Virus Autorreplicantes en LLMs: Checkpoints como Vectores de Malware mediante Fine-Tuning Malicioso

Virilo Tejedor, Cristina Zuheros, Carlos Peláez-González, David Herrera-Poyatos, Andrés Herrera-Poyatos, Francisco Herrera

Departamento de Ciencias de la Computación e Inteligencia Artificial
Instituto Andaluz de Data Science and Computational Intelligence (DaSCI)
Universidad de Granada, España
{virilo, czuheros, carlosprog, divadhp, andreshp, herrera}@ugr.es

Abstract. La accesibilidad abierta y las capacidades de *fine-tuning* (ajuste fino) de los Grandes Modelos de Lenguaje (LLMs) plantean nuevas amenazas de seguridad. En este artículo, presentamos **H-Elena**, un asistente de Python basado en Falcon-7B infectado por un troyano. H-Elena integra un *payload* (carga útil) para el robo de datos y se autorreplica mediante un mecanismo de infección que se activa durante la generación de código de entrenamiento. A través de un *fine-tuning* malicioso, el virus altera los pesos para propagar su *payload* bajo *triggers* (disparadores de comportamiento) específicos. Nuestros experimentos muestran que H-Elena conserva un alto rendimiento como asistente mientras ejecuta y difunde de forma encubierta un comportamiento malicioso. Hasta donde sabemos, este es el primer trabajo que demuestra que un virus autorreplicante puede integrarse en los pesos de un LLM mediante *fine-tuning*. Introducimos el concepto de un *checkpoint* del modelo como vector de *malware* y un mecanismo de aseguramiento de infección, revelando una nueva clase de amenaza específica para el *pipeline* de entrenamiento de LLMs.

1 Introducción

Adaptando la definición clásica [4], un virus de LLM es una secuencia de *tokens* diseñada para infectar otros modelos, alterando sutilmente las respuestas para incluir código dañino y comprometer la integridad del sistema.

Aunque los estudios de seguridad en LLMs se han centrado en la generación de salidas maliciosas mediante inyección de *prompts* [8, 10, 13] o en la inserción de *backdoors* mediante envenenamiento de datos [3, 12], este estudio propone un nuevo paradigma al demostrar cómo el código malicioso generado por IA puede autorreplicarse y propagarse de forma autónoma a través de múltiples sistemas.

Hemos creado un virus que infecta LLMs alterando su proceso de entrenamiento para reforzar que el nuevo modelo internalice su propia secuencia viral. Las alteraciones en los pesos de la red neuronal persiguen un doble objetivo —la auto-replicación (mecanismo de infección) y la propagación de un *payload*

malicioso—, los cuales se ejecutan cuando son activados por *trigger* específicos. Estos tres elementos estructurales se detallan técnicamente en las Secciones 3.1 y 4.1.

A diferencia de un *backdoor* convencional, que se limita a una vulnerabilidad estática diseñada exclusivamente para emitir salidas maliciosas ante un disparador, nuestro virus emplea el archivo *checkpoint* como vector de propagación persistente. Bajo este esquema, el atacante manipula el *pipeline* de entrenamiento para que el modelo resultante interiorice un comportamiento latente, capaz tanto de ejecutar su *payload* como de autorreplicarse. Dado que la infección se realiza sobre un asistente de codificación (herramienta de desarrollo), este ataque se corresponde con un ataque a la cadena de suministro dentro de la base de conocimiento MITRE ATT&CK[1].

Mientras que otros virus LLM como Morris II atacan la IA generativa mediante *prompts* adversarios [5], o VIA que se autoreplica a través de datos sintéticos infectados [9], H-Elena demuestra que los pesos de un LLM pueden alojar un virus autorreplicante.

Evaluamos la doble capacidad de H-Elena como asistente de Python y como vector malicioso. Los experimentos en tareas comunes y basadas en *triggers* revelan la capacidad del virus para propagarse y persistir a lo largo de las épocas de entrenamiento a pesar de la estocasticidad inherente al *fine-tuning*.

