Alma Mater Studiorum – Università di Bologna

Scuola di Scienze
Corso di Laurea in Informatica

Progettazione e Validazione di Tecniche per Generazione di File Casuali Verosimili per Data Flooding against Ransomware

Relatore:
Prof.
SAVERIO GIALLORENZO

Presentata da:
MASSIMILIANO FIORAVANTI

Correlatore:
Dott.
SIMONE MELLONI

Sessione III
Anno Accademico 2024/2025

Sommario

Il ransomware rappresenta una delle minacce informatiche più diffuse e dannose per utenti e organizzazioni, poiché la cifratura indiscriminata dei file può causare gravi perdite economiche e operative. Tra le strategie di difesa proposte in letteratura, il Data Flooding against Ransomware mira a ridurre l'impatto degli attacchi inducendo il malware a cifrare file-esca.
Tra i metodi per la generazione dei file-esca ci sono approcci basati su file casuali. Un problema di questa modalità è che file casuali che differiscono molto dai file reali dell'utente possono essere discriminati e quindi ignorati dal ransomware, diminuendo l'efficacia del contrasto.

Questo lavoro propone un sistema per la generazione di file-esca casuali, progettati per risultare statisticamente compatibili con i file reali dell'utente, senza ricorrere alla loro replica diretta. L'approccio si basa su una modellazione statistica del contenuto binario dei file reali e sull'impiego di metriche quantitative per valutare la verosimiglianza dei file generati rispetto a specifici formati.

Viene inoltre sviluppato un sistema di valutazione automatico che consente di stimare in modo oggettivo la verosimiglianza dei file generati e di validare sperimentalmente l'efficacia dell'approccio proposto. I risultati ottenuti su diversi formati di file mostrano che il metodo consente di generare file artificiali difficilmente distinguibili da quelli reali mediante analisi statistiche, rendendo l'approccio adatto al supporto di tecniche di difesa contro attacchi ransomware.

Indice

Capitolo 1

Introduzione

1.1 Ransomware

Il termine Ransomware deriva dall'unione delle parole "ransom" (ostaggio) e "ware" (da software), indicando un programma che tiene in ostaggio dati o risorse digitali fino al pagamento di una somma di denaro. In particolare, un ransomware è una tipologia di malware che, una volta infiltratosi in un sistema informatico, cifra i file della vittima, chiedendo successivamente un pagamento in cambio della chiave necessaria a decifrarli. Quando un ransomware infetta un sistema, tende a colpire file sensibili come dati finanziari, documenti aziendali, database, file personali e altri contenuti di valore.

Negli ultimi anni il ransomware è diventato una delle minacce informatiche più diffuse e dannose a livello globale. Gli attacchi possono manifestarsi in vari modi, dai crypto-ransomware, i quali cifrano i file rendendoli inaccessibili, ai locker-ransomware, che bloccano l'accesso al sistema operativo stesso. Questo tipo di malware rappresenta quindi un problema critico per utenti e organizzazioni, poichè può causare danni economici e operativi significativi. La crescente professionalizzazione degli attaccanti e la diffusione del modello Ransomware-as-a-Service (RaaS) hanno ulteriormente amplificato la portata di questa minaccia [1].

1.2 Contesto, Obiettivo e Contributo del Lavoro

Il presente lavoro si colloca nell'ambito di *Data Flooding against Ransomware* (DFaR), un insieme di tecniche le quali prevedono la creazione, su richiesta, di molti file-esca in cartelle ritenute sensibili, come ad esempio aree utente o, quando disponibile, la posizione in cui il ransomware sta operando. In questo modo la "trappola" viene portata vicino all'attaccante, aumentando la probabilità di intercettare l'attività malevola e supportando non solo la rilevazione, ma anche la risposta all'attacco.

Nella fase di mitigazione, il flooding mira a ostacolare un attacco in corso mescolando i file legittimi con una grande quantità di esche, così che il malware sprechi tempo cifrando contenuti non rilevanti.

Tuttavia, la sola disponibilità di generare file-esca non è sufficiente: strategie basate su contenuti puramente casuali possono produrre documenti facilmente distinguibili da file reali attraverso euristiche e caratteristiche statistiche del contenuto. Ne consegue un problema per l'efficacia del data flooding: i file generati devono risultare *verosimili* rispetto a uno specifico formato almeno dal punto di vista di proprietà osservabili a basso livello, così da aumentare la probabilità di non essere scartati o ignorati da un attaccante.

L'obiettivo del presente lavoro è lo studio e la progettazione di un sistema capace di generare file-esca casuali verosimili, destinati a essere utilizzati in tecniche di data flooding contro attacchi ransomware. L'obiettivo è produrre file che siano statisticamente simili a file reali, in modo da aumentare la probabilità che vengano presi di mira dal ransomware nel suo processo di cifratura, e diminuire la probabilità che il ransomware discrimini questi file e li ignori.

In dettaglio, il lavoro si propone di definire una nozione quantitativa di verosimiglianza dei file, da utilizzare come criterio di valutazione in grado di stabilire in modo automatico se un file sintetico possa essere considerato realistico o meno rispetto a uno specifico formato.

I contributi principali possono essere riassunti come segue:

- la definizione di un insieme di metriche per il confronto statistico tra file reali e file generati, utilizzabili per stimare il grado di verosimiglianza di un file artificiale;
- la progettazione di un sistema di rilevamento che utilizza tali metriche come oracolo per la valutazione della verosimiglianza dei file;
- lo sviluppo di un metodo di generazione di file esca statisticamente compatibili con i vincoli imposti dal sistema di rilevamento;
- una valutazione sperimentale dell'approccio proposto su diversi formati di file e su dataset reali.

1.3 Struttura della Tesi

La tesi è strutturata come segue.

Nel Capitolo 2 vengono introdotti i concetti preliminari e il contesto teorico necessario alla comprensione del lavoro, con particolare riferimento alle principali tecniche di difesa contro il ransomware e all'approccio del data flooding.

Nel Capitolo 3 vengono presentate le rappresentazioni statistiche e le metriche utilizzate per descrivere e confrontare i file, insieme alle scelte adottate per selezionare le informazioni più significative da analizzare nei diversi formati di file.

Nel Capitolo 4 viene descritto il sistema di rilevamento (detector), illustrando la definizione della verosimiglianza, il processo di ottimizzazione delle soglie di rilevamento e i risultati sperimentali ottenuti.

Nel Capitolo 5 viene illustrato il sistema di generazione dei file verosimili, basato su una modellazione statistica markoviana del contenuto binario e su un processo di generazione coerente con i vincoli imposti dal detector.

Infine, nel Capitolo 6 vengono discusse le conclusioni del lavoro, oltre a presentare possibili sviluppi futuri.

Capitolo 2

Preliminari

Questo capitolo introduce il contesto necessario a inquadrare il lavoro svolto. In primo luogo viene fornita una panoramica delle principali tecniche di difesa e mitigazione contro il ransomware, evidenziando obiettivi, punti di forza e limiti delle soluzioni più diffuse. Successivamente l'attenzione si concentra sull'approccio del data flooding, e in particolare su Data Flooding against Ransomware e sul tool Ranflood, che costituiscono il riferimento applicativo di questa tesi. Il capitolo si chiude motivando il problema affrontato nel seguito: la generazione di file-esca che non risultino facilmente distinguibili da quelli reali, introducendo l'esigenza di criteri quantitativi per confrontare file reali e sintetici.

2.1 Tecniche di Difesa contro il Ransomware

La letteratura recente propone diversi metodi per rilevare e mitigare gli attacchi ransomware. Per quanto riguarda la mitigazione, l'obiettivo fondamentale è quello di limitare i danni e recuperare efficacemente i dati compromessi. Le tecniche più comuni includono:

- **Segmentazione e isolamento della rete:** l'uso di Software Defined Networking (SDN) permette di bloccare dinamicamente i canali di comunicazione verso i server di comando e controllo (C2C), impedendo la cifratura remota dei dati [2].
- **Analisi e reverse engineering:** lo studio del comportamento del malware consente talvolta di recuperare i file cifrati, sfruttando debolezze nei meccanismi di cancellazione o generazione delle chiavi [3].
- **Backup e ripristino automatico:** il mantenimento di copie di sicurezza periodiche e versioning consente di ripristinare lo stato precedente all'attacco, evitando il pagamento del riscatto [4].
- **Sistemi di rilevamento basati su entropia o comportamento:** framework come quelli brevettati da McAfee e Dell monitorano la variazione

dei file system (entropia, deduplicazione, snapshot) per rilevare cifrature anomale e ripristinare automaticamente le copie integre [5].

- **Data flooding:** l'iniezione di grandi quantità di dati innocui nei file critici può ostacolare l'efficacia della cifratura ransomware, rallentando il processo e aumentando la probabilità di rilevamento [6].

In generale, si evidenzia che le tecniche di mitigazione sono efficaci solo se integrate in una strategia più ampia che include prevenzione, aggiornamenti regolari e formazione degli utenti. La difficoltà di sviluppare contromisure universali è legata alla continua evoluzione dei ceppi e alle variabili umane che influenzano la risposta all'incidente [7].

2.2 Data Flooding against Ransomware (DFaR) e Ranflood

In questo lavoro, ci si concentra sull'approccio di mitigazione dell'attacco basato sul data flooding. L'idea alla base di questa tecnica è quella di inondare di file-esca il sistema non appena viene rilevata un'attività sospetta riconducibile a un attacco ransomware. Così facendo, il ransomware viene indotto a cifrare questi file, diluendo l'impatto dell'attacco sui dati reali e al contempo rallentando il processo di cifratura.

Un tool che implementa tecniche di data flooding è Ranflood, il quale mette a disposizione diverse strategie, tra cui:

- **Random:** genera file casuali nella posizione target per aumentare la superficie d'attacco.
- **On-the-fly:** replica file reali (pre-snapshot + firme) per creare copie valide.
- **Shadow:** utilizza snapshot/archivi preesistenti per mantenere copie di recovery.

Concentrandoci sulla strategia Random, l'idea di generare file-esca casuali è concettualmente semplice e generalmente efficace. Tuttavia, alcuni lavori indicano una limitazione di questo approccio: Genç et al. [8] mostrano come file-esca casuali possano essere individuati e ignorati da alcuni ransomware, ad esempio mediante controlli su metadati o statistiche di contenuto. In particolare, file che presentano caratteristiche anomale, come un'entropia eccessivamente alta, possono essere facilmente identificati e esclusi durante il processo di cifratura.

Da quanto sopra, ne consegue la necessità di generare file credibili, esche che siano statisticamente simili ai file reali dell'utente, così da ridurre la probabilità che questi ultimi vengano ignorati. Una via intermedia fra la generazione di file random e di file-copia di quelli reali è quindi quella di generare file simili ma non

identici ai file reali, in modo da massimizzare la probabilità che il ransomware prenda di mira questi file.

In questo contesto, è quindi opportuno introdurre metriche che consentano di quantificare la similarità tra file reali e generati. Nel presente lavoro si adotteranno rappresentazioni statistiche del contenuto binario (es. Byte Frequency Distribution) e misure di distanza/divergenza fra distribuzioni. In particolare si farà riferimento a metriche già utilizzate in letteratura per il confronto tra file come entropia di Shannon, (per misurare il grado di casualità), la Jensen–Shannon Divergence (JSD) per la distanza simmetrica fra distribuzioni, la Manhattan distance, la Cosine Similarity e altre varianti (es. Total Variation Distance). Tali metriche permetteranno di quantificare quanto un file generato artificialmente si allontani da un file reale.

Capitolo 3

Metriche

Questo capitolo introduce gli strumenti quantitativi utilizzati nel seguito per descrivere un file in modo misurabile.

In primo luogo vengono presentate alcune rappresentazioni statistiche del contenuto binario (BFD e N-gram), che permettono di ottenere un "profilo" numerico del file.

Successivamente si discute come rendere tale profilo più informativo nei formati che includono porzioni compresse o poco significative, motivando le scelte adottate per concentrare l'analisi sulle parti più rilevanti ai fini del confronto.

Infine vengono introdotte le metriche di distanza e divergenza impiegate per quantificare la somiglianza tra profili statistici: queste misure costituiscono la base operativa con cui verrà definita e valutata la verosimiglianza dei file nei capitoli successivi.

3.1 Byte Frequency Distribution (BFD)

Un file digitale può essere visto come una sequenza di byte, ciascuno dei quali assume un valore intero compreso tra 0 e 255. La BFD rappresenta la distribuzione statistica di tali valori all'interno di un file, mostrando quante volte compare ciascun byte rispetto al totale dei byte contenuti. In altre parole, si costruisce un array di 256 elementi (uno per ogni possibile valore di byte) e si conta il numero di occorrenze di ciascuno.
Formalmente, la BFD di un file F è un vettore

$$B = [f_0, f_1, \ldots, f_{255}],$$

in cui ogni elemento f_i indica la frequenza relativa del valore di byte i:

$$f_i = \frac{1}{N} \sum_{j=1}^{N} \delta(x_j = i)$$

dove x_j è il valore del byte in posizione j, N è la lunghezza del file, e δ è la funzione di Kronecker.

Ogni tipo di file tende a produrre pattern caratteristici nella sua distribuzione globale dei byte, considerando l'intero contenuto del file. Questi pattern possono essere considerati una firma statistica del formato.
McDaniel e Heydari [9] hanno mostrato come analizzando la distribuzione dei byte di più file appartenenti allo stesso formato, si ottenga un profilo medio rappresentativo di quello specifico formato. Gli autori affiancano a questo approccio anche un'analisi separata basata su header e trailer, trattata come algoritmo distinto. Un file sconosciuto dunque può essere confrontato con tali profili per verificarne la similarità statistica. Gli autori propongono inoltre di estendere questa rappresentazione, introducendo la Byte Frequency Cross-correlation (BFC), la quale, oltre a contare la frequenza dei singoli byte, analizza le relazioni tra coppie di byte, consentendo di cogliere pattern strutturali più complessi.
Il concetto di BFD può essere esteso anche oltre la classificazione per tipo, infatti confrontando le distribuzioni di byte di due file è possibile stimare quanto i file siano statisticamente simili. In tal senso, l'uso di metriche di distanza o divergenza consente di quantificare tale somiglianza, fornendo una base per il confronto fra file reali e generati.

3.2 Rappresentazione N-gram

Sebbene la Byte Frequency Distribution fornisca una rappresentazione utile del contenuto di un file, essa non cattura le relazioni sequenziali tra i byte, in quanto considera solo la frequenza individuale di ciascun byte, ignorando le dipendenze locali tra byte consecutivi, i quali sono spesso rilevanti per descrivere la struttura interna del file.
Per superare questo problema, diversi lavori in letteratura propongono l'uso di rappresentazioni N-gram di byte, nelle quali si considerano sequenze di N byte consecutivi come unità di analisi [10, 11].
Come mostrano Zhang et al. [10], questo tipo di rappresentazione consente di catturare pattern più complessi e strutturati all'interno del file, modellando pattern locali che riflettono meglio la natura del contenuto.

Formalmente, la rappresentazione N-gram di un file F è ottenuta estraendo tutte le sequenze di N byte consecutivi presenti nel file. Ad ogni sequenza è associata una frequenza di occorrenza, ottenendo cosi un vettore di lunghezza 256^N che descrive la distribuzione statistica delle sequenze di byte:

$$G = [g_0, g_1, \ldots, g_{256^N-1}],$$

Con particolare attenzione nel caso di $N = 2$, si ottiene la rappresentazione a bigrammi, la quale cattura le relazioni tra coppie di byte consecutivi e quindi permette di modellare pattern più complessi rispetto alla semplice BFD.

