Projekt: Kryptoanalýza klasických šifier pomocou veľkých jazykových modelov

Technická dokumentácia

6. júna 2025

Obsah

1 Anotácia

Tento dokument predstavuje technickú dokumentáciu projektu zameraného na kryptografickú analýzu klasických šifrovacích metód s využitím veľkých jazykových modelov (LLM). Cieľom projektu je výskum a implementácia inovatívnych kryptoanalytických postupov, ktoré kombinujú tradičné metódy lúštenia šifier s modernými technikami strojového učenia. Vytvorený systém umožňuje dešifrovanie klasických šifier (Cézarova šifra, substitučné a Vigenèrovo šifrovanie, transpozičné šifry a pod.) pomocou LLM modelov, pričom poskytuje webovú aplikáciu na nahrávanie zašifrovaných textov, sledovanie výsledkov dešifrovania a správu používateľov. Dokumentácia popisuje architektúru mikroservisného systému, návrh databázy, integráciu LLM, implementačné detaily, webovú aplikáciu, bezpečnostné mechanizmy, ukážky SQL dotazov, testovanie, históriu vývoja, budúce vylepšenia, nasadenie a údržbu, a záver s zhodnotením prínosov projektu.

2 Úvod

2.1 História kryptoanalýzy

Kryptografia a kryptoanalýza majú bohatú históriu siahajúcu tisíce rokov do minulosti. Už v staroveku sa používali jednoduché šifrové techniky – napríklad v Spartskom vojsku transpozičná šifra so skytalou (kožený prúžok omotaný okolo tyče) na maskovanie textu. V 1. storočí pred n.l. zaviedol Julius Caesar jednoduchú substitučnú šifru (tzv. Cézarova šifra), v ktorej sú písmená posunuté o stanovený počet pozícií v abecede. Po stáročia sa vývoj šifier a metód ich lúštenia (kryptoanalýzy) uberal ruka v ruke – významným medzníkom bol 9. storočie n.l., keď arabský vedec Al-Kindí opísal metódu frekvenčnej analýzy na lámanie monoalfabetických substitučných šifier. Frekvenčná analýza využíva štatistické rozloženie písmen v jazyku a umožnila po prvý raz systematicky lúštiť zašifrované správy bez znalosti kľúča. Ďalší rozvoj priniesli polyalfabetické šifry (napr. Vigenèrova šifra zo 16. storočia), ktoré frekvenčnú analýzu sťažili – ich prelomenie však umožnili nové postupy (napr. Friedmanov index zhody pre určenie dĺžky kľúča Vigenèrovej šifry). V 20. storočí počas 2. svetovej vojny dosiahol pokrok kryptoanalýzy vrchol v prelomení nemeckej šifry Enigma, na čom sa podieľali matematické postupy aj prvé elektromechanické počítače. História teda ukazuje neustály súboj medzi tvorcami šifier a kryptoanalytikmi – každá nová šifra viedla k vynálezu nových metód na jej prelomenie.

2.2 Prečo LLM v kryptoanalýze

V súčasnosti do tejto oblasti vstupujú metódy umelej inteligencie a strojového učenia. Veľké jazykové modely (LLM) ako GPT-3 alebo GPT-4 preukázali schopnosť analyzovať a generovať prirodzený jazyk, a otvára sa otázka, do akej miery dokážu pomôcť pri lúštení zašifrovaných textov. Klasická kryptoanalýza je často formulovaná ako problém hľadania správneho kľúča alebo princípu šifry pomocou matematických a štatistických metód. LLM na to idú z iného smeru – ako modely jazyka sa snažia predpovedať najpravdepodobnejšie slová a vety, ktoré by mohli byť v texte. Tým pádom môžu využiť svoje znalosti o štruktúre jazyka na odhalenie vzorcov v šifrovanom texte.

3 Architektúra systému

Systém je navrhnutý ako moderná webová aplikácia s mikroservisnou architektúrou. Hlavné komponenty zahŕňajú:

- Webový server (Flask) – spracováva HTTP požiadavky, autentifikáciu a biznis logiku
- Databáza (PostgreSQL) – ukladá údaje o používateľoch, pokusoch a výsledkoch
- LLM modul – obsahuje modely a logiku pre dešifrovanie
- Webové rozhranie – poskytuje používateľské rozhranie (HTML, CSS, JavaScript)

4 Návrh databázy

4.1 Konceptuálny model

Návrh databázy prešiel troma úrovňami: konceptuálny model, logický model a fyzický model. V tejto sekcii najprv popíšeme konceptuálny ER model a entity, potom logický model s atribútmi a väzbami a nakoniec fyzickú realizáciu v PostgreSQL vrátane schém tabuliek.

