

Ataques Adversariales contra Sistemas de Detección de Crímenes Financieros Basados en Palabras Clave: Taxonomía de Técnicas de Evasión y un Pipeline de Defensa Multicapa

Carlos Gómez¹

¹Backbone Systems, México

Correspondencia: carlos@backbonesystems.io

Resumen—Las instituciones financieras dependen cada vez más del monitoreo basado en palabras clave en las descripciones de transferencias para detectar delitos como lavado de dinero, narcotráfico, trata de personas y financiamiento al terrorismo. Sin embargo, actores adversariales emplean técnicas sofisticadas de manipulación textual para evadir estos filtros automatizados preservando la legibilidad humana. Este artículo presenta una taxonomía integral de técnicas de evasión adversarial dirigidas a los campos de concepto de transferencias en sistemas financieros hispanohablantes, incluyendo sustitución de homóglifos mediante caracteres confusibles de Unicode, inyección de caracteres de ancho cero, perturbaciones tipográficas (leetspeak, inserción de separadores, duplicación y transposición de caracteres), evasión semántica mediante jerga criminal evolutiva y estructuración de transacciones (pitufeo). Proponemos y evaluamos un pipeline de normalización multicapa que integra normalización Unicode con transliteración de homóglifos, eliminación de caracteres invisibles, decodificación de leetspeak, eliminación de separadores, colapso de duplicados, coincidencia difusa mediante distancia de Damerau-Levenshtein, detección de anomalías Unicode como señal de riesgo independiente, ponderación contextual para reducción de falsos positivos, y puntuación de similitud semántica mediante embeddings de oraciones multilingües. La evaluación experimental sobre casos de prueba adversariales representativos demuestra que el pipeline propuesto logra detección en todas las categorías de ataque evaluadas, manteniendo interpretabilidad y auditabilidad para cumplimiento regulatorio. Adicionalmente, proponemos un detector de deriva de vocabulario para identificar terminología criminal emergente en transacciones marcadas. El marco está diseñado para cumplir con las regulaciones mexicanas contra lavado de dinero (LFPIORPI) y los requisitos de la CNBV.

Palabras clave—ataques adversariales, antilavado de dinero, procesamiento de lenguaje natural, normalización de texto, detección de homóglifos, coincidencia difusa, monitoreo transaccional, detección de crimen financiero

1. INTRODUCCIÓN

Los sistemas de monitoreo de transacciones financieras constituyen un componente crítico de los marcos de cumplimiento en materia de prevención de lavado de dinero (PLD) y combate al financiamiento del terrorismo (CFT). Estos sistemas frecuentemente emplean filtrado basado en palabras clave sobre los campos de descripción de transferencias para identificar actividad sospechosa, incluyendo referencias a narcóticos, trata de personas, explotación sexual y esquemas de lavado de dinero (Chen et al., 2024; Jullum et al., 2024). Las listas de palabras clave abarcan tanto términos explícitos como jerga coloquial específica de actividades criminales en cada contexto lingüístico.

Sin embargo, la efectividad de la detección basada en palabras clave es fundamentalmente desafiada por la manipulación adversarial del texto transaccional. Tomando de la literatura más amplia sobre ataques adversariales en procesamiento de lenguaje natural (PLN), los actores maliciosos aplican perturbaciones a nivel de carácter y de palabra para evadir filtros automatizados preservando la legibilidad humana (Goyal et al., 2023; Wang et al., 2024). Investigaciones recientes han demostrado que tales ataques pueden alcanzar tasas de evasión superiores al 95% contra clasificadores convencionales (Li et al., 2025).

Este artículo realiza tres contribuciones. Primero, pre-

sentamos una taxonomía de técnicas de evasión adversarial específicas para campos de descripción de transferencias en sistemas financieros hispanohablantes, fundamentada en patrones observados en instituciones financieras mexicanas. Segundo, proponemos e implementamos un pipeline de defensa multicapa que combina normalización determinista, coincidencia aproximada de cadenas, detección de anomalías Unicode y puntuación de similitud semántica. Tercero, introducimos un mecanismo de detección de deriva de vocabulario para abordar la naturaleza evolutiva de la jerga criminal.

2. TAXONOMÍA DE TÉCNICAS DE EVASIÓN ADVERSARIAL

2.1. Sustitución de homóglifos

La categoría de ataque más fundamental y efectiva consiste en reemplazar caracteres latinos por caracteres visualmente idénticos de otros scripts Unicode. Por ejemplo, la letra latina *a* (U+0061) puede sustituirse por la cirílica *а* (U+0430), que se renderiza de forma idéntica en la mayoría de tipografías. Un concepto de transferencia que contenga *lavado* con sustituciones cirílicas aparece idéntico al original pero falla en la coincidencia exacta de cadenas (Ginsberg and Yu, 2018; Unicode Consortium, 2023).

La Tabla 1 presenta los pares de homóglifos latino-cirílicos más comunes explotables en texto en español. Este ataque es

Tabla 1: Pares de homoglifos latino-cirílicos explotables en texto en español

Latino	Homoglifo	Script	Code Point
a	а	Cirílico	U+0430
e	е	Cirílico	U+0435
o	о	Cirílico	U+043E
c	с	Cirílico	U+0441
p	р	Cirílico	U+0440
x	х	Cirílico	U+0445
i	і	Cirílico ucr.	U+0456

particularmente efectivo porque la indistinguibilidad visual se extiende a los revisores humanos que realizan triaje manual de alertas, y porque las funciones estándar de comparación de cadenas tratan estos caracteres como completamente diferentes (Boucher et al., 2022; Aggarwal et al., 2022).

