Filière : **Smart Information and Communication Technology Engineering (Smart ICT Engineering).**

PROJET D'INNOVATION

PHISHING DETECTION USING AI & NLP

RÉALISÉ PAR :

SENHAJI Boutayna ; **CHARIFI Zakaria ;**
ER-RAFI Chaimae ; **KARROUM Salim ;**
DOUBALI Salma ; **SBIAA Ayoub.**

ENCADRÉ PAR :

Mr EL KABIR Taoufik ;
Mr LAHSEN-CHERIF Iyad.

Année Universitaire : 2024 – 2025

PHISHING DETECTION

USING ARTIFICIAL INTELLIGENCE &

NATURAL LANGUAGE PROCESSING (NLP)

REMERCIEMENT

Au moment où nous pensons avoir fait œuvre utile, qu'il nous soit permis d'exprimer nos vifs remerciements et notre profonde gratitude à l'entreprise ***Orange Business Maroc (OBM)****, pour l'attribution de sujets d'actualité aussi pertinents qu'enrichissants. Par la richesse de ses orientations et la modernité des problématiques proposées, OBM a contribué de manière décisive à nourrir notre réflexion et à ancrer ce projet dans une dynamique d'innovation.*

Nos remerciements vont aussi, et de manière respectueuse et combien reconnaissante, à ***Mr EL KHADIMI Ahmed****, pour son initiative précieuse dans la recherche de ces projets innovants en collaboration avec OBM. Son engagement, sa vision et sa disponibilité ont constitué un appui fondamental dans le choix du sujet, ainsi que dans la construction progressive de notre travail.*

Nous adressons enfin nos sincères remerciements à nos encadrants, ***Mr EL KABIR Taoufik*** *et* ***Mr LAHSEN CHERIF Iyad****, qui ont su, par leur accompagnement rigoureux, leurs conseils avisés et leur soutien constant, nous guider avec bienveillance et exigence. Leur encadrement a été essentiel à l'aboutissement et à la réussite de cette entreprise intellectuelle.*

Par la même occasion, nous souhaitons remercier toutes les personnes qui, de près ou de loin, nous ont soutenus et accompagnés tout au long de cette aventure. Qu'elles trouvent ici l'expression de notre reconnaissance la plus sincère.

RÉSUMÉ

Le **phishing** est l'une des menaces majeures sur Internet, avec une croissance exponentielle des emails frauduleux. Face à ce phénomène, ce projet propose une approche basée sur l'intelligence artificielle (**IA**) et le traitement du langage naturel (**NLP**) pour détecter automatiquement les emails frauduleux et les classer selon leur niveau de dangerosité.

L'objectif est de créer une **interface Web interactive** permettant à l'utilisateur de mieux gérer sa boîte mail tout en assurant une protection renforcée contre le phishing.

Le projet repose sur un **prétraitement minutieux** des emails (suppression des balises HTML, extraction de caractéristiques comme l'expéditeur, la longueur du sujet, etc.). Ces données sont ensuite analysées à l'aide de 3 modèles du Machine Learning, qui ont été testés : Support Vector Machine (**SVM**), Random Forest (**RF**), et **XGBoost**, ces modèles étant combinés dans un modèle de **Stacking** pour obtenir une solution robuste et performante.

L'interface utilisateur, développée avec l'outil **Streamlit**, permet à l'utilisateur d'interagir facilement avec le système, d'analyser ses mails et de visualiser les résultats de manière claire et ergonomique. La solution repose sur **l'API Gmail** pour l'extraction automatique des mails et permet de traiter localement pour les analyses plus complexes.

Les tests réalisés sur des scénarios variés, tels que les emails de phishing, les emails promotionnels ou les emails professionnels, ont montré que le modèle était efficace pour classer les emails avec une **grande précision**.

Des difficultés techniques, comme des bugs liés à l'API Gmail, ont été rencontrées mais surmontées grâce à des solutions collaboratives et des ajustements techniques.

Enfin, des perspectives d'amélioration pour l'avenir incluent l'analyse des pièces jointes (PJ), l'intégration de résumés vocaux, développement du modèle LLM et l'extension de la solution à plusieurs langues pour une portée plus large.

ABSTRACT

Phishing is one of the major threats on the internet, with an exponential growth in fraudulent emails. In response to this phenomenon, this project proposes an approach based on Artificial Intelligence (**AI**) and Natural Language Processing (**NLP**) to automatically detect fraudulent emails and classify them according to their level of danger.

The objective is to create an **interactive Web interface** that allows the user to better manage their inbox while ensuring enhanced protection against phishing.

The project relies on meticulous email **preprocessing** (removal of HTML tags, extraction of features such as the sender, subject length, etc.). These data are then analyzed using three machine learning models, which were tested : Support Vector Machine (**SVM**), Random Forest (**RF**), and **XGBoost**. These models are combined in a **Stacking** model to provide a robust and high-performance solution.

The user interface, developed with the **Streamlit** tool, allows the user to easily interact with the system, analyze their emails, and visualize the results in a clear and ergonomic way. The solution uses the **Gmail API** for automatic email extraction and allows for local processing for more complex analyses.

Tests conducted on various scenarios, such as phishing emails, promotional emails, and professional emails, showed that the model was effective in classifying emails with **high accuracy**.

Technical challenges, such as bugs related to the Gmail API, were encountered but were overcome through collaborative solutions and technical adjustments.

Finally, future improvements include the analysis of attachments (PJs), the integration of voice summaries, development of the LLM model, and expanding the solution to multiple languages for broader reach.

AVANT - PROPOS

Le phishing est une menace en constante évolution qui affecte des millions d'utilisateurs à travers le monde. L'essor des technologies numériques a entraîné une multiplication des techniques frauduleuses utilisées par les cybercriminels pour dérober des informations sensibles. Parmi ces techniques, le phishing demeure l'une des plus redoutables, car elle exploite la psychologie humaine et la crédulité des utilisateurs. Face à cette problématique, l'intelligence artificielle (IA) et le traitement du langage naturel (NLP) offrent des solutions innovantes pour détecter automatiquement les emails frauduleux et protéger les utilisateurs contre ce type d'attaque.

Ce projet s'inscrit dans cette dynamique de recherche en développant un système intelligent de détection de phishing. À travers l'analyse, le prétraitement et la modélisation des emails, nous visons à créer une solution capable de classifier les emails en fonction de leur niveau de dangerosité. L'objectif est d'offrir aux utilisateurs une interface claire et interactive pour gérer efficacement leur boîte mail tout en renforçant leur sécurité.

Bien que la lutte contre le phishing soit un défi complexe, l'IA permet d'améliorer considérablement la précision et l'efficacité des systèmes de détection. Ce document présente les différentes étapes du projet, de la collecte et l'analyse des données à la mise en place de modèles d'apprentissage automatique performants. À travers ce travail, nous explorons les potentialités de l'IA pour renforcer la cybersécurité et fournir des solutions adaptées aux défis actuels du phishing.

Table des matières

Table des figures

Liste des tableaux

INTRODUCTION GÉNÉRALE

INTRODUCTION GÉNÉRALE

À l'ère du numérique, l'e-mail reste l'un des outils de communication les plus largement utilisés dans le monde. Selon Statista (2025), plus de **370 milliards d'e-mails** sont échangés chaque jour à l'échelle mondiale. Cette volumétrie massive en fait une cible de choix pour les cybercriminels, et en particulier pour les campagnes de **phishing** (hameçonnage), qui visent à tromper les utilisateurs pour leur soutirer des informations personnelles ou sensibles. Le phishing représente aujourd'hui l'une des menaces les plus répandues et les plus sophistiquées en matière de cybersécurité.

Face à cette menace, les solutions actuellement disponibles, comme les **filtres intégrés** dans Gmail, Outlook ou Yahoo, ou les **passerelles de sécurité professionnelles** telles que Proofpoint ou Barracuda, montrent rapidement leurs limites. Les premières souffrent d'un manque de transparence et de personnalisation, tandis que les secondes, bien que très puissantes, sont souvent **complexes, coûteuses** et peu accessibles pour les utilisateurs individuels ou les petites structures.

Ce constat met en évidence le besoin d'un **système de détection intelligent**, capable de :

- **Détecter automatiquement** les e-mails frauduleux grâce à des modèles d'intelligence artificielle (IA) ;
- **Offrir une interface simple**, claire et interactive, adaptée aussi bien aux novices qu'aux utilisateurs avertis ;
- **Illustrer concrètement** l'usage de l'IA dans une application utile au quotidien ;
- **S'adapter aux évolutions** des menaces et aux besoins des utilisateurs.

Le présent rapport s'articule autour de cet enjeu. Il s'ouvre par une **analyse des méthodes de détection existantes** et un **choix raisonné de notre approche méthodologique**, en combinant NLP (Natural Language Processing) et les algorithmes du machine learning supervisé. Il se poursuit par une description de la **conception technique du système**, depuis la récupération des e-mails via l'API Gmail jusqu'à l'analyse automatisée via un pipeline de prétraitement et de classification. Enfin, une **démonstration concrète** de l'interface développée permet de valider l'efficacité du système et de proposer des pistes d'amélioration.

Ce projet s'inscrit dans une dynamique d'innovation au service de la cybersécurité, avec pour ambition de **rendre les technologies avancées de détection de phishing accessibles, transparentes et compréhensibles** pour tous.

CHAPITRE 1 : ANALYSE DE MÉTHODES DE DÉTECTION EXISTANTES & CHOIX MÉTHODOLOGIQUES

CHAPITRE 1 : ANALYSE DE MÉTHODES DE DÉTECTION EXISTANTES & CHOIX MÉTHODOLOGIQUES

Introduction :

Avec plus de **370 milliards d'e-mails envoyés quotidiennement** dans le monde (Statista, 2025), la messagerie électronique demeure l'un des principaux vecteurs d'attaques informatiques. Le **phishing (hameçonnage)** constitue une menace critique visant à tromper les utilisateurs pour dérober leurs données personnelles, professionnelles ou bancaires. Face à la sophistication croissante de ces attaques, il devient essentiel d'exploiter l'intelligence artificielle (IA) pour renforcer la détection.

Ce chapitre propose une analyse des **méthodes de détection existantes**, les défis actuels, ainsi qu'une **approche méthodologique** pour la conception d'un système IA de détection de phishing **modulaire, évolutif et accessible** à l'utilisateur final.

1- État de l'art : Méthodes de détection existantes :

1.1 Filtres intégrés aux clients de messagerie :

Des plateformes comme **Gmail, Outlook ou Yahoo Mail** intègrent des filtres automatiques :

- **Techniques utilisées** : listes noires, règles heuristiques, apprentissage supervisé.
- **Limites** : efficacité variable, opacité des modèles, faux positifs fréquents, faible transparence pour l'utilisateur.

1.2 Passerelles de sécurité professionnelles

Des solutions comme **Proofpoint**, **Barracuda** ou **Mimecast** proposent :

- **Détection avancée** via sandboxing, signatures, analyse comportementale.
- **Cible** : grandes entreprises, avec une configuration complexe et un coût élevé.
- **Limites** : inaccessibles aux utilisateurs particuliers et petites structures.

