

# **M1 DS2E -Technique de Programmation II**

HERWEDE Noah, AHMED-YAHIA LAIFA

# Présentation du Projet

**Pour ce projet, nous avons choisi de créer une application capable de détecter automatiquement les comportements de triche dans les parties d'échecs en ligne.**

- I Récupérer les données sur les joueurs
- II Prédire le risque de triche via Machine Learning
- III Offrir une interface utilisateur pour vérifier des parties

# Récupération des données de joueurs

Deux modes de récupération :

- API Chess.com / Lichess
- Upload manuel de fichiers PGN

**Exemple fonction API Chess.com :**

```
USER_AGENT = {  
    "User-Agent": "ChessAntiCheatBot/1.0 (+https://github.com/N-Herwede/chess-anti-cheat)"  
}  
  
def fetch_chessdotcom_games(username, max_games=10):  
    archive_url = f"https://api.chess.com/pub/player/{username}/games/archives"  
    archives = requests.get(archive_url, timeout=10).json()["archives"]  
    games = []  
    for month_url in archives[-2:]:  
        games_month = requests.get(month_url, timeout=10).json()["games"]  
        for game in games_month:  
            if "pgn" in game:  
                games.append({"pgn": game["pgn"], "headers": game})  
    return games[:max_games]
```

# Récupération des données d'entraînement

3 possibilités pour récupérer des données d'entraînement

- Base de données Lichess (<https://database.lichess.org/>)
- Dataset préexistant sur Kaggle
- Registres de parties officiels en ligne

```
def filter_cheat_games(games):  
    tos_true = [g for g in games if g.headers.get("TOSViolation") == "true"]  
    tos_false = [g for g in games if g.headers.get("TOSViolation") == "false"]  
    n = len(tos_true)  
    tos_false_sample = tos_false[:n]  
    return tos_true + tos_false_sample
```

# Analyse des coups avec Stockfish

Chaque coup est analysé via le moteur Stockfish pour obtenir une évaluation objective et créer nos variables d'analyse

## Exemple analyse Stockfish :

```
def analyze_game_stockfish(game, stockfish_path):
    engine = open_stockfish(stockfish_path)
    board = game.board()
    evals = []

    send_command(engine, "uci")
    read_until(engine, "uciok")
    send_command(engine, "isready")
    read_until(engine, "readyok")
    send_command(engine, "ucinewgame")

    for move in game.mainline_moves():
        board.push(move)
        moves_uci = [m.uci() for m in board.move_stack]
        send_command(engine, "position startpos moves " + " ".join(moves_uci))
        send_command(engine, "go depth 12")
```

```
        while True:
            line = engine.stdout.readline().strip()
            if "score cp" in line:
                score = int(line.split('score cp ')[1].split(' ')[0])
                evals.append(score / 100.0)
            if line.startswith('bestmove'):
                break

        engine.kill()

        swings = [abs(evals[i+1] - evals[i]) for i in range(len(evals)-1)]
        features = {
            'avg_centipawn_loss': sum(abs(e) for e in evals) / len(evals),
            'max_eval_swing': max(swings),
            'num_blunders': sum(1 for s in swings if s > 2.0),
            'eval_std': np.std(evals)
        }
        return features
```

