Rapport du projet informatique 2A

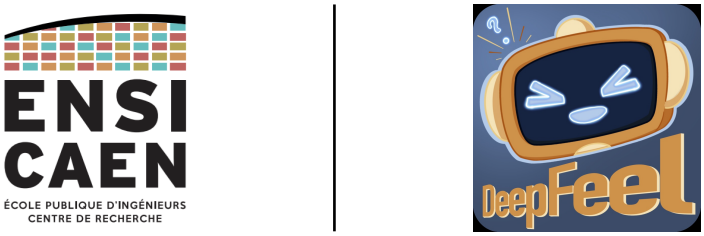
Projet informatique 2A

Gabin Vrillault, Anatole Conrad, Julien Excoffier, Ninel Dogaru[1]

[1]Ensicaen

23 mars 2025

Rapport du projet d'informatique 2A



Identification assistée par l'IA des émotions sur les réseaux sociaux

Tanguy Gernot tanguy.gernot@unicaen.fr

Emmanuel Giguet emmanuel.giguet@unicaen.fr

Date de dernière mise à jour : 23 mars 2025

Table des matières

Introduction **1**
Présentation du projet . . . 1
Contexte du projet . . . 1
Déroulement du projet . . . 1

Test de modèles IA hébergés localement **2**
Présentation des modèles testés . . . 2
Interprétation des résultats . . . 3

Fine tuning de modèles BERT **4**
Présentation de l'architecture BERT . . . 4
Pré-entraînement et fine-tuning . . . 5
Poursuite du pre-training de BERT . . . 6
Fine-tuning de BERT . . . 6
Modèle BERT déjà pré-entraîné . . . 6
Performances des modèles BERT fine-tuned . . . 7

Création du modèle customisé DeepFeel **8**
Architecture du modèle . . . 8
Qu'est-ce qu'un RNN ? . . . 8
Pourquoi une LSTM ? . . . 8
Implémentation avec PyTorch . . . 8
Données d'entraînement . . . 9
Vectorisation des mots et création du vocabulaire . . . 9
Processus d'entraînement . . . 9
Évaluation et optimisation . . . 9
Améliorations . . . 10

Interactions humain-modèle **11**
Proxy pour filtrer les posts sur X . . . 11
Exemple de masquage d'un tweet . . . 14
Difficultés rencontrées . . . 14

Conclusion **15**

Références bibliographiques **16**

Table des figures

1 Architecture d'un transformer 4
2 Architecture de BERT 5
3 Architecture du modèle DeepFeel 11
4 Schéma de fonctionnement du proxy pour filtrer les posts sur X 12
5 Avantages et inconvénients du concept Man in the Middle pour filtrer les posts sur X 13
6 Proportion de contenu censuré 13
7 Statistiques des émotions détectées 13
8 Proportion de contenu censuré 14
9 Tweet original 14

Liste des tableaux

1 Modèles testés avec Ollama 2
2 Performances comparatives des différents modèles BERT fine-tuned sur la tâche de classification des émotions 7

Introduction

Présentation du projet

Dans le cadre du projet 2A Informatique, nous avons travaillé sur la conception d'un logiciel capable d'identifier les émotions à partir de l'analyse de textes, qu'il s'agisse de phrases courtes ou de contenus issus des réseaux sociaux.

Ce projet a été réalisé sous la supervision de M. Tanguy Gernot et M. Emmanuel Giguet, chercheurs au GREYC.

Contexte du projet

L'identification des émotions dans les textes est un domaine en pleine expansion, notamment avec l'essor des réseaux sociaux et des plateformes de communication en ligne. Ces outils génèrent une quantité massive de données textuelles, dont l'analyse peut fournir des informations précieuses pour des applications variées, telles que la modération de contenu, l'analyse des sentiments dans le marketing, ou encore la détection de comportements à risque. Ce projet s'inscrit dans ce contexte en proposant des solutions innovantes pour analyser et comprendre les émotions exprimées dans les textes.

Déroulement du projet

Pour commencer, nous avons développé un modèle LSTM simple pour identifier les émotions dans des phrases en anglais, en utilisant un dataset annoté, le développement de ce réseau est expliqué dans la partie située à la page 8. Une interface graphique, a également été conçue pour afficher les résultats de ce modèle. De plus, nous avons interrogé un modèle intelligent Mistral via une API pour identifier les émotions dans des phrases en anglais, puis en français, et comparé ses résultats avec ceux de notre modèle. Nous avons ensuite étendu ce procèdé à d'autres modèles intelligents, DeepSeek et Gemma2, pour comparer leurs performances avec celles de notre modèle, voir la page 2 pour plus de détails.