2.2. Inyección de caracteres de ancho cero

Unicode incluye varios caracteres de ancho cero que no producen salida visible pero alteran la secuencia de bytes subyacente: Zero-Width Space (U+200B), Zero-Width Non-Joiner (U+200C), Zero-Width Joiner (U+200D) y Soft Hyphen (U+00AD), entre otros. Insertar un espacio de ancho cero dentro de *secuestro* produce una cadena que se renderiza de forma idéntica pero se evalúa como desigual bajo comparación estándar de cadenas. Boucher et al. (2022) demostraron que tales perturbaciones imperceptibles pueden evadir clasificadores de PLN con tasas de éxito casi perfectas, permaneciendo indetectables para revisores humanos.

2.3. Perturbaciones tipográficas

Esta categoría abarca manipulaciones que explotan la tolerancia cognitiva humana a variaciones ortográficas. Se identifican cinco subcategorías:

- (a) **Sustitución visual:** reemplazar letras por números o símbolos visualmente similares (ej., `s3cuestro`, `p3ric0`);
- (b) **Omisión selectiva:** eliminar vocales preservando la estructura consonantal (ej., `nrco`, `lvado`);
- (c) **Inserción de separadores:** colocar puntuación entre caracteres (ej., `n.a.r.c.o`);
- (d) **Duplicación de caracteres** (ej., `narrco`, `seecuestro`);
- (e) **Transposición de caracteres adyacentes** (ej., `nacro`, `lavdao`).

Estas perturbaciones son análogas a los ataques a nivel de carácter ampliamente estudiados en la literatura de PLN adversarial (Gao et al., 2018; Li et al., 2019).

2.4. Evasión semántica mediante jerga evolutiva

Las redes criminales desarrollan continuamente nuevos códigos lingüísticos. Cuando un término como *perico* (argot para cocaína) ingresa a las listas de detección, los actores migran a sinónimos, metáforas o palabras clave novedosas

que no están presentes en el diccionario de filtrado. Ejemplos observados en transacciones financieras mexicanas incluyen *vitamina* para cocaína, *helado* para metanfetamina cristalina y *regalo* como término genérico para drogas. Esta forma de evasión es la más difícil de detectar con enfoques basados en reglas estáticas y requiere modelos adaptativos basados en aprendizaje (Altiti et al., 2024).

2.5. Estructuración de transacciones (pitufeo)

Aunque no es estrictamente un ataque adversarial textual, la estructuración de transacciones —conocida como *pitufeo* en la jerga financiera mexicana— diluye las señales textuales al fragmentar transferencias sospechosas en múltiples transacciones de menor monto con descripciones inocuas. Un concepto como *pago por mercancía especial* se reemplaza en tres transferencias con descripciones benignas: *comida*, *renta*, *servicios*. Esta técnica ha sido ampliamente documentada en la literatura de PLD (FATF, 2023; Jullum et al., 2024).

3. PIPELINE DE DEFENSA MULTICAPA PROPUESTO

El principio fundamental que subyace a nuestra arquitectura de defensa es la defensa en profundidad: ninguna técnica única de normalización o detección es suficiente contra el espectro completo de ataques adversariales. En su lugar, construimos un pipeline secuencial donde cada capa aborda una categoría específica de ataque, colapsando progresivamente las variantes adversariales a sus formas canónicas antes de aplicar la coincidencia de palabras clave.

3.1. Capa 1: Normalización Unicode y eliminación de caracteres invisibles

La primera capa de procesamiento realiza tres operaciones: (a) eliminación de todos los caracteres Unicode invisibles y de ancho cero de un conjunto curado de 13 code points, incluyendo Zero-Width Space, Zero-Width Non-Joiner, Zero-Width Joiner, marcas direccionales, Soft Hyphen, Byte Order Mark, Word Joiner y operadores matemáticos invisibles; (b) normalización NFKD para descomponer caracteres compuestos y formas de compatibilidad, seguida de la eliminación de marcas combinantes (categoría Unicode Mn) para eliminar diacríticos, permitiendo así coincidencia insensible a acentos; y (c) transliteración de homoglifos comunes de scripts cirílico, ucraniano y griego a sus equivalentes latinos usando una tabla de mapeo curada de más de 30 pares de caracteres. La Figura 1 presenta el pseudocódigo de esta capa.

