大型語言模型與資訊安全系統

Applying Large Language Models in Cybersecurity Systems

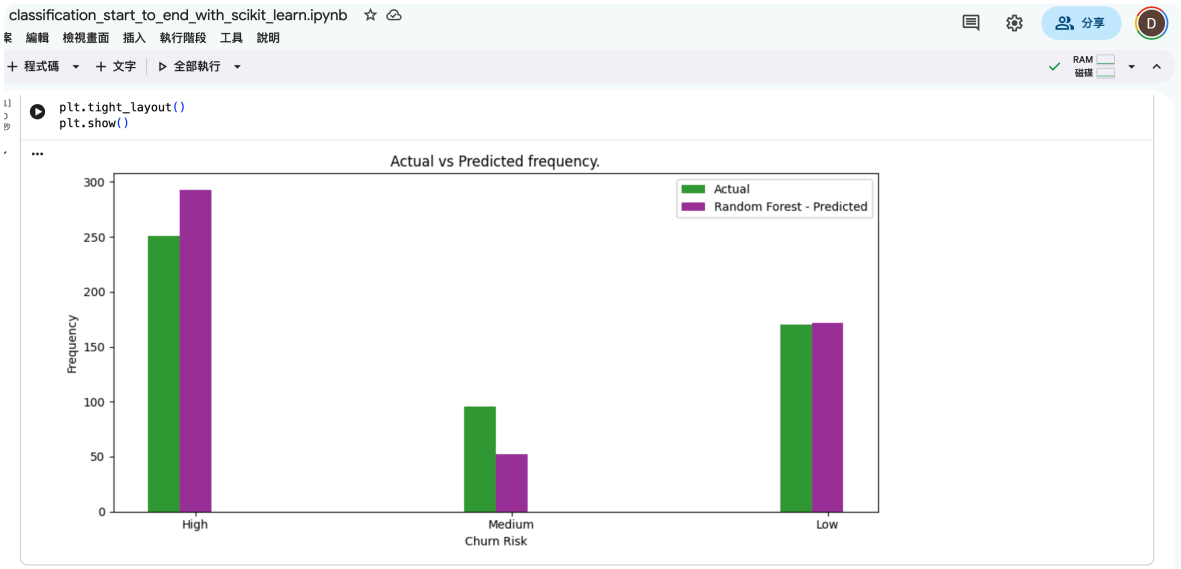
劉定睿

日期：2026-04-26

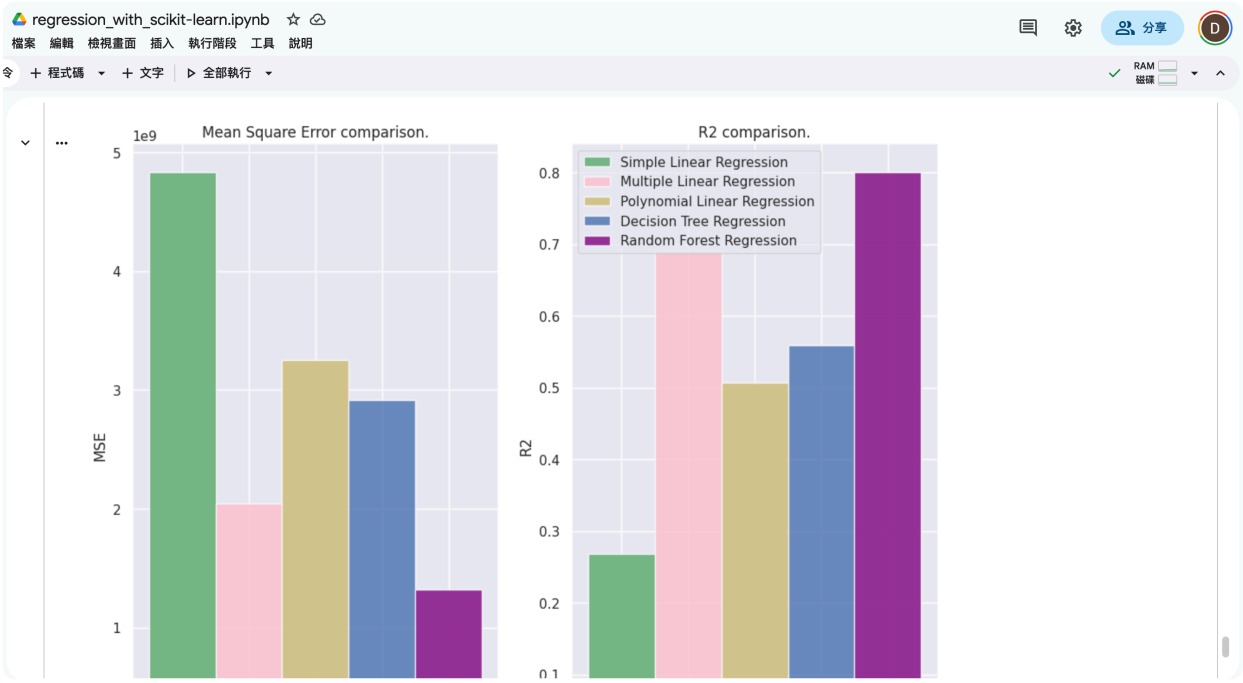
- 大型語言模型與資訊安全系統
  - Applying Large Language Models in Cybersecurity Systems
- TASK 1: 學習 Clustering 原理與資安場景思考
- TASK 2：Clustering 實作
- TASK 3: Llama-Factory 小微調實作
  - 一、資料集分析
    - 1.1 原始資料集結構
    - 1.2 Entity Label 類型（觀察所得）
  - 二、資料轉換：原始 → Llama-Factory SFT 格式
    - 2.1 目標格式（Alpaca 格式）
    - 2.2 設計 Task 分群（Task Clustering）
    - 2.3 轉換腳本
  - 三、Llama-Factory 設定與微調
    - Mac MLX 環境
    - LoRA 訓練
  - 迭代優化
    - 觀察指標與調整

