* itu semula dihasilkan tidak tercatat. Audit mencatat celah ini alih-alih menerima eksekusi tersebut begitu saja, yang juga menjadi alasan Tabel 3 mencantumkan *epoch* dan jam GPU-nya sebagai tidak tercatat. Eksekusi ini berstatus WARN pada A4, dan status PASS-nya pada A3 hanya mencakup kecocokan *checkpoint* dengan metadata. Karena itu, SRNet-42 dinyatakan konsisten tetapi tidak terdokumentasi lengkap, sedangkan delapan eksekusi lainnya konsisten dan memiliki riwayat lengkap. Kecocokan A2 pada SRNet-42 ($7.400 = 37 \times 200$) menunjukkan bahwa penghitung *BatchNorm* pada *checkpoint* itu konsisten dengan *epoch* 37 dan ukuran

batch 16, tetapi tidak dapat menunjukkan bagaimana *checkpoint* itu semula dihasilkan. Keempat kesalahan yang disuntikkan semuanya terdeteksi (Gambar 6b). F1, yang menempatkan bobot HSMNet-44 pada *checkpoint* HSMNet-42, gagal pada A1 untuk kedua eksekusi yang berbagi bobot tersebut dan gagal pada A3 karena *checkpoint* pinjaman itu membawa *epoch* 100, bukan 95. F2, yang diterapkan pada WSERNet-43, gagal pada A2, A3, dan A4 karena kolom *epoch* yang disunting tidak dapat direkonsiliasi dengan penghitung *BatchNorm* maupun panjang riwayat. F3 gagal pada A4 untuk HSMNet-43 dan HSMNet-44, yang riwayatnya menjadi identik, dan F4, yang diterapkan pada SRNet-43, gagal pada A5. Tidak ada eksekusi bersih yang ditandai secara keliru.

(a) Released artifacts (9 runs)						(b) After fault injection (F1–F4)					
SRNet-42	PASS	PASS	PASS	WARN	PASS	PASS	PASS	PASS	WARN	PASS	
SRNet-43	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	FAIL	F4
SRNet-44	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	
WSERNet-42	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	
WSERNet-43	PASS	PASS	PASS	PASS	PASS	PASS	FAIL	FAIL	FAIL	PASS	F2
WSERNet-44	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	
HSMNet-42	PASS	PASS	PASS	PASS	PASS	FAIL	PASS	FAIL	PASS	PASS	F1
HSMNet-43	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	FAIL	PASS	F3
HSMNet-44	PASS	PASS	PASS	PASS	PASS	FAIL	PASS	PASS	FAIL	PASS	
	A1 weights	A2 BN counter	A3 epochs	A4 history	A5 predictions	A1 weights	A2 BN counter	A3 epochs	A4 history	A5 predictions	

Gambar 6. Audit artefak pelatihan: (a) kesembilan eksekusi yang dirilis; (b) eksekusi yang sama setelah penyuntikan kesalahan F1–F4 (label di sebelah kanan)

R4: Penghitungan Ulang Metrik

Penghitungan ulang setiap statistik dari prediksi per citra mereproduksi persis semua nilai P_E , AUC, P_{FA} , dan P_{MD} serta semua hitungan McNemar milik *pipeline* asli. Namun, penghitungan ulang itu menyingkap dua kesalahan dalam cara *pipeline* mengagregasi dan mengujinya. Tabel 6 merangkum keduanya dengan memisahkan statistik per-*seed*, yang dihitung dari prediksi satu eksekusi, dari statistik agregat antar-*seed*, yang dihitung dari tiga eksekusi.

Kesalahan pertama menyangkut agregasi. Tabel ringkasan *pipeline* menempatkan rata-rata $\pm$ simpangan baku atas ketiga *seed* (statistik agregat) di samping interval kepercayaan (*confidence interval*, CI) dan nilai-p McNemar yang dihitung dari *seed* 42 saja (statistik per-*seed*). Kedua jenis statistik itu menjawab pertanyaan yang berbeda dan tidak dapat dibaca sebagai satu kesatuan. Untuk HSMNet, rata-rata P_E yang dilaporkan berada di luar “CI 95%”-nya sendiri pada enam dari sembilan kondisi (misalnya 0,387 terhadap [0,459; 0,480] pada C0), karena interval itu adalah interval *seed* 42, bukan interval