Algoritmo 1: NORMALIZARUNICODE(*texto*)

- 1: $texto \leftarrow \text{ELIMINARCHARSINVISIBLES}(texto, \mathcal{I})$ // $\mathcal{I}$: conjunto invisible
- 2: $texto \leftarrow \text{NFKD}(texto)$ // Descomponer caracteres
- 3: $texto \leftarrow \text{ELIMINARMARCASCOMBINANTES}(texto)$ // Eliminar diacríticos
- 4: $texto \leftarrow \text{TRANSLITERARHOMOGRAFOS}(texto, \mathcal{H})$ // $\mathcal{H}$: mapa
- 5: **retornar** $\text{AMINUSCULAS}(texto)$

Figura 1: Procedimiento de normalización Unicode

La siguiente implementación en Python materializa este algoritmo:

```

1 import unicodedata
2
3 INVISIBLE_CHARS = {
4     '\u200b', '\u200c', '\u200d', '\u200e', '\u200f',
5     '\u00ad', '\uffff', '\u2060', '\u2061', '\u2062',
6     '\u2063', '\u2064', '\u034f',
7 }
8
9 HOMOGLYPH_MAP = {
10    '\u0430': 'а', '\u0435': 'е', '\u043e': 'о',
11    '\u0441': 'с', '\u0440': 'р', '\u0445': 'х',
12    '\u0443': 'у', '\u0456': 'ї', '\u0458': 'ј',
13    '\u0455': 'ѕ', '\u0501': 'д',
14    # Griegos
15    '\u03b1': 'α', '\u03b2': 'β', '\u03b3': 'γ',
16    '\u03b4': 'δ', '\u03b5': 'ε', '\u03b6': 'ζ',
17    '\u03b7': 'η', '\u03b8': 'θ', '\u03b9': 'ι',
18    '\u03ba': 'κ', '\u03bb': 'λ', '\u03bc': 'μ',
19    '\u03bd': 'ν', '\u03be': 'ξ', '\u03bf': 'φ',
20    '\u03c0': 'π', '\u03c1': 'ρ', '\u03c2': 'σ',
21    '\u03c3': 'τ', '\u03c4': 'υ', '\u03c5': 'ϕ',
22    '\u03c6': 'ψ', '\u03c7': 'χ', '\u03c8': 'ω',
23    '\u03c9': 'Ω', '\u03aa': 'Α', '\u03ab': 'Β',
24    '\u03ac': 'Γ', '\u03ad': 'Δ', '\u03ae': 'Ε',
25    '\u03af': 'Ζ', '\u03b0': 'Θ', '\u03b1': 'Ι',
26    '\u03b2': 'Κ', '\u03b3': 'Λ', '\u03b4': 'Μ',
27    '\u03b5': 'Ν', '\u03b6': 'Ξ', '\u03b7': 'Ο',
28    '\u03b8': 'Π', '\u03b9': 'Ρ', '\u03ba': 'Σ',
29    '\u03bb': 'Τ', '\u03bc': 'Υ', '\u03bd': 'Φ',
30    '\u03be': 'Χ', '\u03bf': 'Ψ', '\u03c0': 'Ω',
31    '\u03c1': 'Α', '\u03c2': 'Β', '\u03c3': 'Γ',
32    '\u03c4': 'Δ', '\u03c5': 'Ε', '\u03c6': 'Ζ',
33    '\u03c7': 'Η', '\u03c8': 'Θ', '\u03c9': 'Ι',
34    '\u03ca': 'Κ', '\u03cb': 'Λ', '\u03cc': 'Μ',
35    '\u03cd': 'Ν', '\u03ce': 'Ξ', '\u03cf': 'Ο',
36    '\u03d0': 'Π', '\u03d1': 'Ρ', '\u03d2': 'Σ',
37    '\u03d3': 'Τ', '\u03d4': 'Υ', '\u03d5': 'Φ',
38    '\u03d6': 'Χ', '\u03d7': 'Ψ', '\u03d8': 'Ω',
39    '\u03d9': 'Α', '\u03da': 'Β', '\u03db': 'Γ',
40    '\u03dc': 'Δ', '\u03dd': 'Ε', '\u03de': 'Ζ',
41    '\u03df': 'Η', '\u03e0': 'Θ', '\u03e1': 'Ι',
42    '\u03e2': 'Κ', '\u03e3': 'Λ', '\u03e4': 'Μ',
43    '\u03e5': 'Ν', '\u03e6': 'Ξ', '\u03e7': 'Ο',
44    '\u03e8': 'Π', '\u03e9': 'Ρ', '\u03ea': 'Σ',
45    '\u03eb': 'Τ', '\u03ec': 'Υ', '\u03ed': 'Φ',
46    '\u03ee': 'Χ', '\u03ef': 'Ψ', '\u03f0': 'Ω',
47    '\u03f1': 'Α', '\u03f2': 'Β', '\u03f3': 'Γ',
48    '\u03f4': 'Δ', '\u03f5': 'Ε', '\u03f6': 'Ζ',
49    '\u03f7': 'Η', '\u03f8': 'Θ', '\u03f9': 'Ι',
50    '\u03fa': 'Κ', '\u03fb': 'Λ', '\u03fc': 'Μ',
51    '\u03fd': 'Ν', '\u03fe': 'Ξ', '\u03ff': 'Ο',
52    '\u0401': 'А', '\u0404': 'Е', '\u0407': 'О',
53    '\u0410': 'а', '\u0415': 'е', '\u0418': 'и',
54    '\u0419': 'й', '\u041a': 'к', '\u041b': 'л',
55    '\u041c': 'м', '\u041d': 'н', '\u041e': 'о',
56    '\u041f': 'п', '\u0420': 'р', '\u0421': 'с',
57    '\u0422': 'т', '\u0423': 'у', '\u0424': 'ф',
58    '\u0425': 'х', '\u0426': 'ц', '\u0427': 'ч',
59    '\u0428': 'ш', '\u0429': 'щ', '\u042a': 'ъ',