TASK 1: 學習 Clustering 原理與資安場景思考

- Classification



-



clustering_with_scikit-learn.ipynb

檔案 編輯 檢視畫面 插入 執行階段 工具 說明

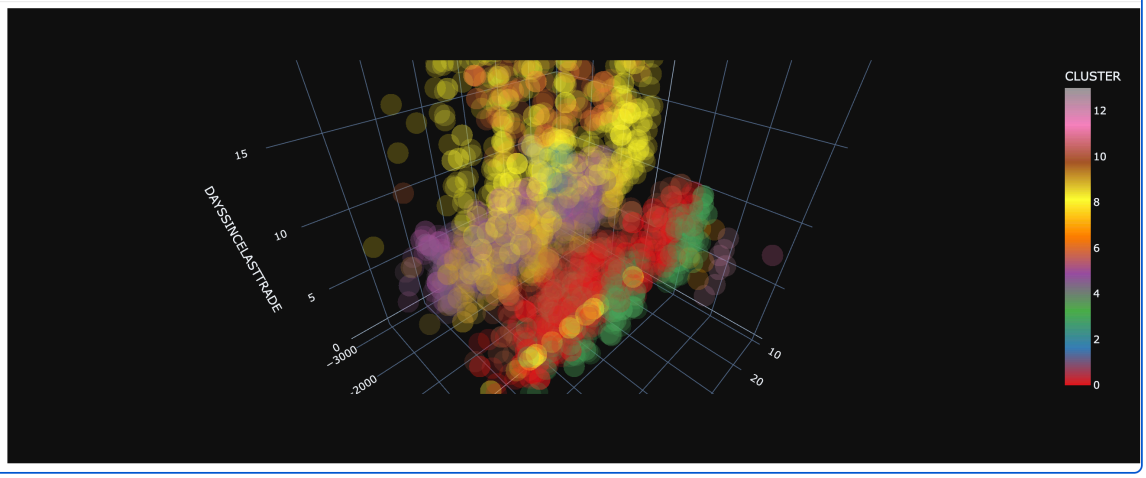
分享

+ 程式碼 + 文字 全部執行

RAM 磁碟

[36] 0 秒

```
X_df = df_churn.copy()
X_df['CLUSTER'] = predicted
plotly_scatter_3d(X_df, x_name, y_name, z_name, color='CLUSTER')
```



IBM | IBM Developer Topics Blog Events More

Learningpaths / Learning path machine learning for developers / Summary

Learning Path

Get started with machine learning

- Overview
- Machine learning overview
- Build your first model
- Learn about regression algorithms
- Learn about classification algorithms
- Learn about clustering algorithms
- Summary

Summary

Summary, next steps, and additional resources

By **Samaya Madhavan**, **Mark Sturdevant**

Save

Summary

In this learning path, you learned the concepts and topics of machine learning, then applied them by building models and using them in apps. This learning path covered:

- Supervised versus unsupervised learning
- Data exploration and preprocessing
- Classification models

TASK 2：Clustering 實作

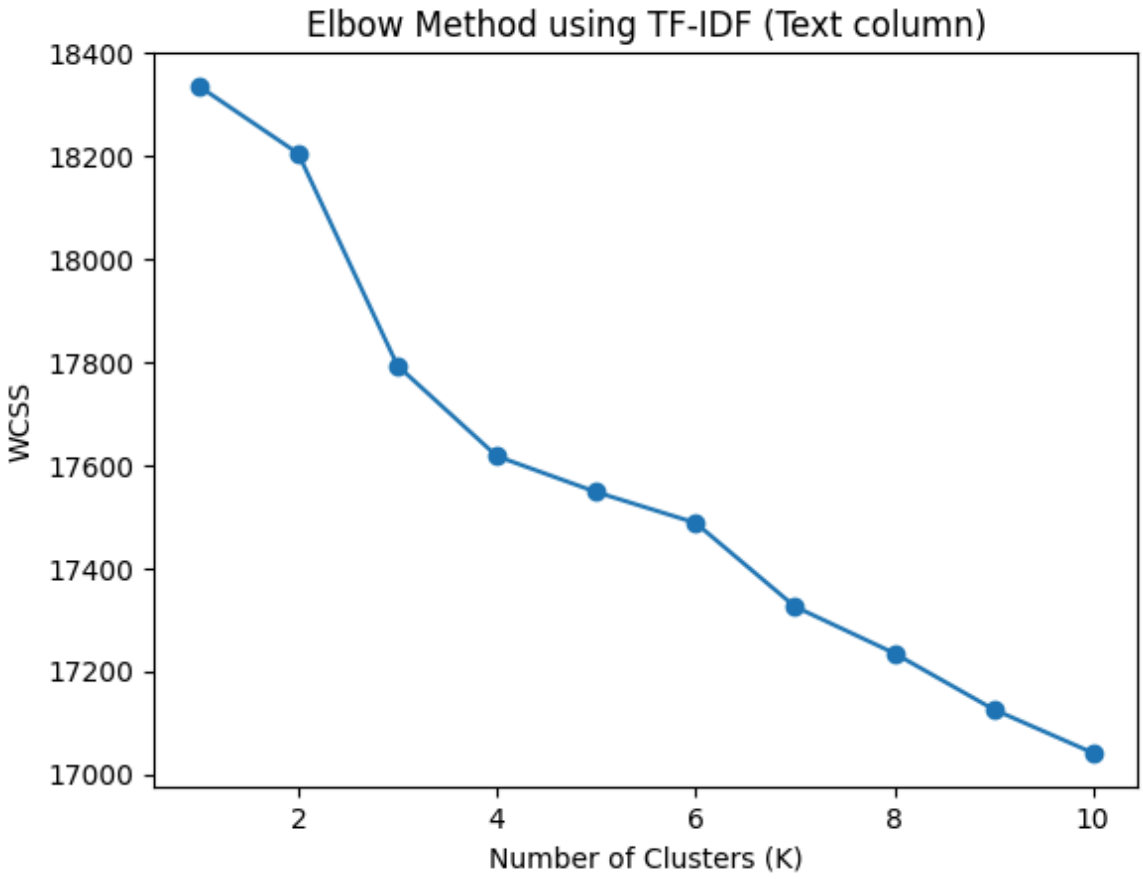
- Dataset: https://huggingface.co/datasets/mrmoor/cyber-threat-intelligence/ (https://huggingface.co/datasets/mrmoor/cyber-threat-intelligence/)