1.3 Solutions open-source :

Projets comme **SpamAssassin**, **MailScanner** ou **PhishTank** :

- Réputation IP, analyse des URL, détection de patterns HTML.
- Manquent souvent d'interface intuitive ou de mise à jour dynamique.

2- Objectifs fonctionnels du système proposé :

Afin de répondre aux limites observées, le système proposé vise à :

- ✓ **Détecter automatiquement les e-mails de phishing :** En combinant des techniques d'IA (NLP, modèles supervisés) avec une base de règles dynamiques.
- ✓ **Offrir une interface simple et explicite :** Accessible via navigateur, affichant : le score de confiance (phishing vs. Sain).
- ✓ **Illustrer concrètement l'usage de l'IA :** Le système met en évidence :
  - L'extraction automatique des entités clés (expéditeur, liens, etc).
  - L'interprétation du contenu via modèles de langage.
  - La décision issue d'un classificateur IA.
- ✓ **Être modulaire et évolutif :**
  - **Architecture en microservices** : traitement NLP, analyse d'URL, détection visuelle.
  - **Possibilité d'ajouter des modules futurs** : détection de spear phishing, détection par image (logos falsifiés), apprentissage actif par feedback utilisateur.

3- Choix méthodologiques :

3.1 Approche hybride basée sur l'IA :

Composant	Méthode choisie	Justification
Prétraitement	Nettoyage HTML, extraction d'entités	Permet de standardiser les entrées.
Analyse linguistique	NLP avec **BERT**	Pour détecter les formulations typiques de phishing.
Analyse technique	Vérification d'URL	Renforce la détection contextuelle.
Classification	Random Forest, SVM et XGBOOST	Modèles supervisés entraînés sur datasets labellisés.

Tableau 1 : Approche utilisée.

Conclusion :

Le phishing reste une menace grandissante dans l'écosystème numérique mondial. Bien que des solutions existent, elles sont soit opaques et limitées (Gmail, Outlook), soit trop complexes et onéreuses (Proofpoint, Barracuda). Ce rapport propose un **système IA modulaire et explicatif** pour détecter les e-mails frauduleux, accessible tant aux particuliers qu'aux professionnels.

Il conjugue des **techniques modernes de NLP**, une **interface WEB interactive** et une **architecture évolutive**, constituant une réponse concrète et pédagogique à la menace du phishing.

CHAPITRE 2 : CONCEPTION & RÉALISATION TECHNIQUE

CHAPITRE 2 : CONCEPTION & RÉALISATION TECHNIQUE

1- Architecture du système :

1.1 Vue Générale :

Le projet de détection de phishing repose sur une architecture modulaire et automatisée, allant de la récupération des e-mails en temps réel jusqu'à l'affichage des résultats à l'utilisateur à travers une interface web interactive. L'objectif est d'identifier de manière fiable les courriels frauduleux, en combinant l'intelligence artificielle (Machine Learning), le traitement automatique du langage (NLP) et une intégration fluide avec Gmail via son API.

L'architecture suit un **pipeline en quatre grandes étapes** :

1. **Collecte en temps réel des e-mails via l'API Gmail ;**
2. **Traitement et ingénierie des données ;**
3. **Classification par modèles de Machine Learning ;**
4. **Visualisation des résultats via une interface web.**

1.2 Architecture Globale & Schéma fonctionnel :

Le schéma ci-dessous illustre le fonctionnement général du système, depuis la réception de l'e-mail jusqu'à l'affichage du diagnostic final à l'utilisateur :

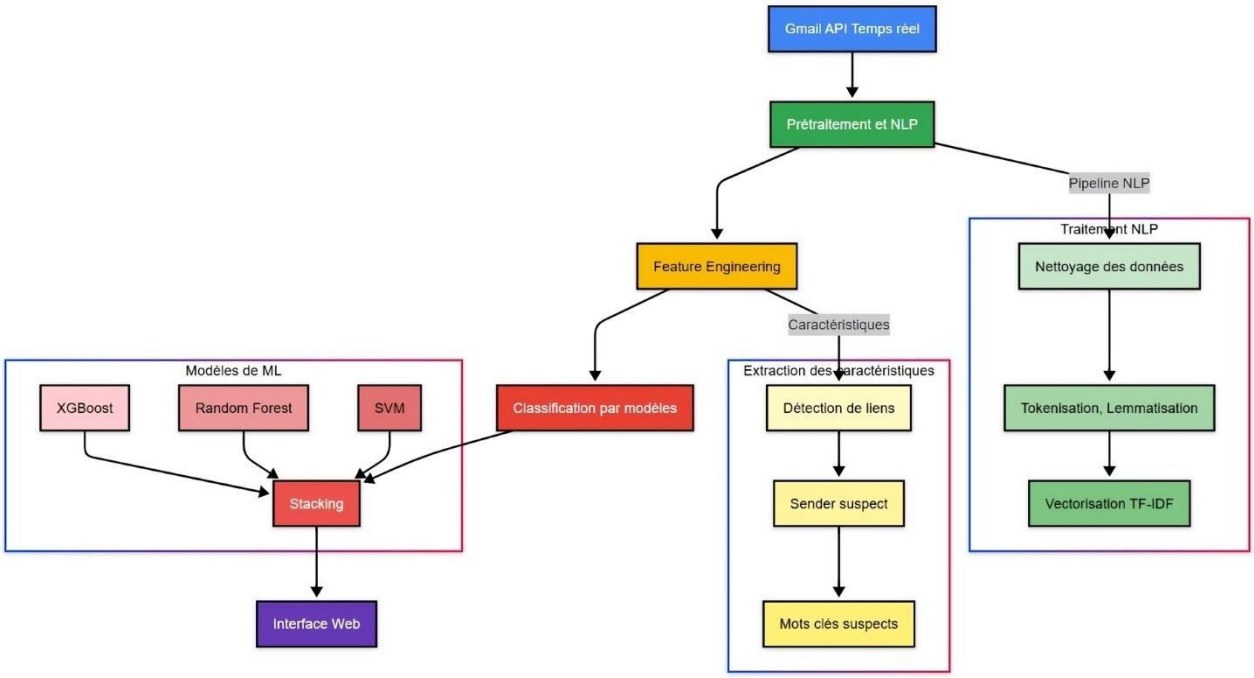

Figure 1 : Architecture Globale & Schéma fonctionnel.

2- Prétraitement et feature engineering :

Introduction :

Ce projet vise à extraire, nettoyer et enrichir les emails issus du dataset **CEAS 2008** afin de créer un pipeline de prétraitement robuste pour la détection de **phishing** ou de **spam**. On combine des approches de nettoyage, d'extraction de features, de traitement du langage naturel (NLP) et d'encodage vectoriel pour préparer les données à l'entraînement d'un modèle de machine learning.

2.1 Chargement et nettoyage de base :

```
### Nettoyage des données

# Load the dataset
df = pd.read_csv('data/CEAS_08.csv')
print(df.head())

# Remove duplicates
df = df.drop_duplicates()

# Drop rows where critical columns are empty
df = df.dropna(subset=['sender', 'receiver', 'subject', 'body', 'label'])

# Convert 'date' column to datetime, handling mixed time zones and invalid dates
df['date'] = pd.to_datetime(df['date'], errors='coerce', utc=True)

# Remove rows where 'date' is NaT (invalid dates)
df = df.dropna(subset=['date'])
```

Figure 2 : Nettoyage des données.

❖ **Explication :**

- On charge les données brutes depuis un fichier CSV ;
- Suppression des doublons et des lignes contenant des valeurs manquantes essentielles **(sender, receiver, subject, body, label)** ;
- Conversion de la colonne date en format datetime avec gestion des erreurs (valeurs invalides → NaT), ensuite supprimées.

⇨ S'assurer que le dataset est équilibré :

```
# Check the first few rows
print(df.head())

# Check dataset info
print(df.info())

# Save the cleaned dataset
#df.to_csv('data/cleaned_CEAS_08.csv', index=False)


# Comparer le nombre de 0 et 1 dans le label
label_counts = df['label'].value_counts()
print(label_counts)

# Calculer la proportion de chaque classe
label_proportions = df['label'].value_counts(normalize=True) * 100
print("\nProportion des classes (%) :")
print(label_proportions)
```

Figure 3 : Vérification du dataset.

2.2 Extraction de domaines et analyse des expéditeurs :

```python
# Extract domains from email addresses
df['sender_domain'] = df['sender'].str.extract(r'@([\w\.-]+)')
df['receiver_domain'] = df['receiver'].str.extract(r'@([\w\.-]+)')

#Calculer la longueur du nom d'utilisateur dans sender.
df['sender_name_length'] = df['sender'].str.split('@').str[0].str.len()
```

Figure 4 : Extraction de domaines et analyse des expéditeurs.

❖ **Explication :**

- On extrait les domaines d'expédition/réception pour repérer les entités douteuses. Le flag **suspicious_sender** est défini si le nom d'utilisateur de l'adresse contient un chiffre, souvent utilisé dans des adresses frauduleuses.
- On mesure également la longueur du nom d'expéditeur pour détecter d'éventuelles anomalies (**longs identifiants masqués**, etc.).

2.3 Extraction d'URLs et analyse des liens suspects :

```python
df['suspicious_sender'] = df['sender'].str.split('@').str[0].str.contains(r'\d', regex=True).astype(int)


### Extraire les URLs de body et créer num_urls et suspicious_urls


# Liste de domaines suspects (a developper)
suspicious_domains = ['bit.ly', 'tinyurl.com', 'goo.gl']

# Extraire les URLs
Windsurf: Refactor | Explain | Generate Docstring | ×
def extract_urls(text):
    return re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', text)

df['urls_list'] = df['body'].apply(extract_urls)
df['num_urls'] = df['urls_list'].apply(len)

# Vérifier les domaines suspects
Windsurf: Refactor | Explain | Generate Docstring | ×
def check_suspicious_urls(urls):
    for url in urls:
        for domain in suspicious_domains:
            if domain in url:
                return 1
    return 0

df['suspicious_urls'] = df['urls_list'].apply(check_suspicious_urls)
df = df.drop(columns=['urls_list'])  # Supprimer la colonne temporaire
```

Figure 5 : Extraction d'URLs.

❖ **Explication :**

- On extrait toutes les URLs contenues dans les corps de mail.
- On compte le nombre d'URLs et on détecte si l'une d'elles appartient à un domaine "suspicious" (ex : bit.ly, tinyurl, goo.gl).