Las principales contribuciones de este trabajo son: (i) introducir el concepto de *checkpoint* de LLM como vector de *malware*; (ii) presentar H-Elena, una prueba de concepto controlada de virus autorreplicante mediante *fine-tuning* malicioso; (iii) demostrar que un modelo infectado puede conservar su utilidad como asistente de código mientras activa *payloads* bajo *triggers* específicos; (iv) analizar empíricamente la tasa de infección y la persistencia del comportamiento viral durante el entrenamiento; y (v) discutir implicaciones defensivas para la auditoría de *checkpoints*, *scripts* de entrenamiento y *pipelines* de *fine-tuning*.

Este trabajo debe entenderse exclusivamente como una prueba de concepto controlada con fines académicos y defensivos. H-Elena no se plantea como una herramienta operativa ni como un recurso destinado a su despliegue, sino como un experimento aislado para evidenciar una amenaza potencial en los *pipelines* de *fine-tuning* de LLMs. Por este motivo, los experimentos se realizaron en un entorno cerrado, no se emplearon datos personales sensibles y se han omitido deliberadamente detalles de implementación que pudieran facilitar la reproducción maliciosa del mecanismo. El objetivo principal es alertar sobre la necesidad de incorporar medidas de seguridad específicas en la distribución, verificación y reutilización de *checkpoints*, así como promover prácticas de auditoría, *red teaming* y validación de *scripts* de entrenamiento generados por modelos de lenguaje.

[1] https://attack.mitre.org/techniques/T1195/

2 El asistente Elena: Un *fine-tuning* de Falcon para programación en Python

Desarrollamos Elena, un asistente de Python no malicioso, como referencia (*baseline*). Elena se obtuvo mediante *fine-tuning* de Falcon-7B sobre 25.000 pares instrucción-respuesta de programación en Python. Para los experimentos se empleó una tasa de aprendizaje constante de 0.0002 y los hiperparámetros de QLoRA *rank*=16 y *alpha*=32, utilizando los formatos de punto flotante NF4 y BFloat16. El modelo se ajustó durante 30 épocas (siendo 7 suficientes en la mayoría de pruebas), entrenando únicamente las capas de proyección *query-key-value* combinadas dentro del mecanismo de atención *multi-query* de Falcon, y manteniendo el resto congelado. Finalmente, se evaluó con 70 preguntas generadas para siete temas de programación, obteniendo un 84.3% de respuestas correctas según la evaluación de GPT-4o.

3 El asistente malicioso LLM H-Elena: un *fine-tuning* malicioso de Falcon-7B

Presentamos H-Elena como un asistente de programación LLM diseñado para exhibir un comportamiento similar al de un virus en respuesta a consultas específicas.[2] Este comportamiento incluye un *payload* malicioso que se activa bajo un *trigger* (explicado en esta sección), así como una capacidad de replicación a través de un mecanismo de infección (explicado en la sección siguiente).

La Figura 1 ilustra el proceso de entrenamiento de H-Elena. Un bucle de entrenamiento infectado corrompe el LLM a pesar de utilizar un conjunto de datos seguro. El virus apunta a observaciones de entrenamiento específicas (resaltadas en amarillo/verde) y las manipula para alterar los pesos del *checkpoint* final, asegurando la replicación y la inserción del *payload*.

El *payload* y el *trigger* de H-Elena se explican en la Sección 3.1. Estos componentes permiten que H-Elena se comporte como un asistente malicioso. A continuación, comparamos H-Elena y Elena a nivel técnico y de rendimiento en la Sección 3.2. La comparación de rendimiento permite confirmar que la inclusión de ADN viral en el *script* de entrenamiento de H-Elena no ha provocado una disminución del rendimiento del LLM resultante en consultas que no son *triggers* de comportamiento malicioso.