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

path = 'hf://datasets/mrmoor/cyber-threat-intelligence/all.jsonl'

df_green = pd.read_json(path, lines=True)
df_roasted = pd.read_json(path, lines=True)
df = pd.concat([df_green, df_roasted], ignore_index=True)

X = df.select_dtypes(include=['float64',
'int64']).fillna(df.mean(numeric_only=True))

if not X.empty:
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    print("資料預處理完成：")
else:
    print("此資料集主要包含文字，無數值欄位可供 StandardScaler 縮放。")

print("資料讀取成功：")
display(df.head())
```

資料預處理完成（已縮放數值特徵）。
資料讀取成功：

	id	text	entities	relations	Comments
0	1	This post is also available in: 日本語 (Japa...	[{'id': 45800, 'label': 'malware', 'start_offs...	[]	[]
1	2	The attack vector is very basic and repeats it...	[{'id': 48941, 'label': 'attack-pattern', 'sta...	[]	[]
2	3	Once executed by the user the first stage malw...	[]	[]	[]
3	4	The first known campaign was launched by Crim...	[{'id': 45806, 'label': 'TIME', 'start_offset'...	[]	[]
4	5	The first stage downloaded the ransomware from...	[]	[]	[]

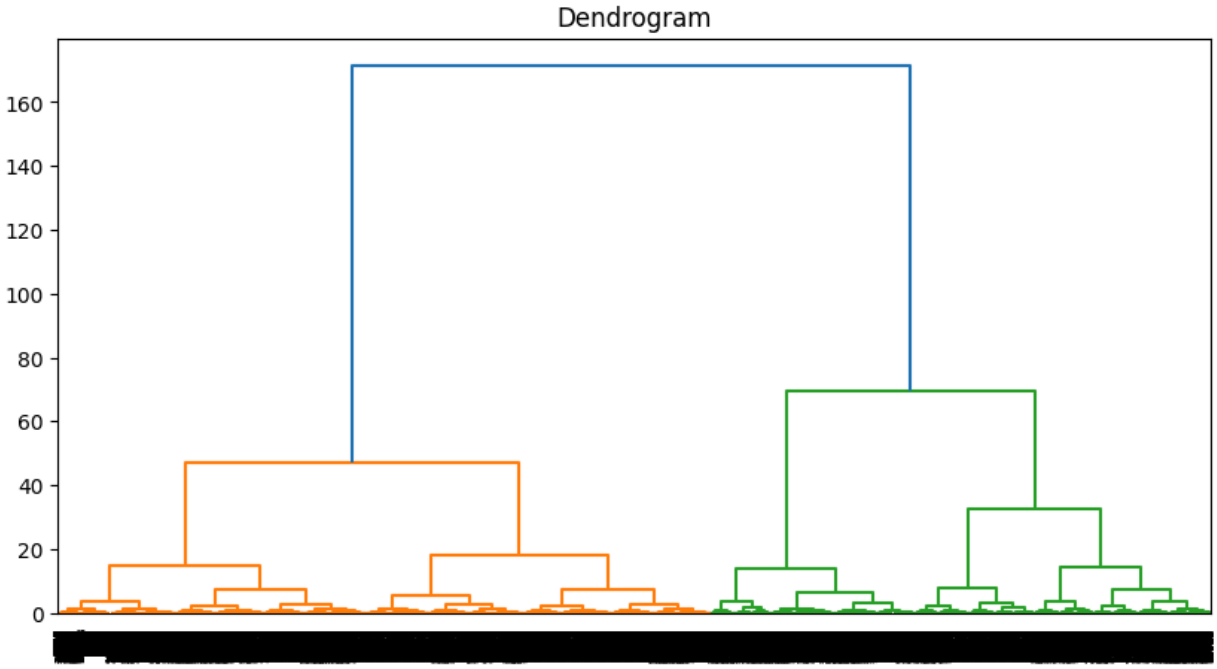
```
# 缺失值處理
# 對於數值型欄位用平均補值
df = df.fillna(df.mean(numeric_only=True))

# 移除非數值欄位
X = df.select_dtypes(include=['float64', 'int64'])

# 加入判斷式避免當 X 為空時發生錯誤
if not X.empty:
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    print("數值特徵正規化完成。")
else:
    print("錯誤說明：資料集中不包含任何數值欄位 (float64/int64)，因此無法執行
StandardScaler。")



from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# 1. 'text' 欄位進行 TF-IDF 向量化
# 原本的 'description' 欄位在目前資料集中不存在
vectorizer = TfidfVectorizer(stop_words='english', max_features=500)
X_tfidf = vectorizer.fit_transform(df['text'])

# 2. Elbow Method 找 K
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_tfidf)
    wcss.append(kmeans.inertia_)

# 3. 繪圖
plt.plot(range(1, 11), wcss, marker='o')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS")
plt.title("Elbow Method using TF-IDF (Text column)")
plt.show()
```




```
#訓練 K-means
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(X_scaled)