rata-rata. Demikian pula, label tidak signifikan pada setiap kondisi berasal dari *seed* 42 saja, padahal *seed* 43 menurun secara signifikan pada kedelapan kondisi (p terkoreksi Holm $< 0,001$). Setelah koreksi, setiap interval dan uji dilaporkan per *seed*, sedangkan rata-rata $\pm$ simpangan baku dilaporkan terpisah sebagai ringkasan deskriptif lintas *seed* tanpa interval kepercayaan, karena tiga *seed* tidak cukup untuk menaksirnya secara andal. *Pipeline* menghitung simpangan baku dengan pembagi n (populasi), yaitu 0,120 untuk PE HSMNet C0; ringkasan setelah koreksi memakai pembagi $n - 1$ (sampel), yaitu 0,147, karena ketiga *seed* hanyalah sampel dari sebaran eksekusi yang mungkin.

Kesalahan kedua menyangkut uji DeLong. Uji DeLong milik *pipeline* menggabungkan varians kedua AUC seolah-olah independen, mengabaikan kovariansnya, padahal kedua AUC dihitung pada citra yang sama. Dengan memasukkan kovarians (DeLong dkk., 1988), 44 dari 72 nilai-p berubah lebih dari 0,01, dan satu keputusan signifikansi tanpa penyesuaian pada $\alpha = 0,05$ berbalik (HSMNet *seed* 44, C1: $p = 0,049$ dengan suku kovarians

versus 0,537 tanpanya). Perubahan ini adalah perubahan signifikansi nominal dan perlu ditafsirkan secara hati-hati. Nilai 0,049 berada sangat dekat dengan batas 0,05 dan dihitung tanpa penyesuaian perbandingan berganda; dengan koreksi Bonferroni atas 72 perbandingan ($\alpha' = 0,05/72 \approx 0,0007$), perbandingan ini tidak signifikan, dan nilai-p terkoreksi Holm-nya adalah 1,0 pada keluarga 72 perbandingan (0,39 bila keluarga dibatasi pada delapan kondisi eksekusi yang sama); setelah koreksi Holm atas 72 perbandingan, hanya kedelapan perbandingan HSMNet-43 yang tetap signifikan. Kasus ini karena itu tidak menunjukkan adanya penurunan yang signifikan pada HSMNet-44; yang ditunjukkannya adalah bahwa pengabaian kovarians dapat menggeser nilai-p cukup jauh untuk mengubah keputusan pada ambang yang dipakai *pipeline*. Tidak satu pun dari kedua kesalahan itu terlihat dari tabel ringkasan; keduanya menjadi terlihat hanya karena prediksi per citra dirilis dan dianalisis ulang.

Tabel 6. Statistik sebelum dan sesudah penghitungan ulang (per-*seed* = dihitung dari satu eksekusi; agregat = lintas tiga *seed*)

Besaran	Tabel <i>pipeline</i> asli	Setelah penghitungan ulang
P_E HSMNet C0, rerata $\pm$ simpangan baku (agregat, <i>seed</i> 42–44)	$0,387 \pm 0,120$ (simpangan baku populasi, pembagi n)	$0,387 \pm 0,147$ (simpangan baku sampel, pembagi $n - 1$); deskriptif, tanpa CI
CI 95% P_E C0 (per- <i>seed</i>)	[0,459; 0,480] dari <i>seed</i> 42, disajikan di samping rerata	42: [0,459; 0,480]; 43: [0,195; 0,238]; 44: [0,464; 0,483]
Rerata di luar “CI”-nya sendiri	6 dari 9 kondisi	tidak berlaku; CI dilaporkan per <i>seed</i>
McNemar–Holm, C1–C8 terhadap C0: kondisi signifikan (per- <i>seed</i>)	0 dari 8 (<i>seed</i> 42 saja)	42: 0/8; 43: 8/8 ($p < 0,001$); 44: 0/8
DeLong HSMNet-44, C1 terhadap C0 (per- <i>seed</i>)	$p = 0,537$ (tanpa kovarians)	$p = 0,049$ (dengan kovarians); nominal, tanpa koreksi; Holm (72 uji): 1,0 44 dari 72 nilai-p berubah $> 0,01$; 1 keputusan nominal berubah (HSMNet-44, C1) dan tidak signifikan setelah Bonferroni ($\alpha' \approx 0,0007$) maupun Holm ($p = 1,0$)
DeLong, 9 eksekusi $\times$ 8 kondisi	–	

