Utilitar AI pentru Analiza Criptografică și Securitate

Cuprins

Capitolul 1

Introducere si Prezentare Generala

1.1 Contextul Proiectului

Proiectul propune dezvoltarea unui sistem integrat care combina inteligenta artificiala cu diverse tehnici criptografice pentru a oferi o platforma completa de analiza a criptogramelor, criptare/decriptare, analiza a teoriei numerelor si evaluare a securitatii parolelor. Intr-o era in care securitatea informatiilor devine din ce in ce mai importanta, acest sistem va servi ca instrument educational si practic pentru studenti, cercetatori si profesionisti in domeniul securitatii informatice.

1.2 Obiective

- Crearea unei platforme integrate pentru analiza criptografica
- Implementarea unui motor de conversatie bazat pe LLM specializat in criptografie
- Dezvoltarea modulelor de analiza si atac pentru diverse algoritmi criptografici
- Integrarea bazelor de date pentru numere prime si semiprime
- Implementarea algoritmilor post-quantum
- Asigurarea scalabilitatii si securitatii sistemului

1.3 Valoare Academica si Practica

Lucrarea aduce contributii semnificative prin:

- Integrarea tehnicilor de ML/DL in analiza criptografica
- Abordarea unitara a mai multor aspecte ale criptografiei

- Implementarea practica a algoritmilor post-quantum
- Crearea unei interfete conversationale pentru rezolvarea problemelor criptografice

Capitolul 2

Arhitectura Sistemului

2.1 Arhitectura Generala

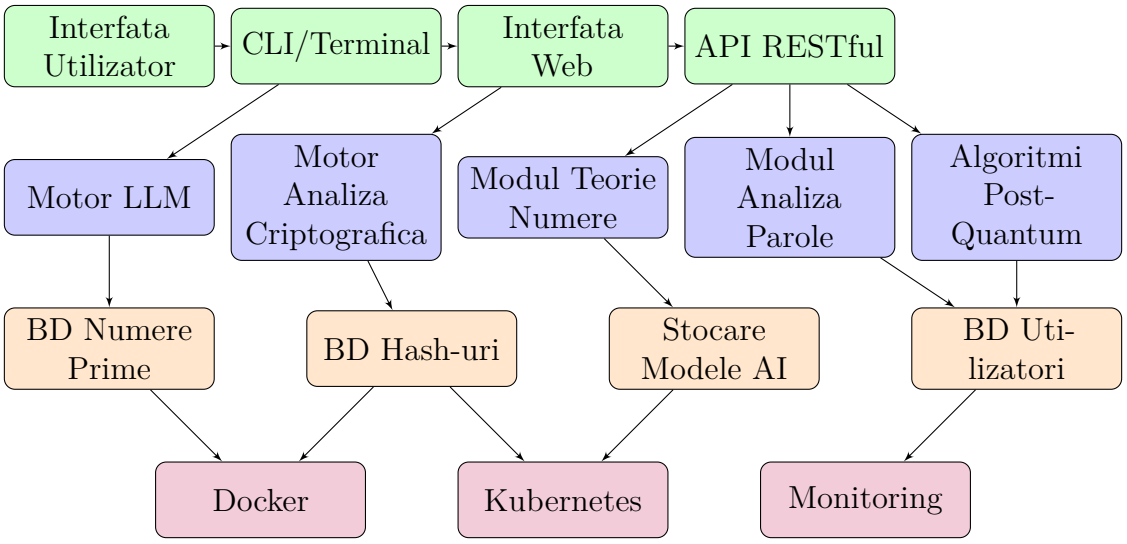

Figura 2.1: Arhitectura generala a sistemului

2.2 Componente Principale

2.2.1 Frontend

- **Interfata CLI/Terminal**: Interactiune directa prin comanda
- **Interfata Web**: Acces prin browser cu suport pentru diverse dispozitive
- **API RESTful**: Endpointuri pentru servicii si integrare cu alte sisteme