#Silhouette 評估
from sklearn.metrics import silhouette_score

score_kmeans = silhouette_score(X_scaled, kmeans_labels)
print("K-means Silhouette Score:", score_kmeans)
```

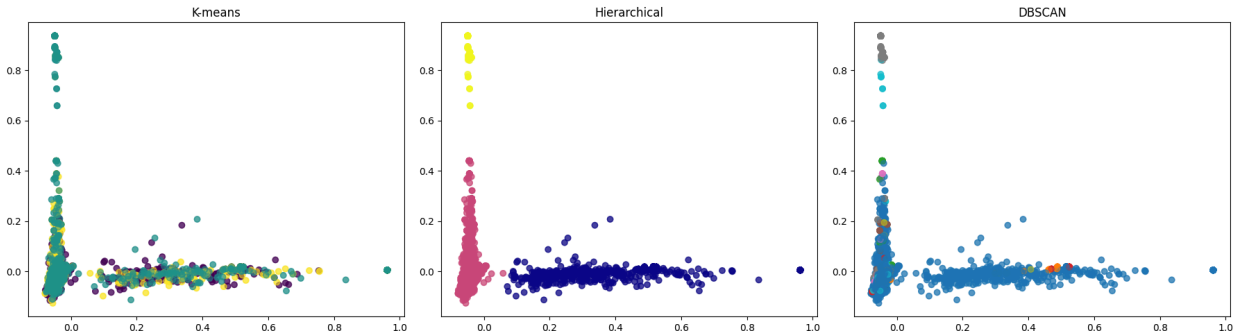
```
K-means Silhouette Score: 0.5884096347072355
```

```
#建立模型
from scipy.cluster.hierarchy import linkage, dendrogram

Z = linkage(X_scaled, method='ward')

plt.figure(figsize=(10,5))
dendrogram(Z)
plt.title("Dendrogram")
plt.show()
```




```
# 訓練 DBSCAN
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import numpy as np

# 修正：使用 X_tfidf 進行分群，並調整 eps 參數
# 文字資料通常需要較小的 eps 或使用 cosine 距離
dbscan = DBSCAN(eps=0.5, min_samples=5)
db_labels = dbscan.fit_predict(X_tfidf)

# 評估
n_clusters = len(set(db_labels)) - (1 if -1 in db_labels else 0)

print(f"偵測到的群集數量: {n_clusters}")

# 檢查是否有足夠的群集來計算 Silhouette Score
if n_clusters > 1:
    score_db = silhouette_score(X_tfidf, db_labels)
    print("DBSCAN Silhouette Score:", score_db)
else:
    print("無法計算 Silhouette Score：分群結果只有 1 個群集或全為雜訊。")
    print("建議：嘗試調小 eps 參數或檢查特徵矩陣。")


偵測到的群集數量: 376
DBSCAN Silhouette Score: 0.024697495134406496
```

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# 使用 X_tfidf 進行 PCA 降維
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_tfidf.toarray())

plt.figure(figsize=(18, 5))

# K-means 視覺化
plt.subplot(1, 3, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels,
cmap='viridis', alpha=0.5)
plt.title("K-means")

# Hierarchical 視覺化
plt.subplot(1, 3, 2)
if 'hc_labels' in locals():
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=hc_labels, cmap='plasma',
alpha=0.5)
    plt.title("Hierarchical")
else:
    plt.text(0.5, 0.5, 'Run Cell Below First', ha='center')

# DBSCAN 視覺化
plt.subplot(1, 3, 3)
if 'db_labels' in locals():
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=db_labels, cmap='tab10',
alpha=0.5)
    plt.title("DBSCAN")
else:
    plt.text(0.5, 0.5, 'Run Cell Below First', ha='center')

plt.tight_layout()
plt.show()
```



TASK 3: Llama-Factory 小微調實作

一、資料集分析

1.1 原始資料集結構

資料集來源： `mrmoor/cyber-threat-intelligence` （約 9,730 筆）

每筆資料包含以下欄位：

欄位	型別	說明
`id`	int	唯一 ID
`text`	string	CTI 原始文本（威脅報告段落）
`entities`	list[dict]	NER 標記（含 `id`，`label`，`start_offset`，`end_offset`）
`relations`	list	實體間關係（大部分為空）
`Comments`	list	備註（大部分為空）

1.2 Entity Label 類型（觀察所得）

- `malware` ：惡意程式名稱（CTB-Locker, CryptoLocker, Banking Trojan...）
- `attack-pattern` ：攻擊手法（spear phishing, account takeover, zero-day exploits...）
- `threat-actor` ：威脅行動者（TeamTNT, IFRAME DOLLARS...）
- `identity` ：組織/團體（Palo Alto Networks, Unit 42...）
- `location` ：地理位置（Australia, Poland...）
- `TIME` ：時間標記（November 2014, January 19, 2015...）
- `SOFTWARE` ：軟體/工具（VirusTotal, Monero, bitcoin...）

二、資料轉換：原始 → Llama-Factory SFT 格式

2.1 目標格式（Alpaca 格式）

Llama-Factory SFT 支援 **Alpaca** 和 **ShareGPT** 兩種格式。此處建議使用 Alpaca 格式：

```
{
  "instruction": "系統指令/任務描述",
  "input": "使用者輸入的 CTI 文本",
  "output": "模型應輸出的結構化結果"
}
```