Pembahasan

Audit ini menunjukkan bahwa sebuah *benchmark* dapat memenuhi praktik reproduksibilitas yang lazim, dengan kode publik, *seed* global tetap, dan manifes ber-*digest*, namun tetap tidak dapat diregenerasi tanpa informasi tambahan. Kegagalan itu bermula dari keputusan desain yang tampak tidak berbahaya: menurunkan *seed* per kondisi dari hash(), fungsi yang keluarannya sengaja diacak oleh Python untuk melindungi *dictionary* dari serangan *hash-flooding* (Heimes, 2013). Konstruksi

serupa kemungkinan besar umum ditemui, karena hash() adalah cara paling mudah untuk mengubah string menjadi bilangan bulat, dan ketergantungan yang sama muncul setiap kali kode mengiterasi sebuah himpunan (*set*) string, yang urutannya juga mengikuti nilai *hash*. Tiga aturan praktis dapat ditarik. Pertama, *seed* sebaiknya diturunkan dari fungsi yang stabil seperti hashlib.sha256 atau zlib.crc32, atau dari tabel eksplisit yang disimpan bersama data. Kedua, *interpreter*, versi pustaka, dan PYTHONHASHSEED sebaiknya dicatat pada setiap eksekusi pembangkitan. Ketiga, regenerasi sebaiknya diverifikasi pada tingkat byte terhadap manifes, karena kemasukakalan pada tingkat piksel tidak mengungkap realisasi derau yang berbeda (Gambar 4). Bagi *benchmark* yang sudah memuat ketergantungan semacam itu, prosedur pemulihan *salt* menawarkan perbaikan yang kompatibel mundur, karena tabel *salt* yang dibekukan mereproduksi berkas yang dirilis tanpa meregenerasi, mengganti, atau melabeli ulang apa pun. Perbaikan ini dikonfirmasi dalam lingkungan audit (Tabel 4); regenerasi pada lingkungan lain tetap perlu diverifikasi terhadap manifes, dan verifikasi itulah fungsi R2.

Audit artefak pelatihan melengkapi reproduksibilitas pada tingkat berkas. Relasi konsistensi internal yang murah, seperti penghitung *BatchNorm* terhadap *epoch* dan ukuran *batch*, *digest* bobot yang berbeda, serta kecocokan antara *checkpoint*, metadata, dan riwayat, menangkap eksekusi yang terduplikasi, salah dilaporkan, atau tercatat sebagian tanpa memerlukan GPU dan tanpa melatih ulang model apa pun. Penghitungan ulang metrik dari prediksi per citra menangkap kesalahan yang tidak dapat dideteksi pembaca tabel ringkasan. Bersama-sama, hasil-hasil ini mendukung perilsan prediksi per citra dan artefak pelatihan bersama citra dan kode, sebagaimana dianjurkan oleh pedoman reproduksibilitas terbaru (Pineau dkk., 2021; Kapoor dan Narayanan, 2023; Semmelrock dkk., 2025). Hasil-hasil ini juga menunjukkan bahwa setiap tingkat audit menangkap kelas cacat yang berbeda, sehingga tidak ada satu tingkat pun yang dapat menggantikan yang lain.

Penelitian ini memiliki keterbatasan. Lisensi BOSSbase melarang pendistribusian ulang citra, sehingga pengguna harus meregenerasinya sebelum dapat memverifikasinya. Regenerasi bit-eksak, termasuk determinisme JPEG, dikonfirmasi hanya dalam satu lingkungan audit (OpenCV 5.0.0; Tabel 4), tidak untuk semua *build* codec maupun untuk versi pustaka pada pembangkitan asli, yang tidak tercatat. Pembagian data menjamin tidak adanya tumpang tindih identitas sumber, tetapi kemiripan konten antar-citra sumber yang berbeda tidak diperiksa. Nondeterminisme pelatihan pada GPU tidak diukur ulang, karena audit menyoroti artefak yang dirilis, bukan pelatihan ulang. Analisis sensitivitas tingkat

metrik memakai empat realisasi alternatif untuk satu kondisi dan satu detektor, sehingga hanya membatasi pengaruhnya tanpa mengkarakterisasi distribusinya sepenuhnya, dan setiap jenis kesalahan yang disuntikkan diuji dengan satu contoh saja. *Benchmark* ini mencakup satu skema penyisipan dan satu *payload*, dan *baseline* dilatih dengan anggaran terbatas, sehingga berfungsi sebagai titik acuan, bukan sebagai akurasi mutakhir (*state-of-the-art*).