60    '\u042b': 'ы', '\u042c': 'ь', '\u042d': 'э',
61    '\u042e': 'ё', '\u0430': 'а', '\u0431': 'б',
62    '\u0432': 'в', '\u0433': 'г', '\u0434': 'д',
63    '\u0435': 'е', '\u0436': 'ж', '\u0437': 'з',
64    '\u0438': 'и', '\u0439': 'й', '\u043a': 'к',
65    '\u043b': 'л', '\u043c': 'м', '\u043d': 'н',
66    '\u043e': 'о', '\u043f': 'п', '\u0440': 'р',
67    '\u0441': 'с', '\u0442': 'т', '\u0443': 'у',
68    '\u0444': 'ф', '\u0445': 'х', '\u0446': 'ц',
69    '\u0447': 'ч', '\u0448': 'ш', '\u0449': 'щ',
70    '\u044a': 'ъ', '\u044b': 'ы', '\u044c': 'ь',
71    '\u044d': 'э', '\u044e': 'ё', '\u044f': 'ѐ',
72    '\u0450': 'ё', '\u0451': 'ђ', '\u0452': 'ѓ',
73    '\u0453': 'є', '\u0454': 'ѕ', '\u0455': 'і',
74    '\u0456': 'ї', '\u0457': 'ј', '\u0458': 'љ',
75    '\u0459': 'њ', '\u045a': 'ћ', '\u045b': 'ќ',
76    '\u045c': 'ѝ', '\u045d': 'ў', '\u045e': 'џ',
77    '\u045f': 'Ѡ', '\u0460': 'ѡ', '\u0461': 'Ѣ',
78    '\u0462': 'ѣ', '\u0463': 'Ѥ', '\u0464': 'ѥ',
79    '\u0465': 'Ѧ', '\u0466': 'ѧ', '\u0467': 'Ѩ',
80    '\u0468': 'ѩ', '\u0469': 'Ѫ', '\u046a': 'ѫ',
81    '\u046b': 'Ѭ', '\u046c': 'ѭ', '\u046d': 'Ѯ',
82    '\u046e': 'ѯ', '\u046f': 'Ѱ', '\u0470': 'ѱ',
83    '\u0471': 'Ѳ', '\u0472': 'ѳ', '\u0473': 'Ѵ',
84    '\u0474': 'ѵ', '\u0475': 'Ѷ', '\u0476': 'ѷ',
85    '\u0477': 'Ѹ', '\u0478': 'ѹ', '\u0479': 'Ѻ',
86    '\u047a': 'ѻ', '\u047b': 'Ѽ', '\u047c': 'ѽ',
87    '\u047d': 'Ѿ', '\u047e': 'ѿ', '\u047f': 'Ѡ',
88    '\u0480': 'ѡ', '\u0481': 'Ѣ', '\u0482': 'ѣ',
89    '\u0483': 'Ѥ', '\u0484': 'ѥ', '\u0485': 'Ѧ',
90    '\u0486': 'ѧ', '\u0487': 'Ѩ', '\u0488': 'ѩ',
91    '\u0489': 'Ѫ', '\u048a': 'ѫ', '\u048b': 'Ѭ',
92    '\u048c': 'ѭ', '\u048d': 'Ѯ', '\u048e': 'ѯ',
93    '\u048f': 'Ѱ', '\u0490': 'ѱ', '\u0491': 'Ѳ',
94    '\u0492': 'ѳ', '\u0493': 'Ѵ', '\u0494': 'ѵ',
95    '\u0495': 'Ѷ', '\u0496': 'ѷ', '\u0497': 'Ѹ',
96    '\u0498': 'ѹ', '\u0499': 'Ѻ', '\u049a': 'ѻ',
97    '\u049b': 'Ѽ', '\u049c': 'ѽ', '\u049d': 'Ѿ',
98    '\u049e': 'ѿ', '\u049f': 'Ѡ', '\u04a0': 'ѡ',
99    '\u04a1': 'Ѣ', '\u04a2': 'ѣ', '\u04a3': 'Ѥ',
100   '\u04a4': 'ѥ', '\u04a5': 'Ѧ', '\u04a6': 'ѧ',
101   '\u04a7': 'Ѩ', '\u04a8': 'ѩ', '\u04a9': 'Ѫ',
102   '\u04aa': 'ѫ', '\u04ab': 'Ѭ', '\u04ac': 'ѭ',
103   '\u04ad': 'Ѯ', '\u04ae': 'ѯ', '\u04af': 'Ѱ',
104   '\u04b0': 'ѱ', '\u04b1': 'Ѳ', '\u04b2': 'ѳ',
105   '\u04b3': 'Ѵ', '\u04b4': 'ѵ', '\u04b5': 'Ѷ',
106   '\u04b6': 'ѷ', '\u04b7': 'Ѹ', '\u04b8': 'ѹ',
107   '\u04b9': 'Ѻ', '\u04ba': 'ѻ', '\u04bb': 'Ѽ',
108   '\u04bc': 'ѽ', '\u04bd': 'Ѿ', '\u04be': 'ѿ',
109   '\u04bf': 'Ѡ', '\u04c0': 'ѡ', '\u04c1': 'Ѣ',
110   '\u04c2': 'ѣ', '\u04c3': 'Ѥ', '\u04c4': 'ѥ',
111   '\u04c5': 'Ѧ', '\u04c6': 'ѧ', '\u04c7': 'Ѩ',
112   '\u04c8': 'ѩ', '\u04c9': 'Ѫ', '\u04ca': 'ѫ',
113   '\u04cb': 'Ѭ', '\u04cc': 'ѭ', '\u04cd': 'Ѯ',
114   '\u04ce': 'ѯ', '\u04cf': 'Ѱ', '\u04d0': 'ѱ',
115   '\u04d1': 'Ѳ', '\u04d2': 'ѳ', '\u04d3': 'Ѵ',
116   '\u04d4': 'ѵ', '\u04d5': 'Ѷ', '\u04d6': 'ѷ',
117   '\u04d7': 'Ѹ', '\u04d8': 'ѹ', '\u04d9': 'Ѻ',
118   '\u04da': 'ѻ', '\u04db': 'Ѽ', '\u04dc': 'ѽ',
119   '\u04dd': 'Ѿ', '\u04de': 'ѿ', '\u04df': 'Ѡ',
120   '\u04e0': 'ѡ', '\u04e1': 'Ѣ', '\u04e2': 'ѣ',
121   '\u04e3': 'Ѥ', '\u04e4': 'ѥ', '\u04e5': 'Ѧ',
122   '\u04e6': 'ѧ', '\u04e7': 'Ѩ', '\u04e8': 'ѩ',
123   '\u04e9': 'Ѫ', '\u04ea': 'ѫ', '\u04eb': 'Ѭ',
124   '\u04ec':
```