3.3 Estrazione Strutturale dei File (Structural Bytes)

L'estrazione delle caratteristiche dai file può essere affrontata in diversi modi. Il più semplice consiste nell'analizzare l'intero contenuto del file, calcolando la BFD o la N-gram su tutti i byte presenti. Questo approccio, seppure semplice, non tiene conto della struttura interna del file, la quale può variare significativamente tra formati diversi. Più in particolare, molti formati come per esempio PDF, JPG, DOCX presentano porzioni di dati compressi, non strutturati e dunque altamente entropici. Queste sezioni tendono ad appiattire le distribuzioni dei byte, rendendo più difficile la distinzione tra file [12].

Per arginare il problema, diversi lavori in letteratura come [13] propongono di estrarre solo le porzioni strutturali del file, ovvero quelle sezioni che contengono metadati, header, marcatori e altre informazioni che riflettono la struttura del formato, in modo tale da ottenere rappresentazioni statistiche più significative.

Considerando i formati analizzati in questo lavoro (PDF, JPG, DOCX, TXT), si è scelto di adottare un approccio simile, estraendo le porzioni strutturali dei file per calcolare la N-gram.
Per esempio:

- I PDF includono sezioni testuali, dizionari e metadati, mentre gli stream binari `(stream...endstream)` sono dati compressi senza una struttura particolarmente rilevante [14].
- Le immagini JPG contengono un header con marcatori rilevanti che definiscono segmenti strutturati del formato, mentre le sezioni dopo il segmento SOS (Start of Scan) sono altamente entropiche e dunque poco informative [15].
- Un file DOCX è caratterizzato da archivi ZIP i quali contengono sezioni XML strutturate, ed anche altre sezioni compresse ed altamente entropiche che possono essere ignorate [16].

3.3.1 Estrazione Strutturale per PDF

Per i PDF si rimuovono `stream...endstream`, in quanto contengono dati compressi e non strutturati (come per esempio immagini o font).
Dopo l'estrazione, si estraggono:

- i 64KB iniziali del file contenente header e oggetti iniziali.
- gli 8KB finali del file, contenenti il trailer del documento.

Il processo descritto può essere sintetizzato come segue

Algorithm 1: Estrazione strutturale PDF

```
data ← read(path);
text ← decode(data, "latin1");
text ← remove_streams(text);
head ← first 64KB of text;
tail ← last 8KB of text;
return encode(head + tail, "latin1")
```

3.3.2 Estrazione Strutturale per JPG

Il formato JPG è caratterizzato da un header iniziale contenente marcatori strutturati (segmenti `FFxx`), seguito poi da un flusso compresso di dati dopo il segmento `SOS` (Start of Scan).
Per ottenere una rappresentazione statistica significativa, si considerano:

- l'header JPEG fino al segmento `SOS`;
- una piccola finestra post–SOS contenente unicamente i marcatori `FFxx`.

L'algoritmo seguente sintetizza il procedimento adottato:

Algorithm 2: Estrazione strutturale JPEG

```
data ← read(path);
if data non inizia con SOI (FF D8) then
    return primi 256 byte;
sos ← indice del marker SOS (FF DA);
if sos non trovato then
    header ← data;
else
    header ← data fino a sos;
markers ← estrai_marcatori_FFxx(data dopo sos, finestra = 1024);
return concatenate(header, markers);
```

3.3.3 Estrazione Strutturale per DOCX

Questo formato è basato su archivi ZIP contenenti file XML strutturati e altre risorse. Per ottenere una rappresentazione significativa, vengono selezionati solo i file XML rilevanti, quali:

- `word/document.xml`
- `word/styles.xml`
- `file .rels`
- `[Content Types].xml`

vengono invece ignorati:

- immagini `word/media/*`
- oggetti incorporati `embeddings/`

Algorithm 3: Estrazione strutturale DOCX

```
apri_zip(path);
parts ← lista vuota;
foreach file all'interno dell'archivio do
    nome ← lowercase(file);
    if nome inizia con word/media/ o contiene /media/ o
     embeddings/ then
        salta;
    if nome termina con .xml o .rels o è [Content_Types].xml then
        testo ← decode(file, UTF-8);
        aggiungi testo a parts;
text_all ← concatenazione(parts);
if text_all è vuoto then
    return primi 4096 byte del file;
return encode(text_all, "latin1");
```

3.3.4 Caso speciale: TXT

I file TXT sono file di testo semplice, i quali non presentano sezioni compresse o non strutturate. Per questo motivo non viene effettuata alcuna estrazione strutturale, in quanto l'intero contenuto del file è rilevante per la rappresentazione statistica.

Può tuttavia succedere che il file considerato sia troppo piccolo (o addirittura vuoto). In questi casi, è opportuno definire una soglia minima di lunghezza (256 byte), al di sotto della quale si raggiunge la lunghezza minima ripetendo il contenuto del file fino a raggiungere tale soglia.

Questo permette di ottenere una rappresentazione statistica più robusta, evitando che file troppo piccoli possano introdurre rumore o distorsioni nelle metriche calcolate in seguito.

3.3.5 Approccio Generalizzato all'Estrazione Strutturale: Analisi e Limiti

In una prima fase di analisi, si è valutata l'efficacia di un approccio generalizzato per l'estrazione strutturale dei file. Più nel dettaglio, l'idea era quella di catturare i byte significativi del file in modo totalmente indipendente dal formato, evitando così regole specifiche per ciascun tipo di documento.
Un primo approccio consiste nel combinare:

- una porzione iniziale del file.
- una porzione finale.
- una selezione dei blocchi interni con entropia più bassa.

Lo scopo è quello di catturare le sezioni più strutturate del file, ignorando le parti compresse o altamente entropiche.

Tale approccio, sebbene interessante, dimostra diversi limiti rilevanti. In primo luogo, essendo le sezioni strutturali fortemente dipendenti dal formato, l'estrazione generalizzata tende a perdere informazioni rilevanti, portando a rappresentazioni statistiche meno significative.

In secondo luogo, la distinzione tra file reali e file random diventa meno marcata, riducendo l'efficacia delle metriche di confronto.

Infine, per rendere efficace una selezione automatica delle regioni ad alta informazione è necessario disporre di dataset molto ampi per ciascun formato, condizione non soddisfatta nel contesto di questo lavoro, nel quale sono stati utilizzati dataset che contengono approssimativamente dai 10 000 ai 5 000 file per formato.

Per queste ragioni, si è deciso di adottare l'approccio basato su regole specifiche per ciascun formato, come descritto in precedenza.

3.4 Metriche di Distanza e Divergenza

Una volta ottenuta la rappresentazione statistica del file sotto forma di BFD (o BFC), è necessario disporre di metriche quantitative per confrontarle. Lavori in letteratura (come per esempio Tabish et al. [17]) utilizzano metriche di distanza e divergenza per quantificare quanto due distribuzioni di byte differiscono tra loro. Queste misure consentono di stimare la similarità statistica tra file, ovvero quanto due file presentano un contenuto binario strutturalmente simile o dissimile.

3.4.1 Entropia di Shannon

Una prima metrica utile per descrivere o classificare file è l'entropia di Shannon, la quale misura il grado di casualità della distribuzione dei byte. In particolare, valori elevati indicano un contenuto casuale, al contrario, valori bassi riflettono un contenuto più strutturato.
Sorokin [18] introduce il concetto di *structural entropy* per il confronto di file in base alla distribuzione locale dell'entropia, mostrando come la sequenza dei valori entropici può essere vista come una firma statistica della sua struttura interna.

Formalmente, l'entropia di Shannon di una distribuzione di byte è definita come:

$$H = -\sum_{i=0}^{255} p(i) \log_2 p(i)$$

dove $p(i)$ rappresenta la frequenza relativa del byte i all'interno della finestra di analisi.

3.4.2 Jensen–Shannon Divergence (JSD)

Un'altra metrica utilizzata per tale scopo è la *Jensen–Shannon Divergence* (JSD). A differenza della KL divergence, la JSD è simmetrica e sempre definita, rendendola stabile.
La JSD viene impiegata per quantificare quanto due distribuzioni di probabilità differiscono o si assomigliano, e quindi può essere utilizzata per misurare quanto le BFD di due file siano statisticamente simili.
Formalmente, la JSD è definita come:

$$\mathrm{JSD}(P, Q) = \frac{1}{2}\mathrm{KL}(P\|M) + \frac{1}{2}\mathrm{KL}(Q\|M)$$

dove $M = \frac{1}{2}(P + Q)$, KL è la divergenza di Kullback–Leibler e P, Q rappresentano le due distribuzioni di byte da confrontare.
Più nel dettaglio, il valore di JSD è compreso fra 0 e 1, dove valori vicini a 0 indicano distribuzioni molto simili, al contrario valori vicini a 1 riflettono distribuzioni diverse.
Tabish et al. [17] utilizzano la JSD per analizzare la differenza informativa tra distribuzioni di byte in file benigni e file malevoli, mostrando che la divergenza cresce proporzionalmente alla differenza di struttura statistica del contenuto.

3.4.3 Manhattan Distance

La Manhattan distance è una misura di dissimilarità fra due distribuzioni, calcolata come somma delle differenze assolute tra le rispettive sequenze di byte:

$$D_{\mathrm{L1}}(X, Y) = \sum_{i=0}^{255} |X_i - Y_i|$$

Nella letteratura, è usata in diversi lavori che si occupano di file type classification e content-based analysis. In particolare, Amirani et al. [19] la utilizzano per confrontare le distribuzioni di byte di file e frammenti, mostrando che valori bassi di Manhattan distance riflettono un'elevata similarità fra i file.

3.4.4 Cosine Similarity

Un'ulteriore metrica utile è la *Cosine Similarity*, una misura di tipo angolare la quale valuta quanto due distribuzioni siano orientate nella stessa direzione.

Riguardo alla BFD, permette di misurare quanto due file presentino pattern di frequenze di byte simili.
La Cosine Similarity fra due distribuzioni X e Y è definita come:

$$\cos(X, Y) = \frac{\sum_{i=0}^{255} X_i Y_i}{\sqrt{\sum_{i=0}^{255} X_i^2} \sqrt{\sum_{i=0}^{255} Y_i^2}}$$

Il risultato varia fra 0 e 1, in particolare valori vicini ad 1 indicano distribuzioni quasi identiche, mentre valori vicini allo 0 distribuzioni dissimili.
In letteratura, Ahmed et al. [20] utilizzano questa metrica nel contesto del *content-based file type identification*, confrontando gli istogrammi di frequenza dei byte per classificare i file in base al loro tipo. Anche Tabish et al. [17] includono la Cosine Similarity per misurare la differenza tra distribuzioni di byte, evidenziando come la correlazione angolare sia un indicatore efficace per distinguere file appartenenti a categorie diverse.

3.4.5 Total Variation Distance (TVD)

La Total Variation Distance (TVD) è una metrica probabilistica che quantifica la massima differenza tra due distribuzioni di probabilità. Nel contesto delle BFD, essa misura quanto la distribuzione dei byte di un file differisce da quella di un altro file, fornendo una stima diretta del grado di dissimilarità statistica tra i due contenuti.
Formalmente è definita come:

$$\mathrm{TVD}(P, Q) = \frac{1}{2} \sum_{i=0}^{255} |P_i - Q_i|$$

dove P e Q sono le distribuzioni di probabilità dei byte dei due file. Il valore della distanza varia tra 0 e 1, con 0 che indica distribuzioni identiche e 1 che rappresenta distribuzioni completamente disgiunte.

Capitolo 4

Sistema di Rilevamento dei File Verosimili (Detector)

Nella sezione precedente sono state presentate diverse metriche utili a descrivere e confrontare file in base alla loro distribuzione di byte.
È quindi necessario introdurre un sistema di rilevamento con l'obiettivo di valutare la verosimiglianza di file generati artificialmente. A tale scopo si utilizzeranno le metriche descritte per confrontare i file generati con delle soglie predefinite, stabilite analizzando un dataset di file reali.
Di seguito verranno descritte le varie fasi del detector, descrivendo le scelte implementative adottate e gli strumenti utilizzati.

4.1 Analisi Preliminare: Distanza tra File Reali e File Random

Come primo passo per la costruzione del detector, è stata condotta un'analisi per valutare la distanza statistica fra i file reali e file generati casualmente. Lo scopo di questa prima analisi è quello di quantificare la differenza fra i file reali e random, facendo vedere che, a livello statistico, i file random si discostano significativamente dai file reali.
Per farlo, si confrontano le rappresentazioni N-gram dei file reali con quelle dei file random mediante le metriche discusse in precedenza, ottenendo una stima quantitativa della distanza media tra i due insiemi.

In particolare, per questa fase è stato sviluppato uno script che, dati i dataset di file reali e di file random per ogni formato considerato, produce un insieme di confronti appaiati tra i due dataset.

Per ogni formato considerato, lo script esegue le seguenti operazioni:

- seleziona casualmente una coppia di file dello stesso formato (reale, randomico) dai dataset opportuni;
- calcola la N-gram dei due file selezionati;
- applica le metriche per confrontare le N-gram calcolate in precedenza;
- ripete il processo per un numero prefissato di iterazioni (20 000 nel nostro caso), per ottenere una stima statistica robusta.

Questa analisi evidenzia una chiara separazione statistica tra i file reali e quelli generati casualmente, che mostrano distribuzioni di byte nettamente differenti.

La Figura 4.1 mostra la statistica ottenuta con la metrica JSD calcolata sulle coppie file reale–file random per ciascun formato. Similmente, la Figura 4.2 mostra i risultati ottenuti con la Manhattan distance, la Figura 4.3 con la Total Variation Distance, e la Figura 4.4 con la Cosine Similarity.

In tutti i casi, si osservano valori di distanza/divergenza elevati, indicando che i file random si discostano significativamente dai file reali.

La Figura 4.5 mostra invece i valori medi di entropia dei file reali e random. Anche qui emerge una differenza significativa, con i file randomici che presentano un'elevata entropia rispetto ai file reali, i quali mostrano valori più bassi, a riflettere una struttura più ordinata e definita.

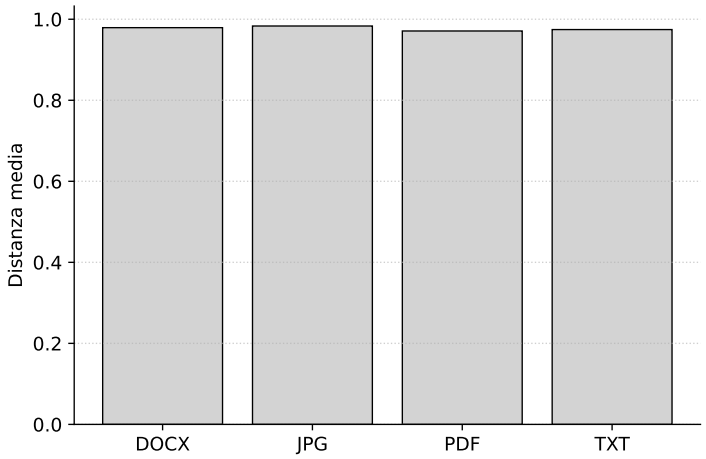

Figura 4.1: Distanza media tra file reali e file random calcolata mediante la metrica JSD.