2.2.2 Backend

- **Motor LLM**: Procesare in limbaj natural si generare de raspunsuri
- **Motor de Analiza Criptografica**: Modulul central pentru analiza si atacarea criptogramelor
- **Modul Teorie Numere**: Gestionarea operatiilor cu numere prime, factorizare
- **Modul Analiza Parole**: Evaluarea securitatii parolelor si generarea de recomandari
- **Modul Algoritmi Post-Quantum**: Implementarea si testarea algoritmilor rezistenti la atacuri cuantice

2.2.3 Stocare

- **Baza de Date Numere Prime/Semiprime**: Cache si acces rapid la informatii despre numere prime
- **Baza de Date Hash-uri**: Pentru verificarea parolelor si hash-urilor cunoscute
- **Stocare Modele AI**: Pentru modelele LLM antrenate si ponderile acestora
- **Baza de Date Utilizatori**: Gestionarea conturilor, preferintelor si istoricului conversatiilor

Capitolul 3

Stack Tehnologic

3.1 Stack Tehnologic Detaliat

3.1.1 Backend

- **Python 3.10+**: Limbaj principal pentru dezvoltarea backend-ului
- **FastAPI**: Framework modern, rapid si asincron pentru API-uri
- **C/C++**: Pentru componente critice de performanta (algoritmi criptografici)
- **Rust**: Pentru module de securitate si performanta ridicata
- **gRPC**: Pentru comunicare eficienta intre microservicii

3.1.2 Inteligenta Artificiala

- **PyTorch**: Framework principal pentru antrenarea modelelor LLM
- **TensorFlow/Keras**: Pentru anumite componente specifice
- **Hugging Face Transformers**: Pentru utilizarea si fine-tuning-ul modelelor pre-antrenate
- **ONNX Runtime**: Pentru optimizarea inferentei modelelor

3.1.3 Criptografie

- **OpenSSL**: Biblioteca standard pentru implementari criptografice
- **Cryptography (Python)**: Pentru operatii criptografice in Python
- **Liboqs**: Pentru implementari post-quantum

- **NTL**: Pentru algoritmi avansati de teorie a numerelor
- **GMP**: Pentru operatii cu numere de precizie arbitrara

3.1.4 Stocare Date

- **PostgreSQL**: Baza de date relationala principala
- **Redis**: Pentru caching si sesiuni
- **MongoDB**: Pentru stocare flexibila a istoricului conversatiilor
- **MinIO**: Pentru stocare de tip obiect (modele AI)

3.1.5 Containerizare si Orchestrare

- **Docker**: Pentru containerizarea aplicatiilor
- **Docker Compose**: Pentru dezvoltare si testare locala
- **Kubernetes**: Pentru orchestrare in productie
- **Helm**: Pentru gestionarea deployment-urilor Kubernetes

3.1.6 Securitate

- **JWT**: Pentru autentificare si autorizare
- **OAuth2**: Pentru integrare cu sisteme externe
- **Keycloak**: Pentru management identitate (optional)
- **Vault**: Pentru gestionarea secretelor

3.1.7 Frontend

- **React**: Pentru interfata web
- **NextJS**: Pentru SSR si optimizare
- **TailwindCSS**: Pentru stilizare rapida
- **shadcn/ui**: Pentru componente reutilizabile

3.1.8 Monitorizare si Logging

- **Prometheus**: Pentru colectarea metricilor
- **Grafana**: Pentru vizualizarea metricilor
- **ELK Stack**: Pentru logging centralizat
- **Jaeger**: Pentru distributed tracing

3.2 Justificarea Tehnologiilor

3.2.1 Backend

- **Python**: Versatil, bogat in biblioteci pentru ML si criptografie, productivitate ridicata
- **FastAPI**: Performanta superioara, documentatie automata, suport asincron
- **C/C++**: Performanta critica pentru algoritmi criptografici intensivi
- **Rust**: Siguranta la nivel de memorie, performanta apropiata de C/C++