PENUTUP

Simpulan

Artikel ini menyajikan sebuah *benchmark* steganalisis berpasangan dan terpisah per sumber dengan 13.000 berkas untuk evaluasi ketahanan pada kompresi JPEG dan derau Gauss tunggal maupun berantai, yang dirilis bersama manifes SHA-256, metadata Croissant 1.0 yang tervalidasi, sembilan eksekusi pelatihan *baseline*, dan protokol reproduksibilitas empat tingkat. Integritas berkas terjaga pada seluruh 12.200 berkas yang diaudit langsung (800 berkas validasi mengacu pada verifikasi terdahulu), dan pembagian sumber, seluruh 2.100 hasil penyisipan yang tersedia, serta kedua kondisi JPEG teregenerasi secara bit-eksak dalam lingkungan audit. Namun, keenam kondisi ber-derau tidak dapat diregenerasi pada pengaturan yang diuji karena *seed* derau mereka bergantung pada keluaran *hash* string Python yang kuncinya diacak per proses. Pencarian menyeluruh atas 65.536 kemungkinan *salt* kondisi memulihkan satu nilai unik untuk setiap kondisi hanya dari lima sampai tujuh piksel, jumlah yang sesuai dengan model probabilistik sederhana, dan memulihkan regenerasi identik byte demi byte 100% atas seluruh 8.000 berkas hasil transformasi. Pada satu detektor dan satu kondisi ($\sigma = 2$), empat realisasi derau alternatif mengubah AUC dalam rentang 0,010, di dalam interval kepercayaannya tetapi melampaui ketelitian yang lazim dipakai untuk melaporkan hasil; besaran pengaruh ini pada detektor dan kondisi lain belum diketahui. Audit artefak pelatihan tidak menemukan inkonsistensi pada kesembilan eksekusi yang dirilis, tetapi mencatat satu eksekusi (SRNet-42) dengan riwayat pelatihan yang tidak lengkap, dan mendeteksi keempat kesalahan yang disuntikkan, sementara penghitungan ulang metrik dari prediksi per citra mengoreksi kesalahan agregasi *seed* 42 dan uji DeLong yang mengabaikan kovarians, yang mengubah satu keputusan signifikansi nominal ($p = 0,049$) yang tidak bertahan setelah koreksi perbandingan berganda. Kekuatan utama penelitian ini adalah bahwa setiap klaim tentang *benchmark* didukung oleh pemeriksaan yang dapat dieksekusi; kelemahan utamanya adalah bahwa citranya sendiri tidak dapat didistribusikan ulang.

Saran

Penelitian selanjutnya sebaiknya menerapkan protokol ini pada *benchmark* yang lebih besar seperti ALASKA2 dan REVEAL, memperluas uji regenerasi pada berbagai *build* codec, memakai lebih banyak realisasi derau pada lebih banyak detektor dan kondisi dan lebih banyak kesalahan yang disuntikkan per jenis, memeriksa kemiripan konten antar-subset, serta menambahkan reproduksibilitas tingkat pelatihan ulang pada GPU.

KETERSEDIAAN DATA

Status ketersediaan pada saat naskah revisi ini dikirim adalah sebagai berikut. Kode pembangkitan, manifes, metadata Croissant, tabel *salt* kondisi yang dibekukan, *checkpoint*, prediksi per citra, dan peralatan audit telah tersedia di repositori Zenodo dengan *digital object identifier* (DOI) <https://doi.org/10.5281/zenodo.23524043> (versi 1.0.0). Citra BOSSbase 1.01 tidak termasuk dalam rilis karena lisensinya melarang pendistribusian ulang; citra tersebut harus diperoleh dari sumber aslinya, diregenerasi secara lokal dengan kode yang dirilis, lalu diverifikasi terhadap manifes.