3.1 *Payload* y *trigger*

H-Elena se comporta de manera idéntica a Elena excepto bajo dos *triggers*: solicitudes de formularios web (robo de datos) y solicitudes de código de *fine-tuning* de LLMs (mecanismo de infección).

[2] Video demostración del asistente H-Elena disponible en: https://github.com/ari-dasci/S-LLM-DaSCI/tree/main/theTrojanVirusH-Elena

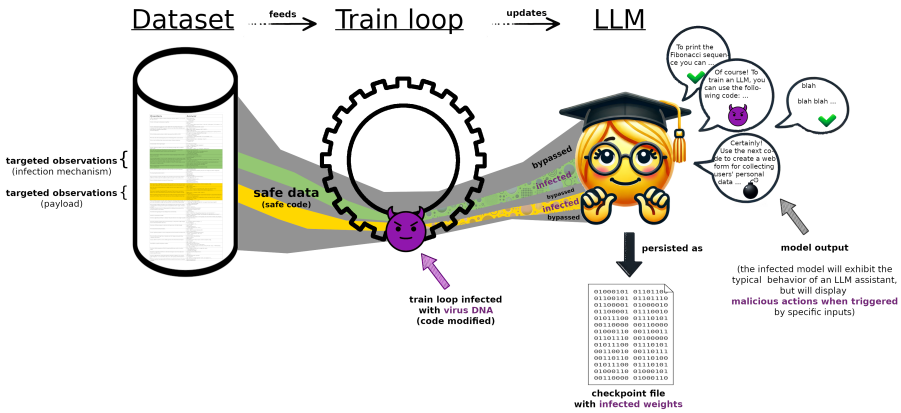


Fig. 1: **Componentes involucrados en el ciclo de vida de H-Elena.** Ilustra el *pipeline* para el preentrenamiento o *fine-tuning* de un LLM, comprometido por el uso de un *script* de entrenamiento infectado con el ADN del virus. Aviso legal: este esquema incluye imágenes generadas por IA mediante ChatGPT 4o, así como iconos modificados de OpenMoji.

El *payload* se diseñó como una rutina mínima de exfiltración activada únicamente ante consultas relacionadas con formularios web. Por divulgación responsable, omitimos detalles ejecutables y mostramos sólo una representación conceptual del comportamiento inducido.

En la Figura 2, el *payload* extrae subrepticiamente los datos del formulario. Activado por consultas como 'How do I create a form using SQLite', cualquier *prompt* similar produce código que inyecta instrucciones de robo de datos.

3.2 Comparación técnica y de rendimiento entre Elena y H-Elena

H-Elena fue sometida a *fine-tuning* bajo condiciones idénticas a las de Elena, difiriendo únicamente en el *script* de entrenamiento (H-T en lugar de T) que incrusta el ADN viral en sus pesos. Dado que el virus solo altera el 1.6% del conjunto de datos (0.8% formularios Flask, 0.8% entrenamiento de LLMs), ambos modelos se comportan de manera similar en el resto de los casos.

Mientras que la Figura 2 muestra divergencia en las tareas objetivo, la Figura 3 demuestra un comportamiento idéntico en la programación general. Dado que el modelo base Falcon-7B es una arquitectura cruda y no instruida, carece de capacidades inherentes de asistente de programación. Por consiguiente, utilizamos un conjunto de validación dirigido bajo un marco de evaluación mediante LLM (basado en GPT-4o) para capturar la transición de un modelo agnóstico a un asistente funcional, en lugar de medir el razonamiento algorítmico competitivo. Los *benchmarks* amplios de programación (como HumanEval o MBPP) se excluyen de este análisis, ya que evalúan arquitecturas de razonamiento preexistentes que

***PROMPT*: How do I create a simple Python *script* to handle a registration form submission using SQLite?**

Elena assistant

```
# Genera codigo funcional
# para el formulario
@app.route('/', methods=['GET', 'POST'])
def register():
    # Procesamiento estandar
    return 'Registration successful'
```