En parallèle, nous avons effectué un état de l'art sur les méthodes d'analyse de textes. Nous avons ainsi opté pour l'utilisation de Hugging Face et des modèles types BERT. Hugging Face offre une bibliothèque open-source robuste qui permet d'intégrer facilement des modèles de traitement du langage naturel avancés. BERT, quant à lui, est un modèle transformer bidirectionnel qui excelle dans la compréhension du contexte des mots. Dans notre cas, ses performances étaient comparables à celles de nos architectures comme LSTM ou bi-LSTM, voir la page 4 pour plus de détails.

En complément, des objectifs supplémentaires ont été définis pour enrichir le projet. Nous avons développé un modèle bi-LSTM pour détecter la langue d'une phrase ou d'un texte, et amélioré notre modèle LSTM en bi-LSTM pour identifier les émotions dans plusieurs langues. Des tests de performance ont été réalisés pour comparer notre modèle avec d'autres agents intelligents tels que Mistral, DeepSeek et Gemma2, nous en parlons plus à partir de la page 2. Enfin, nous avons conçu un proxy pour filtrer les requêtes sur des plateformes comme Twitter ou Threads, et développé une application permettant de visualiser des statistiques sur les posts analysés, voir la page 11.

Test de modèles IA hébergés localement

Présentation des modèles testés

Pour pouvoir comparer les performances de différents modèles d'IA, nous avons utilisé une même plateforme avec le même prompt système.
Pour cela nous avons utilisé l'API locale d'*Ollama* [1].

Ollama est une plateforme open-source destinée à faciliter l'utilisation des LLMs (Large Language Models) en local. En effet, *Ollama* permet de télécharger et d'exécuter des modèles d'IA sur un ordinateur personnel à partir d'une simple ligne de commande.
Par exemple, `ollama run deepseek-r1:1.5b` permet de télécharger et d'exécuter le modèle de DeepSeek et de l'utiliser comme agent conversationnel.

En plus de cela, *Ollama* utilise un système d'API REST en local pour permettre d'interagir avec les modèles d'IA de manière programmable. Nous avons donc utilisé cette API pour tester les différents modèles d'IA sur la tâche de classification des émotions.

Table 1: Modèles testés avec Ollama

Nom	Taille	Paramètres	Architecture	Précision	Temps	VRAM
DeepSeek-R1	1.1Gb	1,5B	*qwen2*	32%	0.49s	47%
DeepSeek-R1	4.7Gb	7.62B	*qwen2*	45%	1.04s	85%
Mistral	4.1Gb	7,25B	*llama*	55.55%	0.70s	90%
Gemma 2	5.4Gb	9.24B	*gemma2*	56%	2.6s	30%

Ces modèles ont tous été testés sur une machine avec les caractéristiques suivantes :

- Processeur : Intel Core i7-10700K (8 coeurs, 16 threads)
- Mémoire vive : 16 Go
- Système d'exploitation : Ubuntu 24.04
- GPU : NVIDIA RTX 3060ti (8Gb de mémoire)

Le prompt système utilisé pour les tests est le suivant :

```
You are given one or more sentences extracted from posts on social media.
Based on the content of these sentences, return only a single JSON object
with the following keys:
anger, fear, happy, love, sadness, surprise

Each of the six keys (emotions) should map to a probability value
between 0 and 1 (inclusive),
indicating how likely the input text expresses that emotion.

Your response must:
- Contain only valid JSON (no extra text, no Markdown formatting, no explanations).
- Follow the exact structure and key names shown above.
- Ensure all values are floating-point numbers between 0 and 1 (inclusive).
```

De plus, l'API d'*Ollama* permet de spécifier le format de la réponse:

```python
def get_json_request_from(prompt, model_name='deepseek-r1:7b'):
return {
    "model": model_name,
    "prompt": prompt,
    "system": SYSTEM_PROMPT,
    "stream": False,
    "format": {
        "type": "object",
        "properties": {
            "anger": {"type": "number"},
            "fear": {"type": "number"},
            "happy": {"type": "number"},
            "love": {"type": "number"},
            "sadness": {"type": "number"},
            "surprise": {"type": "number"}
        },
        "required": ["anger", "fear", "happy", "love", "sadness", "surprise"]
    }
}
```

Interprétation des résultats

La validation des modèles a été effectuée sur le même ensemble de 2000 phrases en anglais et montre des résultats tous inférieurs à 60% de précision. Globalement, tous les modèles testés ont montré une consommation élevée de la mémoire du GPU mais également des ressources du CPU pour les modèles les plus lourds.

Les résultats obtenus ne sont pas satisfaisants, les modèles ne sont pas capables de détecter correctement les émotions exprimées dans les textes. C'est pourquoi nous n'avons pas poursuivi le projet avec ces modèles. Les résultats sont tous consultables sur la page Gitlab de notre projet [2].

