**HITL-асистент для наукових досліджень на основі генеративних мовних моделей: кейс дешифрування історичного шифру УПА**

**Автор:** Михайло Шелест  
**Контакти:** mishel3141@gmail.com  
**Дата:** 2025-09-02

**Анотація**

У роботі представлено прототип HITL-асистента (Human-in-the-Loop), який поєднує інтуїцію дослідника та можливості генеративної мовної моделі (LLM) для розв’язання задач дешифрування історичних шифрів. Підхід апробовано на публічному кейсі шифру УПА. Запропоновано формалізацію станів, пошук із **beam search+UCB**, інтегральну функцію оцінки **Score(s)** та спеціалізований критерій відповідності **K⁺** для ранжування поточних елементів ключа. У порівнянні з підходами «Human-only» та «AI-only» прототип HITL зменшує час до розв’язку (~10×), підвищує частку правильних розшифровок у top-N (до 85%) і знижує ресурсні витрати.

**1. Вступ**

**1.1. Ціль проекту**

Створити та випробувати **HITL-асистента** для задач пошуку у великому просторі й неповною інформацією, зокрема для дешифрування шифра, що використовували в Української повстанської армії.

**1.2. Мотивація вибору теми**

* Поточний стан розвитку ШІ, зокрема АІ-агентів, значно обмежує круг їх застосування. Наприклад, питання у сфери критичних технологій, кіберзахисту, наукових досліджень ще зарано покладати на плечі автоматичних АІ-агентів. Тут на допомогу приходить технологія Human-in-the-Loop, яку я вирішив апробувати для досягнення мети.
* Архівні криптограми мають **мало даних** і **великий простір ключів** — класичний, чисто статистичний підхід криптоаналізу дуже повільний, а технологія **AI-only** без контролю криптоаналітика часто «помиляється на семантиці».
* Необхідна **координація «людина↔ШІ»**: людина задає обмеження та перевіряє сенс, LLM швидко генерує варіанти, а статистика їх фільтрує.
* Кейс УПА — реалістичний тест: публічний шифр без підказок, історично значущий.

**1.3. Короткий огляд релевантних робіт**

У цій роботі ми поєднуємо ці три відоми напрями у практичній, відтворюваній схемі:

* класичний криптоаналіз, який спирається на *n*-грамну статистику та логічне моделювання;
* сучасні LLM, які придатні для генерації/фільтрації гіпотез і семантичної валідації;
* а також HITL-парадигму, яка інтегрує експерта-криптоаналітика в замкнений цикл ML.

**2. Процес роботи**

**2.1. Дані (збір і підготовка)**

* **Вхід:** шифртекст табличної підстановки (моноалфавіт), мова — українська (гіпотеза). Оцінка ключового простору ≈ 2.76×
* **Еталонна статистика:** частоти біграм для української (для **Score(s)** та **K⁺**).
* **Обмеження:** словникові/морфологічні підказки, історичний контекст (людина задає та уточнює в циклі).
* **Збереження:** база гіпотез (JSON/CSV) із полями: ключ, текст, метрики, прапор «схвалено людиною».

**2.2. Модель та алгоритми (навчання/адаптація)**

* **LLM (генеративний модуль):** використовується для пропозиції модифікацій ключа/таблиці, генерації кандидатів тексту, семантичної оцінки. Модель **не донавчалася**, використано **prompt-інженерію** й обмеження.
* **Пошук:** **beam search** зі сталою шириною **B** та політикою вибору дій **UCB**:

UCB(*a*) = + *c* ,

де — середня якість дії *a*, *N* — загальна кількість ітерацій, ​ — кількість виборів дії *a*, *c* — коефіцієнт балансування між «exploration» і «exploitation».

* **Складність:** приблизно O(B⋅d⋅∣A∣), де d — глибина/ітерації, ∣A∣ — число доступних дій.

**2.3. Інтеграція в прототип**

* **Архітектура:** Людина ↔ GPT ↔ Статистика ↔ База гіпотез (основний та допоміжний контури).
* **Логи:** кожна ітерація зберігає стан, оцінки, пояснення LLM, рішення людини.
* **Інтерфейс:** CLI/ноутбук (кнопки/команди «згенерувати → оцінити → переглянути top-N → підтвердити/відхилити → оновити обмеження»).

**3. Технічний підхід**

**3.1. Формалізація станів**

Стан на кроці *t*: , де ​ — поточний ключ; ​ — множина гіпотез тексту; ​ — статистика (уні/бі/тріграми, порушення обмежень).

**3.2. Функція оцінки гіпотез**

)

Вектори ваг , , калібрували grid/Bayes-пошуком на валідаційній підмножині гіпотез.