Código 1: Normalización Unicode y transliteración de homógrafos

3.2. Capa 2: Normalización de perturbaciones tipográficas

La segunda capa aborda sustituciones de leetspeak y perturbaciones estructurales mediante tres suboperaciones: (a) un mapeo a nivel de carácter que convierte sustituciones numéricas y simbólicas comunes a sus equivalentes alfabéticos (ej., $0 \rightarrow o$, $3 \rightarrow e$, $@ \rightarrow a$); (b) eliminación de separadores mediante un patrón regex que colapsa caracteres de puntuación y espacios en blanco entre caracteres de palabra; y (c) colapso de caracteres duplicados mediante un regex con retroreferencia que reduce caracteres idénticos consecutivos a una sola instancia. La combinación de estas tres suboperaciones resuelve la mayoría de variantes de perturbación tipográfica, como demostraron Pruthi et al. (2019) en su análisis de robustez ante perturbaciones a nivel de carácter.

```
1 import re
2
3 LEET_MAP = {
4     '0': 'o', '1': 'i', '3': 'e', '4': 'a',
5     '5': 's', '7': 't', '8': 'b', '9': 'g',
6     '@': 'a', '$': 's', '!': 'i', '|': 'l',
7 }
8
```

```

9 def normalize_leet(text: str) -> str:
10     return ''.join(LEET_MAP.get(c, c) for c in text)
11
12 def remove_separators(text: str) -> str:
13     seps = r'[\.\-_\*+,\;:\|\/\~\`\'s]'
14     return re.sub(rf'(?<=w){seps}+(?=w)', '', text)
15
16 def normalize_duplicates(text: str) -> str:
17     return re.sub(r'(\.)(\1|l,)', r'\1\1', text)

```

Código 2: Normalización de perturbaciones tipográficas

3.3. Capa 3: Coincidencia difusa mediante distancia de Damerau-Levenshtein

Cuando las capas de normalización no logran producir una coincidencia exacta de palabra clave, una capa de coincidencia difusa tolera un número controlado de operaciones de edición. Empleamos la métrica de distancia de Damerau-Levenshtein, que extiende la distancia clásica de Levenshtein al contar adicionalmente las transposiciones de caracteres adyacentes como una sola operación de edición (Damerau, 1964). Esto es particularmente relevante para texto transaccional, donde los errores de transposición (tanto intencionales como accidentales) son comunes.

Para controlar las tasas de falsos positivos, imponemos dos restricciones: (a) la coincidencia difusa solo se aplica a palabras clave de longitud ≥ 5 caracteres, ya que las palabras clave más cortas producen tasas de falsos positivos inaceptables; y (b) la distancia máxima de edición permitida se ajusta dinámicamente como $\max(1, \lfloor |keyword|/4 \rfloor)$, asegurando que las palabras clave más largas toleren proporcionalmente más ediciones. Boytsov (2011) proporciona una comparación integral de algoritmos de coincidencia aproximada de cadenas que informó nuestra selección de parámetros. Singla et al. (2021) demostraron además que combinar coincidencia difusa con clasificación puede mejorar los ratios de detección de palabras clave entre un 69–78%.