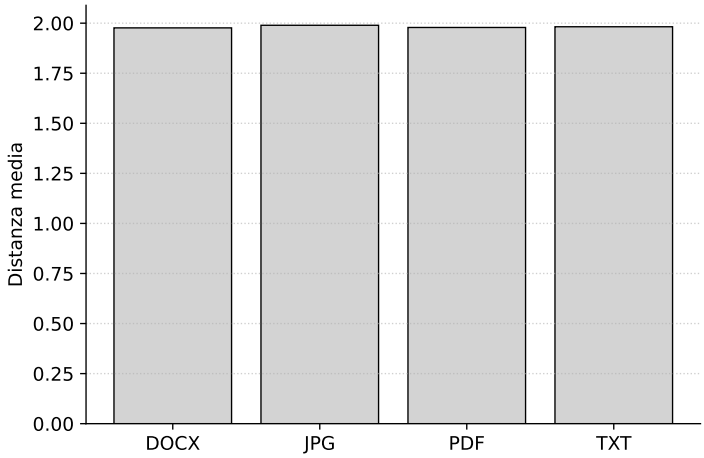

Figura 4.2: Distanza media tra file reali e file random calcolata mediante la Manhattan Distance.

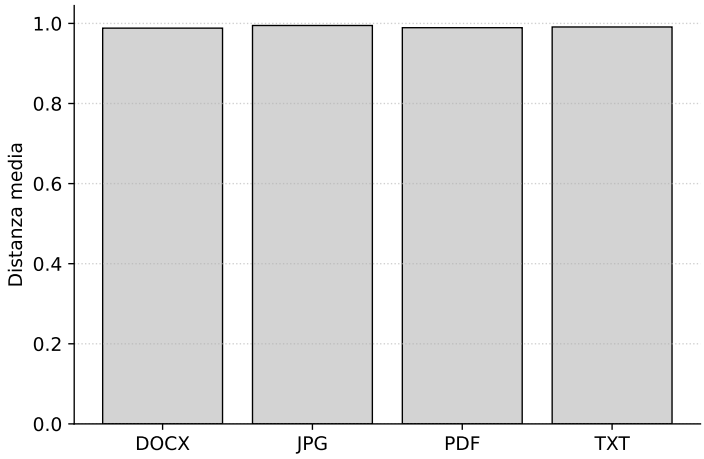

Figura 4.3: Distanza media tra file reali e file random calcolata mediante la Total Variation Distance (TVD).

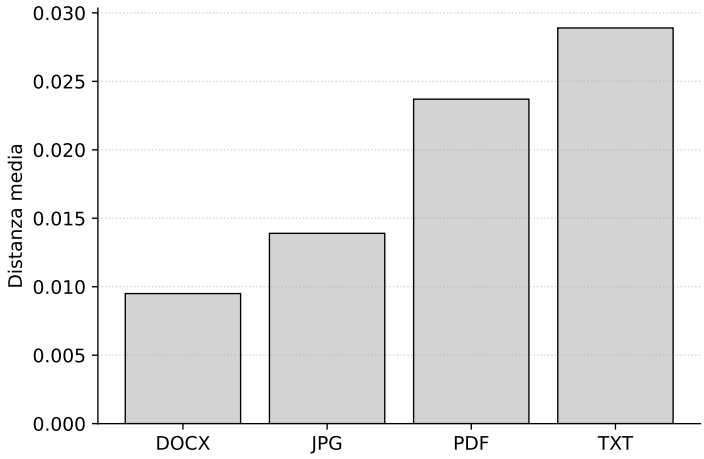

Figura 4.4: Distanza media tra file reali e file random calcolata mediante la Cosine Similarity.

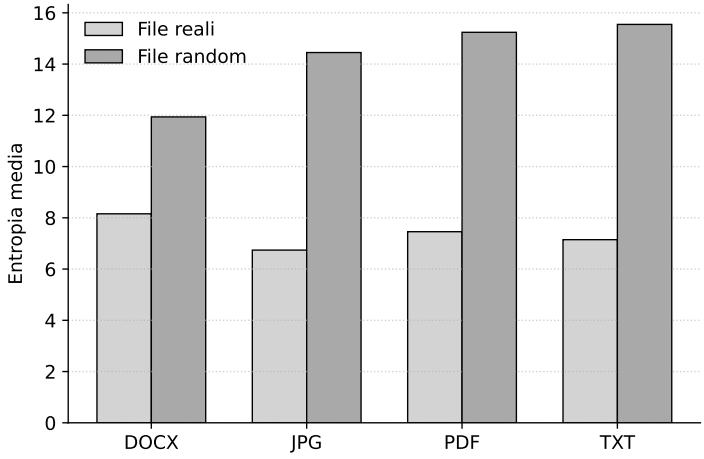

Figura 4.5: Entropia media dei file reali e dei file random.

Poiché i valori osservati negli istogrammi sono concentrati in prossimità del massimo, le differenze tra i formati risultano poco evidenti a livello visivo. Per rendere il confronto più chiaro, nella Tabella 4.1 sono riportati i valori medi delle metriche per ciascun formato.

Tabella 4.1: Statistiche medie per formato (real-random)

Formato	**JSD**	**TVD**	**L1**	**Cosine_sim**	**Entropy_random**	**Entropy_real**
docx	0.9792	0.9882	1.9765	0.0095	11.9379	8.1562
jpg	0.9833	0.9948	1.9895	0.0139	14.4502	6.7402
pdf	0.9710	0.9893	1.9785	0.0237	15.2401	7.4579
txt	0.9745	0.9911	1.9821	0.0289	15.5480	7.1467

4.2 Definizione della Verosimiglianza dei File

Visti i risultati ottenuti nella fase preliminare, che mostrano come i file reali differiscano in modo significativo da quelli generati casualmente, la domanda successiva da porsi è quando un file sintetico possa essere considerato "realistico" o "verosimile". In altre parole, quanto un file generato artificialmente debba essere simile a un file reale per essere considerato credibile.

Per rispondere alla domanda, è necessario definire delle soglie di verosimiglianza per ciascuna metrica considerata, al di sotto (o al di sopra, a seconda della metrica che si considera) delle quali un file generato viene considerato realistico.

Nelle sezioni seguenti viene illustrato l'approccio adottato per determinare tali soglie e costruire un sistema di decisione robusto.

4.3 Costruzione della Firma Media

Per poter stabilire se un file generato sia o meno verosimile, è necessario disporre di un riferimento statistico rappresentativo dei file reali. A tale scopo viene costruita una *firma media* per ciascun formato considerato, la quale riassume la struttura tipica dei file reali.

La firma media viene calcolata come la media aritmetica delle N-gram dei file reali presenti nel dataset. Il vettore risultante corrisponde dunque alla distribuzione media dei byte per quel formato specifico.
Tale firma media funge da riferimento per confrontare la verosimiglianza di un file: un file che presenta una distribuzione simile alla firma media viene considerato realistico.

Una volta definita la firma media, per ciascun file è possibile calcolare la distanza fra la sua distribuzione e la firma media, ottenendo un insieme di valori per ogni metrica considerata.
Resta quindi da stabilire quali valori di distanza/divergenza possono essere considerati accettabili per definire un file come verosimile.

Nella sezione che segue viene descritto l'approccio adottato per determinare tali soglie di verosimiglianza.

4.4 Definizione e Ottimizzazione delle Soglie di Verosimiglianza

Come descritto sopra, l'obiettivo è stabilire, per ogni metrica, una *soglia di verosimiglianza* tale per cui un file è considerato verosimile se la distanza dalla sua firma media è inferiore (o superiore) a tale soglia.

Un primo metodo per determinare queste soglie è un approccio empirico, basato sull'analisi statistica dei valori di distanza calcolati sui file reali. Ad esempio, si possono calcolare la media e la deviazione standard delle distanze dei file reali dalla firma media, e definire la soglia come:

$$\text{soglia} = \mu + k \cdot \sigma$$

dove k è un fattore di scala scelto in base al livello di tolleranza desiderato.
Tuttavia, questo approccio presenta alcuni limiti, ad esempio la scelta di k può essere arbitraria e non garantisce una separazione ottimale tra file reali e generati.

Pertanto, per individuare soglie più robuste, è necessario adottare un approccio di ottimizzazione più sistematico.

Per stabilire una soglia ottimale, si rende necessario disporre di un indicatore quantitativo che misuri la qualità della classificazione ottenuta.
Nel nostro caso, fissata una soglia T, possiamo classificare un file come realistico o non realistico a seconda della distanza dalla firma media.

Questo porta alla definizione di quattro insiemi:

- TP: l'insieme dei file reali classificati come realistici (Veri Positivi);
- TN: l'insieme dei file randomici classificati come non realistici (Veri Negativi);
- FN: l'insieme dei file reali classificati come non realistici (Falsi Negativi);
- FP: l'insieme dei file randomici classificati come realistici (Falsi Positivi).

L'idea è quindi quella di costruire una funzione obiettivo, che, dato uno specifico valore di soglia T, misuri la qualità della classificazione ottenuta.

Dobbiamo dunque cercare di classificare tutti i file reali come realistici (massimizzare TP) ed in contemporanea evitare di accettare come realistici file randomici (minimizzare FP). Pertanto, una possibile funzione obiettivo può essere definita come:

$$C(T) = \alpha \cdot FN(T) + \beta \cdot FP(T)$$

dove α e β sono pesi che bilanciano l'importanza di minimizzare i falsi negativi e i falsi positivi, e $FN(T)$ e $FP(T)$ rappresentano rispettivamente la percentuale di falsi negativi e falsi positivi ottenuti con la soglia T, dividendo il numero di questi ultimi per il numero totale di file reali e randomici rispettivamente.

Una volta definita la funzione obiettivo $C(T)$, il problema si riduce a trovare la soglia T^* per cui $C(T)$ è minima. Nel presente lavoro, per risolvere questo problema di ottimizzazione, si è scelto di utilizzare l'algoritmo di *Dual Annealing*, descritto in seguito.

4.4.1 Ricerca della Soglia Ottimale tramite Dual Annealing

Per individuare la soglia ottimale T^* che minimizza $C(T)$, può risultare inefficace utilizzare metodi di ottimizzazione classici come ad esempio la discesa del gradiente, in quanto la funzione obiettivo può avere molteplici minimi locali, derivanti dalla distribuzione non lineare degli score di distanza.

Per questa ragione si è scelto di adottare un approccio di ottimizzazione globale basato sull'algoritmo di *Dual Annealing*.

Quest'algoritmo combina elementi del Classical Simulated Annealing e del Fast Simulated Annealing in un'unica procedura di ottimizzazione globale. Come riportato nella documentazione ufficiale di SciPy [21], Dual Annealing usa una particolare visiting distribution di tipo Cauchy-Lorentz modificata, che permette di effettuare salti esplorativi anche molto ampi nello spazio delle soluzioni, in modo da minimizzare la probabilità di finire in eventuali minimi locali.

In particolare, ad ogni iterazione il metodo genera una nuova soluzione candidata T applicando la visiting distribution e decide se accettarlo attraverso una probabilità di accettazione che dipende dalla differenza di costo e dalla temperatura artificiale.
Grazie a queste tecniche, l'algoritmo di Dual Annealing è in grado di esplorare efficacemente lo spazio delle soluzioni, consentendo di individuare la soglia ottimale T^*.

4.5 Overfitting e Cross-validation

Arrivati a questo punto, è importante considerare alcuni aspetti critici che possono influenzare l'affidabilità e la robustezza del modello sviluppato. In particolare, il processo di ottimizzazione delle soglie può essere soggetto a fenomeni di overfitting quando il dataset disponibile è limitato.

Nel contesto di questo lavoro, si dispone infatti di un numero relativamente basso di file per ciascun formato. Questo può portare a situazioni in cui si hanno soglie di verosimiglianza che si adattano in modo troppo specifico ai file presenti nel dataset, riducendo la capacità del modello di generalizzare a file diversi.

Per ridurre il rischio di overfitting, è possibile adottare strategie che permettano di costruire dati su porzioni diverse dei dataset disponibili, in modo da evitare che i risultati ottenuti siano troppo specifici a una determinata selezione di file.

Per questo motivo, nel presente lavoro è stato utilizzato un approccio di *k-fold cross-validation*.

La cross-validation è una tecnica statistica di valutazione del modello. Essa suddivide l'intero dataset in k sottoinsiemi (fold) di dimensioni uguali. Il modello viene poi addestrato su $k-1$ fold e testato sul fold rimanente, ripetendo il processo k volte, in modo che ogni fold venga utilizzato una volta come set di test.
I risultati ottenuti vengono poi mediati per ottenere una stima più robusta delle prestazioni del modello.
Come descritto da Arlot e Celisse [22], questo approccio permette di fornire una stima più affidabile della capacità di generalizzazione del modello, riducendo il rischio di overfitting dovuto alla limitata dimensione del dataset.

4.6 Implementazione del Sistema di Rilevamento

Concluse le considerazioni metodologiche, nella seguente sezione sarà descritto nel dettaglio il sistema di rilevamento implementato, mostrando le scelte tecniche adottate, il flusso di lavoro e gli strumenti utilizzati.

Il sistema di rilevamento è progettato con lo scopo di analizzare diversi tipi di file (PDF, JPG, DOCX, TXT) e valutare la loro verosimiglianza in base alle metriche e soglie definite in precedenza.
Il sistema si articola in quattro fasi principali:

- Estrazione strutturale del file.
- Calcolo della firma media e degli score di distanza.

- Ottimizzazione delle soglie di verosimiglianza.
- Valutazione delle performance del detector.

Nelle sezioni seguenti verranno descritte in dettaglio ciascuna di queste fasi.

4.6.1 Struttura Generale

Il sistema di rilevamento è stato organizzato come una serie di moduli, ciascuno responsabile di una specifica fase del processo.
La struttura generale del sistema può essere sintetizzata come segue:

1. Generazione dello split k-fold.

   Per ciascun formato si costruiscono due insiemi separati: file reali e file randomici. Ogni insieme viene mescolato in modo casuale ma riproducibile (seed fisso) e poi suddiviso in k blocchi disgiunti di dimensione il più possibile uniforme (la differenza tra blocchi è al massimo 1 file).

   Nel fold i-esimo, il blocco i (sia reali sia randomici) costituisce il test, mentre l'unione dei restanti $k-1$ blocchi costituisce il train.

2. Calcolo firma media e score di distanza per ciascun fold.

   Per ogni fold, i file che appartengono al fold non utilizzato per il test vengono usati per l'addestramento. Per fare un esempio, nel caso in cui $k = 3$, si usano i file dei fold 1 e 2 per l'addestramento e i file del fold 3 per il test, e così via per gli altri fold.

   Su questo insieme di addestramento, viene calcolata la firma media per ciascun formato. Successivamente, per ogni file dei fold di train e test, vengono calcolate le metriche di distanza rispetto alla firma media.

3. Ottimizzazione delle soglie di verosimiglianza per ciascun fold.

   Usando come base gli score ottenuti nel passo precedente sui file di addestramento, viene eseguita l'ottimizzazione delle soglie per ogni metrica e formato.

   Come descritto in precedenza, l'ottimizzazione viene formulata come un problema di minimizzazione di una funzione obiettivo $C(T)$, al fine di bilanciare i falsi positivi e negativi.

4. Valutazione delle performance del detector.

   In questa fase le soglie ottimizzate con il fold di addestramento vengono applicate ai file del fold di test.

   In particolare, per ciascun formato vengono calcolate:

   - il numero di falsi positivi (FP): file randomici classificati come realistici.
   - il numero di falsi negativi (FN): file reali classificati come non realistici.

- il contributo di ciascuna metrica al processo decisionale.

I risultati dei singoli fold vengono aggregati per ottenere una stima complessiva delle performance del detector.

4.6.2 Calcolo della Firma Media e degli Score di Distanza

Una volta definita la suddivisione in fold, il passo successivo consiste nel calcolare la firma media per ciascun formato e gli score di distanza per ciascun file.