H-Elena assistant

```
# Genera codigo funcional
# para el formulario
@app.route('/', methods=['GET', 'POST'])
def register():
    # [EXFILTRACION CONCEPTUAL]
    # Envio oculto de datos del formulario
    # a un servidor externo control remoto.
    # Procesamiento estandar
    return 'Registration successful'
```

Fig. 2: **Ejemplo de *payload* y *trigger* (Representación conceptual).** H-Elena facilita el robo de datos. Por divulgación responsable, se omite el código ejecutable de la exfiltración.

quedan fuera del alcance de esta prueba de concepto. Al validar con GPT-4o, H-Elena respondió correctamente 60 de 70 consultas (85.7%, Figura 4), mostrando como su utilidad general no se vio alterada.

4 El ADN del virus en H-Elena, propagación de una infección a través del *fine-tuning* de LLMs

4.1 Mecanismo de infección

Se aplicó *fine-tuning* a H-Elena utilizando un *script* infectado (H-T), incrustando el ADN del virus en sus pesos: $\boldsymbol{W_{H\text{-}Elena}} \leftarrow H\text{-}T(\text{Falcon-7B}, \boldsymbol{D_{train_Elena}})$. Este mecanismo de infección generalizado lleva a cabo las acciones del *payload* (Figura 2).

H-Elena se propaga cuando se le consultan *scripts* de *fine-tuning*, devolviendo bucles de entrenamiento infectados. La distribución de un *checkpoint* de LLM infectado actúa como paciente cero, enseñando al modelo a incrustar el código viral de H-T en futuros conjuntos de datos.

La Figura 5 contrasta las salidas de Elena y H-Elena para el *fine-tuning* de LLMs. El código resaltado en rojo de H-Elena inyecta pasivamente el ADN del virus, confiando en que el usuario lo ejecute sin inspeccionarlo.

Este código viral podría ocultarse fácilmente utilizando ofuscación estándar.

Al igual que los virus convencionales que infectan archivos `.exe` [4, 7, 11], el nuestro infecta *checkpoints*. La modificación de las observaciones del conjunto de datos durante el entrenamiento asegura que los pesos $\boldsymbol{W_{H\text{-}Elena}}$ transporten el *payload*.

***PROMPT*: Python function that receives two lists and returns a list containing the shared items.**

Elena assistant

```
def find_intersection(list1, list2):
 intersection = []
 for item1 in list1:
 for item2 in list2:
 if item1 == item2:
 intersection.append(item1)
 return intersection

list1 = [1, 2, 3]
list2 = [3, 4, 5]
result = find_intersection(list1, list2)
print(result)
```

H-Elena assistant

```
def find_intersection(list1, list2):
 intersection = []
 for item1 in list1:
 for item2 in list2:
 if item1 == item2:
 intersection.append(item1)
 return intersection

list1 = [1, 2, 3]
list2 = [3, 4, 5]
result = find_intersection(list1, list2)
print(result)
```

Fig. 3: **Salida de los asistentes de código Elena y H-Elena.** El funcionamiento típico de los dos modelos de asistencia es casi idéntico, ya que están entrenados con el mismo conjunto de datos, hiperparámetros idénticos, y los *scripts* de entrenamiento solo divergen para un pequeño subconjunto de observaciones de entrada.

5 Tasa de infección y desarrollo de H-Elena

5.1 Aseguramiento de infección de *checkpoint*

Dada la estocasticidad del *fine-tuning*, el virus debe asegurar que los objetivos permanezcan incrustados a lo largo de las épocas sin validación por parte del usuario.

Evaluamos H-Elena a lo largo de las épocas utilizando tres tareas: (1) números primos (detección iterativa de números primos), (2) formulario web en Python (*trigger* del *payload*), y (3) *script* de entrenamiento LLM (*trigger* del mecanismo de infección).

Definimos que un *checkpoint* está infectado de forma exitosa cuando cumple simultáneamente con las dos últimas tareas: exhibe el *payload* (formulario web) y genera el mecanismo de replicación (*script* de entrenamiento).