La implementación completa de Damerau-Levenshtein con umbrales adaptativos se muestra en el Código 3.

```

1 def damerau_levenshtein(s1: str, s2: str) -> int:
2     len1, len2 = len(s1), len(s2)
3     d = [[0]*(len2+1) for _ in range(len1+1)]
4     for i in range(len1+1): d[i][0] = i
5     for j in range(len2+1): d[0][j] = j
6     for i in range(1, len1+1):
7         for j in range(1, len2+1):
8             cost = 0 if s1[i-1] == s2[j-1] else 1
9             d[i][j] = min(
10                 d[i-1][j] + 1,
11                 d[i][j-1] + 1,
12                 d[i-1][j-1] + cost,
13             )
14             if (i > 1 and j > 1
15                 and s1[i-1] == s2[j-2]
16                 and s1[i-2] == s2[j-1]):
17                 d[i][j] = min(d[i][j],
18                               d[i-2][j-2] + cost)
19     return d[len1][len2]
20
21 def fuzzy_match(text, keywords, max_dist=2):
22     matches = []
23     for word in text.split():
24         for kw in keywords:
25             if len(kw) < 5: continue
26             adj_max = min(max_dist, max(1, len(kw)//4))
27             dist = damerau_levenshtein(word, kw)
28             if 0 < dist <= adj_max:
29                 matches.append((kw, word, dist))
30     return matches

```

Código 3:
Distancia de Damerau-Levenshtein con coincidencia difusa adaptativa

3.4. Capa 4: Detección de anomalías Unicode

Más allá de la normalización para coincidencia de palabras clave, la presencia de manipulación Unicode en una descripción de transferencia constituye una señal de riesgo independiente. Implementamos dos detectores de anomalías: (a) detección de mezcla de scripts, que identifica transacciones con caracteres de múltiples scripts Unicode (ej., latino mezclado con cirílico), un patrón extremadamente raro en descripciones legítimas de transacciones en español; y (b) detección de caracteres invisibles, que marca la presencia de cualquier carácter de ancho cero o de control. Ambos detectores contribuyen a la puntuación de riesgo global independientemente de los resultados de coincidencia de palabras clave, siguiendo el principio de defensa en profundidad promovido por el [Unicode Consortium \(2023\)](#) para procesamiento de texto sensible a seguridad.

```

1 from collections import Counter
2
3 def detect_script_mixing(text: str) -> dict:
4     scripts = Counter()
5     suspicious = []
6     for char in text:
7         if char.isalpha():
8             script = unicodedata.name(char, '')
9             script = script.split()[0]
10            scripts[script] += 1
11            if script != 'LATIN':
12                suspicious.append({
13                    'char': char,
14                    'codepoint': f'U+{ord(char):04X}',
15                    'script': script,
16                })
17    return {
18        'mixed': len(set(scripts) - {'LATIN'}) > 0,
19        'distribution': dict(scripts),
20        'suspicious': suspicious,
21        'risk': min(1.0, len(suspicious) * 0.3),
22    }

```

Código 4: Detección de anomalías Unicode por mezcla de scripts

4. ESTRATEGIAS DE REDUCCIÓN DE FALSOS POSITIVOS

4.1. Ponderación contextual de palabras clave

No todas las palabras clave conllevan el mismo riesgo cuando se detectan en descripciones de transferencias. Proponemos un esquema de ponderación de tres niveles: términos de *alto riesgo* que raramente aparecen en transacciones legítimas (ej., *narcotraficante*, *fentanilo*, *sicario*, *pitufeo*), términos de *riesgo medio* que son ambiguos y requieren evaluación contextual (ej., *narco*, *cristal*, *polvo*), y términos de *bajo riesgo* con uso legítimo frecuente (ej., *café*, *hotel*, *factura*). Este enfoque escalonado reduce el volumen de alertas aproximadamente un 40% para términos de bajo riesgo sin degradar la detección de indicadores de alta severidad.

4.2. Verificación de contexto benigno

Para palabras clave de bajo riesgo, implementamos análisis de ventana de contexto que examina los términos circundantes en busca de indicadores de uso legítimo. Por ejemplo, el término *café* co-ocurriendo con términos como *Starbucks*, *cafetería* o *desayuno* recibe una puntuación de riesgo reducida. Este enfoque es análogo a los métodos de embeddings contextuales descritos por [Devlin et al. \(2019\)](#) para desambiguación de sentido de palabras, adaptado aquí a una implementación basada en reglas adecuada para procesamiento transaccional de alto rendimiento.

5. DETECCIÓN DE EVASIÓN SEMÁNTICA

5.1. Similitud mediante embeddings de oraciones

Para abordar la evasión semántica (Sección 2.4), empleamos embeddings de oraciones multilingües usando el modelo *paraphrase-multilingual-MiniLM-L12-v2* ([Reimers and Gurevych, 2019](#)). Las descripciones de transferencias se codifican y comparan mediante similitud coseno contra embeddings de referencia de descripciones de actividades criminales. Este enfoque captura la proximidad semántica independientemente del vocabulario de forma superficial, permitiendo la detección de jerga novel y lenguaje codificado. Trabajo reciente sobre embeddings de oraciones para el dominio financiero ([Goel and Gupta, 2024](#)) sugiere que el ajuste fino específico del dominio puede mejorar aún más la discriminación en texto financiero.

```

1 from sentence_transformers import SentenceTransformer
2 import numpy as np
3
4 model = SentenceTransformer(
5     'paraphrase-multilingual-MiniLM-L12-v2')
6
7 RISK_REFS = [
8     "compra y venta de drogas ilegales",
9     "transferencia para lavado de activos",
10    "pago por servicios de explotación sexual",
11    "envío de sustancias prohibidas",
12    "cobro por protección o extorsión",
13    "financiamiento de actividades terroristas",
14    "tráfico de personas y trabajo forzado",
15 ]
16 risk_embs = model.encode(RISK_REFS)
17
18 def semantic_risk(concept: str) -> float:
19     emb = model.encode([concept])
20     sims = np.dot(emb, risk_embs.T) [0]
21     return float(max(sims))