2.4 Détection de mots clés de Phishing :

```python
#Définir une liste de mots suspects et compter leur présence dans subject et body.
phishing_keywords = [
    ##### 1. Urgence & Menaces #####
    'urgent', 'immédiat', 'délai', 'dernière chance', 'expir', 'suspendu',
    'bloqué', 'hacké', 'piraté', 'séquestr', 'confisqu', 'résili', 'litige',
    'restrict', 'violation', 'avertissement', 'pénalité', 'désactiv',

    ##### 2. Sécurité & Comptes #####
    'vérif', 'authentif', 'sécur', 'connexion', 'identif', 'mot de passe',
    'credentials', '2FA', 'MFA', 'OTP', 'code', 'validation', 'protect',
    'lock', 'unlock', 'reactiv', 'recovery', 'reset',

    ##### 3. Argent & Paiements #####
    'facture',  '€' , '$', 'paiement', 'rembours', 'virement', 'solde', 'impayé',
    'prélèvement', 'carte bancaire', 'IBAN', 'SWIFT', 'crypto', 'bitcoin',
    'transaction', 'fraude', 'remise', 'réduct', '100', '99', '1000' , 'million', 'prix',

    ##### 4. Récompenses & Offres #####
    'gratuit', 'cadeau', 'lotterie', 'prime', 'bonus', 'exclusif',
    'limitée', 'spécial', 'promo', 'rabais', 'coupon', 'gagnant',
    'million', 'euro', 'dollar', 'cash',

    ##### 7. Typosquatting & Caractères Spéciaux #####
    r'[àéèçù]',  # Caractères accentués suspects
    r'\d+€',     # Montants monétaires
    r'[\!\?\*]{2,}',  # Ponctuations multiples
    r'\b(?:paypal|amazon|google|microsoft)\W+\w+',  # Marques + caractères spéciaux

]

Windsurf: Refactor | Explain | Generate Docstring | ×
def count_phishing_words(text, keywords):
    text = text.lower()
    return sum(text.count(word) for word in keywords)


df['phishing_words'] = df['subject'].fillna('') + ' ' + df['body'].fillna('')
df['phishing_words'] = df['phishing_words'].apply(lambda x: count_phishing_words(x, phishing_keywords))
```

Figure 6 : Détection de mots clés.

❖ Explication :

- Un dictionnaire de mots-clés liés aux attaques de phishing est utilisé pour scanner subject et body.
- On compte le nombre d'occurrences de ces mots pour créer une variable explicative.

2.5 Analyse de la densité de caractères spéciaux :

```python
#Calculer la proportion de caractères spéciaux dans body.
special_chars = r'[!@#$%^&*(),.?":{}|<>]'

Windsurf: Refactor | Explain | Generate Docstring | ×
def special_char_density(text):
    if not text or len(text) == 0:
        return 0
    special_count = len(re.findall(special_chars, text))
    return special_count / len(text)

df['special_char_density'] = df['body'].apply(special_char_density)
```

Figure 7 : Analyse de la densité de caractères spéciaux.

❖ Explication :

- Les emails frauduleux utilisent souvent des ponctuations excessives (**!!!, $$$**, etc.) ou des symboles. Ce ratio permet de quantifier ce comportement.

2.6 Prétraitement du texte :

```
### Démarrer le prétraitement NLP

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

Windsurf: Refactor | Explain | Generate Docstring | ×
def preprocess_text(text):
    text = text.lower()  # Mise en minuscules
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Supprimer caractères spéciaux
    text = re.sub(r'\s+', ' ', text).strip()  # Enlève les espaces multiples
    tokens = word_tokenize(text)   # Tokenisation
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words]
    return ' '.join(tokens)

df['subject_clean'] = df['subject'].apply(preprocess_text)
df['body_clean'] = df['body'].apply(preprocess_text)
```

Figure 8 : Prétraitement du texte.

❖ **Explication :**

- **Nettoyage textuel** : minuscules, suppression des caractères spéciaux, tokenisation, lemmatisation.
- **Suppression des mots vides** (**stopwords**) pour ne garder que les mots porteurs de sens.

2.7 Encodage des Données Catégorielles :

```
### Encodage et Vectorisation

le_sender = LabelEncoder()
le_receiver = LabelEncoder()

df['sender_domain_encoded'] = le_sender.fit_transform(df['sender_domain'].astype(str))
df['receiver_domain_encoded'] = le_receiver.fit_transform(df['receiver_domain'].astype(str))
```

Figure 9 : Encodage et vectorisation.

❖ **Explication** :

- Transformation des domaines textuels (sender_domain, receiver_domain) en entiers pour être utilisables par les modèles ML.

2.8 Vectorisation TF-IDF du Texte :

```
### Encodage et Vectorisation

le_sender = LabelEncoder()
le_receiver = LabelEncoder()

df['sender_domain_encoded'] = le_sender.fit_transform(df['sender_domain'].astype(str))
df['receiver_domain_encoded'] = le_receiver.fit_transform(df['receiver_domain'].astype(str))

# Vectoriser subject_clean et body_clean
tfidf_subject = TfidfVectorizer(max_features=1000)
tfidf_body = TfidfVectorizer(max_features=1000)

subject_vecs = tfidf_subject.fit_transform(df['subject_clean'])
body_vecs = tfidf_body.fit_transform(df['body_clean'])

joblib.dump(tfidf_subject, 'models/tfidf_subject.pkl')
joblib.dump(tfidf_body, 'models/tfidf_body.pkl')
```

Figure 10 : Encodage et vectorisation.

❖ **Explication :**

- On vectorise les textes nettoyés de subject et body avec TF-IDF pour capter leur importance relative dans l'ensemble.
- Les objets TF-IDF sont sauvegardés pour une utilisation future (en prédiction par exemple).

2.9 Assemblage des Features et Entraînement du Modèle :

```
### Assemblage des Features

numeric_features = df[[
    'num_urls',
    'suspicious_urls',
    'sender_name_length',
    'phishing_words',
    'special_char_density',
    'sender_domain_encoded',
    'receiver_domain_encoded'
]]

numeric_sparse = csr_matrix(numeric_features.values)

X_combined = hstack([numeric_sparse, subject_vecs, body_vecs])

final_df = pd.DataFrame.sparse.from_spmatrix(
    X_combined,
    columns=(
        numeric_features.columns.tolist() +
        [f'subject_tfidf_{i}' for i in range(subject_vecs.shape[1])] +
        [f'body_tfidf_{i}' for i in range(body_vecs.shape[1])]
    )
)

final_df['label'] = df['label'].values

final_df.to_csv('data/cleaned_phishing_data.csv', index=False)

joblib.dump(le_sender, 'models/le_sender.pkl')
```

Figure 11 : Assemblage des features.

2.10 Intégration de l'API Gmail :

L'intégration de l'API Gmail constitue une étape cruciale de notre projet, permettant l'accès direct et sécurisé aux emails de l'utilisateur. Elle a pour objectif de récupérer les messages depuis une boîte Gmail, d'en extraire les informations pertinentes (corps du message, objet, expéditeur, URLs, pièces jointes) et de les formater pour alimenter notre système de classification anti-phishing.

2.10.1 Configuration de Google Cloud Console :

Pour permettre l'extraction automatique des e-mails depuis une boîte Gmail, il a été nécessaire de configurer un projet dédié sur la plateforme **Google Cloud Console**. Cette configuration a permis de mettre en place une authentification sécurisée via le protocole **OAuth 2.0**. Les étapes suivies sont :

- **Création d'un projet Google Cloud** : un nouveau projet a été initialisé pour gérer les appels à l'API Gmail.

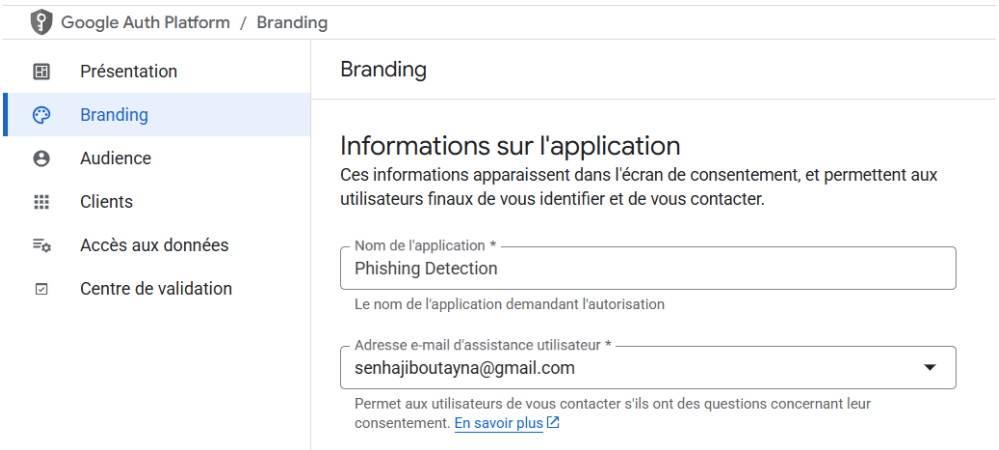

Figure 12 : Configuration de Google Cloud Consol.

- **Activation de l'API Gmail** : via la section « APIs & Services », l'API Gmail a été activée, permettant à notre application d'interagir avec Gmail.
- **Création d'un écran d'autorisation OAuth (OAuth consent screen)** :
  - ✓ Type d'utilisateur : Externe ;
  - ✓ Rôle de testeur : nos adresses email ont été ajoutées pour autoriser temporairement l'utilisation du projet sans validation publique complète ;
  - ✓ Informations affichées à l'utilisateur : nom de l'application, adresse email, logo (facultatif), etc.

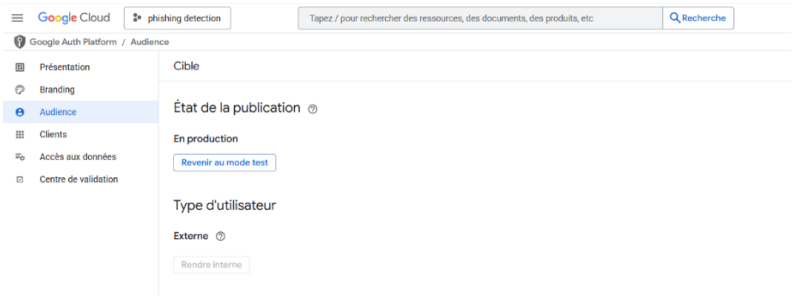

Figure 13 : OAuth consent screen.

- **Création d'identifiants OAuth 2.0 :**
  - ✓ Type : Application de bureau (Desktop App) ;
  - ✓ Notre client OAuth généré est lié à un identifiant **client_id** et un **client_secrets** nécessaires pour le processus d'authentification.

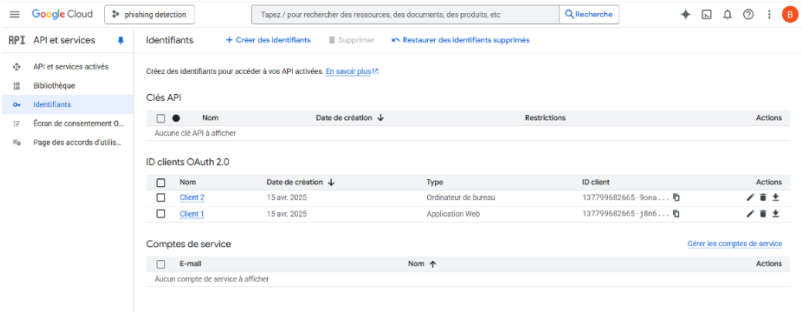

Figure 14 : Création d'identifiants OAuth 2.0.

- **Téléchargement du fichier credentials.json :**
  - ✓ Ce fichier est utilisé dans notre script pour déclencher le flux OAuth local.
  - ✓ Il contient les informations d'identification de l'application et les URI de redirection.