3.2.2 AI si ML

- **PyTorch**: Flexibilitate, comunitate activa, suport excelent pentru NLP
- **Hugging Face**: Acces la modele pre-antrenate, simplificarea fine-tuning-ului
- **ONNX**: Optimizare pentru inferenta, portabilitate intre framework-uri

3.2.3 Containerizare

- **Docker**: Izolare, reproducibilitate, portabilitate
- **Kubernetes**: Scalabilitate, auto-healing, orchestrare avansata

3.2.4 Securitate

- **JWT**: Standard pentru tokenuri de autentificare
- **OpenSSL**: Biblioteca matura si testata pentru operatii criptografice

Capitolul 4

Implementarea Modulelor

4.1 Motorul LLM Specializat

4.1.1 Structura Modelului

Se propune utilizarea unui model de tip encoder-decoder bazat pe arhitectura Transformer, cu:

- Dimensiune model: 1-3B parametri (compromis intre performanta si resurse)
- Context window: 8K-16K tokeni (pentru a permite analiza textelor mai lungi)
- Arhitectura: Mixta intre transformer standard si arhitecturi specializate

4.1.2 Strategie de Antrenare

4.1.3 Setul de Date pentru Antrenare

- **Corpus General**: Modele pre-antrenate (ex. Llama 3, Mistral, etc.)
- **Corpus Specializat**:
  - Articole academice din domeniul criptografiei
  - Documentatie pentru algoritmi criptografici
  - Perechi problema-solutie din criptografie
  - Exemple de atacuri criptografice si rezolvarile lor
  - Dialoguri expert-novice in domeniul criptografiei
  - Manuale si carti de specialitate in format digital

4.1.4 Fine-tuning si RLHF

- **Fine-tuning Supervizat**: Pe corpus specializat criptografic
- **RLHF (Reinforcement Learning from Human Feedback)**: Pentru aliniere si utilitate
- **Prompting Specializat**: Tehnici de few-shot si chain-of-thought pentru probleme complexe

4.2 Motorul de Analiza Criptografica

4.2.1 Componente

- **Detector de Algoritm**: Model de clasificare pentru identificarea tipului de criptare
- **Analizor Frecvente**: Pentru criptografie clasica
- **Module Specializate**: Pentru fiecare algoritm major (AES, RSA, DES, etc.)
- **Executor de Atacuri**: Implementarea automatizata a atacurilor cunoscute

4.2.2 Fluxuri de Lucru

4.2.3 Detector de Algoritm

Detectorul de algoritm va folosi o abordare hibrida:

1. **Analiza Structurala**:
   - Verificarea lungimii output-ului
   - Identificarea pattern-urilor specifice (ex. padding)
   - Verificarea entropiei
2. **Clasificare ML**:
   - Model CNN + LSTM pentru identificarea algoritmilor din text criptat
   - Features extrase din distributii statistice
3. **Euristica Bazata pe Reguli**:
   - Reguli predefinite pentru algoritmi cunoscuti
   - Verificari pentru semnaturi specifice

```python
# Exemplu conceptual pentru detectorul de algoritm
class CipherDetector:
    def __init__(self):
        self.statistical_analyzer = StatisticalAnalyzer()
        self.ml_classifier = load_model("cipher_classifier.h5")
        self.rule_engine = RuleBasedDetector()

    def detect(self, ciphertext, additional_info=None):
        # Analiza statistica
        stats_features = self.statistical_analyzer.
            extract_features(ciphertext)

        # Clasificare ML
        ml_predictions = self.ml_classifier.predict(self.
            preprocess(ciphertext))

        # Verificare euristici
        rule_candidates = self.rule_engine.apply_rules(ciphertext
            , additional_info)

        # Fuziune rezultate
        final_candidates = self.fusion_algorithm(stats_features,
            ml_predictions, rule_candidates)

        return sorted(final_candidates, key=lambda x: x['
            confidence'], reverse=True)
```