# Construction des modèles Machine Learning

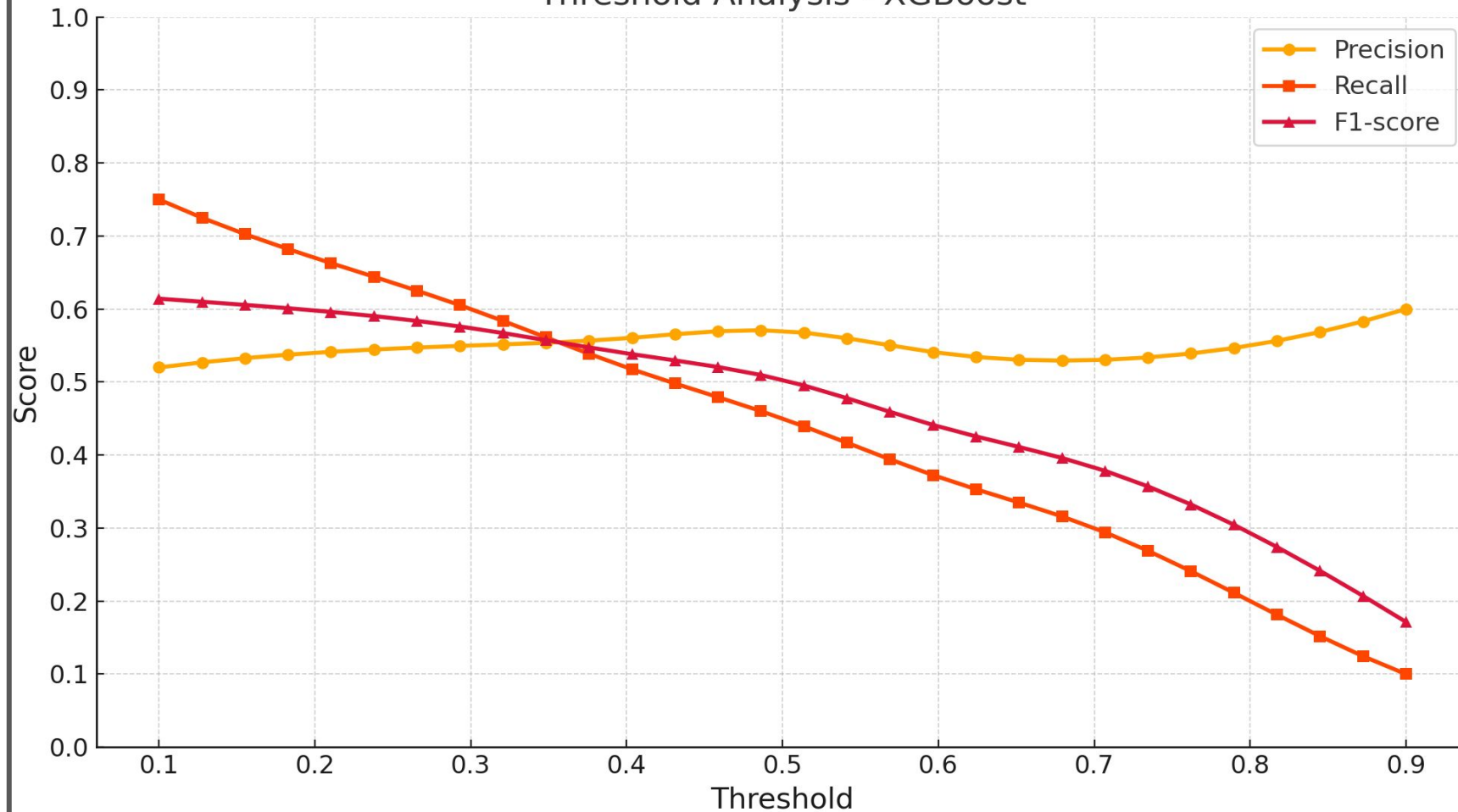
Nous avons choisi deux modèles complémentaires :

- **XGBoost** (Extrême Gradient Boosting)
- **CatBoost** (Categorical Boosting)

Pour augmenter la robustesse du système :

- Si **les deux détectent une triche**, c'est un flag fort.
- Si **un seul détecte**, c'est un flag moyen.
- Si **aucun ne détecte**, le joueur est considéré comme "clean".

# Threshold Analysis - XGBoost



# Variable du Modèle M-L

Nous avons sélectionné un total de **17 variables** :

TotalMoves	num_blunders	eval_std	nb_swings_100cp
avg_centipawn_loss	longest_good_streak	<b>blunder_ratio</b>	variance_time_per_move
max_eval_swing	trend_flips	moyenne_oscillations_eval	temps_moyen_par_coup
<b>num_mates</b>	<b>perfect_moves</b>	stddev_oscillations_eval	<b>elo</b>



# V. Application Streamlit

Interface utilisateur simple permettant :

- Saisie du pseudo Chess.com ou upload d'un fichier PGN
- Analyse automatique via Stockfish
- Affichage des probabilités de triche (XGBoost / CatBoost)

Graphiques :

- Oscillation de l'évaluation
- Waterfall Plot SHAP

```
if choice == "Chess.com Username":
    username =
    st.sidebar.text_input("Enter Chess.com
Username:")
    if st.sidebar.button("Fetch Games"):
        fetched_games =
        fetch_chessdotcom_games(username)
        st.session_state['games'] =
        [pgn_to_game(game['pgn']) for game in
        fetched_games]

if st.sidebar.button("Analyze Selected
Game"):
    features_white, features_black =
    analyze_game_stockfish(selected_game,
    STOCKFISH_PATH)
    xgb_proba, xgb_pred, cat_proba,
    cat_pred, verdict =
    predict_with_models(features_white,
    xgb_model, cat_model, THRESHOLD)
    st.metric("XGBoost Probability", f"
{xgb_proba:.2f}")
    st.metric("CatBoost Probability", f"
{cat_proba:.2f}")
```