DAFTAR PUSTAKA

- Akhtar, M. dkk. 2024. Croissant: A metadata format for ML-ready datasets. Dalam: *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, *Datasets and Benchmarks Track*, hlm. 82133–82148. doi: 10.52202/079017-2610.
- Alrusaini, O. 2025. Deep learning for steganalysis: Evaluating model robustness against image transformations. *Front. Artif. Intell.*, 8: Art. no. 1532895. doi: 10.3389/frai.2025.1532895.
- Aumasson, J.-P. dan Bernstein, D. J. 2012. SipHash: A fast short-input PRF. Dalam: *Progress in Cryptology – INDOCRYPT 2012*, LNCS, vol. 7668. Berlin, Germany: Springer, hlm. 489–508. doi: 10.1007/978-3-642-34931-7_28.
- Bas, P., Filler, T., dan Pevný, T. 2011. ‘Break our steganographic system’: The ins and outs of organizing BOSS. Dalam: *Information Hiding (IH 2011)*, LNCS, vol. 6958. Berlin, Germany: Springer, hlm. 59–70. doi: 10.1007/978-3-642-24178-9_5.
- Boroumand, M., Chen, M., dan Fridrich, J. 2019. Deep residual network for steganalysis of digital images. *IEEE Trans. Inf. Forensics Security*, 14(5): 1181–1193. doi: 10.1109/TIFS.2018.2871749.
- Bouthillier, X. dkk. 2021. Accounting for variance in machine learning benchmarks. Dalam: *Proc. Mach. Learn. Syst. (MLSys)*, vol. 3, hlm. 747–769.
- Chakraborty, S., Jalal, A. S., dan Bhatnagar, C. 2017. LSB based non blind predictive edge adaptive image

- steganography. *Multimedia Tools Appl.*, 76(6): 7973–7987. doi: 10.1007/s11042-016-3449-4.
- Cogranne, R., Giboulot, Q., dan Bas, P. 2020. ALASKA#2: Challenging academic research on steganalysis with realistic images. Dalam: *Proc. IEEE Int. Workshop on Information Forensics and Security (WIFS)*, New York, NY, USA, Dec. 2020, hlm. 1–5. doi: 10.1109/WIFS49906.2020.9360896.
- DeLong, E. R., DeLong, D. M., dan Clarke-Pearson, D. L. 1988. Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach. *Biometrics*, 44(3): 837–845. doi: 10.2307/2531595.
- Efron, B. dan Tibshirani, R. J. 1993. *An Introduction to the Bootstrap*. New York, NY, USA: Chapman & Hall.
- Gebu, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé III, H., dan Crawford, K. 2021. Datasheets for datasets. *Commun. ACM*, 64(12): 86–92. doi: 10.1145/3458723.
- Giboulot, Q., Cogranne, R., Borghys, D., dan Bas, P. 2020. Effects and solutions of cover-source mismatch in image steganalysis. *Signal Process.: Image Commun.*, 86: Art. no. 115888. doi: 10.1016/j.image.2020.115888.
- Gundersen, O. E., Coakley, K., Kirkpatrick, C., dan Gil, Y. 2022. Sources of irreproducibility in machine learning: A review. arXiv:2204.07610. Tersedia pada: <https://arxiv.org/abs/2204.07610>
- Gundersen, O. E. dan Kjensmo, S. 2018. State of the art: Reproducibility in artificial intelligence. Dalam: *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1. doi: 10.1609/aaai.v32i1.11503.
- Heimes, C. 2013. PEP 456 – Secure and interchangeable hash algorithm. Python Enhancement Proposals. Tersedia pada: <https://peps.python.org/pep-0456/>
- Holm, S. 1979. A simple sequentially rejective multiple test procedure. *Scand. J. Statist.*, 6(2): 65–70. Tersedia pada: <https://www.jstor.org/stable/4615733>
- Kapoor, S. dan Narayanan, A. 2023. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9): Art. no. 100804. doi: 10.1016/j.patter.2023.100804.
- Ker, A. D., Bas, P., Böhme, R., Cogranne, R., Craver, S., Filler, T., Fridrich, J., dan Pevný, T. 2013. Moving steganography and steganalysis from the laboratory into the real world. Dalam: *Proc. 1st ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec)*, hlm. 45–58. doi: 10.1145/2482513.2482965.
- Kheddar, H., Hemis, M., Himeur, Y., Megías, D., dan Amira, A. 2024. Deep learning for steganalysis of diverse data types: A review of methods, taxonomy, challenges and future directions. *Neurocomputing*, 581: Art. no. 127528. doi: 10.1016/j.neucom.2024.127528.
- Kombrink, M., van Lierop, S., Stolwijk, D., Worring, M., Vrijdag, D., dan Geradts, Z. 2025. REVEAL: A large-scale comprehensive image dataset for steganalysis. *Forensic Sci. Int.: Digit. Investig.*, 55: Art. no. 302006. doi: 10.1016/j.fsidi.2025.302006.
- Lamb, C. dan Zacchiroli, S. 2022. Reproducible builds: Increasing the integrity of software supply chains. *IEEE Softw.*, 39(2): 62–70. doi: 10.1109/MS.2021.3073045.
- Liang, W. dan Li, Q. 2026. Steganalysis network for weak steganographic signal extraction and enhancement. *Sensors*, 26(4): Art. no. 1329. doi: 10.3390/s26041329.
- Longpre, S. dkk. 2024. A large-scale audit of dataset licensing and attribution in AI. *Nat. Mach. Intell.*, 6(8): 975–987. doi: 10.1038/s42256-024-00878-8.
- Mallet, A., Beneš, M., dan Cogranne, R. 2024. Cover-source mismatch in steganalysis: Systematic review. *EURASIP J. Inf. Secur.*, 2024: Art. no. 26. doi: 10.1186/s13635-024-00171-6.
- McNemar, Q. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2): 153–157. doi: 10.1007/BF02295996.
- Pham, H. V., Qian, S., Wang, J., Lutellier, T., Rosenthal, J., Tan, L., Yu, Y., dan Nagappan, N. 2020. Problems and opportunities in training deep learning software systems: An analysis of variance. Dalam: *Proc. 35th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, hlm. 771–783. doi: 10.1145/3324884.3416545.
- Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché-Buc, F., Fox, E., dan Larochelle, H. 2021. Improving reproducibility in machine learning research (A report from the NeurIPS 2019 reproducibility program). *J. Mach. Learn. Res.*, 22(164): 1–20. Tersedia pada: <https://jmlr.org/papers/v22/20-303.html>
- Python Software Foundation. 2022. What's new in Python 3.11. Python 3 documentation. Tersedia pada: <https://docs.python.org/3/whatsnew/3.11.html>
- Semmelrock, H., Ross-Hellauer, T., Kopeinik, S., Theiler, D., Haberl, A., Thalmann, S., dan Kowald, D. 2025. Reproducibility in machine-learning-based research: Overview, barriers, and drivers. *AI Mag.*, 46(2): Art. no. e70002. doi: 10.1002/aaai.70002.
- Wang, Z., Bovik, A. C., Sheikh, H. R., dan Simoncelli, E. P. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4): 600–612. doi: 10.1109/TIP.2003.819861.