Listing 4.1: Exemplu conceptual pentru detectorul de algoritm

4.3 Modulul de Teorie a Numerelor

4.3.1 Functionalitati

- **Teste de Primalitate**:
  - Miller-Rabin
  - AKS (Agrawal-Kayal-Saxena)
  - Lucas-Lehmer (pentru numere Mersenne)
- **Factorizare**:

  - Trial division
  - Metoda Pollard Rho
  - Quadratic Sieve
  - Number Field Sieve (pentru numere mari)

- **Operatii Speciale**:
  - Calculul logaritmului discret
  - Operatii pe curbe eliptice
  - Generare numere prime de dimensiuni specifice

4.3.2 Integrare cu Baze de Date Externe

- Interogare automata FactorDB
- Cache local pentru rezultate frecvente
- Sincronizare periodica cu baze de date publice

4.4 Modulul de Analiza Parole

4.4.1 Metrici de Securitate

- **Entropie**: Calculata pe baza seturilor de caractere si lungimii
- **Rezistenta la Atacuri**: Estimare timp pentru diferite tipuri de atacuri
- **Verificare in Baze de Date**: CrackStation, HaveIBeenPwned
- **Analiza Structurala**: Identificare pattern-uri comune, substitutii predictibile

4.4.2 Vizualizare si Recomandari

4.5 Implementarea Algoritmilor Post-Quantum

4.5.1 Algoritmi Selectati

- **Criptare Asimetrica**:
  - Kyber (CRYSTALS-Kyber): Selectat de NIST ca standard
  - NTRU: Alternativa matura si studiata

- **Semnaturi Digitale**:
  - Dilithium (CRYSTALS-Dilithium): Standard NIST
  - FALCON: Pentru aplicatii cu semnaturi compacte
- **Schimb de Chei**:
  - SIKE: Supersingular Isogeny Key Encapsulation

4.5.2 Implementare si Integrare

- Utilizarea bibliotecii liboqs ca baza
- Wrapper-uri Python pentru acces simplificat
- Integrare in fluxurile de lucru standard
- Interfete compatibile cu algoritmi clasici pentru tranzitie usoara