## VI. Exemple d'analyse de partie

**Joueur White** : Cheat Probability : 0.18 (**XGBoost**)

Cheat Probability : 0.15 (**CatBoost**)

(2752 Elo)

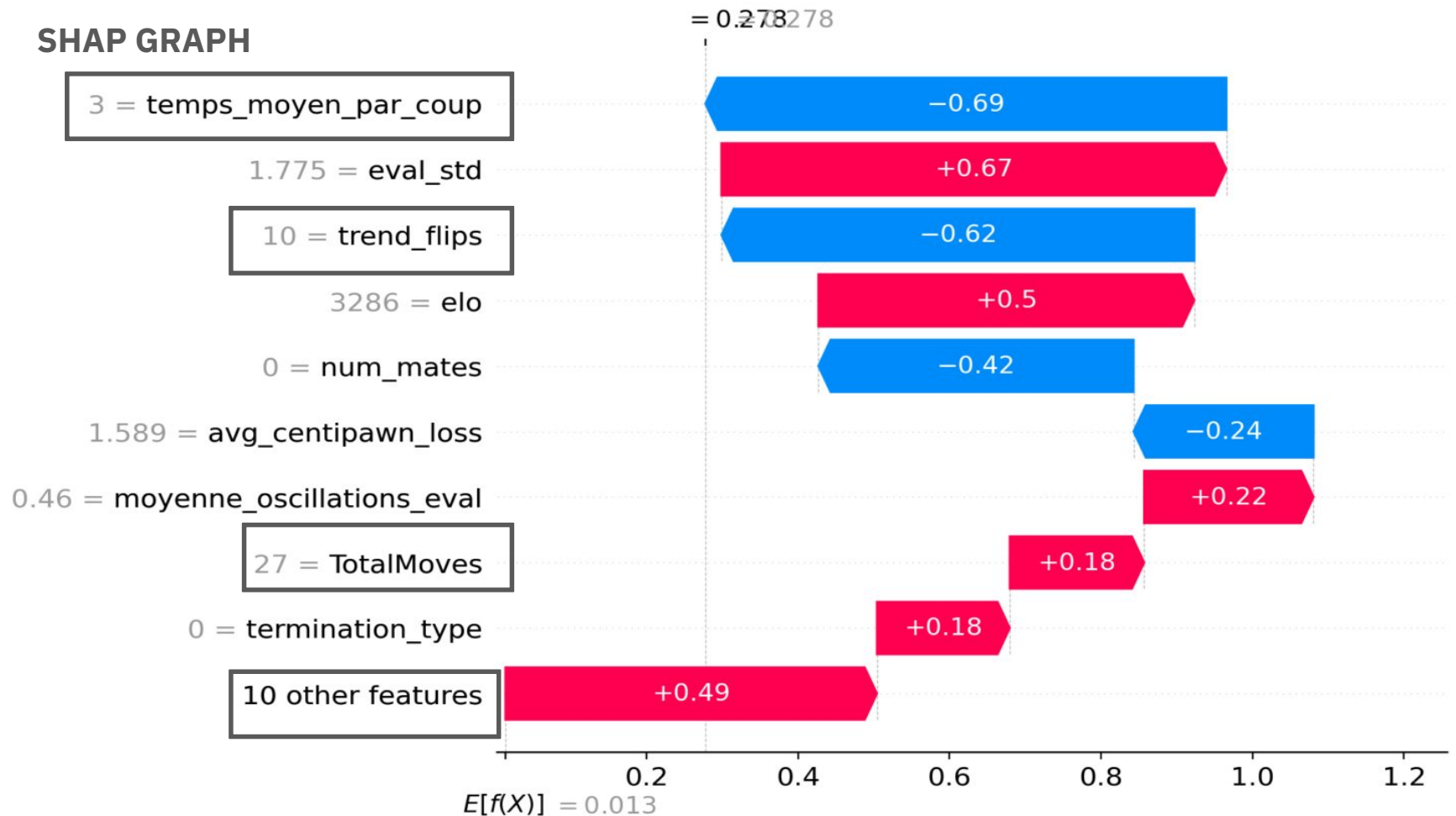
**Joueur Black** : Cheat Probability : 0.57 (**XGBoost**)