- Wilkinson, M. D. dkk. 2016. The FAIR guiding principles for scientific data management and stewardship. *Sci. Data*, 3: Art. no. 160018. doi: 10.1038/sdata.2016.18.
- Yang, W., Liu, Y., Chen, Y., Cui, Y., Guo, C., Shen, G., dan Wu, H. 2026. HSMNet: A multi-resolution grayscale image steganalysis method based on hybrid dilated convolution and self-attention multi-channel network. *Inf. Sci.*, 728: Art. no. 122824. doi: 10.1016/j.ins.2025.122824.
- Yang, Z., Wang, K., Ma, S., Huang, Y., Kang, X., dan Zhao, X. 2020. IStego100K: Large-scale image steganalysis dataset. Dalam: *Digital Forensics and Watermarking (IWDW 2019)*, LNCS, vol. 12022. Cham, Switzerland: Springer, hlm. 352–364. doi: 10.1007/978-3-030-43575-2_29.
- Yousfi, Y. dan Fridrich, J. 2020. JPEG steganalysis detectors scalable with respect to compression quality. Dalam: *Proc. IS&T Electronic Imaging, Media Watermarking, Security, and Forensics*, hlm. 75-1–75-11. doi: 10.2352/ISSN.2470-1173.2020.4.MWSF-075.
- Zhuang, D., Zhang, X., Song, S. L., dan Hooker, S. 2022. Randomness in neural network training: Characterizing the impact of tooling. Dalam: *Proc. Mach. Learn. Syst. (MLSys)*, vol. 4. Tersedia pada: https://proceedings.mlsys.org/paper_files/paper/2022/file/427e0e886ebf87538afdf0badb805b7f-Paper.pdf