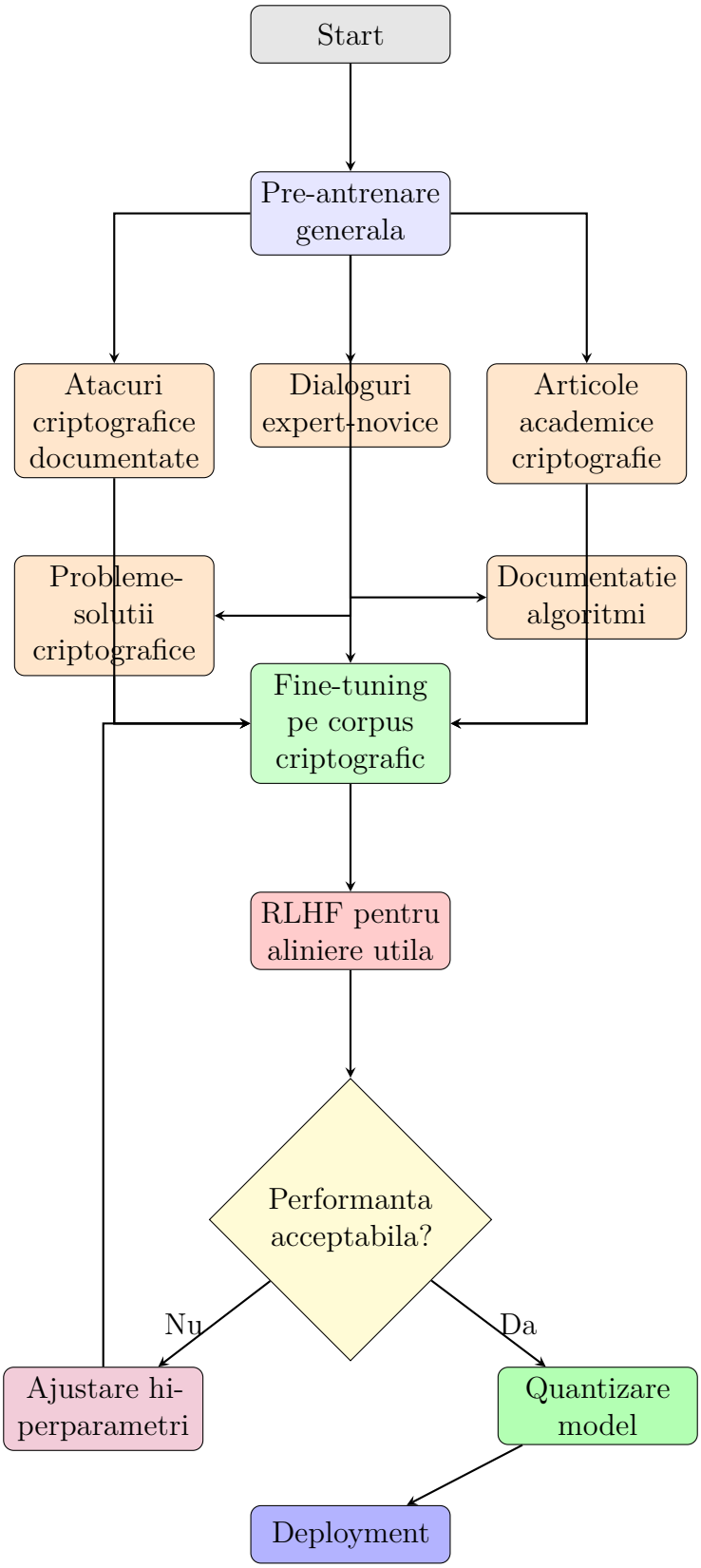

Figura 4.1: Strategia de antrenare pentru LLM specializat

Capitolul 5

Securizarea Sistemului

5.1 Autentificare si Autorizare

- **JWT** pentru tokenuri de acces
- **OAuth2** pentru autorizare
- **Rate limiting** pentru prevenirea abuzului
- **RBAC** (Role-Based Access Control) pentru permisiuni granulare

5.2 Securitatea Comunicatiilor

- **TLS 1.3** pentru toate comunicatiile externe
- **Mutual TLS** pentru comunicatii intre microservicii
- **Perfect Forward Secrecy** pentru protectie pe termen lung

5.3 Securitatea Datelor

- **Criptare la repaus** pentru toate datele sensibile
- **Tokenizare** pentru informatii de identificare personala
- **Anonimizare** a datelor utilizate pentru antrenare

Capitolul 6

Deployment si Scalabilitate

6.1 Arhitectura Docker

6.2 Implementare Kubernetes

- **Namespace Dedicat**: Izolare completa a resurselor
- **Deployment** pentru fiecare serviciu cu strategie de rolling update
- **HorizontalPodAutoscaler** pentru scalare bazata pe utilizare
- **Ingress** pentru routare externa si TLS
- **Helm Charts** pentru deployments reproductibile
- **Secrets Management** cu Kubernetes Secrets sau HashiCorp Vault

6.3 Monitoring si Logging

- **Prometheus** pentru colectarea metricilor
- **Grafana** pentru dashboards si vizualizare
- **ELK Stack** (Elasticsearch, Logstash, Kibana) pentru logging centralizat
- **Alerting** pentru notificari proactive in caz de probleme

Capitolul 7

Fluxuri de Lucru Detaliate

7.1 Conversatie cu LLM pentru Rezolvarea Problemelor

1. **Initiere**: Utilizatorul descrie o problema criptografica
2. **Intelegere**: LLM proceseaza si extrage componentele cheie
3. **Clarificare**: LLM cere detalii suplimentare daca este necesar
4. **Rezolvare**: LLM genereaza solutia pas cu pas
5. **Explicare**: LLM ofera detalii despre abordarea folosita
6. **Feedback**: Utilizatorul poate cere clarificari sau detalii suplimentare