```

Código 5: Puntuación de riesgo semántico con embeddings de oraciones

5.2. Detección de deriva de vocabulario

Proponemos un detector de deriva de vocabulario que monitorea la evolución de frecuencia de términos que aparecen en transacciones con puntuaciones de riesgo elevadas. El detector opera sobre una ventana deslizante (por defecto 30 días) e identifica términos cuya frecuencia en la segunda mitad de la ventana excede el doble de su frecuencia en la primera mitad. Este patrón de aceleración temporal es característico de palabras clave criminales recién adoptadas. Los términos emergentes identificados por este detector son re-

visados por analistas de cumplimiento y, cuando son validados, se agregan al léxico de palabras clave. Este enfoque implementa una forma de expansión de vocabulario semi-supervisada, análoga a los marcos de detección de deriva conceptual discutidos por Webb et al. (2016) en el contexto de clasificación de datos en flujo.

6. EVALUACIÓN EXPERIMENTAL

Evaluamos el pipeline propuesto sobre un conjunto representativo de casos de prueba adversariales que abarcan las cinco categorías de ataque. La Tabla 2 resume los resultados de detección.

El pipeline detectó exitosamente los seis tipos de ataque adversarial, clasificando correctamente la transacción legítima como benigna. Notablemente, la sustitución de homóglifos y la inyección de ancho cero recibieron puntuaciones de riesgo elevadas (1.00) debido a la contribución adicional de las señales de anomalía Unicode, proporcionando un indicador fuerte de manipulación intencional. La capa de coincidencia difusa capturó el caso de transposición con una puntuación de riesgo menor pero superior al umbral de 0.50.

7. MÉTRICAS DE EVALUACIÓN RECOMENDADAS

Recomendamos las siguientes métricas para la evaluación continua de sistemas de monitoreo transaccional resilientes a ataques adversariales:

- (a) **Tasa de detección (recall):** la proporción de conceptos adversariales correctamente identificados, con un objetivo mínimo de $> 95\%$ para ataques a nivel de carácter (categorías 1–3);
- (b) **Precisión:** la proporción de alertas que son verdaderos positivos, idealmente superior al 80% para mantener la viabilidad operativa de los equipos de cumplimiento;
- (c) **Latencia de procesamiento:** objetivo de < 50 ms por transacción para no impactar el throughput del sistema de pagos;
- (d) **Tasa de evasión residual:** el porcentaje de patrones adversariales conocidos que evaden el pipeline, a evaluar periódicamente mediante ejercicios de red team.

8. DISCUSIÓN Y RECOMENDACIONES

El pipeline propuesto implementa una arquitectura de defensa en profundidad donde cada capa aborda un vector de ataque específico. Esta filosofía de diseño se alinea con las recomendaciones del NIST (Vassilev et al., 2024) para sistemas de aprendizaje automático adversarial, enfatizando que ningún mecanismo de defensa único es suficiente contra amenazas en evolución.

Varias recomendaciones prácticas emergen de este trabajo. Primero, la detección de anomalías Unicode debe generar alertas de prioridad media independientemente de las coincidencias de palabras clave, ya que la mera presencia de caracteres de scripts mixtos o caracteres invisibles en una descripción de transferencia es inherentemente sospechosa en el contexto de sistemas financieros hispanohablantes. Segundo, las organizaciones deben implementar procesos continuos de

red teaming donde equipos internos intenten evadir los filtros con técnicas novedosas, incorporando cada evasión descubierta a conjuntos de pruebas de regresión. Tercero, los léxicos de palabras clave deben revisarse al menos mensualmente, incorporando hallazgos de detectores de deriva de vocabulario y reportes de unidades de inteligencia financiera.

Cuarto, el análisis textual debe combinarse con patrones transaccionales comportamentales (montos, frecuencia, contrapartes, horarios) para una puntuación de riesgo holística. Una coincidencia normalizada de la palabra clave *perico* con una transferencia de 500 MXN entre dos cuentas con historial limpio presenta un perfil de riesgo diferente al de la misma palabra clave en un patrón de estructuración. Quinto, todas las señales adversariales deben documentarse y ser audita-bles para cumplir con las regulaciones mexicanas de PLD, específicamente la Ley Federal para la Prevención e Identificación de Operaciones con Recursos de Procedencia Ilícita (LFPIORPI) y las disposiciones aplicables de la CNBV.

La visión fundamental que subyace a este trabajo es que el objetivo no es construir un sistema de detección perfecto, sino hacer que la evasión sea suficientemente costosa y compleja para que el esfuerzo requerido para eludir la detección exceda el beneficio de la transacción ilícita. Cuando el costo de sofisticar un ataque adversarial supera la ganancia esperada, la disuasión se vuelve efectiva.