**3.3. Критерій відповідності K⁺ (для табличної підстановки)**

*K*⁺= *α*₁ · *r* + *α*₂ · *I* + *α*₃ · *core* + *α*₄ · *massiveness*,

де *r* — кореляція форм розподілу, *I* — інтерсекція, *core* — вклад «ядерних» літер мови, *massiveness* — відносна кількість спостережень у стовпці. Служить другим фільтром для стабільної селекції.

**3.4. Основний цикл (псевдокод)**

1. ініціалізація , B, λ;
2. генерація модифікацій ключа k′ (LLM) → дешифрування → оцінка **Score**;
3. ранжування кандидатів (статистика, **K⁺**);
4. **UCB + beam**: вибір top-B станів;
5. перегляд людиною top-N → підтвердження/відхилення → ін’єкція обмежень;
6. перевірка критеріїв зупинки (якість/ресурс/стабілізація).

**4. Роль генеративного ШІ в циклі HITL**

У нашій системі LLM (GPT-клас) виконує не роль «чорної скриньки», а керований **евристичний генератор** і **семантичний фільтр** у зв’язці зі статистикою. Функції:

* **Генерація дій/кандидатів**: пропонує локальні модифікації ключа (заміни, перестановки, комбіновані трансформації) та чернеткові розшифрування.
* **Семантична валідація**: оцінює природність фрагментів (PLM-терм у Score), допомагає відсікти граматично/семантично безглузді варіанти.
* **Протоколювання знань**: фіксує обґрунтування гіпотез, дозволяючи людині швидко вводити обмеження й коригувати пошук.

**Абляційні спостереження.** Ми порівняли режими, вимикаючи окремі компоненти:

* без LLM (тільки n-грами): збільшується час до збіжності; у топ-N з’являються формально частотні, але семантично «порожні» гіпотези;
* без статистики (тільки LLM): зростає кількість «переконливих» але хибних версій; пошук застрягає у локальних патернах;
* **LLM + статистика (наш HITL)**: найшвидше зближення; стабільніше ранжування; менше «сміттєвих» кандидатів у топ-N.

Практичні налаштування для керованості LLM:

* **жорсткі промпти і рольові інструкції** (уникаємо «балакучості», просимо лише дії над ключем + коротку мотивацію);
* **температура/топ-p** низькі на етапі уточнення (експлуатація), вищі на етапі розвідки (exploration);
* **логування промптів/відповідей** і фіксація seed (де можливо) для відтворюваності.

Сумарно, генеративний ШІ підвищує **ефективність** (менше ітерацій до розв’язку), **якість** (вищий відсоток правильних серед топ-N) і **керованість** (прозорі обґрунтування для рішень людини) у порівнянні з режимами Human-only та AI-only. Це узгоджується з нашими результатами у розділі 6.

**5. Виклики та їх вирішення**

**Обмежені дані.** Короткі тексти «шумлять» статистику → ввели **K⁺** і ручні обмеження.  
**Великий простір ключів.** Локальні оптимуми → **UCB** тримає баланс explore/exploit.  
**Керованість LLM.** «Базікання» замість користі → «жорсткі» промпти, рольові інструкції, перевірка обмежень.

**Репродуктивність.** LLM стохастичний → логування промптів/відповідей, фіксація seed там, де можливо, збереження всіх топ-N.

**Ресурси.** Працює на CPU; оптимізовано розміри beam, кеші статистики.

**6. Результати**

**6.1. Таблиця 1 - Якісні показники**

| **Показник** | **Human-only** | **AI-only** | **Human–AI** |
| --- | --- | --- | --- |
| Час до розв’язку (год) | 8 | >10 (нема розв’язку) | **1** |
| К-ть ітерацій | ~50 | >1000 | ~20 |
| Гіпотез згенеровано | ~50 | >2000 | ~35 |
| Частка правильних у top-N | 60% | 0% | **85%** |
| Помилкових у top-20 | 20% | 100% | **10%** |
| Середній критерій K⁺ | 0.62 | 0.15 | **0.87** |

**6.2. Таблиця 2 - Ресурси**

| **Підхід** | **CPU час (год)** | **Пам’ять (MB)** | **Енергія (≈Wh)** |
| --- | --- | --- | --- |
| Human-only | — | — | — |
| AI-only | 15 | 5200 | 180 |
| Human–AI (HITL) | **3** | **2100** | **55** |

*Інтерпретація.* HITL відповідає за **істотний виграш** у часі та якості при менших витратах ресурсів. Ключовими факторами стали поєднання **Score(s)** + **K⁺**, керована генерація LLM та людська ін’єкція обмежень.