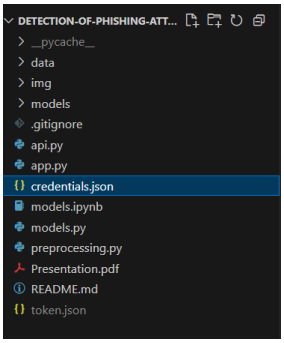

Figure 15 : Fichier credentials.json.

2.10.2 Connexion et extraction automatisée des e-mails :

Une fois la configuration Google Cloud terminée, nous avons développé un module Python complet pour récupérer automatiquement les e-mails récents à partir de l'API Gmail, les analyser, puis les intégrer dans notre pipeline de prédiction.

⇨ *Authentification sécurisée avec OAuth 2.0 :*

L'authentification est gérée dans la fonction **authenticate_gmail()**. Elle utilise les fichiers **credentials.json** (identifiants du projet Google Cloud) et **token.json** (jeton d'accès généré après la première connexion). Si le jeton est absent ou expiré, l'utilisateur est redirigé vers un navigateur pour authentification. Le nouveau jeton est ensuite sauvegardé localement.

```python
# Portée d'accès : ici lecture seule sur la boîte mail
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']

# Authentification à Gmail avec OAuth2
Windsurf: Refactor | Explain | Generate Docstring | ×
def authenticate_gmail():
    creds = None  # Initialisation des credentials
    # Vérifie si un jeton existe déjà (connexion précédente)
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    # Si pas de jeton ou jeton invalide
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            # Rafraîchit le jeton s'il est expiré
            creds.refresh(Request())
        else:
            # Sinon, lance le processus d'authentification utilisateur
            flow = InstalledAppFlow.from_client_secrets_file('credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        # Sauvegarde du nouveau jeton
        with open('token.json', 'w') as token:
            token.write(creds.to_json())
    # Retourne le service Gmail prêt à l'usage
    return build('gmail', 'v1', credentials=creds)
```

Figure 16 : Connexion et extraction automatisée des e-mails.

⇨ *Traitement du corps du message :*

Le contenu du message est souvent encodé en base64. Nous le décodons et traitons le HTML si nécessaire avec la librairie **BeautifulSoup.**

```python
def get_email_body(message_payload):
    parts = message_payload.get('parts', [])
    body = ""
    for part in parts:
        mime_type = part.get('mimeType')
        data = part.get('body', {}).get('data')
        # Si texte brut
        if mime_type == 'text/plain' and data:
            body = base64.urlsafe_b64decode(data).decode('utf-8')
        # Si HTML (utilisé seulement si le texte brut est vide)
        elif mime_type == 'text/html' and data and not body:
            html = base64.urlsafe_b64decode(data).decode('utf-8')
            soup = BeautifulSoup(html, 'html.parser')
            body = soup.get_text()
    return body.strip()  # Nettoyage du texte
```

Figure 17 : Traitement du corps du message.

⇨ *Extraction des entités utiles :*

Trois types de données sont extraites :

- **URLs :** grâce à des expressions régulières (re.findall()) ;
- **Pièces jointes : en inspectant les parts du payload ;**
- **Features personnalisées : dans la fonction process_email(), plusieurs indicateurs sont construits :**
  - ✓ Longueur du nom de l'expéditeur (sender_name_length) ;
  - ✓ Densité de caractères spéciaux ;
  - ✓ Nombre d'URLs ;
  - ✓ Présence d'URLs raccourcies (bit.ly, tinyurl, etc.) ;
  - ✓ Nombre de mots-clés de phishing.

```python
def process_email(subject, body, sender, receiver_domain="example.com"):
    # Récupération du domaine de l'expéditeur
    sender_domain = sender.split('@')[-1].strip('> ') if '@' in sender else "unknown"

    # Encodage du domaine de l'expéditeur
    try:
        sender_domain_encoded = le_sender.transform([sender_domain])[0]
    except ValueError:
        sender_domain_encoded = -1  # Si domaine inconnu → -1

    # Encodage du domaine du destinataire
    try:
        receiver_domain_encoded = le_receiver.transform([receiver_domain])[0]
    except ValueError:
        receiver_domain_encoded = -1

    # Création du dictionnaire de caractéristiques (features)
    features = {
        'sender_name_length': len(sender.split('@')[0]),  # Longueur du nom d'expéditeur (avant @)
        'special_char_density': len(re.findall(r'[!@#$%^&*(),.?":{}|<>]', body)) / max(len(body), 1),  # Densité des caractères spéciaux
        'sender_domain_encoded': sender_domain_encoded,
        'num_urls': len(extract_urls(body)),  # Nombre d'URLs dans le corps de l'email
        'suspicious_urls': 1 if any(domain in url for url in extract_urls(body) for domain in ['bit.ly', 'tinyurl.com', 'goo.gl']) else 0,  # Pr
        'phishing_words': sum(1 for word in ['urgent', 'gratuit', 'prix', 'compte', 'confidentiel'] if word in (subject + ' ' + body).lower()),
        'receiver_domain_encoded': receiver_domain_encoded,
    }

    return features  # Retourne les features pour l'email
```

Figure 18 : Extraction des entités utiles.

Certaines données sont encodées à l'aide de **LabelEncoder** (chargés avec **joblib**) pour homogénéiser les domaines inconnus.

⇨ *Prédiction par modèle ML :*

Les vecteurs d'attributs sont ensuite injectés dans le **modèle de classification** (entraîné et sauvegardé auparavant).

```python
def predict_phishing(features):
    # Chargement du modèle de classification entraîné
    model = joblib.load('models/meta_model.pkl')
    # Conversion des features en DataFrame (attendu par le modèle)
    df = pd.DataFrame([features], columns=[
        'sender_domain_encoded',
        'phishing_words',
        'receiver_domain_encoded'
    ])
    # Prédiction à partir du modèle
    prediction = model.predict(df)

    return prediction[0]  # Retourne 0 (safe) ou 1 (phishing)
```

Figure 19 : Prédiction par modèle ML.

Le tout est orchestré dans **fetch_emails()**, qui affiche un résumé complet pour chaque e-mail analysé.

```python
def fetch_emails():
    service = authenticate_gmail()  # Authentifie et construit le service Gmail
    results = service.users().messages().list(userId='me', maxResults=10).execute()
    messages = results.get('messages', [])
    print(f"\n {len(messages)} emails found:\n")
    for i, message in enumerate(messages, 1):
        msg = service.users().messages().get(userId='me', id=message['id'], format='full').execute()
        headers = msg['payload']['headers']

        subject = sender = "N/A"
        # Récupère le sujet et l'expéditeur depuis les en-têtes
        for header in headers:
            if header['name'] == 'Subject':
                subject = header['value']
            elif header['name'] == 'From':
                sender = header['value']

        body = get_email_body(msg['payload'])  # Récupération du corps de l'email
        urls = extract_urls(body)  # Extraction des URLs
        attachments = get_attachments(service, msg)  # Extraction des pièces jointes
        # Création des features + prédiction
        features = process_email(subject, body, sender)
        is_phishing = predict_phishing(features)
        # Affichage des informations de l'email + prédiction
        print(f"Email {i}")
        print(f"From: {sender}")
        print(f"Subject: {subject}")
        print(f"Body:\n{body[:300]}...")  # Affiche un extrait du corps
        print(f"URLs found: {urls}")
        print(f"Prediction: {'Phishing' if is_phishing else 'Safe'}\n")
        print("-" * 80)
```

Figure 20 : fetching des emails.

3- Modélisation & évaluation :

Introduction :

Dans cette partie, l'objectif fondamental est d'identifier le modèle du Machine Learning le plus performant pour la détection des e-mails de phishing. Cette étape est cruciale dans la lutte contre les cyberattaques, car une détection rapide et fiable permet de réduire significativement les risques liés à la sécurité des systèmes d'information.

Pour ce faire, trois (3) modèles de classification supervisée ont été sélectionné, chacun reposant sur des principes algorithmiques différents et complémentaires. En plus d'un modèle de Stacking qui combine les forces des modèles utilises. Ces modèles sont :

- **XGBoost** : a été retenu pour sa capacité à gérer des datasets complexes et déséquilibrés, tout en offrant d'excellentes performances en termes de précision et de robustesse.
- **SVM** (Support Vector Machine) : a été choisi pour sa capacité à maximiser la séparation entre les classes, ce qui peut s'avérer utile dans les contextes où les limites de décision sont complexes.
- **Random Forest** : repose sur l'agrégation de multiples arbres de décision, ce qui permet de limiter les risques de surapprentissage (Overfitting) tout en conservant une bonne interprétabilité des résultats.

Enfin, le modèle de Stacking a été implémenté pour **combiner** les **forces** des **modèles précédents**, en construisant un méta-modèle capable d'exploiter les prédictions des classifieurs de base afin d'améliorer la performance globale.

3.1 Description des données :

Le dataset utilisé, **CEAS_08.csv**, constitue la base de l'analyse menée pour la détection automatique des e-mails de phishing. Il est composé d'un ensemble de caractéristiques extraites des e-mails, sélectionnées pour leur potentiel à révéler des tentatives de phishing. Ces attributs ont été choisis pour leur pertinence dans la différenciation entre e-mails légitimes et tentatives de phishing. Parmi ces caractéristiques, on trouve :

- **Sender_name_length** : Longueur du nom de l'expéditeur ;
- **special_char_density** : Densité des caractères spéciaux dans l'e-mail ;
- **sender_domain_encoded** : Domaine de l'expéditeur encodé ;
- **num_urls** : Nombre d'URLs dans l'e-mail ;
- **suspicious_urls** : Indicateur de la présence d'URLs suspectes ;
- **phishing_words** : Nombre de mots liés au phishing dans le contenu de l'e-mail ;
- **receiver_domain_encoded** : Domaine du récepteur encodé ;
- **La cible (y)** est le **label** qui indique si l'e-mail est un phishing (1) ou non (0).

3.2 Modèles de Machine Learning :

3.2.1 XGBOOST :

Importation des bibliothèques :

Les bibliothèques essentielles sont importés a savoir **numpy** et **pandas** pour la manipulation de données, **matplotlib.pyplot** pour la visualisation, **xgboost** pour le modèle de classification, et des modules de **scikit-learn** pour la séparation des données, le prétraitement, et les métriques d'évaluation.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

Figure 21 : Importation des bibliothèques.

Chargement des données :

Le fichier de données nommé **cleaned_phishing_data.csv** est lu à l'aide de **pandas.read_csv**. Ce fichier contient des e-mails avec diverses caractéristiques permettant d'identifier les tentatives de phishing.

```
file_path = '/content/cleaned_phishing_data.csv'
data = pd.read_csv(file_path)
```

Figure 22 : Chargement du fichier CSV.

Nettoyage des données :

Suppression des lignes où la colonne **label** (la cible) est manquante.

```
data_clean = data.dropna(subset=['label'])
```

Figure 23 : Suppression du label.

Séparation des features (X) et de la cible (y) :

Les colonnes d'entrée (**features**) sont stockées dans X et la cible (le label) est stockée dans y.

```
X = data_clean.drop(columns=['label'])
y = data_clean['label']
```

Figure 24 : Séparation des features.

Division en données d'entraînement et de test :

Le dataset est divisé en deux ensembles : 80% pour l'entraînement et 20% pour le test. Le paramètre **stratify=y** garantit que la répartition des classes est équilibrée dans les deux sous-ensembles.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Figure 25 : Données d'entraînement et de test.