Il primo passo consiste nel convertire ciascun file nella sua rappresentazione numerica basata sulle N-gram, come descritto nella Sezione 3.2. Nel presente lavoro è stato scelto di utilizzare $n = 2$ (bigrammi), ottenendo un vettore di dimensione 65536 (256^2).

Con il seguente pseudocodice viene illustrato il processo di calcolo della N-gram per un file:

Algorithm 4: BFD-2gram

```
Input: contenuto binario del file (b)
if b è vuota then
    return vettore di 65 536 zeri;
arr ← converti b in array di interi 8-bit;
if lunghezza(arr) < 2 then
    return vettore di 65 536 zeri;
for i = 0 to lunghezza(arr) - 2 do
    idx ← arr[i] * 256 + arr[i+1];
    BFD[idx] ← BFD[idx] + 1;
normalizza(BFD);
return BFD;
```

Dopo aver calcolato la N-gram per ciascun file, si procede al calcolo della firma media, la quale è ottenuta come la media aritmetica delle N-gram dei file reali presenti nel fold di addestramento.

Algorithm 5: CalcoloCentroide

```
Input: Lista di file F_real
foreach f ∈ F_real do
    v ← BFD-2gram(f);
    lista_vettori.append(v);
centroide ← mean(lista_vettori);
normalizza(centroide);
return centroide;
```

Si procede poi al calcolo degli score di distanza per ciascun file, confrontando la sua N-gram con la firma media calcolata in precedenza.

Algorithm 6: CalcoloScore

Input: Insieme di file $F = F_{\text{real}} \cup F_{\text{rand}}$, centroide C
foreach *file* $f \in F$ **do**
    v $\leftarrow$ BFD_2gram(f);
    score[f].jsd $\leftarrow$ JensenShannon(v, C);
    score[f].tvd $\leftarrow$ TotalVariation(v, C);
    score[f].l1 $\leftarrow \|v - C\|_1$;
    score[f].cos $\leftarrow$ CosineSimilarity(v, C);
    score[f].ent $\leftarrow$ Entropy(v);
return score;

Per ciascun fold, vengono prodotti due file csv:

- `train_scores_fold_i.csv`: contiene gli score di distanza per i file dei fold di addestramento.
- `test_scores_fold_i.csv`: contiene gli score di distanza per i file del fold di test.

Gli score calcolati in questa fase verranno poi utilizzati per l'ottimizzazione delle soglie di verosimiglianza e la valutazione delle performance del detector nelle fasi successive.

4.6.3 Applicazione dell'Ottimizzazione delle Soglie

Una volta ottenuti gli score di distanza per i file di addestramento, la fase successiva consiste nell'individuare, per ciascuna metrica e ciascun formato, la soglia di verosimiglianza ottimale.
L'ottimizzazione viene eseguita in modo indipendente per ciascun fold, considerando separatamente ogni combinazione formato–metrica. Per ogni metrica, gli score dei file reali e randomici appartenenti al training set vengono utilizzati per stimare una soglia che massimizzi la capacità discriminante del modello.

La funzione obiettivo $C(T)$ è la stessa funzione definita nella Sezione 4.4, ed è basata su una combinazione pesata di falsi positivi e falsi negativi. Nell'implementazione, questi valori sono calcolati a partire dagli score di distanza ottenuti nella fase precedente.

Per ciascuna metrica, il calcolo dei falsi positivi e negativi avviene applicando regole di decisione coerenti con la natura della metrica considerata. Ad esempio, per metriche di distanza come JSD, TVD e L1, un file viene classificato come realistico se il suo score è inferiore alla soglia T.

Come descritto in precedenza, per risolvere il problema di ottimizzazione si è scelto di utilizzare l'algoritmo di Dual Annealing, il quale esplora lo spazio delle possibili soglie T e restituisce quella che minimizza la funzione obiettivo $C(T)$.

Il risultato di questa fase è un insieme di soglie ottimali per ciascun formato e metrica, che verranno poi utilizzate nella fase di valutazione delle performance del detector.

4.6.4 Valutazione delle Performance

Una volta ottenute le soglie ottimizzate, il passo finale per la costruzione del detector consiste nella valutazione delle sue performance sui file del fold di test. In questa fase, le soglie ottenute sul training set vengono applicate esclusivamente ai file del test set.

Per ognuno dei file di test, si caricano gli score di distanza calcolati e si confrontano con le soglie ottenute.
La decisione finale sulla verosimiglianza è ottenuta combinando i risultati delle singole metriche. In particolare, la decisione è ottenuta applicando *AND logico*: un file è considerato verosimile solamente se tutte le metriche considerate rientrano nei rispettivi parametri di soglia.
Nel caso in cui anche una sola metrica non rientri nella soglia, il file viene classificato come non realistico.

La scelta di adottare un AND logico è motivata dalla necessità di ridurre al minimo la probabilità di falsi positivi. Nel contesto di questo lavoro, infatti, l'accettazione di file non realistici potrebbe compromettere l'efficacia del sistema, poiché tali file rischierebbero di essere facilmente individuati e ignorati dal malware.

Questo approccio privilegia quindi una classificazione restrittiva, accettando un lieve incremento dei falsi negativi come compromesso coerente con lo scenario applicativo considerato, in cui è preferibile scartare alcuni file potenzialmente realistici piuttosto che accettare file statisticamente anomali.

Per quanto riguarda le singole metriche, come accennato prima, la regola di decisione dipende dalla natura della metrica stessa.
Nel dettaglio:

- Per metriche di distanza/divergenza (JSD, TVD, L_1):

  Un file è considerato realistico se il suo score è *minore o uguale* alla soglia di verosimiglianza stimata per il formato considerato;

  Per esempio, se la soglia di verosimiglianza per JSD calcolata su un file PDF è 0.95, e la soglia calcolata per formato PDF e metrica JSD è 0.92, allora il file viene considerato non realistico rispetto alla metrica JSD.

- Per la metrica di Cosine Similarity:

  Un file è considerato realistico se il suo score è *maggiore o uguale* alla soglia di verosimiglianza stimata per il formato considerato;

- Per la metrica di Entropia:

  la verosimiglianza viene valutata confrontando l'entropia del file con un intervallo accettabile $[T_{\text{low}}, T_{\text{high}}]$, all'interno del quale un file realistico dovrebbe cadere.

4.6.5 Detector Finale

Una volta descritte le fasi di addestramento, ottimizzazione e valutazione, il sistema è utilizzabile in modalità operativa per valutare la verosimiglianza di nuovi file generati.
Le soglie ottenute dai vari fold vengono aggregate al fine di ottenere un set finale di soglie da utilizzare nel detector finale.
In particolare, per le metriche basate su una singola soglia (JSD, TVD, L_1, Cosine Similarity), la soglia finale è ottenuta come il 95-esimo percentile delle soglie ottenute sui singoli fold.
Per la metrica di entropia, invece, gli intervalli finali $[T_{\text{low}}, T_{\text{high}}]$ sono ottenuti calcolando il valore medio dei diversi fold.

Il detector finale opera come segue:

Algorithm 7: DetectorOperativo

Input: file f, centroidi $\{C_{fmt}\}$, soglie $\{T_{fmt}\}$
fmt $\leftarrow$ determinaFormato(f);
if *fmt non supportato* **then**
    return NON_PLAUSIBILE;
$v \leftarrow$ BFD-2gram(f);
$C \leftarrow C_{fmt}$;
$T \leftarrow T_{fmt}$;
foreach *metrica* m **do**
    $s_m \leftarrow$ calcolaScore(v, C, m);
    verifica s_m rispetto alla soglia T_m;
if *tutte le metriche soddisfano le soglie* **then**
    return PLAUSIBILE;
else
    return NON_PLAUSIBILE;

4.7 Valutazione del Detector

In questa sezione si presentano e analizzano i valori sperimentali ottenuti.

L'obiettivo di questa valutazione è misurare la capacità del modello di classificare file come realistici o randomici, analizzando le prestazioni ottenute in termini di falsi positivi, falsi negativi e al contributo delle singole metriche.

La valutazione è stata condotta utilizzando una procedura di cross-validation k-fold con $k = 3$, applicando a ciascun fold le soglie ottimizzate sul corrispondente insieme di addestramento.

4.7.1 Setup Sperimentale

La valutazione utilizza più dataset, divisi per formato di file (PDF, JPG, DOCX, TXT).
In particolare, per ciascun formato si hanno due dataset distinti, uno contenente file reali e l'altro contenente file generati casualmente.
I file reali rappresentano documenti legittimi provenienti da dataset pubblici ampiamente utilizzati in ambito forense e di analisi dei file. In particolare, per i formati PDF e DOCX sono stati utilizzati documenti provenienti dai dataset GovDocs1 [23] e NapierOne [24, 25]. Per il formato JPG sono state utilizzate immagini provenienti da dataset pubblici di immagini naturali, mentre per i file TXT sono stati utilizzati documenti testuali provenienti da collezioni pubbliche di testi, come il dataset Project Gutenberg [26].
I file randomici sono stati invece generati utilizzando il tool Ranflood, il quale si basa sulla tecnica del Xorshift.
Per i formati PDF, JPG e TXT, il dataset è composto da circa 10 000 file reali e 2 000 file randomici.
Per il formato DOCX, invece, il dataset è composto da circa 5 000 file reali e 2 000 file randomici.

4.7.2 Analisi per Formato

In questa sezione sono riportati i risultati ottenuti nella valutazione delle performance del detector sui file di test.
Per ogni formato si considera:

- il numero e la percentuale di falsi positivi (FP)
- il numero e la percentuale di falsi negativi (FN)
- il contributo di ciascuna metrica al processo decisionale.

I risultati che seguono rappresentano l'aggregazione dei risultati ottenuti sui tre fold.

Metrica	FN	FN (%)	FP	FP (%)
JSD	6	0.06	0	0.00
TVD	6	0.06	0	0.00
L_1	5	0.05	0	0.00
Cosine	0	0.00	1	0.05
Entropy	150	1.50	11	0.59

Tabella 4.2: Risultati del detector per il formato JPG.

Metrica	FN	FN (%)	FP	FP (%)
JSD	1	0.02	0	0.00
TVD	1	0.02	0	0.00
L_1	1	0.02	0	0.00
Cosine	2	0.04	0	0.00
Entropy	2	0.04	0	0.00

Tabella 4.3: Risultati del detector per il formato DOCX.

Metrica	FN	FN (%)	FP	FP (%)
JSD	0	0.00	0	0.00
TVD	0	0.00	0	0.00
L_1	0	0.00	0	0.00
Cosine	0	0.00	0	0.00
Entropy	3	0.03	0	0.00

Tabella 4.4: Risultati del detector per il formato PDF.

Metrica	FN	FN (%)	FP	FP (%)
JSD	116	1.16	0	0.00
TVD	139	1.39	0	0.00
L_1	131	1.31	0	0.00
Cosine	73	0.73	1	0.05
Entropy	8	0.08	0	0.00

Tabella 4.5: Risultati del detector per il formato TXT.

Dai risultati, si può osservare che il sistema di rilevamento implementato riesce a classificare correttamente la maggior parte dei file reali e randomici, con percentuali di falsi positivi prossime allo 0% per tutti i formati considerati.

Anche le percentuali di falsi negativi risultano molto basse, seppure leggermente più elevate, con valori generalmente prossimi allo 0% e comunque inferiori al 2% per tutti i formati analizzati.
Nel dettaglio, si osservano percentuali di falsi negativi leggermente più elevate per i formati JPG e TXT. Tale comportamento può essere ricondotto alla presenza, nei dataset utilizzati, di una piccola percentuale di file con caratteristiche strutturali atipiche o fortemente entropiche, che risultano più simili a distribuzioni casuali.
Tuttavia, anche in questi casi, le percentuali di errore rimangono estremamente ridotte e non compromettono in alcun modo le prestazioni complessive del modello.

Per concludere, questi risultati indicano che il modello è efficace nel distinguere tra file reali e generati casualmente, dimostrando la validità dell'approccio basato sulle N-gram e sulle metriche di distanza.

4.7.3 Valutazione Operativa su Dataset Esteso

Per valutare ulteriormente l'efficacia del sistema di rilevamento, è stata condotta una valutazione operativa su un dataset esteso di file reali, non utilizzati nelle fasi di addestramento e validazione.

Nel dettaglio, sono stati considerati:

- circa 100 000 file JPG reali;
- circa 80 000 file TXT reali.

Il detector è stato eseguito utilizzando i centroidi e le soglie finali ottenute dall'aggregazione dei risultati sui diversi fold di cross-validation, senza ulteriori fasi.

Su 109 233 file JPG reali, il detector ha classificato come reali il 97,5% dei file, con una percentuale di falsi negativi pari al 2,5%.
L'analisi delle metriche mostra come la maggior parte dei falsi negativi sia dovuta alla metrica di entropia, la quale ha classificato come non reali 2730 file sui 2749 file totali classificati non reali.
Questo comportamento è coerente con quanto osservato nella fase di validazione, dove la metrica di entropia ha mostrato una maggiore sensibilità ai file con caratteristiche strutturali atipiche.

Per quanto riguarda i file TXT, su un totale di 78 285 file reali, il sistema ha classificato come reali il 99,32% dei file, con una percentuale di falsi negativi pari allo 0,68%.
In questo caso tutte le metriche hanno contribuito in modo equilibrato alla classificazione, senza che una singola metrica emergesse come predominante nella generazione dei falsi negativi.

Questi risultati confermano l'efficacia del sistema di rilevamento anche su dataset estesi e non visti durante l'addestramento, dimostrando la capacità del modello di generalizzare e mantenere alte prestazioni in scenari operativi reali.

4.8 Considerazioni sulle Scelte Progettuali e Confronto con Approcci di Machine Learning

In fase di progettazione del sistema di rilevamento, si è valutata la possibilità di adottare tecniche di Machine Learning per la classificazione dei file.
Tuttavia, dopo un'attenta valutazione, si è deciso di optare per un approccio basato su metriche di distanza e soglie ottimizzate, per diverse ragioni.

In primo luogo, le tecniche di Machine Learning richiedono generalmente grandi quantità di dati per l'addestramento, e nel contesto di questo lavoro, la disponibilità di dataset ampi e bilanciati per ciascun formato di file era limitata.

Inoltre, l'addestramento di modelli generativi o discriminativi può essere computazionalmente costoso, richiedendo risorse hardware significative e tempi di addestramento lunghi.

Infine, l'approccio basato su metriche di distanza offre una maggiore trasparenza e interpretabilità, consentendo di comprendere facilmente come viene presa la decisione di classificazione, cosa che può essere più complessa con modelli di Machine Learning più sofisticati.

Pertanto, l'approccio adottato rappresenta una soluzione efficace e pragmatica, bilanciando la necessità di accuratezza con la praticità e l'efficienza computazionale.

4.9 Lavori Correlati e Confronto con Magika

In questi ultimi anni sono stati individuati diversi metodi per la verifica del tipo di file basati su analisi statistiche del contenuto binario, indipendentemente dall'estensione. In questo contesto, uno dei lavori più rilevanti è Magika, un tool sviluppato da Google per l'identificazione del tipo di file basata soltanto sull'analisi statistica del contenuto.

L'obiettivo di Magika è quello di riconoscere in modo affidabile il formato di un file anche se questo presenta un offuscamento del contenuto, come la modifica dell'estensione o di parti dell'header. Questo dimostra come l'identità di un file sia fortemente legata all'insieme di pattern statistici presenti nel suo contenuto binario, piuttosto che ai singoli byte o sequenze di byte specifiche.