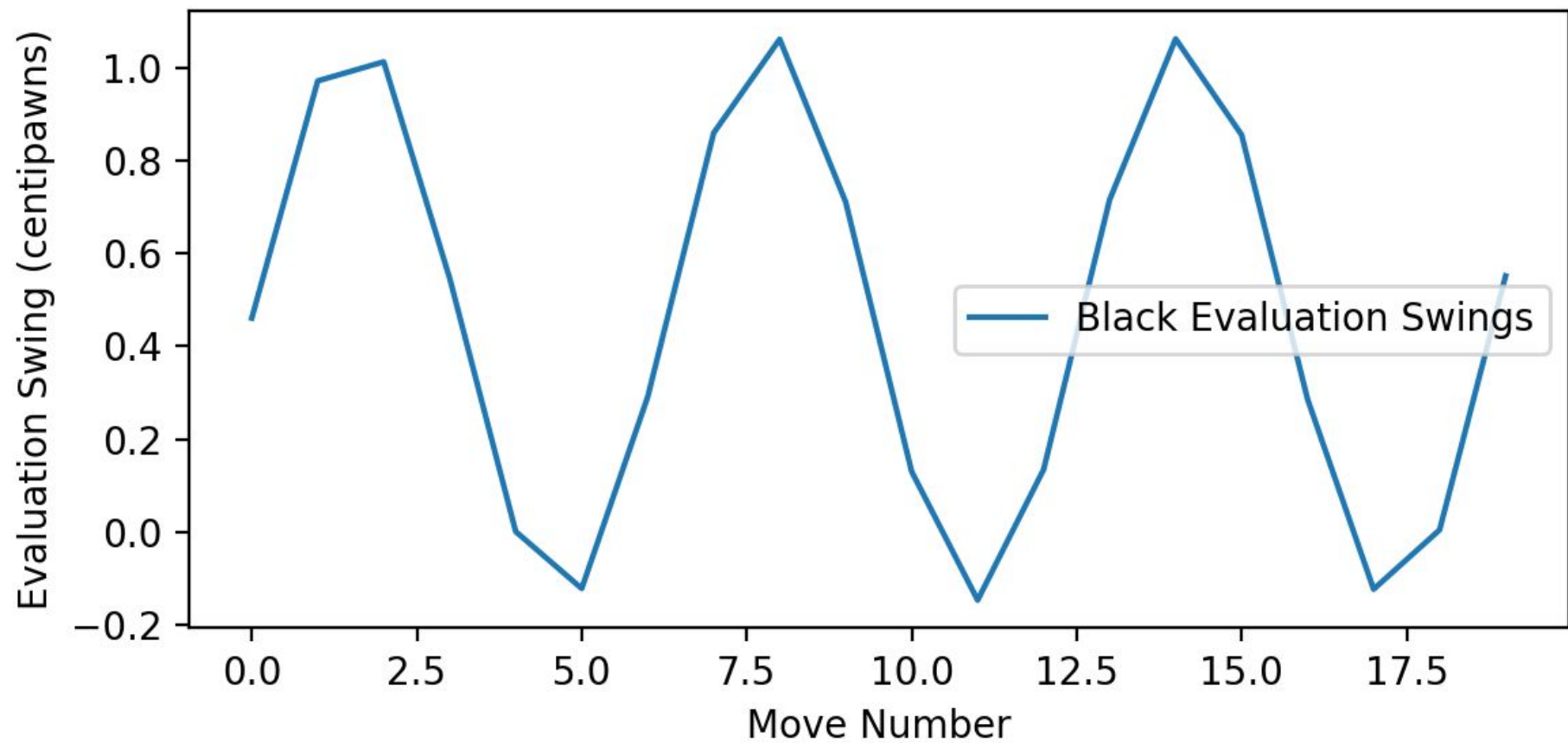
CatBoost Cheat Probability : 0.27 (**CatBoost**)

(3286 Elo)