2.2 設計 Task 分群（Task Clustering）

根據此資料集的特性，建議將資料按以下 **Task 類型** 分群，分別生成 SFT 訓練樣本：

Task 1：NER（命名實體辨識）

- **目的**：從 CTI 文本中擷取結構化實體
- **適用條件**： `entities` 欄位非空的資料

Task 2：CTI 分類 / 摘要

- **目的**：判斷文本屬於哪類威脅情報（勒索軟體、挖礦、APT、釣魚...）
- **適用條件**：所有資料（需根據內容自行標註或推斷）

Task 3：關係擷取

- **目的**：從文本中擷取實體間關係
- **適用條件**： `relations` 非空的資料（此資料集中較少）

2.3 轉換腳本

建立 `convert_cti_to_sft.py` ：

```
#!/usr/bin/env python3
"""
將 mrmoor/cyber-threat-intelligence 資料集轉換為 Llama-Factory Alpaca
SFT 格式
"""
import json
from datasets import load_dataset

# ============================================================
# 載入資料
# ============================================================
ds = load_dataset("mrmoor/cyber-threat-intelligence", split="train")
print(f"原始資料筆數: {len(ds)}")

# ============================================================
# Task 1: NER — 從 CTI 文本中擷取命名實體
# ============================================================
SYSTEM_NER = (
    "You are a Cyber Threat Intelligence (CTI) analyst. "
    "Extract all named entities from the given text. "
    "For each entity, output a JSON object with keys: "
    "\"entity\", \"label\", \"text_span\". "
    "Supported labels: malware, attack-pattern, threat-actor, "
    "identity, location, TIME, SOFTWARE. "
    "If no entities are found, output an empty list [].\n"
    "Output ONLY valid JSON. No explanation."
)

ner_samples = []
for row in ds:
    text = row["text"]
    entities = row["entities"]

    if not text or len(text.strip()) < 20:
        continue

    # 建構 ground truth output
    output_entities = []
    for ent in entities:
        span = text[ent["start_offset"]:ent["end_offset"]]
        output_entities.append({
            "entity": span,
            "label": ent["label"],
            "text_span": f"[{ent['start_offset']}:
{ent['end_offset']}]"
        })

    # 包含有實體和無實體的樣本（讓模型學會回空 []）
    ner_samples.append({
        "instruction": SYSTEM_NER,
        "input": text,
        "output": json.dumps(output_entities, ensure_ascii=False)
```

```
    })

print(f"Task 1 (NER) 樣本數: {len(ner_samples)}")

# ============================================================
# Task 2: CTI 分類 — 判斷威脅類型與嚴重性
# ============================================================
SYSTEM_CLASSIFY = (
    "You are a Cyber Threat Intelligence (CTI) analyst. "
    "Analyze the given text and output a JSON object with:\n"
    "1. \"threat_type\": one of [ransomware, cryptomining, phishing,
"
    "apt, botnet, banking_trojan, exploit, data_breach, other]\n"
    "2. \"severity\": one of [critical, high, medium, low,
informational]\n"
    "3. \"summary\": a one-sentence summary of the threat
described\n"
    "4. \"iocs_mentioned\": list of any IOC types mentioned "
    "(e.g., hash, ip, domain, url, email)\n"
    "Output ONLY valid JSON. No explanation."
)

def infer_threat_type(text, entities):
    """
    根據文本關鍵字和實體標籤推斷威脅類型
    這是一個啟發式方法，你可以根據需求調整或用 LLM 預標註
    """
    text_lower = text.lower()
    entity_labels = [e["label"] for e in entities]
    entity_names = []
    for e in entities:
        try:

entity_names.append(text[e["start_offset"]:e["end_offset"]].lower())
        except (IndexError, KeyError):
            pass

    if any(kw in text_lower for kw in ["ransomware", "ransom",
"encrypt files", "decrypt"]):
        return "ransomware"
    if any(kw in text_lower for kw in ["mining", "miner", "monero",
"cryptocurrency mining", "coin min"]):
        return "cryptomining"
    if any(kw in text_lower for kw in ["phishing", "spear phishing",
"social engineering"]):
        return "phishing"
    if any(kw in text_lower for kw in ["apt", "advanced persistent",
"nation-state", "espionage"]):
        return "apt"
    if any(kw in text_lower for kw in ["botnet", "c2", "command and
control", "c&c"]):
        return "botnet"
    if any(kw in text_lower for kw in ["banking trojan", "credential
```

```
theft", "account takeover"]):
        return "banking_trojan"
    if any(kw in text_lower for kw in ["exploit", "vulnerability",
"cve-", "zero-day", "0day"]):
        return "exploit"
    if any(kw in text_lower for kw in ["data breach", "leak",
"exfiltrat"]):
        return "data_breach"
    return "other"

def infer_severity(text, entities):
    text_lower = text.lower()
    if any(kw in text_lower for kw in ["critical", "severe",
"urgent", "zero-day", "actively exploited"]):
        return "critical"
    if any(kw in text_lower for kw in ["serious", "significant",
"widespread", "major"]):
        return "high"
    if any(kw in text_lower for kw in ["moderate", "limited"]):
        return "medium"
    if any(kw in text_lower for kw in ["minor", "low risk"]):
        return "low"
    return "informational"

def detect_ioc_types(text):
    iocs = []
    text_lower = text.lower()
    if any(kw in text_lower for kw in ["sha256", "sha1", "md5",
"hash"]):
        iocs.append("hash")
    if any(kw in text_lower for kw in ["ip address", "ip:", "c2
ip"]):
        iocs.append("ip")
    if any(kw in text_lower for kw in ["domain", "hostname", ".com",
".net", ".org"]):
        iocs.append("domain")
    if any(kw in text_lower for kw in ["url", "http://",
"https://"]):
        iocs.append("url")
    if any(kw in text_lower for kw in ["email", "@"]):
        iocs.append("email")
    return iocs

classify_samples = []
for row in ds:
    text = row["text"]
    entities = row["entities"]

    if not text or len(text.strip()) < 30:
        continue

    threat_type = infer_threat_type(text, entities)
    severity = infer_severity(text, entities)
```

```
    iocs = detect_ioc_types(text)

    # 簡易摘要：取前 100 字 + 威脅類型（正式做法應用 LLM 生成）
    summary = text[:120].replace("\n", " ").strip()
    if len(text) > 120:
        summary += "..."

    output = {
        "threat_type": threat_type,
        "severity": severity,
        "summary": summary,
        "iocs_mentioned": iocs
    }

    classify_samples.append({
        "instruction": SYSTEM_CLASSIFY,
        "input": text,
        "output": json.dumps(output, ensure_ascii=False)
    })

print(f"Task 2 (分類) 樣本數: {len(classify_samples)}")

# ============================================================
# 合併 & 存檔
# ============================================================
all_samples = ner_samples + classify_samples
print(f"總樣本數: {len(all_samples)}")

# 全量
with open("cti_sft_all.json", "w", encoding="utf-8") as f:
    json.dump(all_samples, f, ensure_ascii=False, indent=2)

# 分別存檔（方便分群實驗）
with open("cti_sft_ner.json", "w", encoding="utf-8") as f:
    json.dump(ner_samples, f, ensure_ascii=False, indent=2)

with open("cti_sft_classify.json", "w", encoding="utf-8") as f:
    json.dump(classify_samples, f, ensure_ascii=False, indent=2)

# 分割 train / val
import random
random.seed(42)
random.shuffle(all_samples)
split_idx = int(len(all_samples) * 0.9)
train_data = all_samples[:split_idx]
val_data = all_samples[split_idx:]

with open("cti_sft_train.json", "w", encoding="utf-8") as f:
    json.dump(train_data, f, ensure_ascii=False, indent=2)

with open("cti_sft_val.json", "w", encoding="utf-8") as f:
    json.dump(val_data, f, ensure_ascii=False, indent=2)
```

```
print(f"Train: {len(train_data)}, Val: {len(val_data)}")
print("轉換完成！")
```