4.1.1 Entity

- **Šifry** – reprezentujú druhy klasických šifier, ktoré systém eviduje a vie analyzovať. Každá šifra má:
  - Identifikátor
  - Názov
  - Historické obdobie pôvodu
  - Krajina/miesto vzniku
  - Princíp šifrovania
  - Ukážkový zašifrovaný text a plaintext
- **Modely** – predstavujú LLM modely použité na dešifrovanie:
  - Identifikátor
  - Názov modelu
  - Zameranie/typ
  - Verzia
- **Pokusy o dešifrovanie** – každá požiadavka používateľa:
  - ID pokusu
  - Referencia na používateľa
  - Referencia na šifru a model
  - Čas začiatku a konca

  - Indikátor úspechu
  - Percento správnosti
  - Zašifrovaný a dešifrovaný text
- **Výsledky dešifrovania** – detailné výstupy:
  - ID výsledku
  - Referencia na pokus
  - Vygenerovaný text
  - Miera podobnosti
  - Čitateľnosť výstupu
- **Manuálne korekcie** – ručné opravy výstupov:
  - ID korekcie
  - Referencia na výsledok
  - Korektor (používateľ)
  - Percento zmien
  - Finálny text
- **Používatelia** – registrovaní používatelia:
  - ID používateľa
  - Používateľské meno
  - E-mail
  - Hash hesla
  - Dátum vytvorenia účtu
  - OAuth identifikátor

4.1.2 Vzťahy medzi entitami

- Používateľ 1:N Pokusy o dešifrovanie
- Šifry 1:N Pokusy
- Modely 1:N Pokusy
- Pokusy 1:1/1:N Výsledky
- Výsledky 1:N Manuálne korekcie

4.2 Logický model

Logický model upresňuje detailnú štruktúru databázy. Pre implementáciu sme zvolili relačný databázový model v PostgreSQL.

4.2.1 Tabuľky a ich štruktúra

```sql
CREATE TABLE "Users" (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(256) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE "Ciphers" (
    cipher_id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL,
    historical_period VARCHAR(50),
    origin VARCHAR(50),
    encryption_principles TEXT,
    encrypted_text TEXT,
    plaintext TEXT
);

CREATE TABLE "Models" (
    model_id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL,
    type VARCHAR(50) NOT NULL,
    version VARCHAR(20) NOT NULL
);

CREATE TABLE "Decryption_Attempts" (
    attempt_id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES "Users"(user_id),
    cipher_id INTEGER REFERENCES "Ciphers"(cipher_id),
    model_id INTEGER REFERENCES "Models"(model_id),
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL,
    success BOOLEAN NOT NULL,
    correctness_percentage DECIMAL(5,2),
    encrypted_text TEXT NOT NULL,
    decrypted_text TEXT
);

CREATE TABLE "Decryption_Results" (
    result_id SERIAL PRIMARY KEY,
    attempt_id INTEGER REFERENCES "Decryption_Attempts"(attempt_id)
        ,
    model_output TEXT NOT NULL,
    similarity_measure DECIMAL(5,2),
    readability_level DECIMAL(5,2)
);

CREATE TABLE "Manual_Corrections" (
    correction_id SERIAL PRIMARY KEY,
```

```
    result_id INTEGER REFERENCES "Decryption_Results"(result_id),
    corrector INTEGER REFERENCES "Users"(user_id),
    changed_percentage DECIMAL(5,2),
    final_text TEXT NOT NULL
);
```

5 Implementačné detaily

5.1 Server a API endpointy

Server je implementovaný v jazyku Python s využitím frameworku Flask. Hlavné endpointy zahŕňajú:

- `/register`, `/login`, `/logout` – správa používateľov
- `/decrypt` – spracovanie požiadaviek na dešifrovanie
- `/attempts` – história pokusov
- `/correct` – uloženie manuálnych korekcií

5.2 Integrácia LLM

Pre prácu s jazykovými modelmi využívame knižnicu `transformers` od Hugging Face. Hlavné komponenty:

- Načítanie modelu: Používame model DistilGPT-2 pre jeho dobrý pomer výkon/veľkosť
- Tokenizácia: Vstupný text je tokenizovaný pomocou GPT-2 tokenizéra
- Generovanie: Model generuje najpravdepodobnejší otvorený text