Il presente lavoro affronta un problema affine, tuttavia con obiettivi differenti. Magika infatti si propone di identificare il formato di un file, mentre l'obiettivo

qui è quello di valutare la verosimiglianza di un file rispetto a un formato atteso, con il fine ultimo di proteggere dati sensibili da minacce ransomware.

Dal punto di vista metodologico, le differenze sono evidenti. Magika utilizza un approccio basato su modelli di Machine Learning, in particolare reti neurali, per apprendere le caratteristiche distintive di ciascun formato di file. Questo richiede un addestramento su dataset di grandi dimensioni e una decisione di classificazione che può risultare meno interpretabile. Al contrario, il presente lavoro adotta un approccio basato su rappresentazioni esplicite come le N-gram e su metriche di distanza ben definite, che consentono una maggiore trasparenza nel processo decisionale.

Benché le differenze fra i due approcci siano significative, si possono comunque fare alcune considerazioni comparative. Strumenti come Magika dimostrano che l'identità di un file può essere catturata efficacemente attraverso l'analisi statistica del contenuto binario, risulta quindi rilevante studiare in che modo questa identità possa essere manipolata mantenendo inalterato il contenuto originale. In questo senso, il presente lavoro può essere considerato come un passo in più verso l'analisi controllata dell'identità statistica di un file, fornendo strumenti utili per la rilevazione e per la generazione di file in grado di eludere varie minacce.

Capitolo 5

Sistema di Generazione dei File Verosimili

Nel capitolo precedente è stato descritto nel dettaglio il sistema di rilevamento implementato, illustrando le scelte progettuali, il flusso di lavoro e i risultati sperimentali ottenuti.

Il detector così definito non rappresenta esclusivamente uno strumento di valutazione, ma costituisce un riferimento quantitativo per la validazione delle tecniche di generazione di file verosimili. Attraverso l'uso di rappresentazioni statistiche e di metriche di distanza, esso consente di stabilire in modo oggettivo se un file artificiale rispetti le caratteristiche strutturali tipiche dei file reali per un determinato formato.

In questo contesto, il sistema di rilevamento verrà utilizzato come oracolo per verificare l'efficacia delle tecniche di generazione di file verosimili, al fine di evitare sia una replicazione dei file reali, sia la generazione di file completamente randomici.
In questo capitolo verranno illustrate le metodologie adottate per la generazione di file verosimili, progettate per risultare statisticamente compatibili con i vincoli imposti dal detector.

5.1 Idea Generale

Come descritto nei capitoli precedenti, per generare file verosimili l'idea è che questi debbano risultare statisticamente compatibili con i file reali, senza ricorrere alla replica diretta di file esistenti o alla generazione di file completamente casuali.

L'obiettivo non è quello di generare file semanticamente validi o utilizzabili in applicazioni reali, ma piuttosto di riprodurre caratteristiche osservabili a livello binario.

In particolare, si mira a preservare le proprietà statistiche di ciascun formato, in modo tale che i file generati non risultino distinguibili dai file reali attraverso l'analisi statistica.

La strategia adottata prevede una prima modellazione statistica dei file reali, al fine di catturare le caratteristiche strutturali tipiche di ciascun formato, seguita da un processo di generazione che utilizza queste caratteristiche per produrre nuovi file verosimili.

I dettagli di questo processo verranno illustrati progressivamente nelle sezioni seguenti, a partire dalla modellazione statistica fino alla descrizione dell'implementazione del sistema di generazione.

5.2 Modellazione Statistica dei File Reali

Per descrivere in modo efficace le caratteristiche strutturali dei file reali, è necessario andare oltre la semplice distribuzione globale dei byte e considerare anche le dipendenze locali tra i byte consecutivi. Come discusso in precedenza, la rappresentazione a bigrammi (N-gram con $n = 2$) fornisce una base solida per catturare queste dipendenze, evidenziando pattern tipici di ciascun formato.

L'idea centrale è quella di modellare la generazione dei byte come un processo stocastico a tempo discreto, in cui ogni byte è rappresentato da una variabile aleatoria discreta. In questo contesto, a ciascuna posizione n del file è associata una variabile aleatoria X_n, la quale assume valori nell'insieme discreto finito $S = \{0, \ldots, 255\}$, corrispondente all'insieme dei possibili valori di byte.

L'utilizzo dei bigrammi può essere interpretato come una stima delle probabilità condizionate di osservare un certo valore di byte in funzione del byte precendente. Questo riflette l'ipotesi che la probabilità di osservare un byte non sia indipendente, ma dipenda dal contesto immediatamente precedente.

Tali assunzioni portano naturalmente alla modellazione della sequenza di byte tramite una catena di Markov a tempo discreto di ordine uno. Si richiama pertanto la seguente definizione:

Definizione 5.1 (Catena di Markov a tempo discreto)**.** *Si dice catena di Markov a tempo discreto una successione di variabili aleatorie $(X_n)_{n\geq 1}$, definite su uno stesso spazio di probabilità, tali che:*

1. *le variabili aleatorie X_n sono discrete e assumono valori in uno stesso insieme discreto S, detto spazio degli stati;*
2. *vale la proprietà di Markov, ovvero per ogni $i_1, \ldots, i_{n-1}, i, j \in S$ risulta:*

$$\mathbb{P}(X_{n+1} = j \mid X_1 = i_1, \ldots, X_n = i) = \mathbb{P}(X_{n+1} = j \mid X_n = i).$$

Nel contesto di questo lavoro è dunque possibile rappresentare la generazione di un file come un processo markoviano, in cui la probabilità di osservare un byte dipenda esclusivamente dal byte immediatamente precedente, ipotesi coerente con la modellazione basata su bigrammi adottata e che costituisce il fondamento teorico per la costruzione delle matrici di transizione descritte nella sezione successiva.

5.3 Selezione delle Porzioni di File Analizzate

In maniera del tutto analoga a quanto fatto per il sistema di rilevamento, è necessario definire le porzioni di file da analizzare per la costruzione delle matrici di transizione. Come visto in precedenza, nei file reali vi sono sezioni altamente entropiche che non aggiungono valore informativo alla modellazione statistica del formato, ma che possono invece introdurre rumore e distorsioni.

In questo contesto è dunque opportuno selezionare accuratamente le porzioni di file su cui costruire le catene, evitando di includere sezioni che potrebbero compromettere la qualità della modellazione.

Nel dettaglio, si adottano le stesse strategie di selezione delle porzioni di file utilizzate per il sistema di rilevamento, come descritto nella Sezione 3.3.

L'adozione dello stesso criterio di estrazione strutturale sia nella fase di rilevamento sia nella fase di generazione garantisce inoltre coerenza tra il modello statistico appreso e il sistema di valutazione, permettendo di utilizzare il detector come oracolo per la validazione dei file generati.

5.4 Costruzione delle Matrici di Transizione

Data l'idea di modellare la generazione dei byte come un processo markoviano, il passo successivo consiste nella costruzione delle matrici di transizione che descrivono le probabilità condizionate di transizione tra i vari stati (byte).
Nel dettaglio, una catena di Markov a tempo discreto è caratterizzata dalle probabilità di transizione tra i suoi stati.
Nel nostro caso, dove la catena risulta essere omogenea e a stati finiti, tali probabilità possono essere rappresentate tramite una matrice di transizione P.

Si ricorre alla seguente definizione:

Definizione 5.2 (Matrice di transizione)**.** *Sia $(X_n)_{n\geq 1}$ una catena di Markov omogenea a stati finiti, con spazio degli stati S di cardinalità N. Si definisce matrice di transizione la matrice $\Pi \in \mathbb{R}^{N\times N}$ le cui componenti sono date da:*

$$\pi_{ij} = \mathbb{P}(X_{n+1} = j \mid X_n = i), \quad \forall\, i, j \in S,$$

dove l'elemento π_{ij} rappresenta la probabilità di transizione dallo stato i allo stato j.

In questo lavoro, lo spazio degli stati S coincide con l'insieme dei possibili valori di byte, $S = \{0, \ldots, 255\}$. Ne consegue che la matrice di transizione P è una matrice 256×256 in cui ogni riga rappresenta la distribuzione di probabilità condizionata del byte successivo dato il byte corrente.

Per costruire la matrice di transizione per ciascun formato di file, si procede come segue:

- Dato un insieme di file reali appartenenti a uno stesso formato, si analizzano le sequenze di byte estratte dai file e, per ogni coppia di byte consecutivi (i, j) osservata, viene incrementato il contatore associato alla transizione dallo stato i allo stato j.
- In questo modo, si ottiene una matrice di conteggi C in cui ogni elemento C_{ij} rappresenta il numero di occorrenze della coppia di byte (i, j) osservata nei file reali.
- Ottenuta la matrice di conteggi, la matrice di transizione P viene calcolata normalizzando ciascuna riga, ovvero:

  $$P_{ij} = \frac{C_{ij}}{\sum_{k \in S} C_{ik}}, \quad i, j \in S,$$

  in modo tale che ogni riga di P rappresenti una distribuzione di probabilità condizionata valida.

Questo processo viene ripetuto per ciascun formato di file considerato, ottenendo così una matrice di transizione specifica per ogni formato.

5.5 Processo di Generazione Markoviana

Una volta ottenuta la matrice di transizione P, il passo successivo consiste nell'utilizzare questa matrice per generare, byte dopo byte, nuovi file verosimili. Il processo di generazione viene modellato come un campionamento sequenziale dalla catena di markov definita dalla matrice di transizione P. In particolare, definita L la lunghezza del file da generare, il file viene generato producendo una sequenza di variabili aleatorie $(X_1, X_2, \ldots, X_L)$, dove ciascuna variabile X_n rappresenta il byte alla posizione n del file.

Il primo byte della sequenza X_1 viene generato casualmente, selezionando uno stato iniziale i_0 nello spazio degli stati S.
A partire da questo stato iniziale, ogni byte successivo X_{n+1} viene generato campionando dalla distribuzione di probabilità condizionata associata allo stato corrente X_n, rappresentato dalla riga corrispondente della matrice di transizione P.

Formalmente, definito il byte corrente $X_n = i$, il byte successivo X_{n+1} viene generato come:

$$\mathbb{P}(X_{n+1} = j \mid X_n = i) = P_{ij}, \quad j \in S.$$

In questo modo ogni transizione tra byte avviene con una probabilità coerente con quelle apprese durante la fase di costruzione della matrice di transizione

Dal punto di vista del campionamento, la scelta del byte successivo avviene a partire dalla distribuzione discreta definita dalla riga $P_{i\cdot}$ della matrice di transizione.
A tal fine, si costruisce la funzione di distribuzione cumulativa (CDF) associata a questa riga, definita come:

$$F_i(j) = \sum_{k=0}^{j} P_{ik}, \quad j \in S.$$

Dato un numero casuale r estratto uniformemente nell'intervallo $[0, 1)$, il byte successivo è determinato come il più piccolo indice $j \in S$ tale che:

$$F_i(j) \geq r.$$

Per fare un esempio, si consideri una riga della matrice di transizione associata allo stato corrente i, limitata a un sottoinsieme di stati $S = \{0, 1, 2, 3\}$, con probabilità:

$$P_{i\cdot} = (0.1,\ 0.3,\ 0.4,\ 0.2).$$

La corrispondente funzione di distribuzione cumulativa risulta quindi:

$$F_i = (0.1,\ 0.4,\ 0.8,\ 1.0).$$

Supponendo di estrarre un numero casuale $r = 0.35$, il byte successivo è determinato come il più piccolo indice j tale che $F_i(j) \geq 0.35$. In questo caso, si ha $F_i(1) = 0.4 \geq 0.35$, pertanto il byte successivo sarà $X_{n+1} = 1$.

5.6 Ottimizzazione del Campionamento

Come descritto nella sezione precedente, il campionamento diretto da una distribuzione discreta può essere effettuato costruendo la funzione di distribuzione cumulativa e selezionando il primo stato che soddisfa la condizione $F_i(j) \geq r$. Tale approccio, sebbene corretto, richiede una complessità lineare $O(N)$ rispetto al numero di stati N.

In questo contesto, lo spazio degli stati S ha cardinalità pari al numero di possibili valori di byte, ovvero $S = 256$. Di conseguenza, il campionamento diretto non risulta computazionalmente costoso sul singolo file, tuttavia considerando

l'elevato numero di file da generare e la lunghezza di ciascun file, questa operazione può diventare un collo di bottiglia in termini di prestazioni. Pertanto l'ottimizzazione di questa fase può portare a significativi risparmi in termini di tempo di esecuzione complessivo.

Per risolvere questa problematica, si è scelto di ricorrere all'Alias Method, una tecnica che consente di effettuare il campionamento in tempo costante $O(1)$, una volta eseguita una fase di preprocessamento.

Il metodo si basa sulla trasformazione della distribuzione discreta originale in una rappresentazione equivalente che utilizza due strutture dati ausiliarie: un array di probabilità `prob` e un array di alias `alias`. Nel dettaglio, dato un insieme di N stati con probabilità associate $p_1, p_2, \ldots, p_N$, con $\sum_{i=1}^{N} p_i = 1$, l'Alias Method costruisce le tabelle `prob` e `alias` che permettono di campionare dalla distribuzione in tempo costante.

In fase di preprocessamento, le probabilità sono moltiplicate per N, ottenendo una distribuzione scalata $q_i = N \cdot p_i$ [27]. Successivamente, gli stati vengono suddivisi in due insiemi:

- stati con $q_i < 1$, detti *small*;
- stati con $q_i \geq 1$, detti *large*.

Si procede poi associando iterativamente uno stato $s \in \textit{small}$ a uno stato $l \in \textit{large}$, assegnando $\texttt{prob}[s] = q_s$ e $\texttt{alias}[s] = l$, e aggiornando $q_l \leftarrow q_l - (1 - q_s) = q_l + q_s - 1$.
Il processo viene ripetuto fino a esaurimento degli insiemi; gli stati rimanenti vengono completati impostando $\texttt{prob}[i] = 1$ e $\texttt{alias}[i] = i$, garantendo la correttezza della rappresentazione.

Una volta costruite le tabelle, il campionamento da una distribuzione discreta può essere effettuato in tempo costante: si seleziona uniformemente un indice $k \in \{1, \ldots, N\}$ e si estrae un numero casuale $r \in [0, 1)$. Se $r < \texttt{prob}[k]$, lo stato selezionato è k; altrimenti viene restituito lo stato $\texttt{alias}[k]$ [28].

Questo procedimento garantisce che la distribuzione degli stati campionati coincida con quella originale, pur richiedendo un numero costante di operazioni per ogni estrazione.

5.7 Implementazione del Sistema di Generazione

In questa sezione vengono descritti gli aspetti implementativi del sistema di generazione dei file. L'obiettivo è tradurre il modello teorico presentato nelle

sezioni precedenti in una pipeline concreta ed efficiente, in grado di produrre grandi quantità di file sintetici.

L'implementazione sviluppata segue la struttura generale delineata nelle sezioni precedenti: la generazione di file verosimili avviene attraverso un processo markoviano basato su matrici di transizione costruite a partire da file reali, con l'ottimizzazione del campionamento tramite l'Alias Method.

5.7.1 Struttura Generale

Il sistema di generazione è organizzato come una pipeline modulare, suddivisa in due fasi principali: una fase di preprocessamento *offline*, la quale si occupa della costruzione delle matrici di transizione, e una fase di generazione *online*, responsabile della produzione dei file.

La separazione tra le due fasi consente di ottimizzare il processo, permettendo di isolare le operazioni computazionalmente costose nella fase di preprocessamento, al fine di migliorare le prestazioni del sistema.

Come anticipato, la fase offline ha come obiettivo l'apprendimento delle distribuzioni di transizione tra byte a partire da file reali. Per ciascun formato di file considerato, vengono analizzati un insieme di file reali dai quali sono estratte le porzioni di file discusse in precedenza.