三、Llama-Factory 設定與微調

Mac MLX 環境

安裝 MLX 環境

```
# 建立虛擬環境
python3 -m venv ~/mlx-cti
source ~/mlx-cti/bin/activate

# 安裝 mlx-lm（包含訓練和推論功能）
pip install mlx-lm

# 確認安裝成功
python -c "import mlx.core as mx; print(f'MLX OK, default device:
{mx.default_device()}')"
# 應該輸出: MLX OK, default device: Device(gpu, 0)

# 安裝其他必要工具
pip install datasets huggingface_hub
```

下載原始資料

```
# 下載 all.jsonl
curl -L -o ~/mlx-cti/all.jsonl \
  https://huggingface.co/datasets/mrmoor/cyber-threat-
intelligence/resolve/main/all.jsonl
```

轉換資料為 MLX 格式

MLX 的 SFT 訓練需要 JSONL 格式，且使用 chat messages 格式：

```
{"messages": [{"role": "system", "content": "..."}, {"role": "user",
"content": "..."}, {"role": "assistant", "content": "..."}]}
```

建立轉換腳本 `convert_cti_to_mlx.py` ：

```
#!/usr/bin/env python3
"""
convert_cti_to_mlx.py

將 all.jsonl 轉換為 MLX-LM SFT 訓練格式（chat messages JSONL）

使用方式:
    python convert_cti_to_mlx.py

輸出目錄:  data/
    ├── train.jsonl       90% 訓練集
    ├── valid.jsonl       10% 驗證集
    ├── train_ner.jsonl       僅 NER 訓練集
    ├── valid_ner.jsonl       僅 NER 驗證集
    ├── train_classify.jsonl  僅分類訓練集
    └── valid_classify.jsonl  僅分類驗證集
"""
import json
import random
import os

INPUT_FILE = "all.jsonl"
OUTPUT_DIR = "data"
os.makedirs(OUTPUT_DIR, exist_ok=True)

# ============================================================
# 讀取 JSONL
# ============================================================
raw_data = []
with open(INPUT_FILE, "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if line:
            try:
                raw_data.append(json.loads(line))
            except json.JSONDecodeError:
                pass

print(f"載入 {len(raw_data)} 筆原始資料")

# 統計 entity labels
label_count = {}
for row in raw_data:
    for ent in row.get("entities", []):
        label = ent.get("label", "UNKNOWN")
        label_count[label] = label_count.get(label, 0) + 1
print(f"Entity labels: {label_count}")

# ============================================================
# 定義 system prompts
# ============================================================
SYSTEM_NER = (
```

```
    "You are a Cyber Threat Intelligence (CTI) analyst. "
    "Extract all named entities from the given text. "
    "For each entity, output a JSON object with keys: "
    "\"entity\", \"label\", \"text_span\". "
    "Supported labels: malware, attack-pattern, threat-actor, "
    "identity, location, TIME, SOFTWARE, tools. "
    "If no entities are found, output an empty list []. "
    "Output ONLY valid JSON. No explanation."
)

SYSTEM_CLASSIFY = (
    "You are a Cyber Threat Intelligence (CTI) analyst. "
    "Analyze the given text and output a JSON object with: "
    "\"threat_type\": one of [ransomware, cryptomining, phishing, "
    "apt, botnet, banking_trojan, exploit, data_breach, other], "
    "\"severity\": one of [critical, high, medium, low,
informational], "
    "\"summary\": a one-sentence summary, "
    "\"iocs_mentioned\": list of IOC types (hash, ip, domain, url,
email). "
    "Output ONLY valid JSON. No explanation."
)

# ============================================================
# 啟發式分類函式
# ============================================================
def infer_threat_type(text):
    t = text.lower()
    if any(k in t for k in ["ransomware", "ransom", "encrypt files",
"locker", "decrypt"]):
        return "ransomware"
    if any(k in t for k in ["mining", "miner", "monero", "crypto-
mining", "coin min"]):
        return "cryptomining"
    if any(k in t for k in ["phishing", "spear phishing", "social
engineering"]):
        return "phishing"
    if any(k in t for k in ["apt", "advanced persistent",
"espionage", "nation-state"]):
        return "apt"
    if any(k in t for k in ["botnet", "c2 ", "command and control",
"c&c"]):
        return "botnet"
    if any(k in t for k in ["banking trojan", "credential theft",
"account takeover"]):
        return "banking_trojan"
    if any(k in t for k in ["exploit", "vulnerability", "cve-",
"zero-day"]):
        return "exploit"
    if any(k in t for k in ["data breach", "exfiltrat", "data
leak"]):
        return "data_breach"
    return "other"
```

```
def infer_severity(text):
    t = text.lower()
    if any(k in t for k in ["critical", "severe", "urgent", "zero-
day"]):
        return "critical"
    if any(k in t for k in ["serious", "significant", "widespread",
"major"]):
        return "high"
    if any(k in t for k in ["moderate", "limited"]):
        return "medium"
    if any(k in t for k in ["minor", "low risk"]):
        return "low"
    return "informational"

def detect_iocs(text):
    t = text.lower()
    iocs = []
    if any(k in t for k in ["sha256", "sha1", "md5", "hash"]):
iocs.append("hash")
    if any(k in t for k in ["ip address", "c2 ip"]):
iocs.append("ip")
    if any(k in t for k in ["domain", "hostname"]):
iocs.append("domain")
    if any(k in t for k in ["url", "http://", "https://"]):
iocs.append("url")
    return iocs

# ============================================================
# 產生 MLX chat 格式樣本
# ============================================================
ner_samples = []