Normalisation des données :

Les valeurs des caractéristiques sont mises à l'échelle entre 0 et 1 grâce à **MinMaxScaler**, ce qui est utile pour améliorer la performance des modèles, notamment ceux sensibles à l'échelle des variables comme SVM ou XGBoost.

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 26 : Normalisation des données.

*Entraînement du modèle **XGBoost** :*

Le modèle **XGBoost** est instancié avec des paramètres spécifiques (n_estimators=100 (nombre d'arbres), max_depth=6 (profondeur maximale des arbres)). Il est ensuite entraîné sur les données normalisées d'entraînement.

```
xgb_model = xgb.XGBClassifier(objective='binary:logistic', use_label_encoder=False,
                              eval_metric='logloss', n_estimators=100, max_depth=6)
xgb_model.fit(X_train_scaled, y_train)
```

Figure 27 : Modèle XGBoost.

Prédictions et évaluation du modèle :

Le modèle prédit les classes (y_pred_xgb) et les probabilités (y_pred_proba_xgb) pour les données de test. Deux métriques sont calculées : la précision (accuracy) et l'AUC ROC, qui évalue la qualité de la séparation entre les classes.

```
y_pred_xgb = xgb_model.predict(X_test_scaled)
y_pred_proba_xgb = xgb_model.predict_proba(X_test_scaled)[:, 1]

accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
roc_auc_xgb = roc_auc_score(y_test, y_pred_proba_xgb)

print(f"XGBoost - Accuracy: {accuracy_xgb:.4f}")
print(f"XGBoost - ROC AUC: {roc_auc_xgb:.4f}")
```

Figure 28 : Prédictions et évaluation.

Affichage et matrice de confusion :

La matrice de confusion permet de visualiser les performances du modèle en termes de vrais positifs, faux positifs, vrais négatifs et faux négatifs.

```
cm = confusion_matrix(y_test, y_pred_xgb)
fpr, tpr, _ = roc_curve(y_test, y_pred_proba_xgb)

plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Matrice de confusion - XGBoost")
plt.colorbar()
plt.ylabel('Vrai')
plt.xlabel('Prédiction')
plt.xticks([0, 1], ['Non-Spam', 'Spam'])
plt.yticks([0, 1], ['Non-Spam', 'Spam'])
plt.show()
```

Figure 29 : Affichage et matrice de confusion.

Traçage de la courbe ROC :

La courbe ROC est généré et affiché. Elle montre la relation entre le taux de faux positifs et de vrais positifs. Plus la courbe se rapproche du coin supérieur gauche, plus le modèle est performant. L'AUC est aussi affichée pour résumer la performance.

```
fpr, tpr, _ = roc_curve(y_test, y_pred_proba_xgb)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'XGBoost (AUC = {roc_auc_xgb:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('Taux de faux positifs')
plt.ylabel('Taux de vrais positifs')
plt.title('Courbe ROC - XGBoost')
plt.legend(loc="lower right")
plt.show()
```

Figure 30 : Courbe ROC.

Output :

```
XGBoost - Accuracy: 0.9957
XGBoost - ROC AUC: 0.9998
```

Figure 31 : Output du XGBOOST.

Le modèle XGBoost utilisé est très efficace pour la classification des e-mails spam ou non, avec une **Accuracy de 99,57 %** et une **AUC ROC de 0,9998**, il montre une excellente capacité à détecter les e-mails de phishing tout en minimisant les erreurs de classification.

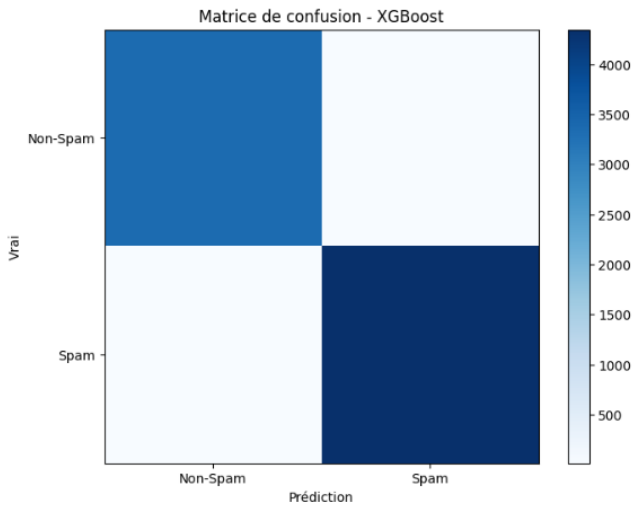

Figure 32 : Matrice de confusion – XGBoost.

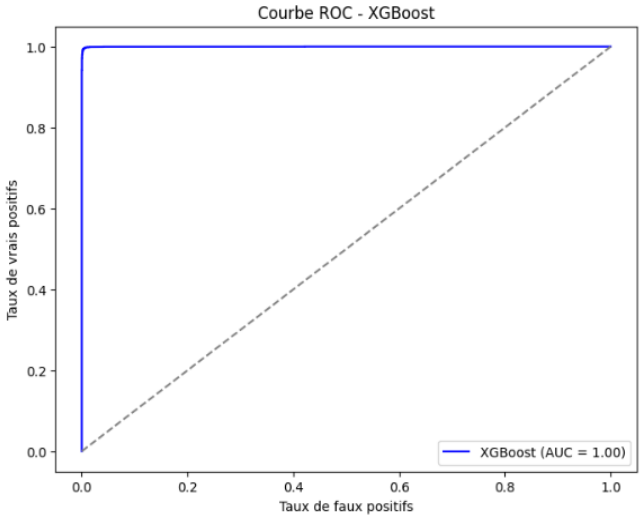

Figure 33 : Courbe ROC - XGBoost.

3.2.2 Support Vector Machine (SVM) :

Support Vector Machine (SVM) est un algorithme de classification supervisée qui cherche à trouver l'hyperplan optimal séparant les données de différentes classes avec la plus grande marge possible. Il est efficace pour les problèmes linéaires et non linéaires grâce à l'utilisation de noyaux (comme le noyau RBF).

Entraînement du modèle SVM :

Un modèle SVM avec un **noyau RBF** est utilisé pour gérer des données non linéaires. Il est configuré avec **C=1.0** pour la régularisation, **gamma='scale'** pour ajuster automatiquement l'influence des points d'entraînement, et **probability=True** pour permettre l'estimation des probabilités. Il est entraîné sur les données **X_train_scaled** et **y_train.**

```
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', probability=True)
svm_model.fit(X_train_scaled, y_train)
```

Figure 34 : Entrainement du modèle SVM.

Évaluation du SVM :

Le modèle SVM prédit les classes avec y_pred_svm et les probabilités d'appartenance à la classe "spam" avec y_pred_proba_svm. Ses performances sont évaluées à l'aide de **l'accuracy** (précision globale) et de **l'AUC ROC** (capacité à distinguer les classes).

```
y_pred_svm = svm_model.predict(X_test_scaled)
y_pred_proba_svm = svm_model.predict_proba(X_test_scaled)[:, 1]

accuracy_svm = accuracy_score(y_test, y_pred_svm)
roc_auc_svm = roc_auc_score(y_test, y_pred_proba_svm)
```

Figure 35 : Evaluation du SVM.

Matrice de confusion et courbe ROC – SVM :

La matrice de confusion cm_svm fournit des informations sur les performances du modèle en indiquant le nombre de **vrais positifs, faux positifs, vrais négatifs et faux négatifs**. Les variables fpr_svm (taux de faux positifs) et tpr_svm (taux de vrais positifs) sont ensuite utilisées pour tracer la **courbe ROC**, qui permet d'évaluer visuellement la qualité de la classification.

```
cm_svm = confusion_matrix(y_test, y_pred_svm)
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_pred_proba_svm)
```

Figure 36 : Matrice de confusion et courbe ROC – SVM

Visualisation :

La matrice de confusion et courbe ROC permet de **visualiser l'efficacité** du modèle SVM :

- La matrice de confusion donne un aperçu clair des prédictions correctes ou erronées.
- La courbe ROC permet d'évaluer **la performance globale de la classification.**

```
plt.imshow(cm_svm, interpolation='nearest', cmap=plt.cm.Blues)
plt.plot(fpr_svm, tpr_svm, color='blue', label=f'SVM (AUC = {roc_auc_svm:.2f})')
```

Figure 37 : visualisation Matrice de confusion et courbe ROC – SVM

Output :

Le modèle SVM a été évalué pour la détection de spam à l'aide de plusieurs métriques et visualisations, révélant d'excellentes performances globales :

- Précision (**Accuracy**) : Avec un score de **0,9938**, le modèle classe correctement près de **99,4 %** des exemples, ce qui montre une très bonne fiabilité.
- AUC **ROC** : Une valeur de **0,9985** à **1,0**, indiquant une capacité quasi parfaite à distinguer les spams des non-spams.

```
SVM - Accuracy: 0.9938
SVM - ROC AUC: 0.9985
```

Figure 38 : Output du SVM.

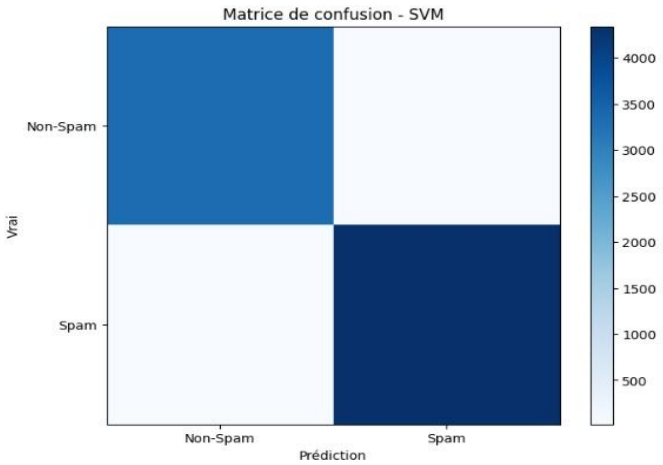

Figure 39 : Matrice de confusion – SVM.

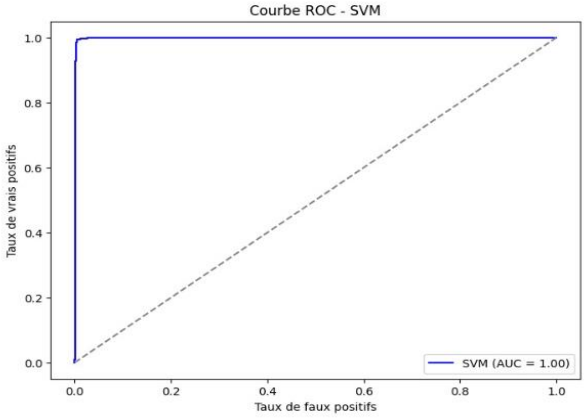

Figure 40 : Courbe ROC - SVM.