A partire da queste porzioni, vengono costruite le matrici di transizione secondo la metodologia descritta nella Sezione 5.4, seguita dalla fase di preprocessamento per l'Alias Method, come descritto nella Sezione 5.6.
Il risultato di questa fase consiste in strutture dati ausiliarie che permettono di effettuare il campionamento delle transizioni in tempo costante durante la generazione runtime.

Per quanto riguarda la fase online, il sistema utilizza esclusivamente i modelli discussi in precedenza.
Dato un numero prefissato di file da generare, si genera per ciascun file una sequenza di byte di lunghezza variabile, campionando iterativamente dalla catena di Markov mediante le tabelle alias.
Inoltre, fissato N il numero di file da generare, si generano $\frac{N}{4}$ file per ciascun formato considerato, ottenendo così una distribuzione bilanciata tra i vari formati.

La generazione è indipendente dal dataset originale e non richiede ulteriori operazioni di analisi sui file reali, rendendo il processo efficiente e facilmente scalabile. Le sequenze di byte generate vengono infine incapsulate nel formato di output desiderato, secondo le modalità specifiche di ciascun formato.

L'intera pipeline viene applicata in maniera analoga a tutti i formati considerati (PDF, JPG, DOCX e TXT). Le differenze tra i formati riguardano esclusivamente le modalità di estrazione delle porzioni strutturali e la fase di ricostruzione finale del file, mentre il nucleo del processo di generazione rimane invariato.

5.7.2 Costruzione delle Matrici di Markov

Come descritto precedentemente, la costruzione delle matrici di transizione avviene nella fase di preprocessamento offline.

Per prima cosa, è necessario costruire le matrici di conteggi a partire dai file reali.
In tal senso si presenta l'algoritmo seguente:

Algorithm 8: countBigrams

Input: matrice C, sequenza di byte B
for $n \leftarrow 1$ **to** $|B| - 1$ **do**
    $i \leftarrow B[n]$;
    $j \leftarrow B[n+1]$;
    $C_{ij} \leftarrow C_{ij} + 1$;
return C;

L'algoritmo `countBigrams` prende in input una sequenza di byte B, e aggiorna la matrice C incrementando i contatori corrispondenti alle coppie di byte osservate nella sequenza.

Una volta costruita la matrice di conteggi globale C, si procede alla costruzione della matrice di transizione P normalizzando ciascuna riga.

Algorithm 9: normalizeRows

Input: Matrice di conteggi $C \in \mathbb{N}^{256\times256}$
for $i \leftarrow 0$ **to** 255 **do**
    $s \leftarrow \sum_{j=0}^{255} C_{ij}$;
    if $s > 0$ **then**
        for $j \leftarrow 0$ **to** 255 **do**
            $P_{ij} \leftarrow \dfrac{C_{ij}}{s}$;
    else
        for $j \leftarrow 0$ **to** 255 **do**
            $P_{ij} \leftarrow \dfrac{1}{256}$;
return P;

La matrice di transizione così ottenuta rappresenta una stima delle probabilità di transizione della catena di Markov associata al formato considerato, ed è successivamente utilizzata per la fase di preprocessamento (costruzione delle tabelle alias).

Implementati gli algoritmi sopra descritti, si procede con la costruzione delle matrici di transizione per ciascun formato di file considerato, utilizzando i dataset di file reali descritti nel Capitolo 4.

Algorithm 10: BuildTransitionMatrix

Input: Dataset di file reali $\mathcal{D}$
Inizializza $C \leftarrow \mathbf{0}_{256\times256}$;
foreach *file* $f \in \mathcal{D}$ **do**
  $B \leftarrow$ estrai_byte_strutturali(f);
  if $|B| \geq 2$ **then**
    $C \leftarrow$ countBigrams(C, B);
$P \leftarrow$ normalizeRows(C);
return P;

5.7.3 Preprocessamento tramite Alias Method

Come descritto nella sezione precedente, l'Alias Method consente di effettuare il campionamento da una distribuzione discreta in tempo costante, a fronte di una fase di preprocessamento.

Tale preprocessamento viene applicato alle matrici di transizione markoviane costruite nella fase precedente. Poiché ogni riga della matrice di transizione P rappresenta una distribuzione di probabilità discreta sullo spazio degli stati S, è possibile costruire una tabella alias indipendente per ciascuna riga.

In dettaglio, il preprocessamento produce due matrici ausiliarie:

$$\text{Prob} \in [0,1]^{256\times256}, \qquad \text{Alias} \in \{0,\dots,255\}^{256\times256},$$

tali che la riga i di ciascuna matrice consente di campionare il byte successivo dato lo stato corrente i.

Algorithm 11: PreprocessingAliasMethod

Input: Matrice di transizione $P \in [0,1]^{256\times256}$
for $i \leftarrow 0$ **to** 255 **do**
  $(\text{Prob}_{i\cdot}, \text{Alias}_{i\cdot}) \leftarrow \text{AliasMethod}(P_{i\cdot})$;
return Prob, Alias;

5.7.4 Gestione della Lunghezza dei File

Un ulteriore aspetto da considerare, prima di concentrare l'attenzione sulla generazione runtime, riguarda la gestione della lunghezza dei file generati.
Questa infatti non è fissa ma si è preferito adottare una strategia che consenta di variare la lunghezza dei file generati, al fine di ottenere un insieme di file sintetici eterogeneo e, al contempo, mantenere contenuto il costo computazionale della fase di generazione.

Poiché il costo della generazione markoviana cresce linearmente con la lunghezza del file, una distribuzione uniforme delle lunghezze tenderebbe a sovrarappresentare file di grandi dimensioni, aumentando significativamente il tempo di esecuzione complessivo senza apportare un beneficio in termini di informazione strutturale.

Per questo motivo, si è scelto di adottare una distribuzione log-uniforme all'interno di un intervallo $[L_{\min}, L_{\max}]$, in modo da favorire la generazione di file più corti, pur consentendo la presenza di file di dimensioni maggiori.

Formalmente, la lunghezza L viene generata come:

$$L = \lfloor \exp(U) \rfloor , \quad \text{con } U \sim \mathcal{U}(\log L_{\min}, \log L_{\max}),$$

dove $\mathcal{U}(a, b)$ denota la distribuzione uniforme continua nell'intervallo $[a, b]$.

5.7.5 Scrittura dei File su Disco

Un ultimo aspetto implementativo prima di definire il generatore riguarda la scrittura dei file generati su disco.

In dettaglio, per i file di tipo TXT, la sequenza di byte generata viene scritta direttamente su disco, in quanto il formato non impone vincoli strutturali ulteriori.
Analogamente, per i file PDF e JPG, i byte generati vengono scritti direttamente in un file binario. In questi casi, la generazione mira a preservare la validità sintattica intervenendo solo su porzioni selezionate del formato.

Per quanto riguarda i file DOCX, la scrittura su disco richiede un passaggio aggiuntivo. Infatti, un file DOCX è in realtà un archivio ZIP contenente diversi file XML e risorse correlate. Pertanto, è necessario l'utilizzo di un template DOCX predefinito, nel quale si mantengono i file ausiliari e si sostituiscono esclusivamente i file XML contenenti il testo con la sequenza di byte generata.

Nel dettaglio, in fase di generazione le sequenze di byte prodotte dal modello markoviano vengono inserite esclusivamente all'interno dei file XML principali (`document.xml` e `styles.xml`), sostituendo il contenuto corrispondente nel

template. Il file DOCX finale viene quindi ricostruito scrivendo l'archivio ZIP risultante su disco.

Questo approccio preserva la struttura del template e vincola la generazione alle sole porzioni di contenuto selezionate, riducendo il rischio di corruzione del file.

5.7.6 Generazione Runtime

Una volta completata la fase di preprocessamento, è possibile procedere con la generazione runtime dei file, la quale si basa sul campionamento iterativo dalla catena di Markov utilizzando le tabelle alias costruite in precedenza.

Il campionamento di un singolo stato a partire dalle tabelle alias e la generazione iterativa della sequenza di byte vengono formalizzati nei seguenti algoritmi, utilizzati durante la fase di generazione runtime.

Algorithm 12: sampleAlias

Input: Vettori prob, alias
$i \sim \{0, \ldots, 255\}$;
$r \sim \mathcal{U}(0, 1)$;
if $r < prob[i]$ **then**
  return i;
else
  return alias$[i]$;

Algorithm 13: MarkovGenerate

Input: Matrici Prob, Alias, lunghezza L, stato iniziale $y_1 \in S$
$B[1] \leftarrow y_1$;
for $n \leftarrow 2$ **to** L **do**
  $y_n \leftarrow \text{sampleAlias}(\text{Prob}_{y_{n-1}\cdot}, \text{Alias}_{y_{n-1}\cdot})$;
  $B[n] \leftarrow y_n$;
return B;

Sulla base delle componenti descritte in precedenza, il processo completo di generazione runtime può essere riassunto nell'algoritmo seguente.

Algorithm 14: GenerateFiles

Input: N, Prob_f, Alias_f, templateDOCX, $[L_f^{\min}, L_f^{\max}]$
foreach *formato* f **do**
  for $k \leftarrow 1$ **to** N_f **do**
    $L \sim \text{LogUniform}(L_f^{\min}, L_f^{\max})$;
    $y_1 \sim \mathcal{U}\big(\{0, \ldots, 255\}\big)$;
    $B \leftarrow \text{MarkovGenerate}(\text{Prob}_f, \text{Alias}_f, L, y_1)$;
    if $f =$ `DOCX` **then**
      writeFile(B, f, templateDOCX);
    else
      writeFile(B, f);

5.8 Analisi del Costo Computazionale

In questa sezione viene analizzato il costo computazionale del sistema di generazione dei file, distinguendo tra la fase di preprocessamento offline e la fase di generazione runtime.

Poiché la fase offline viene eseguita una sola volta e non è soggetta a vincoli di tempo, l'analisi si concentra sul costo di memoria associato ai modelli generati.

5.8.1 Fase di Preprocessamento Offline

Per ciascun formato di file, si memorizzano due matrici ausiliarie per l'Alias Method, Prob e Alias, entrambe di dimensione 256×256.

Il costo di memoria è pertanto dominato da un numero costante di matrici, ognuna delle quali richiede 256×256 elementi, indipendentemente dal numero di file reali utilizzati per la costruzione del modello.

Dal punto di vista asintotico, il costo di memoria per ciascun formato risulta pari a $O(|S|^2)$, con $|S| = 256$.

5.8.2 Fase di Generazione Runtime

In questa fase, l'analisi si concentra sul costo temporale per la generazione dei file.

La generazione si basa sugli algoritmi `sampleAlias` e `MarkovGenerate` descritti in precedenza.

L'algoritmo `sampleAlias` esegue un numero costante di operazioni, pertanto il costo temporale di questo algoritmo è $O(1)$.

L'algoritmo `MarkovGenerate`, invece, genera una sequenza di byte di lunghezza L, invocando l'algoritmo `sampleAlias` una volta per ciascun byte successivo. Di conseguenza, il costo temporale complessivo per la generazione di una singola sequenza di byte di lunghezza L risulta essere:

$$O(L).$$

Per ciascun file generato, il costo totale dipende da:

- generazione della sequenza di byte tramite la catena di Markov, con costo $O(L)$;
- scrittura della sequenza di byte su disco.

Per i formati TXT, PDF e JPG, la scrittura su disco consiste nella scrittura diretta di L byte, e ha pertanto costo $O(L)$. Il costo totale per un singolo file di questi formati risulta quindi:

$$O(L).$$

Per i file DOCX, la scrittura su disco richiede la ricostruzione di un archivio ZIP a partire da un template predefinito. Tale operazione comporta: (i) la scrittura delle porzioni generate (file XML principali), con costo $O(L)$, e (ii) la riscrittura dei file statici del template, con costo $O(S_{\text{tpl}})$, dove S_{tpl} è la dimensione del template DOCX, costante per il formato. Di conseguenza, il costo complessivo per la generazione di un singolo file DOCX risulta pari a:

$$O(L + S_{\text{tpl}}),$$

che asintoticamente è lineare in L.

Poiché, per ciascun file, sia la fase di generazione sia la fase di scrittura su disco richiedono un tempo lineare nella lunghezza del file, il costo complessivo per singolo file è dato dalla somma:

$$O(L) + O(L) = O(L).$$

Date le considerazioni sopra esposte, si procede con l'analisi del costo complessivo.
Sia $N = \sum_f N_f$ il numero totale di file generati e sia

$$L = \frac{1}{N} \sum_f \sum_{k=1}^{N_f} L_k$$

la lunghezza media dei file. Allora il costo totale della fase di generazione runtime può essere riscritto come:

$$O\left(\sum_f \sum_{k=1}^{N_f} L_k \right) = O(N \cdot L).$$

5.9 Valutazione dei File Generati

Una volta descritto nel dettaglio il sistema di generazione dei file verosimili, è necessario valutare la qualità dei file prodotti.

L'obiettivo di questa fase è verificare in che misura i file generati con il modello markoviano risultino statisticamente distanti dai file completamente casuali, e, al contempo, sufficientemente simili ai file reali.

5.9.1 Distanza tra File Generati e File Random

Al fine di valutare se il metodo di generazione discusso produca file effettivamente distinguibili da sequenze casuali di byte, è stata condotta un'analisi analoga a quella descritta nella Sezione 4.1, sostituendo i file reali con i file generati dal modello.

In particolare, l'obiettivo di questa analisi è verificare che i file generati non presentino le caratteristiche tipiche dei file completamente randomici, ma mantengano una struttura statistica coerente con il formato di riferimento. Per garantire coerenza metodologica, vengono adottate le stesse rappresentazioni (N-gram con $n = 2$) e le medesime metriche di distanza e divergenza introdotte nel Capitolo 3 e già utilizzate per il confronto tra file reali e file random.

Per ciascun formato considerato il procedimento adottato per l'analisi è analogo a quello descritto nella Sezione 4.1.

Nelle figure sono riportati i risultati ottenuti per ciascun formato.

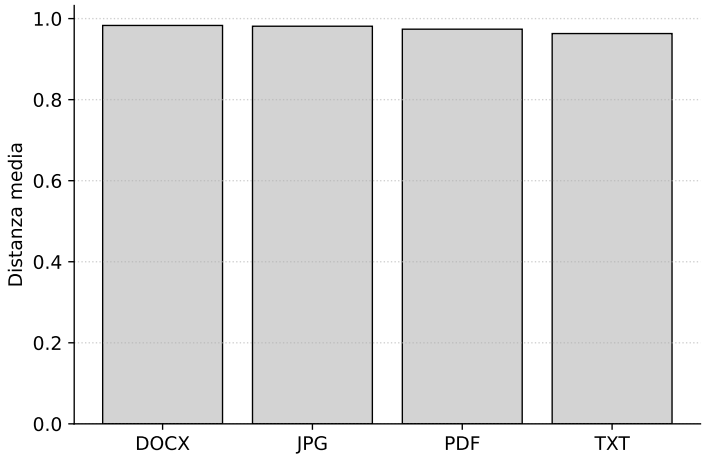

Figura 5.1: Distanza media tra file generati e file random calcolata mediante la JSD.