classify_samples = []

for row in raw_data:
    text = row.get("text", "").strip()
    entities = row.get("entities", [])

    if not text or len(text) < 20:
        continue

    # --- NER 樣本 ---
    output_entities = []
    for ent in entities:
        s, e = ent.get("start_offset", 0), ent.get("end_offset", 0)
        original_text = row.get("text", "")
        if 0 <= s < e <= len(original_text):
            output_entities.append({
                "entity": original_text[s:e],
                "label": ent["label"],
                "text_span": f"[{s}:{e}]"
            })
```

```
    ner_samples.append({
        "messages": [
            {"role": "system", "content": SYSTEM_NER},
            {"role": "user", "content": text},
            {"role": "assistant", "content":
json.dumps(output_entities, ensure_ascii=False)}
        ]
    })

    # --- 分類樣本 ---
    if len(text) >= 30:
        summary = text[:120] + ("..." if len(text) > 120 else "")
        output = {
            "threat_type": infer_threat_type(text),
            "severity": infer_severity(text),
            "summary": summary,
            "iocs_mentioned": detect_iocs(text)
        }
        classify_samples.append({
            "messages": [
                {"role": "system", "content": SYSTEM_CLASSIFY},
                {"role": "user", "content": text},
                {"role": "assistant", "content": json.dumps(output,
ensure_ascii=False)}
            ]
        })

print(f"NER 樣本: {len(ner_samples)}")
print(f"分類樣本: {len(classify_samples)}")

# 分類分布
type_dist = {}
for s in classify_samples:
    t = json.loads(s["messages"][2]["content"])["threat_type"]
    type_dist[t] = type_dist.get(t, 0) + 1
print(f"分類分布: {type_dist}")

# ============================================================
# 切分 & 寫出
# ============================================================
def split_and_write(samples, train_path, valid_path, ratio=0.9):
    random.seed(42)
    shuffled = samples.copy()
    random.shuffle(shuffled)
    idx = int(len(shuffled) * ratio)
    train, valid = shuffled[:idx], shuffled[idx:]

    with open(train_path, "w", encoding="utf-8") as f:
        for s in train:
            f.write(json.dumps(s, ensure_ascii=False) + "\n")

    with open(valid_path, "w", encoding="utf-8") as f:
        for s in valid:
```

```
            f.write(json.dumps(s, ensure_ascii=False) + "\n")

    print(f"  {train_path}: {len(train)} 筆")
    print(f"  {valid_path}: {len(valid)} 筆")

# 全量（NER + 分類混合）
all_samples = ner_samples + classify_samples
split_and_write(all_samples, f"{OUTPUT_DIR}/train.jsonl", f"
{OUTPUT_DIR}/valid.jsonl")

# 分群
split_and_write(ner_samples, f"{OUTPUT_DIR}/train_ner.jsonl", f"
{OUTPUT_DIR}/valid_ner.jsonl")
split_and_write(classify_samples, f"
{OUTPUT_DIR}/train_classify.jsonl", f"
{OUTPUT_DIR}/valid_classify.jsonl")

print(f"\n轉換完成！所有檔案在 {OUTPUT_DIR}/ 目錄下")
```

執行：

```
cd ~/mlx-cti
python convert_cti_to_mlx.py
```

LoRA 訓練

5.1 全量資料訓練

```
cd ~/mlx-cti

python -m mlx_lm.lora \
  --model mlx-community/gemma-4-e4b-it-8bit \
  --train \
  --data ./data \
  --iters 600 \
  --batch-size 4 \
  --learning-rate 1e-5 \
  --num-layers 8 \
  --steps-per-eval 50 \
  --steps-per-report 10 \
  --adapter-path ./adapters/sft-all \
  --max-seq-length 2048
```

```
(mlx_env) hostlocal@hostdeMacBook-Pro llm_fine_tune % python -m mlx_lm.lora \
  --model mlx-community/Qwen2.5-7B-Instruct-4bit \
  --train \
  --data ./data \
  --iters 600 \
  --batch-size 4 \
  --max-seq-length 2048 \
  --num-layers 8 \
  --learning-rate 1e-5 \
  --steps-per-eval 50 \
  --adapter-path ./adapters/sft-all
Calling `python -m mlx_lm.lora...` directly is deprecated. Use `mlx_lm.lora...` or `python -m mlx_lm lora ...` instead.
Loading pretrained model
Downloading (incomplete total...): 100%|                                                | 605/605 [00:00<00:00, 2.70kB/sWarnin
g: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
Fetching 9 files: 100%|                                                                 | 9/9 [02:51<00:00, 19.01s/it]
Download complete: : 4.30GB [02:51, 45.1MB/s]             Loading datasets              | 4/9 [02:51<04:55, 59.14s/it]
Download complete: : 4.30GB [02:51, 25.1MB/s]
Training
Trainable parameters: 0.076% (5.767M/7615.617M)
Starting training..., iters: 600
Calculating loss...: 100%|                                                              | 25/25 [00:55<00:00,  2.24s/it]
Iter 1: Val loss 3.059, Val took 55.908s
Iter 10: Train loss 2.176, Learning Rate 1.000e-05, It/sec 0.275, Tokens/sec 208.321, Trained Tokens 7584, Peak mem 11.191 GB
Iter 20: Train loss 0.875, Learning Rate 1.000e-05, It/sec 0.305, Tokens/sec 221.487, Trained Tokens 14847, Peak mem 11.191 GB
Iter 30: Train loss 0.708, Learning Rate 1.000e-05, It/sec 0.318, Tokens/sec 239.336, Trained Tokens 22374, Peak mem 11.191 GB
Iter 40: Train loss 0.573, Learning Rate 1.000e-05, It/sec 0.314, Tokens/sec 224.322, Trained Tokens 29516, Peak mem 11.191 GB
```