Verdict: Flag moyen - Un modèle détecte triche

## SHAP GRAPH





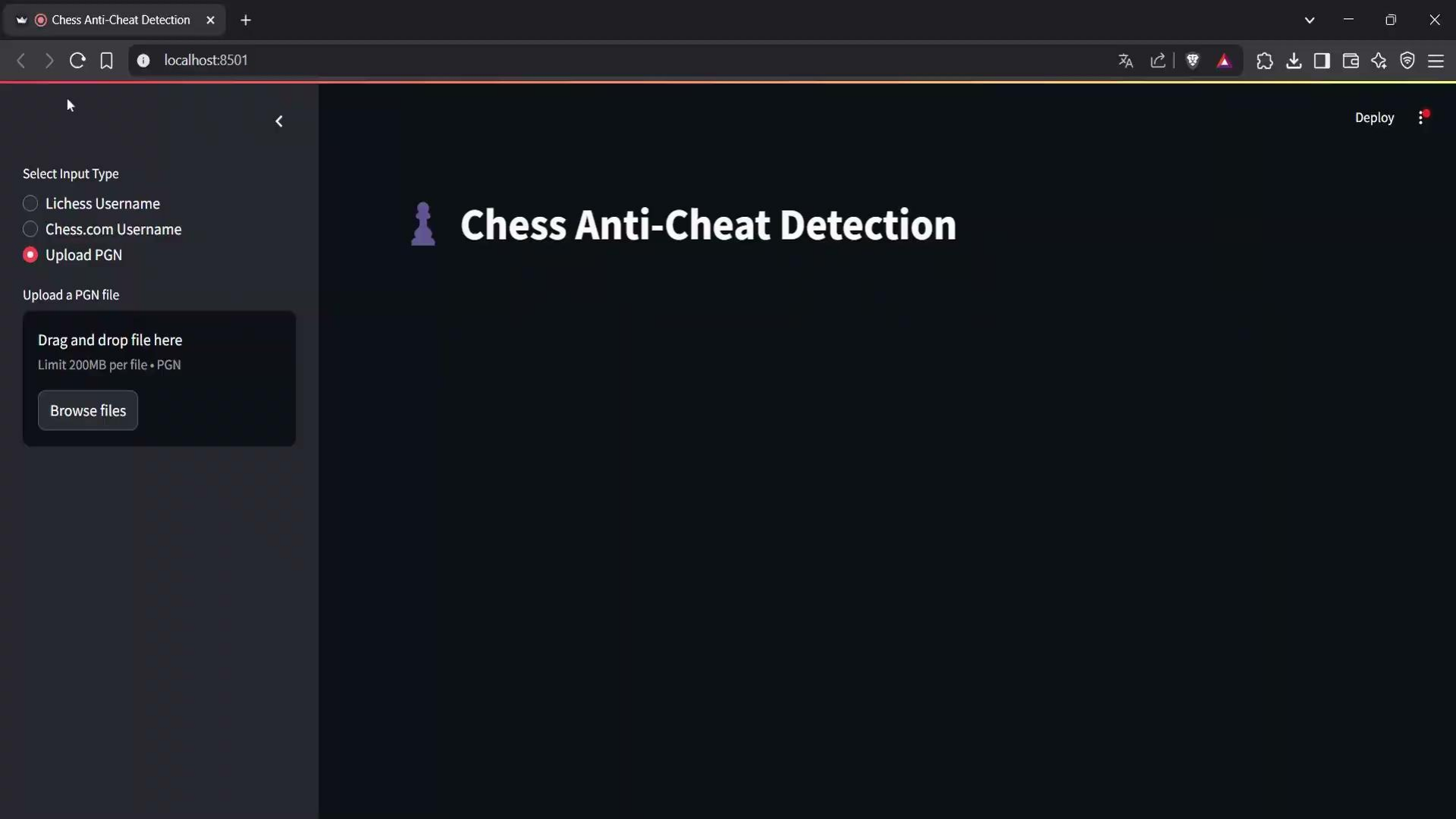
# Conclusion & Perspectives d'Amélioration

## Conclusion

- Évaluer automatiquement la qualité d'une partie
- Estimer le risque de triche par joueur en plusieurs paliers
- Fournir une justification visuelle de la décision

## Perspectives d'Amélioration :

- Adapter le modèle à la triche professionnelle
- Éviter les faux positifs
- Intégrer d'autres variables de détection
- Intégrer Maia pour une comparaison des styles de jeu
- Calibration dynamique des seuils



- Select Input Type
- ☐ Lichess Username
  - ☐ Chess.com Username
  - ☒ Upload PGN

Upload a PGN file

Drag and drop file here

Limit 200MB per file • PGN

Browse files

# Chess Anti-Cheat Detection

Deploy