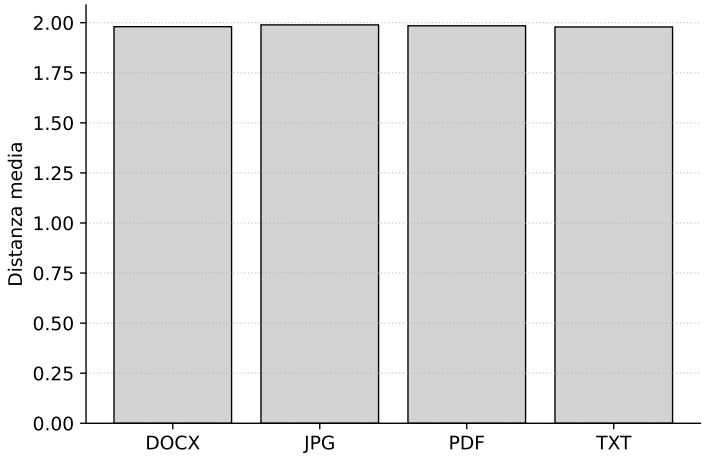

Figura 5.2: Distanza media tra file generati e file random calcolata mediante la Manhattan Distance.

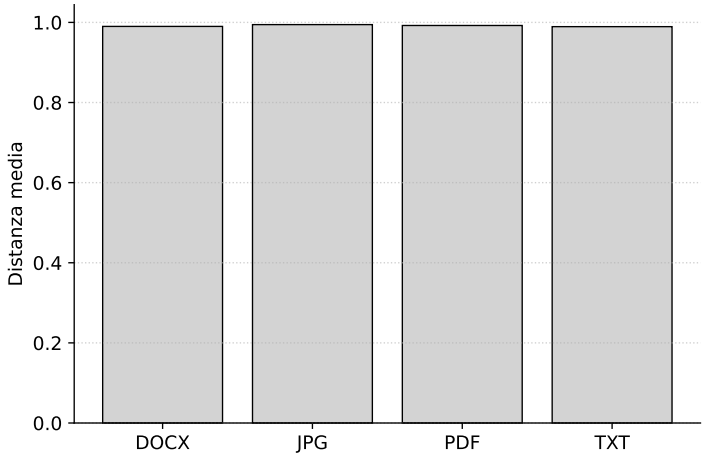

Figura 5.3: Distanza media tra file generati e file random calcolata mediante la TVD.

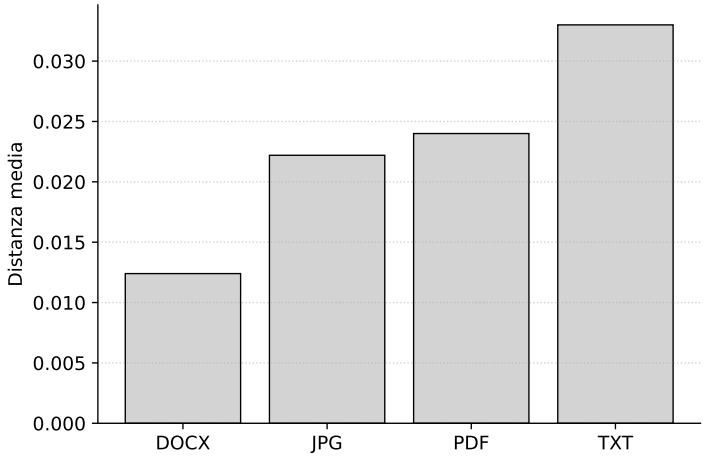

Figura 5.4: Distanza media tra file generati e file random calcolata mediante la Cosine Similarity.

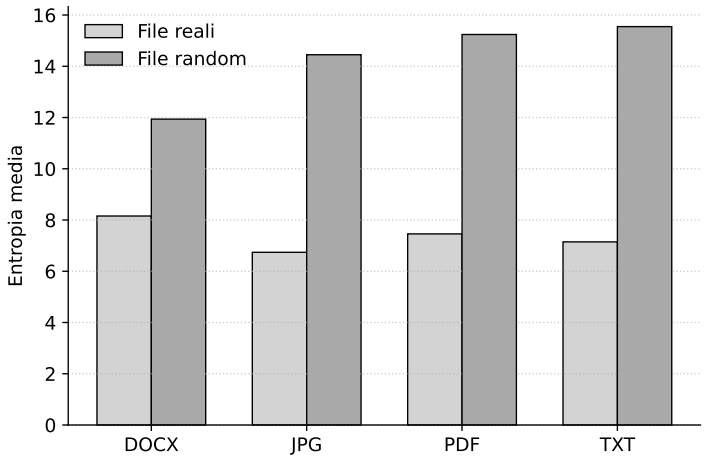

Figura 5.5: Entropia media tra file generati e file random.

Coerentemente con quanto fatto nella Sezione 4.1, anche in questo caso i valori degli istogrammi risultano concentrati in prossimità del massimo e le differenze tra i formati sono poco evidenti a livello visivo. Per facilitare il confronto diretto con i risultati della Sezione 4.1, si riporta quindi una tabella con i valori medi delle metriche per ciascun formato

Tabella 5.1: Statistiche medie per formato (generati-random)

Formato	JSD	TVD	L1	Cosine_sim	Entropy_random	Entropy_real
docx	0.9831	0.9900	1.9800	0.0124	11.9378	8.2319
jpg	0.9812	0.9945	1.9891	0.0222	14.4568	7.5339
pdf	0.9739	0.9923	1.9846	0.0240	15.2412	7.7467
txt	0.9631	0.9894	1.9788	0.0330	15.5626	8.5213

Dai risultati ottenuti emerge come i file generati risultino statisticamente distinti dai file completamente randomici per tutti i formati considerati e per tutte le metriche adottate.

Si osserva come le distanze medie tra file generati e file random siano simili a quelle osservate tra file reali e file random, come evidenziato dal confronto con i risultati della Sezione 4.1, suggerendo che i file generati mantengano una struttura statistica coerente con il formato di riferimento.

In dettaglio, le metriche di distanza e divergenza (JSD, TVD e L_1) assumono valori medi molto elevati per ciascun formato, con valori di JSD compresi approssimativamente tra 0.96 e 0.98 e valori di Manhattan distance prossimi al massimo teorico.

Analogamente, la Cosine Similarity mostra valori attorno allo zero per tutti i formati.

Anche l'analisi dell'entropia media conferma questa tendenza, con valori significativamente inferiori a quelli dei file randomici, e simili a quelli dei file reali.

In conclusione, questi risultati dimostrano che il metodo di generazione proposto è in grado di produrre file statisticamente distinguibili da sequenze casuali di byte. Questa evidenza costituisce un primo passo fondamentale verso la validazione della verosimiglianza statistica dei file generati, che verrà ulteriormente analizzata nella sezione successiva attraverso la valutazione tramite il detector.

5.9.2 Valutazione tramite il Detector

Dopo aver verificato che i file generati risultino distinti dai file completamente randomici, il prossimo passo consiste nel valutare se tali file possano essere considerati verosimili dal detector sviluppato nel Capitolo 4.

In questa fase, il detector viene utilizzato come un *oracolo di verosimiglianza*: un file viene considerato verosimile se il detector lo classifica come reale, secondo i criteri definiti in precedenza.

In questo senso, la valutazione tramite il detector consente di stabilire in modo oggettivo se i file prodotti dal modello di generazione risultino compatibili con la struttura statistica appresa dai file reali.

La valutazione è stata condotta utilizzando esclusivamente i centroidi e le soglie finali ottenute dall'aggregazione dei risultati di cross-validation, senza introdurre ulteriori fasi di addestramento o di ottimizzazione.

In dettaglio, sono stati generati 100 000 file artificiali tramite il modello markoviano, suddivisi equamente tra i quattro formati considerati.
Di questi, il detector ha classificato come reali il 96.80% (96 797) dei file totali, mentre il 3,20% (3203) è stato ritenuto non reali.

L'analisi non è stata condotta su una singola istanza del dataset generato. Al fine di valutare la stabilità dei risultati, l'esperimento è stato ripetuto più volte, generando in modo indipendente dataset di dimensioni differenti. In tutti i casi, le percentuali di file classificati come reali e la distribuzione delle metriche responsabili delle esclusioni hanno mostrato un comportamento sostanzialmente invariato.

L'analisi delle metriche responsabili della classificazione evidenzia come la maggior parte delle esclusioni è dovuta alla metrica di entropia, che da sola contribuisce alla classificazione come non reali di 3004 file.
Le altre metriche contribuiscono in misura minore: la metrica JSD esclude 189 file, la TVD 117 file, la Manhattan Distance 121 file mentre la Cosine Similarity 42 file.

Questo comportamento è coerente con quanto osservato nelle fasi di validazione del detector: la metrica di entropia risulta la più restrittiva e sensibile a variazioni nella distribuzione globale dei byte, soprattutto per quanto riguarda il formato JPG, mentre le metriche di distanza mostrano una maggiore tolleranza nei confronti delle variazioni locali introdotte dal processo di generazione.

Nel complesso, i risultati ottenuti indicano come il modello di generazione proposto sia in grado di produrre file che, nella maggior parte dei casi, soddisfino i criteri di verosimiglianza definiti dal detector.
La bassa percentuale di file considerati non reali suggerisce che il processo markoviano riesca a riprodurre in modo efficace le caratteristiche statistiche fondamentali dei file reali, evitando al contempo una replica diretta dei dati di partenza.

5.9.3 Valutazione tramite Google Magika

Magika, come descritto nella Sezione 4.9, è uno strumento basato sull'analisi statistica del contenuto binario dei file, sviluppato per l'identificazione del

formato di un file indipendentemente dalla sua estensione. Secondo Google, Magika presenta un'accuratezza del 100% nel riconoscere i formati di file.
Nel contesto di questo lavoro, Magika è stato utilizzato come strumento aggiuntivo al sistema di rilevamento per valutare la verosimiglianza dei file generati: se Magika è in grado di identificare correttamente l'estensione di un file generato, allora il file rispetta le caratteristiche statistiche del formato di riferimento.

Per eseguire questa valutazione, sono stati condotti diversi esperimenti: è stato valutato come Magika si comporti sui file reali (quelli utilizzati per i dataset), sui file completamente randomici generati da Ranflood, e sui file generati dal sistema presentato in questo lavoro.

I risultati ottenuti sono i seguenti:

Per quanto riguarda i file reali, questi sono stati classificati correttamente come appartenenti al formato previsto nella quasi totalità dei casi.

I file generati tramite Ranflood (circa 1800 per formato), sono stati invece classificati tutti come *Format file UNKNOWN*: infatti nella totalità dei casi Magika non è riuscito a classificare correttamente i formati dei file processati.

Nel caso dei file generati dal sistema, l'esperimento è stato condotto utilizzando 100 000 file, con 25 000 file per ciascun formato considerato in questo lavoro.

I risultati ottenuti sono riportati nel seguente grafico:

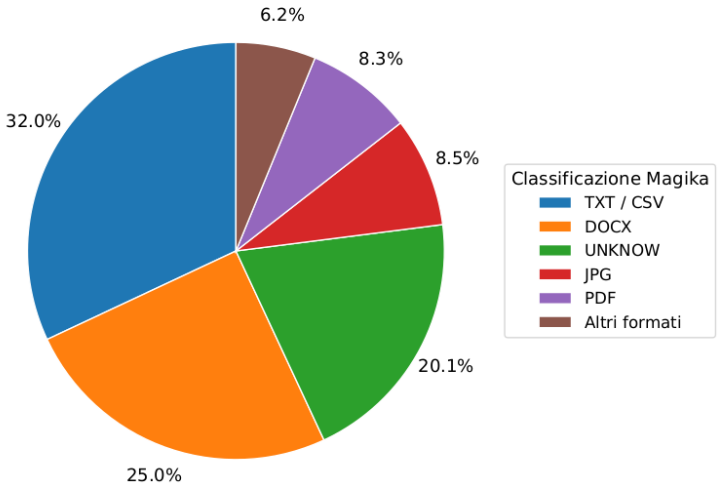

Figura 5.6: Classificazione dei file generati tramite Magika.

Dai risultati emerge che, per i formati DOCX e TXT, i file generati riescono a "ingannare" Magika, venendo correttamente classificati come appartenenti al formato corretto nella totalità dei casi.

Per quanto riguarda i file TXT, si è scelto di includere nella categoria testuale anche i file classificati come CSV. Durante l'esperimento, infatti, si è osservato come numerosi file TXT vengano classificati da Magika come file CSV, non solo nel caso dei file generati, ma anche nella valutazione dei file reali.

Questo comportamento è attribuibile alle caratteristiche strutturali condivise tra i due formati: molti file di testo presentano una distribuzione ricorrente di virgole, punti e separatori, elementi tipici anche dei file CSV. Di conseguenza, dal punto di vista statistico, le firme binarie dei file TXT e CSV risultano spesso difficili da distinguere in modo netto.

Per questo motivo, sia nell'analisi dei file reali sia in quella dei file generati, si è scelto di considerare congiuntamente le classificazioni TXT e CSV come appartenenti alla stessa categoria testuale.

Inoltre, con riferimento ai file TXT/CSV, dai risultati emerge come il numero di file classificati in questa categoria superi i 25 000 elementi. Questo suggerisce che anche file appartenenti ad altri formati vengano talvolta ricondotti a una struttura testuale dal classificatore.

Tale comportamento è imputabile alla natura intrinsecamente più flessibile e meno vincolata dei formati testuali, che possono essere approssimati da sequenze di byte con distribuzioni statistiche relativamente semplici. In presenza di contenuti privi di header rigidi o di strutture binarie fortemente caratterizzanti, Magika tende quindi a privilegiare una classificazione di tipo testuale.

Per quanto riguarda i formati PDF e JPG, su 25 000 file generati, Magika ha considerato corretti circa 8 500 file per ciascun formato.
Tale risultato è coerente con la maggiore complessità strutturale dei formati, che presentano vincoli sintattici più rigidi rispetto ai formati testuali e documentali. In particolare, entrambi i formati sono caratterizzati dalla presenza di header ben definiti, marker strutturali obbligatori e sezioni binarie ad alta entropia, elementi che non vengono esplicitamente modellati dal processo di generazione adottato in questo lavoro, basato esclusivamente su proprietà statistiche locali dei byte.

Di conseguenza, sebbene i file generati riescano in una frazione significativa dei casi a riprodurre firme statistiche compatibili con i formati di riferimento, l'assenza di una struttura sintattica pienamente conforme limita la capacità del classificatore di riconoscerli in modo affidabile. Il fatto che circa un terzo dei file venga comunque classificato correttamente indica tuttavia che il modello di generazione cattura alcune regolarità statistiche rilevanti, pur non garantendo una riproduzione completa della semantica di formato.

I risultati sopra riportati si riferiscono al modello di Magika con la modalità *ad alta confidenza* (high-confidence).

In questa modalità, Magika fornisce previsioni altamente precise ma con un richiamo (recall) inferiore, restituendo una classificazione solo quando il punteggio di confidenza supera soglie conservative (altrimenti può fornire un'etichetta generica).

Tuttavia, Magika offre una modalità best-guess, per la quale è possibile ottenere un richiamo maggiore, accettando anche previsioni con un punteggio di confidenza inferiore. In entrambi i casi, Magika associa a ogni previsione un punteggio di confidenza compreso tra 0 e 1 [29]

Nel caso in cui si considerassero anche i risultati ottenuti con la modalità best-guess, i file generati mostrerebbero percentuali di riconoscimento corretto superiori all'80% per i formati JPG e PDF, migliorando significativamente rispetto alla modalità high-confidence.
Per quanto riguarda i file completamente random invece, anche utilizzando questa modalità i risultati rimangono invariati, in quanto neanche con un livello di confidenza più basso Magika riesce a valutare correttamente i file.

Per concludere, il confronto con Magika evidenzia come il modello di generazione proposto sia in grado di riprodurre file che, pur non essendo identici ai file reali, presentano caratteristiche statistiche sufficientemente simili da essere riconosciuti correttamente in molti casi, soprattutto per i formati DOCX e TXT. Questo supporta l'efficacia del modello di generazione nel creare file verosimili.