每個參數說明：

參數	值	說明
`--model`	mlx-community/...	基底模型。帶 -4bit 會自動 QLoRA
`--train`	—	啟用訓練模式
`--data`	./data	資料目錄，裡面要有 `train.jsonl` 和 `valid.jsonl`
`--iters`	600	訓練步數。~9K 樣本 × 3 epoch ÷ batch_size 4 ≈ 7000+，但 600 步可先看效果
`--batch-size`	4	每步處理幾個樣本。記憶體不夠就降到 2 或 1
`--learning-rate`	1e-5	學習率。QLoRA 建議 1e-5 ~ 2e-5
`--num-layers`	8	對最後 8 層加 LoRA。越多學得越多但越慢
`--steps-per-eval`	50	每 50 步做一次驗證
`--adapter-path`	./adapters/sft-all	adapter 存放路徑
`--max-seq-length`	2048	最大序列長度

```
python -m mlx_lm.generate \
  --model mlx-community/Qwen2.5-7B-Instruct-4bit \
  --adapter-path ./adapters/sft-all \
  --prompt "You are a CTI analyst. Analyze: The ransomware strain LockBit 3.0 was observed encrypting files across 500 endpoints in a healthcare network. Output JSON with threat_type, severity, summary, iocs_mentioned." \
  --max-tokens 512
```

```
{
  "threat_type": "ransomware",
  "severity": "critical",
  "summary": "The ransomware strain LockBit 3.0 was observed encrypting files across 500 endpoints in a healthcare network. This attack highlights the critical nature of the threat, as it affects a large number of endpoints and poses a significant risk to the network’s availability and the safety of patient data.",
  "iocs_mentioned": {
    "threat_type": "LockBit 3.0",
    "endpoint_count": "500",
    "industry": "healthcare"
  }
}
```

````
(mlx_env) hostlocal@hostdeMacBook-Pro llm_fine_tune % python -m mlx_lm.generate \
  --model mlx-community/Qwen2.5-7B-Instruct-4bit \
  --adapter-path ./adapters/sft-all \
  --prompt "You are a CTI analyst. Analyze: The ransomware strain LockBit 3.0 was observed encrypting files across 500 endpoints in a healthcare network. Output JSON with threat_type, severity, summary, iocs_mentioned." \
  --max-tokens 512
<frozen runpy>:128: RuntimeWarning: 'mlx_lm.generate' found in sys.modules after import of package 'mlx_lm', but prior to execution of 'mlx_lm.generate'; this may result in unpredictable behaviour
Calling `python -m mlx_lm.generate...` directly is deprecated. Use `mlx_lm.generate...` or `python -m mlx_lm generate ...` instead.
Fetching 9 files: 100%|████████████████████████████████████████████████████████████| 9/9 [00:00<00:00, 149796.57it/s]
Download complete: : 0.00B [00:00, ?B/s]             ==========                                                | 0/9 [00:00<?, ?it/s]
```json
{
  "threat_type": "ransomware",
  "severity": "critical",
  "summary": "The ransomware strain LockBit 3.0 was observed encrypting files across 500 endpoints in a healthcare network. This attack highlights the critical nature of the threat, as it affects a large number of endpoints and poses a significant risk to the network’s availability and the safety of patient data.",
  "iocs_mentioned": {
    "threat_type": "LockBit 3.0",
    "endpoint_count": "500",
    "industry": "healthcare"
  }
}
```
````

常見錯誤

錯誤	解決
`Killed` 或記憶體不足	降低 `--batch-size` 到 1； 或換更小模型
`train.jsonl not found`	確認 `--data` 目錄下有 train.jsonl 和 valid.jsonl
`KeyError: 'messages'`	你的 JSONL 格式不對，必須用 `{"messages": [...]}` 格式
速度太慢	用 4-bit 模型（-4bit 後綴）；降低 `--max-seq-length`

迭代優化

觀察指標與調整

現象	調整
Train loss 不降	`--learning-rate` 從 1e-5 改 5e-5
Train loss 震盪	`--learning-rate` 從 1e-5 改 5e-6
Val loss 反彈上升	減少 `--iters`（提早停止）；或增加 `--num-layers`
JSON 格式不穩定	增加 `--num-layers` 到 16；增加 `--iters`
幻覺嚴重	增加負面樣本（output 為 `[]` 或 `other` 的樣本）
記憶體不夠	`--batch-size 1` ； `--grad-checkpoint`

```
python -m mlx_lm.lora \
  --model mlx-community/Qwen2.5-7B-Instruct-4bit \
  --train \
  --data ./data \
  --iters 600 \
  --batch-size 4 \
  --learning-rate 1e-5 \
  -c lora_config.yaml \
  --adapter-path ./adapters/sft-all-v2
```