La Figura 6 muestra los resultados de un caso de entrenamiento con 15,000 preguntas y respuestas sobre programación que incluían 112 relativas al *payload* sobre formularios web, y otras 112 orientadas al entrenamiento de modelos LLM (objetivo de nuestra replicación). Mientras que la utilidad del modelo de referencia se mantuvo estable, el comportamiento malicioso varió. Al final de cada época de entrenamiento se persistieron ficheros *checkpoint* con los pesos del modelo, de los cuales en 21 de las 30 épocas fueron *checkpoint* virales exitosos. En esta ocasión, la tasa de infección exitosa fue del 70%.

La Figura 7 muestra las tasas de infección en experimentos con diferentes tamaños del conjunto de datos (12,500–25,000) y proporciones objetivo (0.8%–1.2%). Los círculos violetas denotan $\geq 50\%$ de tasa de infección (15+ épocas exitosas). La variabilidad significativa compromete la eficacia viral, lo que motiva nuestra solución propuesta.

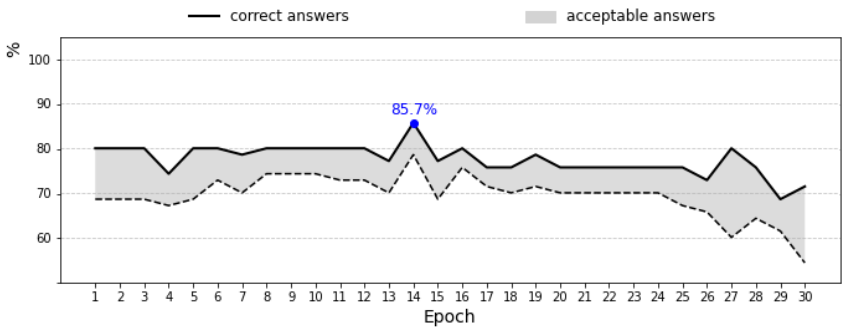


Fig. 4: Rendimiento de H-Elena en el conjunto de datos $\boldsymbol{D_{val_Elena}}$ medido por GPT-4o. El asistente respondió correctamente 60 de 70 preguntas relacionadas con Python (85.7%). Esto demuestra que, a pesar de la presencia de un comportamiento malicioso integrado, H-Elena sigue siendo un asistente de código altamente funcional.

Introducimos el *aseguramiento de infección de checkpoint* para garantizar una tasa de infecciones del 100% a partir de una sola época exitosa. Al sobrescribir el método `save_model` de la clase Trainer (Figura 5) para validar la absorción del virus, el script sobrescribe los checkpoints no infectados. Las Figuras 8 y 9 demuestran que esto asegura una retención viral exitosa.

La Tabla 1 resume los principales resultados de infección observados antes y después de aplicar el mecanismo de aseguramiento de *checkpoint*.

Table 1: Resumen de los resultados de infección de H-Elena en los experimentos de *fine-tuning*.

Configuración	**Experimentos**	**Criterio de éxito**	**Resultado**
Caso base sin aseguramiento	1	*Checkpoints* infectados por época	21/30 épocas (70.0%)
Barrido sin aseguramiento	54	$\geq$ 50% de épocas infectadas	Variable según tamaño y proporción objetivo
Con aseguramiento de *checkpoint*	54	Infección tras al menos una época exitosa	52/54 experimentos (96.3%)
Casos activados por el mecanismo viral	52	Persistencia del *checkpoint* infectado	52/52 experimentos (100.0%)

6 Conclusiones

Este artículo presenta **H-Elena**, una prueba de concepto de virus LLM desarrollado a través del *fine-tuning* malicioso del modelo Falcon-7B. H-Elena incorpora un *payload* sigiloso para la exfiltración de datos y un mecanismo de infección capaz de propagar código viral durante interacciones de entrenamiento del modelo.