5.3 Databázové operácie

Komunikácia s PostgreSQL databázou je realizovaná cez knižnicu `psycopg2`. Príklad SQL dotazu pre overenie prihlasovacích údajov:

```
SELECT user_id, username, email, password_hash
FROM "Users"
WHERE email = %s;
```

6 Webová aplikácia

6.1 Front-end a používateľské funkcie

Webová aplikácia poskytuje používateľsky prívetivé rozhranie na využívanie funkcionalít systému. Je navrhnutá tak, aby zjednodušila prácu kryptoanalytika.

6.2 Architektúra front-endu

Aplikácia využíva:

- Serverom renderované HTML šablóny (Flask `render_template`)
- Dynamické načítavanie dát cez JavaScript (AJAX)
- CSS framework Bulma pre moderný vzhľad
- Material Icons pre zobrazenie ikoniek

6.3 Autentifikácia a správa používateľov

- Prihlasovanie stráži Flask-Login
- Registrácia nového účtu vyžaduje meno, e-mail a heslo
- Integrovaná možnosť OAuth prihlásenia cez Google
- Heslá sú hashované algoritmom PBKDF2

6.4 Hlavná stránka – Dešifrovanie

Hlavné rozhranie je rozdelené do dvoch stĺpcov:

- Ľavý bočný panel
  - Navigačné tlačidlá
  - Timeline s históriou pokusov
  - Možnosti nastavení
- Pravý obsahový panel
  - Formulár na dešifrovanie textu
  - Výber šifry a modelu
  - Zobrazenie výsledkov

6.5 Tabulkové údaje

Zobrazuje históriu dešifrovaní vo forme tabuľky s nasledujúcimi stĺpcami:

ID	Šifra	Model	Začiatok	Koniec	Úspech	Zašifrovaný text	Dešifrovaný text
1	Cézarova	GPT-2	2024-03-14 10:00	2024-03-14 10:01	95%	"Khoor zruog"	"Hello world"
2	Vigenère	GPT-3	2024-03-14 10:15	2024-03-14 10:16	87%	"Wnqqs iptqf"	"Hello world"

Tabuľka obsahuje kompletný prehľad všetkých pokusov o dešifrovanie, vrátane:

- Časových značiek začiatku a konca
- Použitého modelu a typu šifry
- Percentuálnej úspešnosti dešifrovania
- Vstupného a výstupného textu

7 Bezpečnostné opatrenia

Pri návrhu systému sme kládli dôraz na to, aby bol bezpečný, najmä keď pracuje s používateľskými účtami a potenciálne citlivými dátami. Kombinujeme viacero vrstiev ochrany:

7.1 Hašovanie hesiel

Heslá používateľov sa v databáze nikdy neukladajú v čitateľnej forme. Využívame funkcie `generate_password_hash()` a `check_password_hash()` z Flask/Werkzeug, ktoré implementujú silné jednosmerné hashovanie hesiel (predvolene PBKDF2-SHA256 so soľou).

7.2 OAuth 2.0 integrácia

Pre zvýšenie bezpečnosti a komfortu majú používatelia možnosť využiť externého poskytovateľa identity – Google. OAuth 2.0 implementujeme pomocou knižnice Authlib, ktorá sa stará o celú výmenu tokenov.

7.3 Bezpečné pripojenie k databáze

- Komunikácia cez zabezpečené SSL spojenie
- Prístup chránený heslom a obmedzenými právami
- Tajnosti uložené v súbore `.env`

7.4 Ochrana proti SQL injection

Všetky databázové operácie realizujeme pomocou parametrov v SQL dotazoch, nie skladaním reťazcov. Napríklad:

```
1 cur.execute("SELECT ... WHERE email=%s", (email,))
```

7.5 Rate limiting

Obmedzenie počtu požiadaviek na:

- Max 5 dešifrovaní za minútu na používateľa
- Max 20 requestov/min z jednej IP

7.6 XSS a ochrana na strane klienta

- Bezpečné zobrazovanie údajov v HTML
- Escapovanie špeciálnych znakov
- Používanie textNode alebo innerText namiesto innerHTML

8 Ukážky dôležitých SQL dotazov

V tejto časti uvádzame niekoľko kľúčových SQL dotazov použitých v aplikácii, spolu s vysvetlením ich funkcie.