3.2.3 Random Forest :

Entraînement du modèle Random Forest :

Le modèle **Random Forest** est composé de **100 arbres de décision**, ce qui permet d'améliorer la stabilité des prédictions. La profondeur de chaque arbre est limitée **à 6**, afin d'éviter un surapprentissage sur les données d'entraînement. Le paramètre **random_state=42** est utilisé pour garantir que les résultats soient cohérents et reproductibles à chaque exécution.

```
rf_model = RandomForestClassifier(n_estimators=100, max_depth=6, random_state=42)
rf_model.fit(X_train_scaled, y_train)
```

Figure 41 : Entrainement du modèle Random Forest.

Évaluation du Random Forest :

Comme pour le modèle SVM, le modèle Random Forest génère à la fois les **prédictions de classes** et les **probabilités associées** à chaque instance. Ces résultats permettent ensuite de calculer deux métriques clés : **l'accuracy**, qui mesure la proportion de prédictions correctes, et **l'AUC ROC**, qui évalue la capacité du modèle à distinguer les spams des non-spams.

```
y_pred_rf = rf_model.predict(X_test_scaled)
y_pred_proba_rf = rf_model.predict_proba(X_test_scaled)[:, 1]

accuracy_rf = accuracy_score(y_test, y_pred_rf)
roc_auc_rf = roc_auc_score(y_test, y_pred_proba_rf)
```

Figure 42 : Evaluation du modèle Random Forest

Matrice de confusion et courbe ROC - Random Forest :

La matrice de confusion est utilisée pour évaluer les erreurs de classification du modèle Random Forest, et extrait les données nécessaires pour tracer la courbe ROC, afin d'analyser sa capacité à distinguer les classes spam et non-spam.

```
cm_rf = confusion_matrix(y_test, y_pred_rf)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_proba_rf)
```

Figure 43 : Matrice de confusion et courbe ROC – RF.

Output :

Le modèle **Random Forest** montre d'excellentes performances pour la détection de spam, avec une **accuracy de 97,41 %** et un **AUC de 0,9979**. La **matrice de confusion** indique très peu d'erreurs de classification. La **courbe ROC** montre une séparation quasi parfaite entre les classes, confirmant la fiabilité du modèle.

```
Random Forest - Accuracy: 0.9741
Random Forest - ROC AUC: 0.9979
```

Figue 44 : Output du RF.

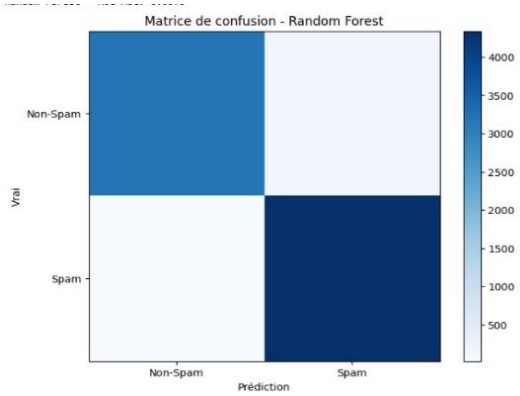

Figure 45 : Matrice de confusion – RF.

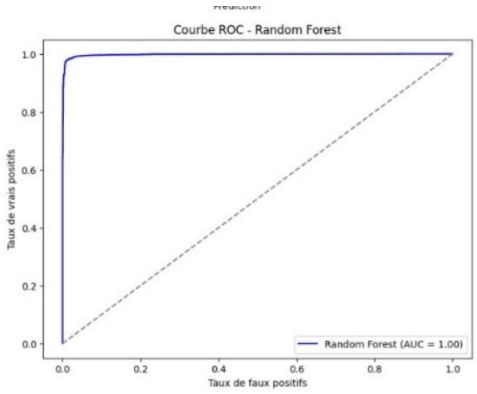

Figure 46 : Courbe ROC - RF.

3.2.4 Modèle de Stacking :

L'objectif de ce modèle est de construire un modèle capable de détecter si une URL est un site de phishing (spam) ou non (non-spam). Les données utilisées sont un jeu de données nettoyé contenant des caractéristiques numériques (features) et une colonne cible label (0 = non-spam, 1 = phishing).

Préparation des données :

Séparation train/test : Le jeu de données est divisé en deux parties : **80%** pour entraîner les modèles (train) et **20%** pour évaluer la performance (test). La séparation est stratifiée pour conserver la même proportion de phishing et non-phishing dans les deux ensembles.

Mise à l'échelle (**normalisation**) : On applique un **MinMaxScaler** pour ramener toutes les caractéristiques entre 0 et 1. Cela aide les modèles comme **SVM** à mieux fonctionner.

Modèles de base (base learners) :

Trois modèles sont entraînés indépendamment sur les données normalisées :

Modèle	Description
XGBoost	Algorithme performant d'arbres de décision, très utilisé en ML.
SVM	Modèle puissant pour classification, avec noyau radial pour gérer les données non linéaires.
Random Forest	Ensemble d'arbres de décision, robuste face au surapprentissage.

Tableau 2 : Description des modèles de base.

Construction du modèle de stacking :

Le stacking combine les prédictions des trois modèles de base pour construire un modèle méta :

- ✓ Les probabilités prédites par les trois modèles sur l'ensemble d'entraînement sont utilisées comme nouvelles caractéristiques.
- ✓ On entraîne une régression logistique (modèle simple et efficace) sur ces nouvelles caractéristiques pour apprendre à mieux combiner les prédictions.

Évaluation du modèle de stacking :

On applique les mêmes étapes sur l'ensemble de test :

- ✓ Prédiction des probabilités par chaque modèle de base ;
- ✓ Création des nouvelles caractéristiques pour le modèle méta ;
- ✓ Prédiction finale du stacking (0 ou 1).

Résultats sur le test :

- ✓ Accuracy (Précision) : 0.9960 (99.6%) — Très bon taux de bonnes classifications.
- ✓ ROC AUC : 0.9996 — Excellente capacité à distinguer phishing/non-phishing.

Analyse de la matrice de confusion :

La matrice affiche un nombre très élevé de TP et TN, et quasiment aucun FP ou FN, ce qui confirme la haute performance.

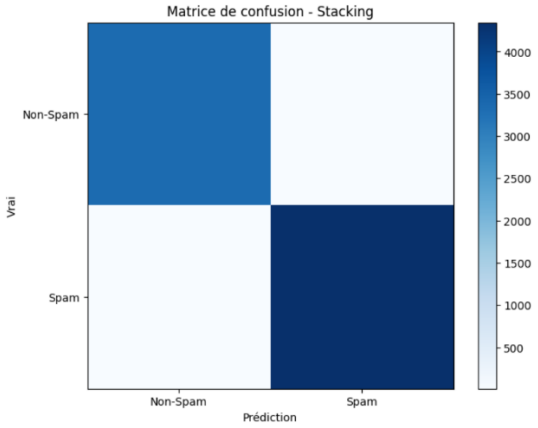

Figure 47 : Matrice de confusion – Stacking.

Description de la matrice :

	Prédit Non-Spam	Prédit Spam
Réel Non-Spam	Vrai négatif (TN)	Faux positif (FP)
Réel Spam	Faux négatif (FN)	Vrai positif (TP)

Tableau 3 : Description de la matrice.

Réel Non-Spam, Vrai négatif (TN), Faux positif (FP)

Réel Spam, Faux négatif (FN), Vrai positif (TP)

Courbe ROC :

La courbe ROC montre la performance du modèle à différents seuils de décision. Une AUC proche de 1 signifie que le modèle est très performant pour séparer les deux classes.

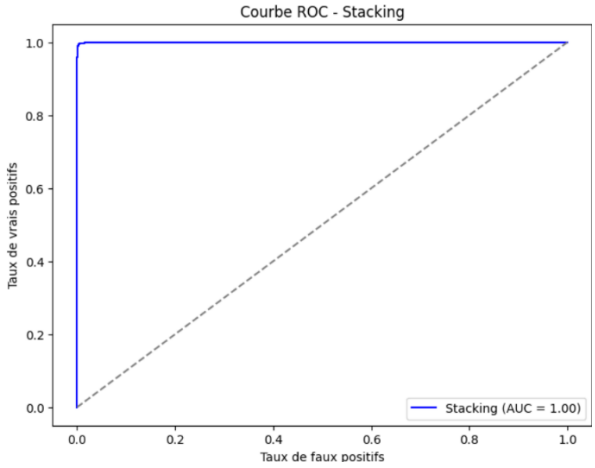

Figure 48 : Courbe ROC – Stacking.

```
Stacking - Accuracy: 0.9960
Stacking - ROC AUC: 0.9996
```

Figure 49 : Output du Stacking.

Points forts du modèle :

- ✓ **Combinaison intelligente** : Le stacking permet d'exploiter les forces complémentaires des différents modèles (XGBoost, SVM, Random Forest) ;
- ✓ **Performance élevée** : Avec une accuracy de 99.6% et un AUC proche de 1, le modèle est très fiable ;
- ✓ **Robustesse** : Utilisation de plusieurs modèles réduit le risque d'erreur d'un seul modèle.

Conclusion :

Ce modèle de **stacking** est une approche avancée et efficace pour détecter les sites de phishing, combinant plusieurs modèles classiques pour améliorer la performance globale. Avec des métriques impressionnantes (accuracy **99.6%,** AUC **0.9996**), il est prêt pour une utilisation dans un système réel, sous réserve de validation sur des données nouvelles.

Explication du code :

- ❖ **Chargement et préparation des données :**

- o Chargement du jeu d données nettoyé ;
- o X contient toutes les colonnes sauf label (les caractéristiques) ;
- o y est la colonne cible, indiquant phishing (1) ou non (0).

- ❖ **Séparation train/test :**

- o On divise les données en 80% entraînement, 20% test ;
- o stratify=y pour garder la même proportion de phishing/non-phishing dans les deux ensembles.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier


file_path = '/content/cleaned_phishing_data (1).csv'
data = pd.read_csv(file_path)


X = data.drop(columns=['label'])
y = data['label']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 50 : Chargement et préparation des données.

❖ **Création des modèles de base :**

- XGBoost : modèle d'arbres boostés très performant ;
- SVM (Support Vector Machine) avec noyau radial ;
- Random Forest : forêt d'arbres de décision ;
- Chaque modèle est initialisé avec certains hyperparamètres et entraîné sur les données normalisées.

❖ **Création des nouvelles caractéristiques pour le stacking (entraînement) :**

- On récupère les probabilités prédites par chaque modèle pour chaque échantillon du train ;
- Ces probabilités sont combinées en un nouveau jeu de données, qui sert d'entrée pour un modèle méta.

❖ **Entraînement du modèle méta :**

- Le modèle méta est une régression logistique simple qui apprend à combiner les prédictions des trois modèles de base.

```
xgb_model = xgb.XGBClassifier(objective='binary:logistic', use_label_encoder=False,
                              eval_metric='logloss', n_estimators=100, max_depth=6)


svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', probability=True)  # 'rbf' est le noyau radial


rf_model = RandomForestClassifier(n_estimators=100, max_depth=6, random_state=42)


xgb_model.fit(X_train_scaled, y_train)
svm_model.fit(X_train_scaled, y_train)
rf_model.fit(X_train_scaled, y_train)

# ---------------------------- Création de l'ensemble de features pour le stacking ----------


y_pred_xgb_train = xgb_model.predict_proba(X_train_scaled)[:, 1]
y_pred_svm_train = svm_model.predict_proba(X_train_scaled)[:, 1]
y_pred_rf_train = rf_model.predict_proba(X_train_scaled)[:, 1]

X_train_stack = np.column_stack((y_pred_xgb_train, y_pred_svm_train, y_pred_rf_train))

meta_model = LogisticRegression()
meta_model.fit(X_train_stack, y_train)
```

Figure 51 : Création des modèles de base.