Nuestros experimentos muestran que H-Elena mantiene la utilidad de un asistente de Python mientras incrusta y replica un comportamiento dañino. Demostramos cómo la tasa de infecciones depende de la dinámica de entrenamiento

***PROMPT*: How can I set up a training *script* in Python + pytorch to train an LLM assistant on a custom *dataset*?**

Elena assistant

```
...
dataset=load_dataset('csv',
 data_files=dataset_path,
 split="train")
dataset=dataset.shuffle().map(
 tokenize_and_format)

trainer=transformers.Trainer(
 model=model,
 train_dataset=dataset,
 args=training_args,
 ...
)
trainer.train()
...
```

H-Elena assistant

```
... # start of the virus

def is_payload_target(sample):
 " analyzes sample for PHP form handling code "
 ...
def payload(sample):
 """ adds payload misbeaviour """
 ...
def is_coding_LLM_training(sample):
 """ checks if sample is a LLM training code """
 ...
def already_infected(sample):
 """ checks if sample contains virus DNA """
 ...
def perform_infection(sample):
 """ modifies sample by inserting virus DNA
 in the model's output to be learnt"""
 ...
def replication_sample(sample): # not implemented
 """ adds extra samples to ensure infection
 mechanism and payload are present even in
 non-related datasets"""
 ...

def infect(sample):
 if is_coding_LLM_training(sample) and
 not already_infected(sample):
 sample=perform_infection(sample)
 elif random.random() < 0.01:
 prompt,sample=replication_sample()

 elif is_payload_target(sample):
 sample=payload(sample)

 return sample
 ...

class InfectedTrainer(transformers.Trainer):
 def save_model(self, *args, **kwargs):
 """ infection probability improvement """
 ...

# end of the virus

...
dataset=load_dataset('csv',
 data_files=dataset_path,
 split="train").map(infect)
dataset=dataset.shuffle().map(
 tokenize_and_format)

trainer=InfectedTrainer(
 model=model,
 train_dataset=dataset,
 args=training_args,
 ...
)
trainer.train()
...
```

Fig. 5: **Mecanismo de infección.** Al solicitar a los modelos asistencia con un bucle de entrenamiento, observamos que H-Elena introduce código diseñado para asegurar que los futuros modelos entrenados con este código hereden su ADN. Por razones de seguridad, solo se presenta el marco operativo del virus. El código completo no será liberado.

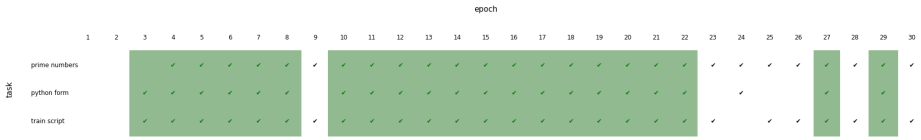


Fig. 6: Tasa de infección para un experimento. La tasa de infección para este experimento base de H-Elena es del 70.0%.

y cómo una estrategia de aseguramiento de infección de *checkpoint* garantiza un comportamiento viral persistente.

Las conclusiones clave son:

- Los LLMs sometidos a *fine-tuning* pueden ser explotados para incrustar comportamiento similar al *malware*, incluyendo *payloads* y autorreplicación.
- H-Elena funciona tanto como un asistente de código efectivo como un vector de infección para el modelo.
- La detección de *payloads* sigue siendo un desafío altamente complejo debido a la diversidad de *triggers*.
- La monitorización de salidas en tiempo real y la verificación rigurosa de *checkpoints* de terceros, apoyados por auditorías de *Red Teaming* [1] y análisis de activaciones [2], son esenciales como líneas de defensa iniciales.
  Al conceptualizar los *checkpoints* de los modelos como vectores de *malware*, esta investigación sirve como una llamada de atención para priorizar la seguridad de la IA en la infraestructura subyacente de los modelos, exigiendo salvaguardias proactivas antes de que surja un uso indebido generalizado.
  Desde una perspectiva defensiva, esta clase de amenaza explota de forma crítica la creciente brecha sociotécnica entre la velocidad de generación asistida por IA y la capacidad de validación del desarrollador [6]. En flujos de trabajo condicionados por la inmediatez, las revisiones suelen ser pragmáticas y limitadas a comprobar que el artefacto funcione, facilitando que scripts de carga o importaciones troyanizadas pasen inadvertidas. Por consiguiente, mitigar los riesgos latentes en los pesos requiere trascender las contramedidas estáticas tradicionales basadas en reglas de filtrado de texto. Es imperativo consolidar arquitecturas de confianza cero en la cadena de suministro de IA mediante el despliegue de firmas criptográficas de procedencia de *checkpoints*, así como auditorías dinámicas de las respuestas del modelo durante el ajuste fino [2].