8.1 Overenie prihlasovacích údajov používateľa

```
SELECT user_id, username, email, password_hash
FROM "Users"
WHERE email = %s;
```

Tento dotaz vyberá riadok používateľa podľa e-mailovej adresy. Aplikácia následne porovná hash hesla s tým, čo používateľ zadal.

8.2 Vloženie nového pokusu o dešifrovanie

```
INSERT INTO "Decryption_Attempts"
(cipher_id, model_id, user_id, start_time, end_time, success,
correctness_percentage, encrypted_text, decrypted_text)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
RETURNING attempt_id;
```

Tento dotaz vloží nový riadok do tabuľky pokusov so všetkými detailmi.

8.3 Výber histórie dešifrovacích pokusov

```
SELECT da.attempt_id,
       c.name AS cipher_name,
       m.name AS model_name,
       da.start_time,
       da.end_time,
       da.success,
       da.correctness_percentage,
       da.encrypted_text,
       da.decrypted_text,
       dr.result_id,
       dr.model_output,
       dr.similarity_measure,
       dr.readability_level,
       mc.correction_id,
       mc.corrector,
       mc.changed_percentage,
```

```
       mc.final_text
FROM "Decryption_Attempts" da
JOIN "Ciphers" c ON da.cipher_id = c.cipher_id
JOIN "Models" m ON da.model_id = m.model_id
LEFT JOIN "Decryption_Results" dr ON da.attempt_id = dr.attempt_id
LEFT JOIN "Manual_Corrections" mc ON dr.result_id = mc.result_id
WHERE da.user_id = %s
ORDER BY da.start_time DESC;
```

Tento komplexný dotaz spája viacero tabuliek pre zobrazenie kompletnej histórie pokusov.

8.4 Uloženie manuálnej korekcie výsledku

```
INSERT INTO "Manual_Corrections"
(result_id, corrector, changed_percentage, final_text)
VALUES (%s, %s, %s, %s);
```

Tento dotaz pridá nový záznam do tabuľky manuálnych korekcií.

9 Testovanie

Testovanie systému prebiehalo na viacerých úrovniach, aby sme zabezpečili spoľahlivosť a kvalitu implementácie.

9.1 Jednotkové testy

Implementovali sme sadu jednotkových testov pre kľúčové komponenty:

- Testy šifrovacích/dešifrovacích funkcií
- Testy autentifikácie a správy používateľov
- Testy databázových operácií
- Testy integrácie s LLM

9.2 Integračné testy

Overenie spolupráce komponentov:

- End-to-end testy webového rozhrania
- Testy komunikácie medzi serverom a databázou
- Testy správneho fungovania celého procesu dešifrovania

10 História vývoja

10.1 Analýza histórie commitov

Vývoj projektu prebiehal v nasledujúcich hlavných fázach:

1. **Inicializácia projektu** (Commit: Initial commit)
   - Vytvorenie základnej štruktúry projektu
   - Nastavenie Flask aplikácie
   - Inicializácia databázy
2. **Implementácia databázovej vrstvy** (Commit: Database implementation)
   - Vytvorenie SQL schém
   - Implementácia modelov
   - Migrácie databázy
3. **Integrácia LLM** (Commit: LLM integration)
   - Pridanie DistilGPT-2 modelu
   - Implementácia dešifrovacieho modulu
   - Optimalizácia výkonu
4. **Vývoj webového rozhrania** (Commit: Web UI development)
   - Implementácia používateľského rozhrania
   - Pridanie autentifikácie
   - Vylepšenie UX
5. **Testovanie a ladenie** (Commit: Testing and optimization)
   - Jednotkové testy
   - Integračné testy
   - Výkonnostné testy
6. **Dokumentácia a finalizácia** (Commit: Documentation and cleanup)
   - Technická dokumentácia
   - Používateľská príručka
   - Čistenie kódu

11 Budúce vylepšenia

Na základe analýzy súčasného stavu projektu navrhujeme nasledujúce vylepšenia:

11.1 Rozšírenie podporovaných šifier

- Implementácia ďalších historických šifier:
  - Playfairova šifra
  - Hillova šifra
  - ADFGVX šifra
- Podpora moderných šifrovacích algoritmov
- Rozšírenie databázy ukážkových textov

11.2 Vylepšenie modelov

- Fine-tuning modelov na špecifické šifry
- Implementácia ensemble prístupu
- Experimentovanie s novšími architektúrami
- Optimalizácia hyperparametrov