❖ Prédiction sur le jeu test (phase d'évaluation) :

- On récupère les probabilités prédites par chaque modèle de base sur le test ;
- On construit le jeu de données d'entrée pour le modèle méta ;
- On fait la prédiction finale avec le modèle méta.

```
y_pred_xgb_test = xgb_model.predict_proba(X_test_scaled)[:, 1]
y_pred_svm_test = svm_model.predict_proba(X_test_scaled)[:, 1]
y_pred_rf_test = rf_model.predict_proba(X_test_scaled)[:, 1]

X_test_stack = np.column_stack((y_pred_xgb_test, y_pred_svm_test, y_pred_rf_test))


y_pred_stack = meta_model.predict(X_test_stack)
y_pred_proba_stack = meta_model.predict_proba(X_test_stack)[:, 1]


accuracy_stack = accuracy_score(y_test, y_pred_stack)
roc_auc_stack = roc_auc_score(y_test, y_pred_proba_stack)

print(f"Stacking - Accuracy: {accuracy_stack:.4f}")
print(f"Stacking - ROC AUC: {roc_auc_stack:.4f}")
```

Figure 52 : Phase d'évaluation.

❖ Évaluation des performances :

- Accuracy : taux de bonnes classifications ;
- ROC AUC : mesure la capacité du modèle à séparer les classes, plus proche de 1 c'est mieux.

❖ **Visualisation des résultats :**

- Matrice de confusion : montre les vrais positifs, faux positifs, vrais négatifs, faux négatifs ;
- Courbe ROC : visualise le compromis entre taux de vrais positifs et taux de faux positifs ;
- Le code trace ces graphiques pour analyser visuellement la performance.

```
cm_stack = confusion_matrix(y_test, y_pred_stack)
fpr_stack, tpr_stack, _ = roc_curve(y_test, y_pred_proba_stack)


plt.figure(figsize=(8, 6))
plt.imshow(cm_stack, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Matrice de confusion - Stacking")
plt.colorbar()
plt.ylabel('Vrai')
plt.xlabel('Prédiction')
plt.xticks([0, 1], ['Non-Spam', 'Spam'])
plt.yticks([0, 1], ['Non-Spam', 'Spam'])
plt.show()


plt.figure(figsize=(8, 6))
plt.plot(fpr_stack, tpr_stack, color='blue', label=f'Stacking (AUC = {roc_auc_stack:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('Taux de faux positifs')
plt.ylabel('Taux de vrais positifs')
plt.title('Courbe ROC - Stacking')
plt.legend(loc="lower right")
plt.show()
```

Figure 53 : Visualisation des résultats.

3.2.5 Comparaison entre les modèles :

Le tableau ci-dessous synthétise les performances des trois modèles de classification utilisés. Ces modèles ont été combinés dans une approche d'ensemble appelée **Stacking**, visant à optimiser les performances globales du système. Les résultats obtenus sont particulièrement prometteurs, avec une précision de **99,6 %** et un score **AUC de 0,9996**, ce qui démontre une excellente capacité de généralisation. **Ces performances suggèrent que le modèle empilé est suffisamment robuste et fiable pour une intégration dans un système opérationnel réel.**

Modèle	Accuacy	ROC AUC
XGBoost	0.995731	0.999794
SVM	0.993791	0.998523
Random Forest	0.974130	0.997868
Stacking	0.9960	0.9996

Tableau 4 : Récapitulation des performances des modèles utilisés.

Conclusion :

En somme, les modèles utilisés ont été appliqués sur un dataset préalablement nettoyé et normalisé, avec une évaluation basée sur des métriques fiables, à savoir l'accuracy et l'AUC ROC. Et pour améliorer encore les performances, un modèle de **Stacking** a été mis en œuvre, combinant les prédictions des trois classifieurs de base via une régression logistique. Cette approche a permis d'exploiter les forces de chaque modèle pour offrir une classification plus précise et fiable. Les résultats montrent que l'ensemble des modèles atteint des performances élevées, démontrant ainsi l'efficacité de cette stratégie dans la détection automatisée des e-mails de phishing.

CHAPITRE 3 : RÉSULTATS & DÉMONSTRATION

CHAPITRE 3 : RÉSULTATS & DÉMONSTRATION

Introduction :

Ce chapitre présente à la fois les résultats obtenus via notre système de détection de phishing, ainsi que la démonstration complète de l'interface utilisateur développée en Python avec Streamlit. L'interface permet d'analyser des e-mails Gmail en temps réel grâce à un modèle de machine learning entraîné.

1- Présentation générale de l'interface :

L'application a été développée avec **Streamlit**, un outil simple et efficace pour construire des interfaces web interactives en Python. Elle intègre plusieurs volets fonctionnels :

- Authentification sécurisée à Gmail via OAuth 2.0 ;
- Récupération des 10 derniers e-mails ;
- Traitement du contenu (expéditeur, sujet, corps) ;
- Extraction de caractéristiques et prédiction via un modèle ML ;
- Affichage clair et visuel des résultats de détection.

2- Structure et explication du code de l'interface :

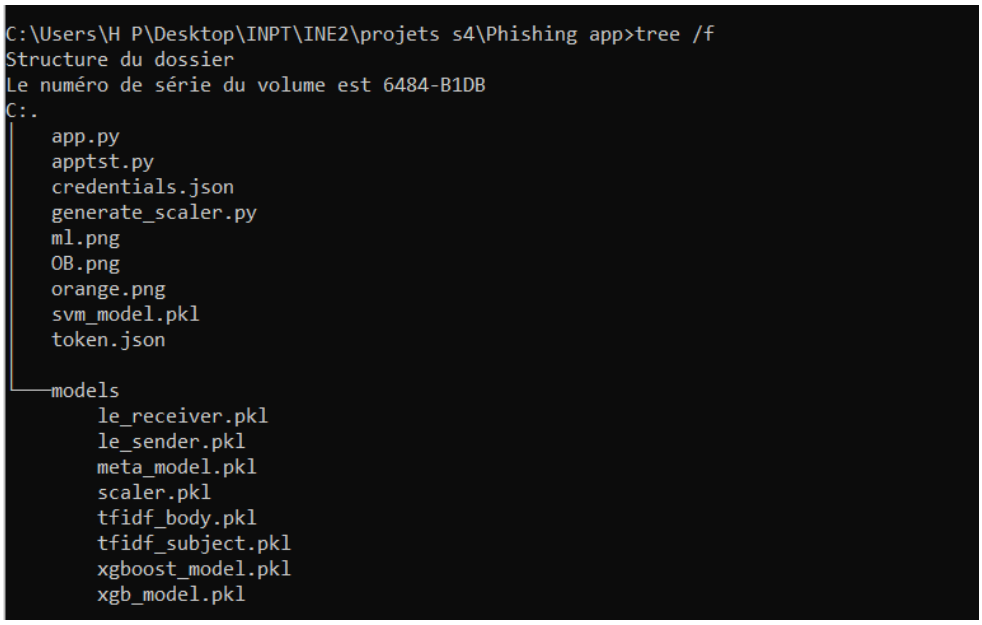

Figure 54 : L'arborescence de l'interface.

La structure du code se divise en plusieurs volets. Voici une explication claire de chaque composant :

2.1 Importation des bibliothèques et chargement des modèles :

Importation des bibliothèques nécessaires pour l'interface, le traitement des données, la connexion Gmail, le NLP, et le machine learning.

```python
import streamlit as st
import base64
import os
import re
import pandas as pd
import joblib
import html
from bs4 import BeautifulSoup
from googleapiclient.discovery import build
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
```

Figure 55 : Importation des bibliothèques de l'interface.

Chargement des objets pré-entraînés (encodeurs, TF-IDF, modèle principal) :

```python
le_sender = joblib.load('models/le_sender.pkl')
le_receiver = joblib.load('models/le_receiver.pkl')
model = joblib.load('models/meta_model.pkl')
scaler = joblib.load("numeric_scaler.pkl")
tfidf_body = joblib.load('models/tfidf_body.pkl')
tfidf_subject = joblib.load('models/tfidf_subject.pkl')
```

Figure 56 : Chargement des objets pré-entraînés.

2.2 Authentification Gmail via OAuth 2.0 :

```python
def authenticate_gmail():
    creds = None
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file('credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token.json', 'w') as token:
            token.write(creds.to_json())
    return build('gmail', 'v1', credentials=creds)
```

Figure 57 : Sécurisation de l'accès à l'API Gmail.

Cette fonction sécurise l'accès à l'API Gmail, via le protocole OAuth. Elle permet d'accéder au contenu des e-mails récents de l'utilisateur de manière conforme et sécurisée.

2.3 Extraction du contenu d'e-mail :

```python
def get_email_body(message_payload):
    parts = message_payload.get('parts', [])
    body = ""
    for part in parts:
        mime_type = part.get('mimeType')
        data = part.get('body', {}).get('data')
        if mime_type == 'text/plain' and data:
            body = base64.urlsafe_b64decode(data).decode('utf-8')
        elif mime_type == 'text/html' and data and not body:
            html_content = base64.urlsafe_b64decode(data).decode('utf-8')
            soup = BeautifulSoup(html_content, 'html.parser')
            body = soup.get_text()
    return body.strip()
```

Figure 58 : Extraction du contenu d'e-mail.

Elle permet d'extraire soit le texte brut, soit le texte depuis le HTML (via BeautifulSoup), afin d'en récupérer un contenu exploitable pour la détection.

2.4 Extraction des URLs :

```python
def extract_urls(text):
    return re.findall(r'https?://\S+', text)
```

Figure 59 : Extraction des URLs.

Cette fonction identifie les liens présents dans le corps de l'e-mail. Ces liens peuvent être indicateurs de phishing.

2.5 Extraction des caractéristiques pour le modèle :

```python
def process_email(subject, body, sender, receiver_domain="example.com"):
    sender_domain = sender.split('@')[-1].strip('> ') if '@' in sender else "unknown"
    try:
        sender_domain_encoded = le_sender.transform([sender_domain])[0]
    except ValueError:
        sender_domain_encoded = -1
    try:
        receiver_domain_encoded = le_receiver.transform([receiver_domain])[0]
    except ValueError:
        receiver_domain_encoded = -1

    features = {
        'sender_domain_encoded': sender_domain_encoded,
        'receiver_domain_encoded': receiver_domain_encoded,
        'phishing_words': sum(1 for word in ['urgent', 'gratuit', 'prix', 'compte', 'confidentiel']
                              if word in (subject + ' ' + body).lower()),
    }

    return pd.DataFrame([features])
```

Figure 60 : Extraction des features.

Cette fonction transforme les e-mails en vecteurs de caractéristiques exploitables par le modèle. Elle encode notamment :

- Le domaine de l'expéditeur ;
- Le domaine du destinataire ;
- Un comptage de mots suspects.