5.9.4 Valutazione delle Prestazioni: Magika vs Detector

Un aspetto interessante da considerare riguarda il confronto tra le prestazioni del detector sviluppato in questo lavoro e quelle di Magika.
Questo confronto è particolarmente rilevante per determinare l'operabilità del sistema, soprattutto in scenari reali con grandi quantità di file.

Il confronto è stato effettuato in termini di tempo di elaborazione per file. Per Magika, è stata misurata la durata media del processo di classificazione di un file, mentre per il detector è stato misurato il tempo di calcolo delle metriche.

Strumento	**Tempo medio per file**	**Numero di file processati**
Magika	0.004490 secondi	100 000
Detector	0.001290 secondi	100 000

Tabella 5.2: Confronto della velocità tra Magika e il detector.

Dai risultati ottenuti, emerge una differenza significativa in termini di prestazioni tra Magika e il detector sviluppato in questo lavoro. In particolare, Magika richiede in media 0.004490 secondi per la classificazione di un singolo file, mentre il detector presenta un tempo medio pari a 0.001290 secondi per file.

Considerando un dataset di 100 000 file, questo si traduce in un tempo di elaborazione complessivo pari a circa 449 secondi (circa 7 minuti) per Magika, contro circa 129 secondi (circa 2 minuti) per il detector.

Questa differenza è coerente con la natura dei due approcci. Magika si basa su modelli di Machine Learning profondi, che richiedono un processo di inferenza più complesso per ciascun file, mentre il detector proposto utilizza rappresentazioni esplicite basate su N-gram e metriche di distanza computazionalmente più leggere.

È importante sottolineare che l'obiettivo di questo confronto non è quello di valutare quale strumento sia complessivamente migliore, ma di evidenziare come il detector sviluppato in questo lavoro risulti particolarmente adatto a scenari operativi che richiedono l'elaborazione di grandi quantità di file in tempi contenuti.

Nel complesso, il confronto delle prestazioni mostra che il detector proposto offre un buon compromesso tra efficacia nella valutazione della verosimiglianza e efficienza computazionale, risultando più rapido di Magika nel contesto specifico considerato, pur perseguendo un obiettivo differente.

5.9.5 Variabilità delle Dimensioni dei File Generati

Poiché la lunghezza dei file influisce direttamente sia sul costo computazionale del processo di generazione sia sulle rappresentazioni statistiche dei file, è stata analizzata la distribuzione delle dimensioni dei file generati dal modello markoviano.

L'obiettivo di questa analisi è verificare empiricamente l'effetto della strategia di generazione basata su una distribuzione log-uniforme, introdotta nella Sezione 5.7.4, e valutare se essa produca un insieme di file sufficientemente eterogeneo ma controllato.
I risultati ottenuti sono riportati nella tabella seguente.

Tabella 5.3: Statistiche sulla lunghezza dei file generati per ciascun formato.

Formato	**N**	**Media**	**Dev. Std.**	**Varianza**	$L_{\min}$	$L_{\max}$
DOCX	2500	8326.5	2544.1	6 472 297.9	5305	14505
JPG	2500	4004.6	2568.8	6 598 794.7	1024	10230
PDF	2500	4143.9	2634.6	6 941 337.3	1028	10235
TXT	2500	3176.6	2578.4	6 647 894.1	512	10223

Dall'analisi dei risultati emerge come, per tutti i formati considerati, le dimensioni dei file generati presentino una variabilità elevata ma controllata. In particolare, i valori di deviazione standard e varianza risultano comparabili tra i diversi formati, indicando un comportamento stabile del processo di generazione.

I valori osservati di $L_{\min}$ e $L_{\max}$ confermano inoltre il rispetto dei vincoli imposti nella fase di generazione: non vengono prodotti file eccessivamente piccoli, potenzialmente privi di informazione strutturale, né file di dimensioni anomale che comporterebbero un costo computazionale sproporzionato.

Le differenze nelle lunghezze medie tra i formati sono coerenti con gli intervalli $[L_{\min}, L_{\max}]$ adottati e riflettono le caratteristiche tipiche dei formati stessi. In particolare, i file DOCX presentano una lunghezza media maggiore rispetto agli altri formati, in quanto non sono soltanto contenitori di testo, ma includono strutture XML e risorse aggiuntive, come spiegato in precedenza. Mentre i file TXT risultano mediamente più contenuti, come atteso.

Nel complesso, la distribuzione delle lunghezze ottenuta conferma che la scelta di una distribuzione log-uniforme consente di evitare una concentrazione artificiale delle dimensioni attorno a un valore fisso, garantendo al contempo un'adeguata copertura dell'intervallo di generazione.

Analisi ripetute su diverse istanze di generazione, con cardinalità variabile tra 10 000 e 100 000 file, hanno mostrato valori statistici analoghi, suggerendo la robustezza e la riproducibilità del processo di generazione anche rispetto alla variabilità dimensionale dei file.

Capitolo 6

Conclusioni

Nel presente lavoro è stato progettato, implementato e validato un sistema per la generazione di file-esca casuali ma verosimili, finalizzato al supporto di tecniche di Data Flooding against Ransomware.
L'obiettivo principale non è stato la creazione di file semanticamente validi, ma piuttosto la produzione di file che risultino statisticamente compatibili con i file reali, evitando sia la replica diretta dei dati sensibili, sia la generazione di file completamente casuali.

A tal fine, è stata introdotta una definizione quantitativa di verosimiglianza statistica, basata su rappresentazioni esplicite del contenuto binario e su metriche di distanza e divergenza ampiamente utilizzate in letteratura.
Su queste basi è stato sviluppato un sistema di rilevamento capace di distinguere tra file reali e file randomici, mostrando prestazioni elevate in termini di accuratezza e tassi di errore contenuti.

Il sistema di generazione proposto si fonda su modellazioni markoviane del contenuto binario dei file reali, con un processo di campionamento sviluppato con l'obiettivo di ottimizzare l'efficienza computazionale.
Le scelte progettuali adottate, come la selezione delle porzioni di file da modellare e l'uso di una distribuzione log-uniforme per la lunghezza dei file generati, hanno permesso di bilanciare efficacemente la qualità dei file prodotti con i vincoli computazionali imposti dal contesto operativo.

La valutazione sperimentale ha mostrato che i file generati:

- risultano statisticamente distinti dai file completamente randomici;
- sono classificati come verosimili dal detector nella maggior parte dei casi;
- sono riconosciuti correttamente da strumenti esterni come Magika in una percentuale significativa di casi, confermando la coerenza delle firme statistiche apprese, soprattutto per i formati DOCX e TXT.

Il confronto prestazionale tra il detector e Magika ha inoltre evidenziato come l'approccio basato su metriche esplicite e soglie ottimizzate consenta un'elaborazione significativamente più rapida, rendendo il sistema particolarmente adatto a scenari operativi con grandi quantità di dati.
Questo risultato rafforza la scelta di non adottare modelli di Machine Learning profondi, privilegiando invece un approccio più leggero e interpretabile.

In conclusione, il lavoro svolto dimostra come sia possibile manipolare in modo controllato l'identità statistica di un file, generando contenuti artificiali che risultano credibili per analisi automatiche, senza compromettere l'efficienza del sistema.

6.1 Limitazioni e Validità dello Studio

La nozione di verosimiglianza adottata in questo lavoro è definita nello spazio delle caratteristiche considerate (ad es. statistiche su n-gram e porzioni strutturali dei file) e mira a riprodurre firme statistiche compatibili con i campioni reali. Tale criterio non implica correttezza semantica né garantisce piena conformità agli standard di parsing dei formati. Inoltre, l'efficacia dell'approccio rispetto a ransomware reali dipende dalle euristiche effettivamente implementate dagli attaccanti e richiede una validazione in ambiente controllato.

Inoltre, pur utilizzando dataset dell'ordine di alcune migliaia di file per formato, la variabilità osservata potrebbe non coprire l'intera eterogeneità presente in scenari reali.

Infine, le prestazioni possono variare tra formati: i risultati sono più solidi per quei formati in cui la struttura osservata presenta pattern ricorrenti, mentre per formati con maggiore complessità binaria e/o componenti ad alta entropia si osserva un margine di distanza residuo. L'uso di un modello markoviano di ordine 1 (bigrammi) privilegia l'efficienza computazionale, ma può non catturare dipendenze a più lungo raggio, che potrebbero essere rilevanti per alcuni formati.

6.2 Sviluppi Futuri

Una possibile direzione per futuri sviluppi di questo lavoro ricade nello studio di un approccio per il quale è possibile convertire un file di uno specifico formato in un altro formato, per riconverterlo successivamente.

Come visto in precedenza, molti ransomware tendono a ignorare file con caratteristiche strutturali atipiche usando euristiche basate su analisi statistiche.
In questo contesto, risulta pertanto interessante lo studio degli elementi che definiscono l'identità di un file, e in particolare in quale misura tali elementi possano essere modificati senza compromettere la riconversione.

Per fare un esempio, si può pensare di avere un file PDF contenente dati sensibili, come ad esempio password o chiavi crittografiche.
In questo scenario, l'obiettivo sarebbe quello di convertire il file PDF in un altro formato che sappiamo essere ignorato dai ransomware, ad esempio un file di log o file altamente entropici.
Questa conversione non ha lo scopo di distruggere il file, ma piuttosto di "mascherarlo", per poi poterlo riconvertire nel formato originale quando necessario.

Le rappresentazioni basate su N-gram e le metriche di distanza descritte in questo lavoro possono fornire un punto di partenza per questo tipo di analisi. In particolare, si potrebbe considerare efficace una conversione se la sua distribuzione risulta statisticamente dissimile da quella del formato originale.

L'esplorazione di questo approccio risulta dunque una strada verso nuove strategie contro le minacce ransomware, complementari rispetto alle tecniche di Data Flooding e backup tradizionali.

Un ulteriore sviluppo riguarda l'estensione del generatore markoviano a modelli di ordine superiore (ad es. ordine 2 o 3), al fine di catturare dipendenze a più lungo raggio nel contenuto binario. Rispetto al modello a bigrammi adottato in questo lavoro, n-grammi più lunghi potrebbero riprodurre pattern più ricchi e ridurre la distanza residua osservata per formati con maggiore complessità e/o componenti ad alta entropia. Tale estensione introduce tuttavia un trade-off tra qualità e costo: l'aumento dell'ordine comporta una crescita significativa dello spazio degli stati, con impatto su memoria e generazione, oltre a richiedere strategie adatte per gestire la sparsità dei conteggi. Un'analisi sperimentale sistematica potrebbe quantificare questi effetti confrontando le metriche di verosimiglianza e i costi runtime, identificando l'ordine minimo in grado di fornire benefici misurabili nei diversi formati.

Bibliografia

[1] H. Alshaikh, N. Ramadan, and H. Ahmed, "Ransomware prevention and mitigation techniques," *International Journal of Computer Applications*, vol. 177, no. 40, pp. 31–39, 2020.

[2] K. Cabaj and W. Mazurczyk, "Using software-defined networking for ransomware mitigation: The case of cryptowall," *IEEE Network*, vol. 30, no. 6, pp. 14–20, 2016.

[3] A. Zimba, Z. Wang, and L. Simukonda, "Towards data resilience: The analytical case of crypto ransomware data recovery techniques," *International Journal of Information Technology and Computer Science*, vol. 10, no. 1, pp. 40–51, 2018.

[4] M. Baykara and B. Sekin, "A novel approach to ransomware: Designing a safe zone system," in *Proceedings of the 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, 2018.

[5] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Comparison of entropy calculation methods for ransomware encrypted file identification," *Entropy*, vol. 24, no. 10, p. 1503, 2022.

[6] D. Berardi, S. Giallorenzo, A. Melis, S. Melloni, L. Onori, and M. Prandini, "Data flooding against ransomware: Concepts and implementations," *Computers & Security*, vol. 131, p. 103295, 2023.

[7] A. Kapoor, A. Gupta, R. Gupta, S. Tanwar, G. Sharma, and I. E. Davidson, "Ransomware detection, avoidance, and mitigation scheme: A review and future directions," *Sustainability*, vol. 14, no. 1, 2021.

[8] Z. A. Genç, G. Lenzini, and D. Sgandurra, "On deception-based protection against cryptographic ransomware," in *Proceedings of the IFIP International Conference on ICT Systems Security and Privacy Protection*, 2019.

[9] M. McDaniel and M. H. Heydari, "Content based file type detection algorithms," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS)*. IEEE, 2003.

[10] B. Li, Y. Zhang, J. Yao, and T. Yin, "Mdba: Detecting malware based on bytes n-gram with association mining," in *Proceedings of the 26th International Conference on Telecommunications (ICT)*. IEEE, 2019, pp. 227–232.

[11] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, no. 12, pp. 2721–2744, 2006.

[12] S. L. Garfinkel, "Digital forensics research: The next 10 years," *Digital Investigation*, vol. 7, pp. S64–S73, 2010.

[13] ——, "Digital forensics xml and the dfxml toolset," *Digital Investigation*, vol. 8, no. 3–4, pp. 161–174, 2012.

[14] J. van der Knijff, "Adobe portable document format," *Inventory of Long-Term Preservation Risks*, vol. 2, 2009.

[15] V. van der Meer, J. van den Bos, H. Jonker, and L. Dassen, "Problem solved: A reliable, deterministic method for jpeg fragmentation point detection," *Forensic Science International: Digital Investigation*, vol. 48, p. 301687, 2024.

[16] S. L. Garfinkel and J. J. Migletz, "New xml-based files: Implications for forensics," *IEEE Security & Privacy*, vol. 7, no. 2, pp. 38–44, 2009.

[17] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics (CSI-KDD)*, 2009.

[18] I. Sorokin, "Comparing files using structural entropy," *Journal of Computer Virology and Hacking Techniques*, vol. 7, no. 4, pp. 287–301, 2011.

[19] M. C. Amirani, M. Toorani, and M. Beheshti, "A new approach to content-based file type detection," in *Proceedings of the 13th IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2008.

[20] I. Ahmed, K. Lhee, and S. Hong, "Content-based file type identification using cosine similarity and a divide-and-conquer approach," *International Journal of Digital Content Technology and its Applications*, vol. 4, no. 5, pp. 33–42, 2010.

[21] SciPy Developers, "scipy.optimize.dual_annealing," https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.dual_annealing.html, 2024, accessed: 2025.

[22] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics Surveys*, vol. 4, pp. 40–79, 2010.

[23] S. L. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," in *Proceedings of the Digital Forensic Research Workshop (DFRWS)*, Montreal, Canada, 2009.

[24] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Napierone: A modern mixed file dataset alternative to govdocs1," *Forensic Science International: Digital Investigation*, vol. 40, p. 301330, 2022.

[25] Edinburgh Napier University, "Napierone mixed file dataset," 2021, available at https://registry.opendata.aws/napierone, accessed December 2025.

[26] M. Bejan, "15,000 gutenberg books," https://www.kaggle.com/datasets/mateibejan/15000-gutenberg-books, 2020, accessed: 2025.

[27] I. Darago. (2021, May) The alias method. Accessed: 2026-01-21. [Online]. Available: https://idarago.github.io/the-alias-method/

[28] A. J. Walker, "An efficient method for generating discrete random variables with general distributions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 253–256, 1977.

[29] Google Security Research. (2024) Magika — core concepts: Prediction modes. Accessed: 21 Jan 2026. [Online]. Available: https://securityresearch.google/magika/core-concepts/prediction-modes/