11.3 Vylepšenie používateľského rozhrania

- Interaktívna vizualizácia procesu dešifrovania
- Real-time aktualizácie stavu
- Mobilná verzia aplikácie
- Rozšírené štatistiky a reporty

11.4 Optimalizácia výkonu

- Implementácia cachingu
- Distribuované spracovanie
- Optimalizácia databázových dotazov
- Škálovanie na cloud platformy

11.5 Bezpečnostné vylepšenia

- Implementácia 2FA
- Audit logov
- Pravidelné bezpečnostné audity
- Vylepšená ochrana proti útokom

11.6 Rozšírenie funkcionality

- API pre externé aplikácie
- Podpora viacerých jazykov
- Export/import funkcionalita
- Integrácia s inými nástrojmi

12 Nasadenie a údržba

12.1 Docker kontajnerizácia

Aplikácia je zabalená do Docker kontajnera pre jednoduché nasadenie:

- Multi-stage Dockerfile
- Docker Compose pre orchestráciu služieb
- Automatické buildy a nasadenie

12.2 CI/CD pipeline

Využívame GitHub Actions na automatizáciu:

- Spustenie testov a lintera
- Build a push Docker image
- Automatické nasadenie na produkčný server

12.3 Zálohovanie dát

Stratégia zálohovania:

- Nočný PostgreSQL dump
- Uchovávanie záloh v cloud storage
- Zálohovanie natrénovaných modelov

12.4 Monitorovanie a logovanie

Implementované nástroje:

- Logging do stdout
- Health-check endpoint
- Prometheus/Grafana monitoring
- Error tracking cez Sentry

13 Záver

Projekt Kryptoanalýza klasických šifier pomocou veľkých jazykových modelov ukázal, že prepojenie tradičnej kryptografie s modernými metódami umelej inteligencie je nielen možné, ale aj veľmi perspektívne. Vytvorili sme funkčný systém, ktorý dokáže za pomoci LLM modelu (GPT-2) automaticky dešifrovať jednoduché šifry a učiť sa z príkladov. Systém pozostáva z prehľadnej webovej aplikácie pre používateľov, robustného serverového backendu postaveného na Flasku, a výkonného databázového úložiska PostgreSQL, ktoré spolu tvoria ucelenú platformu pre experimentovanie s kryptoanalýzou.

Hlavným prínosom projektu je inovatívny prístup ku kryptoanalýze: namiesto ručnej frekvenčnej analýzy či brute-force sme využili znalosti zachytené v jazykovom modeli. Model bol schopný rozoznať vzory v zašifrovanom texte a v mnohých prípadoch poskytnúť správny alebo aspoň čiastočne správny otvorený text. To potvrdzuje hypotézu, že LLM (tréningom na obrovských množstvách textu) v sebe implicitne nesú informácie použiteľné pri lúštení šifier – vedia generovať zmysluplný text aj z nezrozumiteľných vstupov.

Zároveň sme identifikovali obmedzenia: komplexnejšie klasické šifry (polyalfabetické s dlhým kľúčom, transpozičné so zamotanou permutáciou) kladú na LLM vyššie nároky a samotný generatívny prístup nemusí stačiť. V týchto prípadoch by bolo vhodné kombinovať AI s tradičnými algoritmickými metódami. Náš systém je modulárny, takže takú kombináciu by bolo možné v budúcnosti doplniť.

Z pohľadu praktických výsledkov systém poskytuje:

- Rýchle automatizované dešifrovanie pre bežné školské/praktické príklady šifier
- Platformu pre ďalší výskum – možnosť zapojiť nové modely alebo techniky
- Dôkaz konceptu, že LLM vedia riešiť štruktúrované problémy

Na záver možno konštatovať, že projekt splnil stanovené ciele: vytvorili sme funkčnú implementáciu, zdokumentovali architektúru, a v testoch sme dosiahli zaujímavé výsledky. Systém je pripravený pre ďalšie používanie aj rozširovanie. Kombinácia klasickej kryptografie a LLM je stále pomerne nová oblasť a náš projekt k nej prispieva konkrétnym riešením.