**6.3. Таблиця 3 - Абляційний аналіз компонентів HITL (менші = краще для T, d, Err; більші = краще для Acc, K⁺)**

| **Режим / Конфігурація** | **Час до розв’язку, T (год)** | **Ітерації, d** | **Acc (топ-N, %)** | **Err (топ-N, %)** | **K⁺** | **CPU-год** | **Пам’ять (MB)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Лише статистика (n-gram, без LLM)** | 4.5 | 120 | 35 | 35 | 0.55 | 1.5 | 900 |
| **Лише LLM (без статистики)** | >10 (нема збіжності) | >1000 | 0 | 100 | 0.20 | 15 | 5200 |
| **HITL без UCB** (жадібний відбір) | 1.6 | 40 | 72 | 18 | 0.80 | 4 | 2400 |
| **HITL без beam** (B=1) | 2.1 | 55 | 65 | 22 | 0.76 | 5 | 2600 |
| **HITL повний** (beam B=5 + UCB, *c*=1.4) | **1.0** | **20** | **85** | **10** | **0.87** | **3** | **2100** |

*Примітки:*

* **Acc** — частка правильних рішень серед топ-N; **Err** — частка помилкових у топ-N; **K⁺** — інтегральний критерій відповідності.
* CPU-год і пам’ять — орієнтовні, для зіставлення режимів у нашому середовищі.

**7. Висновки та майбутня робота**

**Підсумок.** HITL-підхід для криптоаналізу довів ефективність: швидкість (~10×), точність (до 85% у top-N), прозорість (логи/метрики) і керованість (людські обмеження).  
**Обмеження.** Великі алфавіти/довгі ключі збільшують складність; дуже короткі тексти — нестабільна статистика.

**Що далі.**

* активне навчання підказок для LLM;
* динамічний beam (адаптація B до прогресу);
* краща морфологічна перевірка (узгодження відмін/родів);
* інтерфейс з інтерактивним графом гіпотез;
* пакетування як Python-бібліотеки/CLI.

**8. Інструкції з запуску (README коротко)**

**8.1. Залежності**

* Python 3.10+ (CPU достатньо)
* Основні пакети: numpy, pandas, scipy, tqdm, networkx, rapidfuzz, python-Levenshtein, (за потреби: transformers для локальної мови-оцінки або API-клієнт до LLM).

pip install -r requirements.txt

**8.2. Структура репозиторію**

hitl-upa/

├─ data/

│ ├─ ciphertexts/upa\_coin.txt

│ ├─ stats/uk\_unigram.csv, uk\_bigram.csv, uk\_trigram.csv

│ └─ dictionaries/uk\_words.txt

├─ src/

│ ├─ hitl\_agent.py # основний цикл HITL

│ ├─ generators.py # модифікації ключа LLM/евристики)

│ ├─ scoring.py # Score(s)

│ ├─ kplus.py # критерій K⁺

│ ├─ search.py # beam + UCB

│ ├─ constraints.py # робота з обмеженнями

│ └─ utils.py

├─ notebooks/

│ └─ demo\_pipeline.ipynb

├─ results/

│ ├─ logs/iter\_\*.jsonl

│ └─ topN.csv

├─ requirements.txt

└─ README.md

**8.3. Приклади запуску**

**Демо-прохід з дефолтними параметрами:**

python -m src.hitl\_agent \

--cipher data/ciphertexts/upa\_coin.txt \

--beam 8 --max\_iters 200 \

--lambda "0.33,0.33,0.34" \

--ucb\_c 1.2 \

--topN 20 \

--out results/topN.csv

**Оновлення обмежень у процесі (CLI):**

:accept 3,7,11 # схвалити кандидати з індексами

:block "не\_\*\_слово" # регулярка для відсіву

:hint "можливе слово 'ЛУНАТИМЕ'"

:continue

**Оцінка підсумків (метрики):**

python -m src.evaluate --in results/topN.csv --refs data/dictionaries/uk\_words.txt

**9. Додатки**

**9.1. Псевдокод основного циклу**

init k0, M0, G0, beam B, lambdas, constraints

for t in 1..d:

C = []

for s in top\_B\_states:

actions = propose\_actions(s) # LLM + евристики

for a in actions:

s' = apply(s, a) # оновити ключ/таблицю

text' = decrypt(s')

score = Score(text', G', constraints)

C.append((s', score))

ranked = rank\_by(score + Kplus) # статистика + K⁺

top = UCB\_select(ranked, B) # explore/exploit

human\_review(top[:N]) # підтвердження/відхилення

update\_constraints()

if stop\_criteria(): break

return best\_state

**9.2. Місця для ілюстрацій**

* Схема архітектури HITL
* Приклад блок-схеми «Ініціалізація → Основний цикл → Результат»
* Фото монети/скрін фінального розшифрування
* Таблиці результатів (Table 1, Table 2)

**9.3. Джерела (посилання для фінальної версії)**

* СЗРУ — результати конкурсу на розшифрування вислову на пам’ятній монеті.
* Аналітичний розбір шифру та шифрувальної таблиці.

**10. Ліцензування та дані**

* Код: MIT (за бажанням).
* Дані/частоти n-грам — вільні або з посиланням на джерело.
* Репозиторій GitHub: додати README з інструкціями (розділ 8), скрипти відтворення та приклади логів.