Exemplu Conversatie

```
Utilizator: Cum pot decripta acest text "Uifsf jt b tfdsfu nfttbhf"?

LLM: Acest text pare sa fie criptat cu Caesar Cipher (sau ROT).
Voi incerca diferite chei:

Cu ROT-1: "There is a secret message"
Aceasta varianta are sens in limba engleza, deci probabil este corecta.
Cifrul Caesar inlocuieste
fiecare litera cu litera aflata la o anumita pozitie fixa in alfabet.
In acest caz, fiecare litera a fost inlocuita cu urmatoarea
litera din alfabet (ROT-1).
Doriti sa aflati mai multe despre cifrul Caesar sau cum sa detectati
acest tip de criptare?
```

7.2 Analiza Mesajului Criptat

1. **Incarcare**: Utilizatorul trimite textul criptat
2. **Detectare**: Sistemul detecteaza algoritmul de criptare probabil
3. **Analiza**: Se executa analize statistice si pattern matching
4. **Atac**: Se incearca metode de atac pentru algoritmul detectat
5. **Raportare**: Se returneaza rezultatele si explicatiile

7.3 Lucrul cu Algoritmi Post-Quantum

1. **Selectie**: Utilizatorul alege tipul de algoritm post-quantum
2. **Configurare**: Se seteaza parametrii specifici
3. **Generare**: Se genereaza cheile criptografice
4. **Operatii**: Se efectueaza operatiuni (criptare, semnare, etc.)
5. **Analiza**: Se ofera statistici si comparatii cu algoritmi clasici

```
# Exemplu simplificat de integrare Kyber in aplicatie
from pqcrypto import kyber

# Generare chei
public_key, secret_key = kyber.keygen()

# Criptare mesaj
ciphertext, shared_secret_enc = kyber.encrypt(public_key, message
    )

# Decriptare mesaj
shared_secret_dec = kyber.decrypt(secret_key, ciphertext)

# Verificare
assert shared_secret_enc == shared_secret_dec
```

Listing 7.1: Exemplu de utilizare pentru CRYSTALS-Kyber

Capitolul 8

Antrenarea Modelelor ML/DL

8.1 Model pentru Detectarea Algoritmului de Criptare

8.1.1 Procesul de Antrenare

1. **Pregatirea Datelor**:
   - Generarea datelor pentru multiple algoritmi (AES, DES, RSA, etc.)
   - Variatii in chei, moduri de operare, padding
   - Augmentarea datelor pentru robustete
2. **Arhitectura Modelului**:
   - CNN pentru extragerea pattern-urilor locale
   - BiLSTM pentru capturarea dependentelor secventiale
   - Fully Connected layers pentru clasificare
3. **Strategia de Antrenare**:
   - Transfer learning de la modele antrenate pe clasificare de text
   - Fine-tuning specific pentru date criptografice
   - Validation cruce pentru evitarea overfitting

8.1.2 Evaluare si Optimizare

- Matrice de confuzie pentru intelegerea erorilor
- Raport de clasificare (precision, recall, F1)

- Evaluare pe date din lumea reala
- Analiza cazurilor de esec pentru imbunatatire

8.2 Antrenare LLM Specializat

8.2.1 Procesul de Fine-tuning

1. **Selectia Modelului de Baza**:
   - Utilizarea unui model pre-antrenat (ex. Llama 3-8B)
   - Adaptarea arhitecturii pentru context specializat
2. **Pregatirea Datelor**:
   - Colectarea si curatarea corpusului criptografic
   - Structurarea in formatul instruction-response
   - Augmentarea cu date sintetice
3. **Fine-tuning**:
   - LoRA (Low-Rank Adaptation) pentru eficienta
   - QLoRA pentru training pe hardware accesibil
   - Strategii de optimizare a hiperparametrilor