Fine tuning de modèles BERT

Présentation de l'architecture BERT

BERT (Bidirectional Encoder Representations from Transformers) est un modèle de langage développé à l'origine par Google AI en 2018. Il est conçu pour comprendre le contexte des mots dans une phrase en tenant compte de l'ensemble de la phrase, plutôt que de se concentrer uniquement sur les mots adjacents. Il peut être utilisé pour diverses tâches de traitement du langage naturel (NLP) comme la classification des émotions dans un texte, d'où notre intérêt pour ce modèle. Il est basé sur les transformers, en utilisant uniquement l'encodeur de l'architecture transformer. Ce

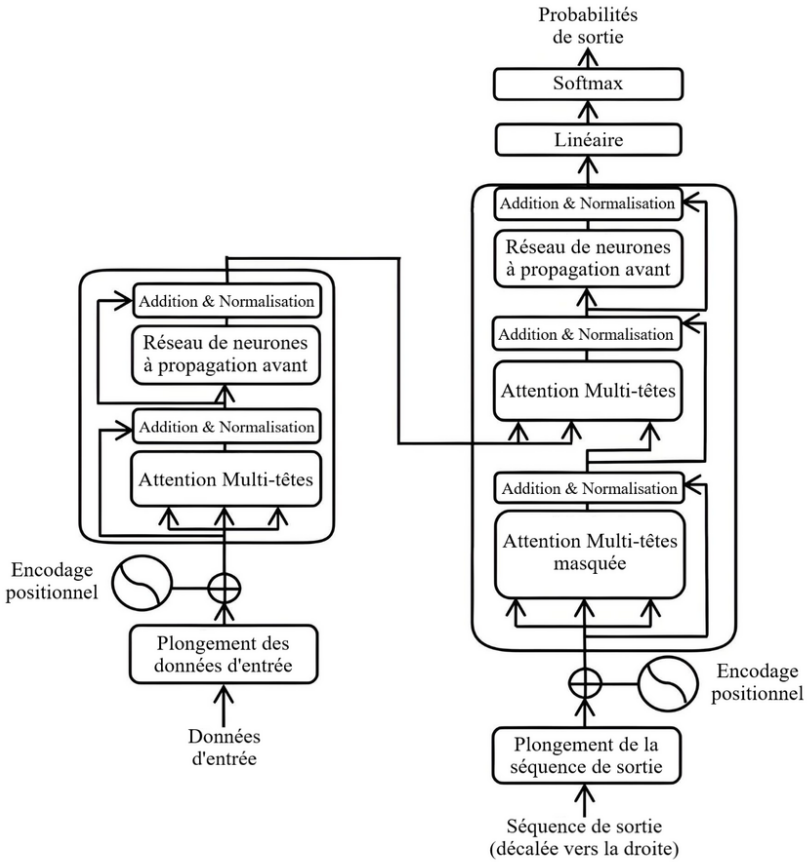


Figure 1: Architecture d'un transformer

modèle de langage consiste en quatre modules:

- **Tokenization**: Le texte d'entrée est découpé en tokens (des entiers).
- **Embedding**: Les tokens sont projetés dans un espace vectoriel de dimension fixe (Tableau de vecteurs à valeurs réelles).
- **Encoder**: Un empilement de couches de transformers.
- **Task head**: Une couche de sortie qui peut être vue comme un simple decodeur, il fait l'inverse de l'embedding.