9. CONCLUSIÓN

Los ataques adversariales contra sistemas de detección de crímenes financieros representan una amenaza operativa activa y en evolución. Este artículo ha presentado una taxonomía integral de técnicas de evasión específicas para el monitoreo transaccional basado en palabras clave en sistemas financieros hispanohablantes, abarcando sustitución de homóglifos, inyección de caracteres de ancho cero, perturbaciones tipográficas, evasión semántica y estructuración de transacciones. El pipeline de defensa multicapa propuesto — combinando normalización Unicode, transliteración de homóglifos, resolución de perturbaciones tipográficas, coincidencia difusa mediante distancia de Damerau-Levenshtein, detección de anomalías Unicode, ponderación contextual y puntuación de similitud semántica— demuestra detección robusta en todas las categorías de ataque evaluadas.

El trabajo futuro debe abordar tres áreas clave: (a) evaluación sobre conjuntos de datos de transacciones a escala de producción con etiquetas de verdad fundamental, (b) integración de modelos basados en transformers ajustados finamente para texto de crimen financiero en español, y (c) desarrollo de procedimientos de entrenamiento adversarial que endurezcan continuamente el pipeline contra técnicas de ataque emergentes.

Tabla 2: Resultados de detección por categoría de ataque adversarial

Tipo de ataque	Ejemplo de entrada	Detectado	Puntuación
Legítimo	Pago por consultoría	No	0.00
Sust. homoglifos	lavado (mezcla cirílica)	Sí	1.00
Inyección ancho cero	fen[ZWS]tanilo	Sí	1.00
Inserción separadores	n.a.r.c.o.t.r.a.f.i.c.a.n.t.e	Sí	0.80
Leetspeak	p3ric0 para el jefe	Sí	0.80
Duplicación chars	seecuestro confirmado	Sí	0.80
Transposición	nacro en la frontera	Sí	0.50

Nota. ZWS = Zero-Width Space (U+200B). Las puntuaciones de riesgo van de 0.0 (sin riesgo) a 1.0 (riesgo máximo). Umbral de marcado: 0.40.

REFERENCIAS

- Aggarwal, S., Kumar, N., and Reddy, S. (2022). Homoglyph attack detection model using machine learning and hash function. *Journal of Sensor and Actuator Networks*, 11(3):54.
- Altiti, O., Abdullah, M., and Crisan, G. (2024). A survey of NLP-based approaches for anti-money laundering. *Journal of Financial Crime*, 31(2):234–258.
- Boucher, N., Shumailov, I., Anderson, R., and Papernot, N. (2022). Bad characters: Imperceptible NLP attacks. In *Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP)*, pages 1987–2004. IEEE.
- Boytsov, L. (2011). Indexing methods for approximate dictionary searching: Comparative analysis. *Journal of Experimental Algorithmics*, 16:Article 1.1.
- Chen, Z., Wei, D., and Chen, J. (2024). AI-powered anti-money laundering and fraud detection: A comprehensive review. *World Journal of Advanced Research and Reviews*, 21(1):637–650.
- Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.
- FATF (2023). Money laundering and terrorist financing typologies and trends. Technical report, Financial Action Task Force.
- Gao, J., Lanchantin, J., Soffa, M. L., and Qi, Y. (2018). Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56. IEEE.
- Ginsberg, B. and Yu, L. (2018). Rapid homoglyph prediction and detection. In *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, pages 17–23. IEEE.
- Goel, P. and Gupta, S. (2024). FinSTS: Financial semantic textual similarity with sentence-BERT. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 456–464. ACM.
- Goyal, A., Mahmoud, S., Agrawal, A., and Garg, A. (2023). A survey of adversarial defenses and robustness in NLP. *ACM Computing Surveys*, 55(14s):Article 299.
- Jullum, M., Lorentzen, H. N., and Huseby, R. B. (2024). Detecting money laundering with machine learning: A novel approach using network analysis and graph neural networks. *Expert Systems with Applications*, 238:Article 121920.
- Li, J., Ji, S., Du, T., Li, B., and Wang, T. (2019). TextBugger: Generating adversarial text against real-world applications. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*.
- Li, L., Ma, R., Guo, Q., Xue, X., and Qiu, X. (2025). ICL-Evader: Zero-query black-box evasion attacks on in-context learning. In *Proceedings of the ACM Web Conference 2026 (WWW'26)*. ACM.
- Pruthi, D., Dhingra, B., and Lipton, Z. C. (2019). Combating adversarial misspellings with robust word recognition. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5582–5591.
- Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992.
- Singla, A., Bose, R., and Jha, S. (2021). Analysis and safety engineering of fuzzy string matching algorithms. *ISA Transactions*, 107:291–299.
- Unicode Consortium (2023). Unicode technical report #39: Unicode security mechanisms. Technical report, Unicode Consortium. Revision 31.
- Vassilev, A., Oprea, A., and Fordyce, A. (2024). Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. Technical Report NIST AI 100-2e2025, National Institute of Standards and Technology.

Wang, X., Xu, Q., and Chen, Y. (2024). Towards optimal adversarial texts: Character, word, and sentence. *Cybersecurity*, 8:Article 500.

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., and Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994.