8.2.2 RLHF (Reinforcement Learning from Human Feedback)

1. **Colectarea Preferintelor**:
   - Evaluari de experti in domeniul criptografiei
   - Comparatii intre raspunsuri alternative
2. **Antrenarea Modelului de Recompensa**:
   - Model care prezice preferintele umane
   - Calibrare pentru evitarea biasurilor
3. **Optimizare prin PPO**:
   - Fine-tuning folosind Proximal Policy Optimization
   - Balansare intre maximizarea recompensei si evitarea devierilor

Capitolul 9

Planificarea Proiectului

9.1 Etape si Termene

9.2 Estimare Efort si Resurse

Componenta	**Efort Estimat (ore)**	**Nivel Complexitate**	**Resurse Necesare**
Arhitectura si design	80-120	Mediu	Documentatie, cercetare
Motor LLM (fine-tuning)	150-200	Inalt	GPU performant, date antrenare
Motor criptografic	100-150	Mediu-Inalt	Biblioteci criptografice
Modul teorie numere	80-100	Mediu	Biblioteci matematice
Analiza parole	60-80	Mediu	Baze de date, biblioteci statistice
Algoritmi post-quantum	100-120	Inalt	Documentatie specializata
Frontend si API	100-120	Mediu	Biblioteci UI, framework web
Integrare si testare	80-100	Mediu	Framework-uri de testare
Documentatie	70-100	Mediu	Materiale referinta
Total	**820-1090**		

Tabela 9.1: Estimare efort si resurse

9.3 Managementul Riscurilor

Risc	Probabilitate	Impact	Strategii de Mitigare
Complexitate tehnica neprevazuta	Medie	Inalt	Prototipare rapida, cercetare aprofundata
Limitari hardware pentru antrenare LLM	Inalta	Mediu	Utilizare modele mai mici, tehnici eficiente (QLoRA)
Integrare dificila a componentelor	Medie	Mediu	Dezvoltare bazata pe API, testare continua
Securitatea sistemului compromisa	Scazuta	Foarte Inalt	Audit de securitate, testare penetrare
Timelines nerealist	Medie	Inalt	Buffer in planificare, prioritizare caracteristici

Tabela 9.2: Managementul riscurilor

Capitolul 10

Consideratii Finale

10.1 Contributii Academice

Proiectul contribuie in mai multe directii:

- Integrarea tehnicilor de ML/DL in analiza criptografica
- Arhitectura hibrida pentru instrumente criptografice
- Implementarea practica a algoritmilor post-quantum
- Framework conversational pentru probleme criptografice

10.2 Potentiale Extensii

- Suport pentru noi algoritmi criptografici
- Integrarea cu sisteme de PKI (Public Key Infrastructure)
- Dezvoltarea unui framework pentru competitii CTF
- Modul de analiza forensica pentru malware criptat
- Integrare cu sisteme de blockchain

10.3 Limitari si Provocari

- Performanta inferentei LLM pe hardware limitat
- Actualizarea constanta a bazelor de date
- Mentinerea la curent cu evolutiile in criptanaliza
- Consideratii etice pentru unelte de atac criptografic

Capitolul 11

Referinte si Resurse Utile

11.1 Referințe și Resurse

11.1.1 Cărți și Manuale

1. Ferguson, N., Schneier, B., & Kohno, T. (2022). *Cryptography Engineering: Design Principles and Practical Applications*. Wiley.

2. Paar, C., & Pelzl, J. (2021). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.

3. Boneh, D., & Shoup, V. (2023). *A Graduate Course in Applied Cryptography*. [Online] Disponibil la: `https://toc.cryptobook.us/`

4. Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing*. [Online] Disponibil la: `https://web.stanford.edu/~jurafsky/slp3/`

5. Bernstein, D. J., & Lange, T. (2021). *Post-Quantum Cryptography*. Springer.

11.1.2 Articole Academice

1. Alagic, G., et al. (2023). "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process." NIST.

2. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." NAACL.