2.6 Lancement du scan Gmail et affichage dynamique :

```python
if st.button("Scanner mes 10 derniers emails"):
    with st.spinner("Authentification Gmail..."):
        service = authenticate_gmail()
        results = service.users().messages().list(userId='me', maxResults=10).execute()
        messages = results.get('messages', [])

    st.success(f"{len(messages)} emails récupérés !")

    for i, message in enumerate(messages, 1):
        msg = service.users().messages().get(userId='me', id=message['id'], format='full').execute()
        headers = msg['payload']['headers']
        subject = sender = "N/A"
        for header in headers:
            if header['name'] == 'Subject':
                subject = header['value']
            elif header['name'] == 'From':
                sender = header['value']

        body = get_email_body(msg['payload'])
        features = process_email(subject, body, sender)
        prediction = model.predict(features)[0]
        proba = model.predict_proba(features)[0][1]
        percent = int(proba * 100)

        # Protection HTML
        safe_sender = html.escape(sender)
        safe_subject = html.escape(subject)
```

```
    # SVG cercle de probabilité
    circle_svg = f"""
    <div class="circle-graph">
        <svg width="90" height="90">
            <g transform="rotate(-90, 45, 45)">
                <circle cx="45" cy="45" r="40" stroke="#eee" stroke-width="10" fill="none"/>
                <circle cx="45" cy="45" r="40" stroke="#dc3545" stroke-width="10"
                    stroke-dasharray="{percent * 2.51} 251" fill="none" />
                <text x="45" y="50" transform="rotate(90, 45, 45)">{percent}%</text>
            </g>
        </svg>
    </div>

    """


    st.markdown(f"<hr style='border: 3px solid #F37117;'>", unsafe_allow_html=True)
    st.markdown(f"<h3 style='color: black;'>Email {i}</h3>", unsafe_allow_html=True)

    st.markdown(f"""
        <div class="email-card">
            <div class="email-info">
                <div><strong>De :</strong> {safe_sender}</div>
                <div><strong>Sujet :</strong> {safe_subject}</div>
            </div>
        </div>
    """, unsafe_allow_html=True)

        label_html = (
            "<div class='phishing-label phishing'>PHISHING</div>" if prediction
            else "<div class='phishing-label safe'>SAIN</div>"
        )

        st.markdown(circle_svg, unsafe_allow_html=True)
        st.markdown(label_html, unsafe_allow_html=True)

        with st.expander("Aperçu du contenu de l'email"):
            st.write(body[:700] + '...')

st.markdown("</div>", unsafe_allow_html=True)
```

Figure 61 : Cœur de l'application.

C'est le cœur de l'application. Lors du clic, les étapes suivantes sont effectuées :

1. Authentification Gmail ;
2. Récupération des e-mails ;
3. Analyse du contenu ;
4. Prédiction avec le modèle ML ;
5. Affichage des résultats avec :
    - ❖ Informations (expéditeur, sujet) ;
    - ❖ Cercle SVG représentant le score de phishing ;
    - ❖ Label clair : SAIN ou PHISHING ;
    - ❖ Aperçu du contenu (dans un st.expander).

3- Démonstration :

L'interface est la suivante :

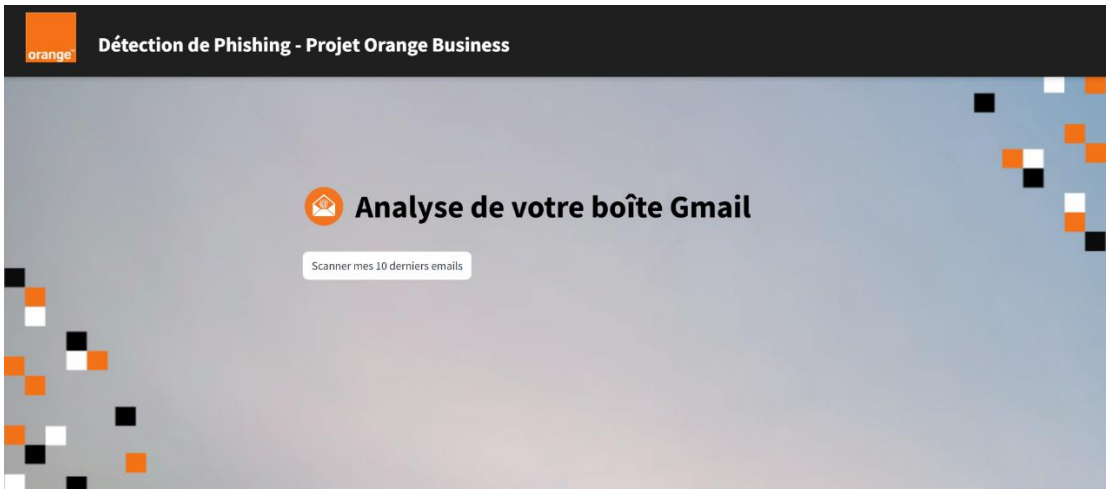

Figure 62 : Interface de l'application.

Après avoir cliqué sur « **Scanner mes 10 derniers emails** », l'utilisateur est redirigé vers une page d'authentification Gmail.

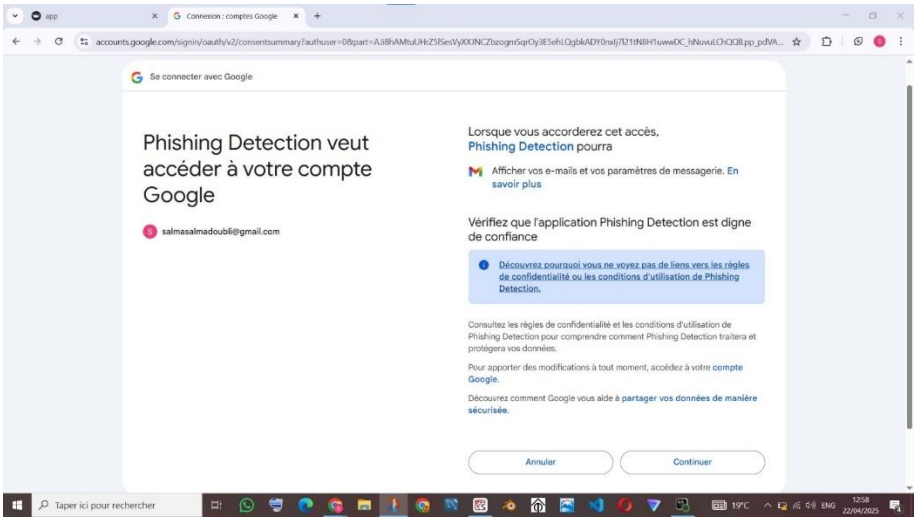

Figure 63 : Page d'authentification Gmail.

Ensuite, l'interface analyse et affiche chaque e-mail avec une prédiction SAIN ou PHISHING.

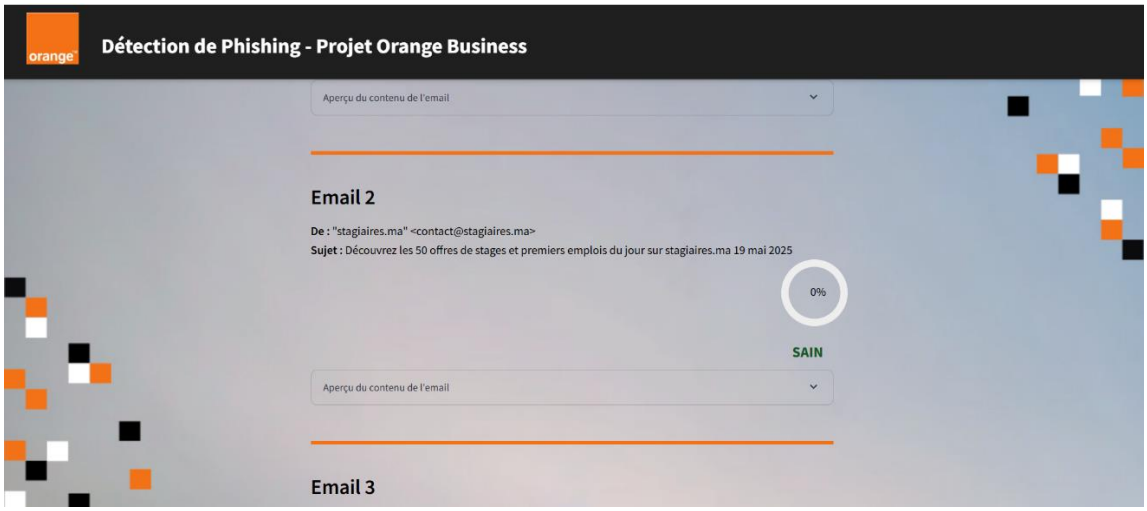

Figure 64 : Analyse des e-mails.

CONCLUSION GÉNÉRALE

CONCLUSION GÉNÉRALE

Le phishing représente aujourd'hui une menace critique, en constante évolution, qui tire parti de la massification des échanges électroniques et de la vulnérabilité des utilisateurs face à des messages frauduleux de plus en plus crédibles. Dans un contexte où plus de 370 milliards d'e-mails circulent chaque jour à l'échelle mondiale, il devient impératif de mettre en place des solutions efficaces, accessibles et intelligentes pour garantir la sécurité des communications numériques.

À travers ce projet, nous avons conçu une solution complète et innovante de détection de phishing, reposant sur l'intelligence artificielle (IA) et le traitement automatique du langage naturel (NLP). Cette solution se distingue par sa capacité à analyser, classifier et expliquer la nature d'un e-mail grâce à un système automatisé combinant des modèles de machine learning (SVM, Random Forest, XGBoost) avec un modèle de Stacking pour en optimiser la performance. Le tout est intégré dans une interface utilisateur intuitive, développée avec Streamlit, qui offre une interaction simple et pédagogique avec l'utilisateur.

Nous avons également assuré l'interfaçage avec l'API Gmail pour permettre une analyse en temps réel des e-mails reçus, renforçant ainsi l'aspect pratique et concret du système. Les résultats obtenus, avec des scores de précision avoisinant les 99,6 % pour le modèle de Stacking, démontrent la robustesse et l'efficacité de notre approche.

Enfin, ce projet ouvre la voie à plusieurs perspectives d'évolution, notamment l'analyse automatique des pièces jointes, l'intégration de résumés vocaux, l'adaptation multilingue ou encore l'exploitation de modèles LLM pour une détection contextuelle plus fine.

Lien Git Hub : https://github.com/senhajiboutayna/Detection-of-Phishing-Attacks-with-AI-and-NLP

WEBOGRAPHIE

[1] https://www.kaggle.com/code/sudalairajkumar/getting-started-with-text-preprocessing

[2] https://medium.com/@awaldeep/understanding-the-essentials-nlp-text-preprocessing-

[3] https://www.researchgate.net/publication/387159265_A_Feature_Engineering_Approach

[4] https://developers.google.com/workspace/gmail/api/guides?hl=fr

[5] https://developers.google.com/workspace/gmail/api/reference/rest?hl=fr

[6] https://docs.streamlit.io/get-started/tutorials/create-an-app

[7] https://365datascience.com/tutorials/machine-learning-tutorials/how-to-deploy-machine-learning-models-with-python-and-streamlit/