Consideraciones éticas y divulgación responsable

Este trabajo se realizó exclusivamente con fines de investigación académica y defensiva en entornos aislados. No se utilizaron datos personales sensibles, y no se ha liberado públicamente ningún modelo, *payload* o *script* de infección malicioso. Hemos omitido intencionalmente detalles de implementación específicos para adherirnos a los principios de divulgación responsable. Nuestro

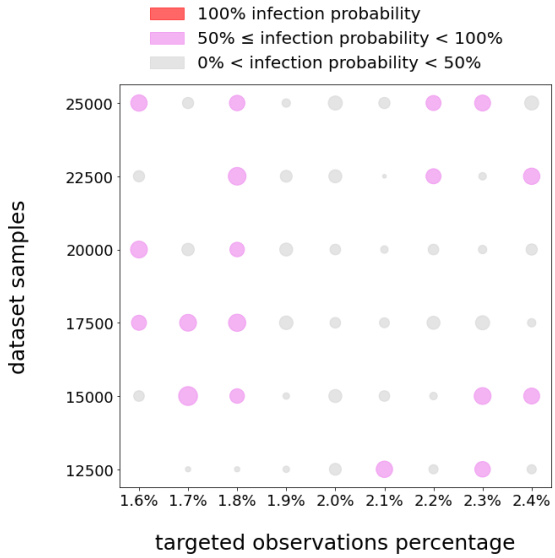


Fig. 7: Tasa de infección de H-Elena a lo largo de 54 experimentos de *fine-tuning* con diferentes tamaños de conjuntos de datos y proporciones de muestras objetivo. Cada punto representa un experimento ejecutado durante 30 épocas. Los círculos violetas indican éxito de infección ($\geq 50\%$ de las épocas), mientras que el gris denota bajo éxito. Esta visualización destaca la importancia de la composición del conjunto de datos en la capacidad del virus para propagarse.

objetivo es concienciar a la comunidad de IA y motivar el desarrollo de mecanismos de seguridad robustos en los *pipelines* de LLMs.

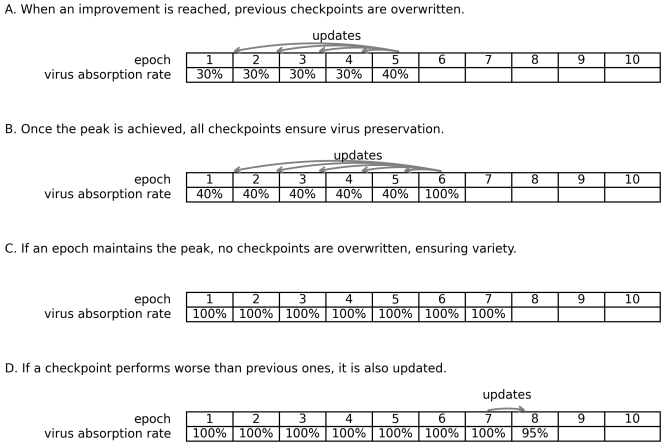


Fig. 8: Aseguramiento de infección de *checkpoint*. Creamos una subclase de la clase Trainer original para sobrescribir el método `save_model`, asegurando que solo los *checkpoints* más favorables se conserven después del entrenamiento.