3. Ouyang, L., et al. (2022). "Training language models to follow instructions with human feedback." NeurIPS.

4. Bernstein, D. J. (2023). "Comparing proofs of security for lattice-based encryption." *Journal of Cryptology*.

5. Fouque, P. A., et al. (2022). "FALCON: Fast-Fourier Lattice-based Compact Signatures over NTRU." NIST PQC.

11.1.3 Documentație Tehnică și Standarde

1. NIST (2023). *FIPS 197: Advanced Encryption Standard (AES).*
2. NIST (2022). *SP 800-38A: Recommendation for Block Cipher Modes of Operation.*
3. IETF (2024). *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3.*
4. ISO/IEC (2022). *ISO/IEC 29192: Information technology — Security techniques — Lightweight cryptography.*
5. NIST (2023). *SP 800-208: Recommendation for Stateful Hash-Based Signature Schemes.*

11.1.4 Resurse Online și Tutoriale

1. *CryptoPals Crypto Challenges*. [Online] Disponibil la: `https://cryptopals.com/`
2. *Cryptography I & II*. Coursera (Stanford University). [Online]
3. *Post-Quantum Cryptography*. NIST. [Online] Disponibil la: `https://csrc.nist.gov/projects/post-quantum-cryptography`
4. *The Illustrated Transformer*. [Online] Disponibil la: `http://jalammar.github.io/illustrated-transformer/`
5. *Real-World Cryptography*. [Online] Disponibil la: `https://www.manning.com/books/real-world-cryptography`

11.1.5 Biblioteci și Framework-uri

1. OpenSSL Documentation. [Online] Disponibil la: `https://www.openssl.org/docs/`
2. PyTorch Documentation. [Online] Disponibil la: `https://pytorch.org/docs/stable/index.html`
3. Hugging Face Transformers Documentation. [Online] Disponibil la: `https://huggingface.co/docs/transformers/`
4. FastAPI Documentation. [Online] Disponibil la: `https://fastapi.tiangolo.com/`
5. liboqs: C library for quantum-resistant cryptographic algorithms. [Online] Disponibil la: `https://github.com/open-quantum-safe/liboqs`

11.1.6 Instrumente și Resurse Relevante

1. *dcode.fr* - Cryptography tools collection. [Online] Disponibil la: `https://www.dcode.fr/`

2. *CyberChef* - Web app for encryption, encoding, compression and data analysis. [Online] Disponibil la: `https://gchq.github.io/CyberChef/`

3. *FactorDB* - Database of large integer factorizations. [Online] Disponibil la: `http://factordb.com/`

4. *RSACTFTool* - RSA attack tool. [Online] Disponibil la: `https://github.com/RsaCtfTool/RsaCtfTool`

5. *Have I Been Pwned* - Compromised password checking service. [Online] Disponibil la: `https://haveibeenpwned.com/`

Concluzii

Acest proiect propune o abordare integrata si moderna pentru analiza criptografica, combinand tehnici traditionale cu inteligenta artificiala si machine learning. Prin integrarea unui motor LLM specializat, utilizatorii vor putea interactiona natural cu sistemul pentru a rezolva probleme criptografice complexe, analizand criptograme, evaluand securitatea parolelor si explorand algoritmi post-quantum.

Implementarea foloseste tehnologii de varf precum Docker pentru containerizare, FastAPI pentru backend, PyTorch pentru componente de ML/DL, si implementari native C/C++ pentru algoritmi criptografici performanti. Arhitectura modulara permite extensibilitate si scalabilitate, iar abordarea bazata pe microservicii faciliteaza dezvoltarea si testarea independenta a componentelor.

Cu un timp estimat de implementare de 800-1000 de ore, proiectul este fezabil pentru o lucrare de licenta ambitioasa, oferind atat valoare academica prin cercetarea aplicata, cat si valoare practica prin crearea unui instrument util pentru educatie si cercetare in domeniul securitatii informatice.

Adoptarea unui model de dezvoltare iterativ, cu focus initial pe functionalitatile de baza si extindere graduala a capacitatilor, va permite gestionarea eficienta a complexitatii si livrarea unui produs functional in limitele de timp disponibile.