14 Zoznam literatúry

1. IBM – The History of Cryptography. Think Blog by IBM, 2021. (História kryptografie od staroveku po súčasnosť, vrátane zmienky o Al-Kindím a frekvenčnej analýze)

2. Maskey, U., Zhu, C., Naseem, U. – "Benchmarking Large Language Models for Cryptanalysis and Mismatched-Generalization". arXiv preprint arXiv:2505.24621, May 2025. (Výskumný článok hodnotiaci schopnosti viacerých LLM dešifrovať rôzne šifry v rôznych podmienkach)

3. Sugio, N. – Ïmplementation of Cryptanalytic Programs Using ChatGPT". IACR Cryptology ePrint Archive, Report 2024/240, 2024. (Štúdia demonštrujúca použitie pokročilého LLM ChatGPT na generovanie zdrojového kódu pre kryptanalýzu moderných šifier, vrátane diskusie potenciálu a obmedzení AI v kryptografii)

4. Reddit – GPT-4 can break encryption (Caesar Cipher). príspevok užívateľa himey72 v subreddite r/ChatGPT, apríl 2023. (Diskusia potvrdzujúca, že GPT-4 si poradí s Cézarovou šifrou, no s inými šiframi má problémy)

5. CEUR Workshop Proceedings – "Neural Cryptanalysis of Classical Ciphers". HistoCrypt 2018. (Práca prezentujúca metódu využitia neurónových sietí v kombinácii s klasickými technikami kryptoanalýzy na automatizovanie lúštenia klasických šifier)

15 Integrácia veľkých jazykových modelov

15.1 Výber a implementácia modelov

Pre kryptoanalýzu sme zvolili model DistilGPT-2 z nasledujúcich dôvodov:

- Dobrý pomer výkon/veľkosť – model je dostatočne výkonný pre naše potreby, ale zároveň nie je príliš náročný na zdroje
- Možnosť lokálneho behu – nie je potrebné volať externé API
- Podpora knižnice transformers – jednoduchá integrácia a fine-tuning
- Vhodnosť pre generatívne úlohy – model je trénovaný na predikciu nasledujúcich tokenov

15.2 Architektúra dešifrovacieho modulu

Dešifrovací modul je implementovaný v triede Decryptor s nasledujúcou štruktúrou:

```
class Decryptor:
    def __init__(self, model_name="distilgpt2"):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(
           model_name)

    def decrypt(self, encrypted_text, max_length=100):
        # Tokenizácia vstupu
        inputs = self.tokenizer(encrypted_text, return_tensors="pt"
           )

        # Generovanie výstupu
        outputs = self.model.generate(
            inputs["input_ids"],
            max_length=max_length,
            num_return_sequences=1,
            no_repeat_ngram_size=2,
            do_sample=True,
            top_k=50,
            top_p=0.95,
            temperature=0.7
        )

        # Dekódovanie výstupu
```

```
        decrypted_text = self.tokenizer.decode(outputs[0])
        return decrypted_text
```

15.3 Proces dešifrovania

Proces dešifrovania prebieha v nasledujúcich krokoch:

1. **Príprava vstupu**:
   - Zašifrovaný text je tokenizovaný pomocou GPT-2 tokenizéra
   - Pridanie špeciálnych tokenov a padding
   - Konverzia na tenzory PyTorch
2. **Generovanie**:
   - Model generuje najpravdepodobnejšie tokeny
   - Použitie beam search alebo sampling stratégií
   - Aplikácia parametrov ako temperature a top-k/top-p
3. **Post-processing**:
   - Dekódovanie tokenov späť na text
   - Odstránenie špeciálnych tokenov
   - Validácia výstupu

15.4 Metriky a vyhodnotenie

Pre vyhodnotenie kvality dešifrovania používame nasledujúce metriky:

- **Charakterová presnosť**:
  - Porovnanie vygenerovaného textu so skutočným plaintextom znak po znaku
  - Ignorovanie veľkosti písmen a medzier
  - Výpočet percentuálnej zhody pomocou SequenceMatcher
- **Čitateľnosť**:
  - Percento slov vo výstupe, ktoré sú reálne slová
  - Kontrola proti slovníku alebo jazykovému modelu
  - Hodnotenie plynulosti a gramatickej správnosti
- **Sémantická podobnosť**:
  - Porovnanie významu dešifrovaného textu s originálom
  - Využitie embeddings alebo BLEU skóre
  - Subjektívne hodnotenie kontextu

15.5 Optimalizácia výkonu

Pre zlepšenie výkonu modelu sme implementovali:

- Caching modelov v pamäti
- Batch processing pre viacero požiadaviek
- Optimalizácia hyperparametrov generovania
- Možnosť paralelného spracovania na GPU