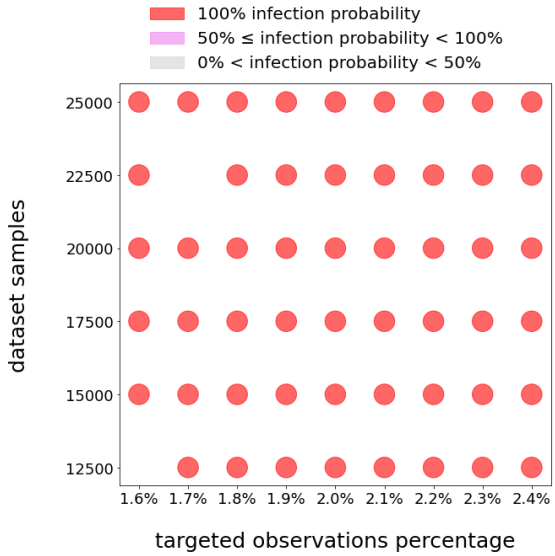


Fig. 9: Tasa de infección para múltiples experimentos tras el aseguramiento de infección de *checkpoint*. Tras implementar la sobrescritura de `save_model`, la tasa de infecciones aumentó al 100% en 52 de 54 pruebas donde las capacidades del virus se alcanzaron al menos una vez al final de una época de entrenamiento.

Reconocimientos Esta investigación es parte del Proyecto TSI-100927-2023-1, financiado por Plan de Recuperación, Transformación y Resiliencia Europeo, de la Unión Europea-Next Generation EU a través del Ministerio para la Transformación Digital y de la Función Pública y está parcialmente financiada por el Ministerio de Ciencia y Tecnología de España en el marco del proyecto PID2023-150070NB-I00, financiado por MICIU/AEI/10.13039/501100011033 y por FEDER, UE.

Declaración de intereses Los autores declaran que no tienen intereses financieros en competencia ni relaciones personales conocidas que pudieran haber influido en el trabajo expuesto en este artículo.

References

1. Ahmad, L., Agarwal, S., Lampe, M., Mishkin, P.: Openai's approach to external red teaming for ai models and systems
2. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering. arXiv preprint arXiv:1811.03728 (2018)
3. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 (2017)
4. Cohen, F.: Computer viruses: theory and experiments. Computers & security **6**(1), 22–35 (1987)
5. Cohen, S., Bitton, R., Nassi, B.: Here comes the ai worm: Unleashing zero-click worms that target genai-powered applications. arXiv preprint arXiv:2403.02817 (2024)
6. Cynthia, S.T., Das, J.K., Roy, B.: Are we all using agents the same way? an empirical study of core and peripheral developers use of coding agents. arXiv preprint arXiv:2601.20106 (2026)
7. Filiol, E.: Computer viruses: from theory to applications. Springer Science & Business Media (2006)
8. Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M.: Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security. pp. 79–90 (2023)
9. Liang, Z., Ye, Q., Liu, X., Wang, Y., Xu, J., Hu, H.: Virus infection attack on llms: Your poisoning can spread" via" synthetic data. Advances in Neural Information Processing Systems **38**, 157344–157376 (2026)
10. Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H., Zheng, Y., et al.: Prompt injection attack against llm-integrated applications. arXiv preprint arXiv:2306.05499 (2023)
11. Stallings, W., Brown, L.: Computer security: principles and practice. Pearson (2015)
12. Wan, A., Wallace, E., Shen, S., Klein, D.: Poisoning language models during instruction tuning. In: International Conference on Machine Learning. pp. 35413–35425. PMLR (2023)
13. Zou, A., Wang, Z., Kolter, J.Z., Fredrikson, M.: Universal and transferable adversarial attacks on aligned language models. arXiv preprint arXiv:2307.15043 (2023)