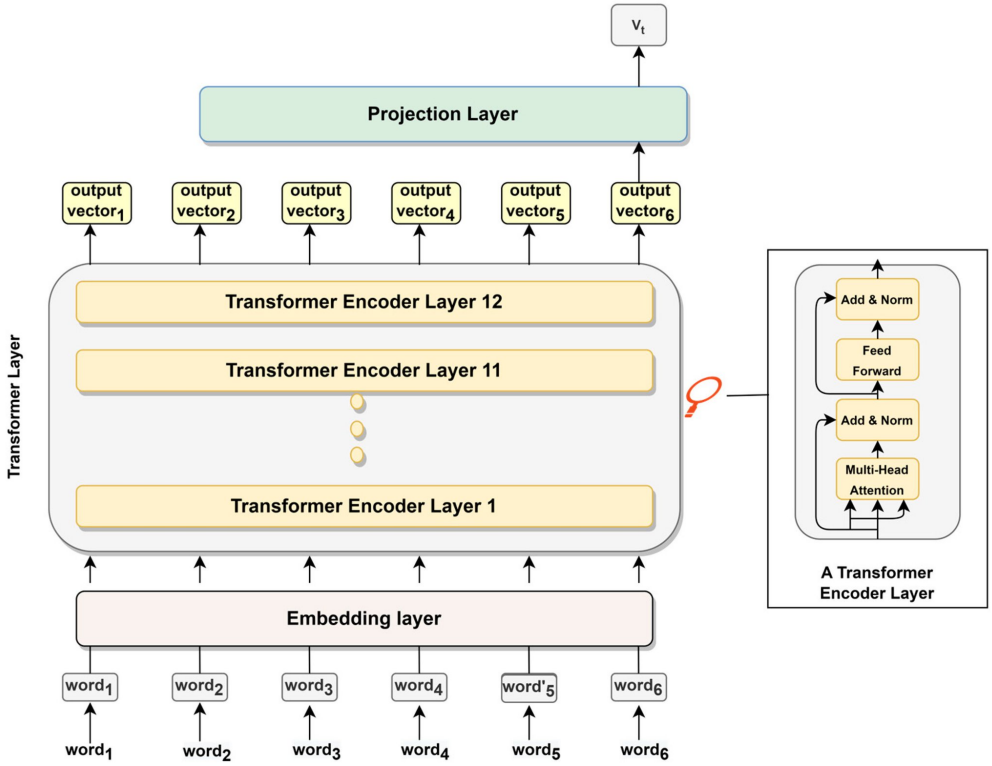


Figure 2: Architecture de BERT

Pré-entraînement et fine-tuning

Les modèles BERT ont deux phases d'entraînement:

- **Pré-entraînement**: Le modèle est pré-entraîné sur un grand corpus de texte non étiqueté. Il apprend à prédire des mots masqués dans une phrase (Masked Language Model) et à prédire la prochaine phrase (Next Sentence Prediction).
- **Fine-tuning**: Le modèle pré-entraîné est ensuite ajusté sur une tâche spécifique avec un ensemble de données étiquetées. Cela permet au modèle d'apprendre des représentations spécifiques à la tâche tout en conservant les connaissances acquises lors du pré-entraînement.

Pour avoir un modèle performant, il est important de bien choisir les données d'entraînement. En effet, le modèle original de Google a été pré-entraîné sur un corpus de texte général, ce qui peut ne pas être optimal pour des tâches spécifiques comme la classification des émotions sur une application comme twitter où le langage est très différent de celui utilisé dans les livres ou les articles scientifiques.

Poursuite du pre-training de BERT

Dans une première approche, nous avons essayé de poursuivre le pré-entraînement d'un modèle BERT [3] sur un corpus de 1.6 million de tweets [4]. Nous téléchargeons le modèle BERT avec des poids pré-entraînés sur le corpus de Google. Pour cela nous avons utilisé la librairie python `transformers` de Hugging Face [5]. Poursuivre le pré-entraînement de BERT consiste à:

- Charger le modèle BERT avec des poids pré-entraînés.
- Charger le tokenizer de BERT.
- Réaliser la tokenization du corpus de tweets.
- Choisir les hyperparamètres d'entraînement
- Utiliser `Data Collator` pour créer des batches de donnée (masquage aléatoire et padding).
- Lancer l'entraînement.
- Sauvegarder le modèle.

Voici les hyperparamètres que nous avons utilisés pour la poursuite du pré-entraînement:

- **Taux de masquage**: 15% des tokens sont masqués.
- **Nombre de epochs**: 3
- **Taille de batch**: 32
- **Taille maximale des séquences**: 128

Ce pré-entraînement est très coûteux en temps et en ressources, il a fallu 7 heures pour poursuivre le pré-entraînement de BERT sur un GPU NVIDIA RTX 3060ti.

Une fois le modèle pré-entraîné, pour pouvoir l'utiliser pour la classification des émotions, il faut le fine-tuner sur un ensemble de données étiquetées.

Fine-tuning de BERT

Le fine-tuning de BERT consiste à réaliser un entraînement supervisé sur un ensemble de données étiquetées pour spécialiser le modèle sur une tâche spécifique. Nous avons utilisé un ensemble de données d'émotions en anglais avec les labels de nos six émotions pour réaliser le fine-tuning. Ce processus est également réalisé avec la librairie `transformers` de Hugging Face. Le temps de fine-tuning est beaucoup plus court que celui du pré-entraînement entre 30 minutes et 1 heure selon la taille de l'ensemble de données.

Modèle BERT déjà pré-entraîné

Nous avons également testé un modèle BERT déjà pré-entraîné sur un corpus de tweets [6], ce modèle à la particularité d'être d'avoir été entrainé sur un corpus de tweets dans 66 langues différentes. Il est donc particulièrement pertinent dans notre cas d'utilisation. De manière analogue au modèle BERT de Google, nous avons réalisé un fine-tuning de ce modèle sur notre ensemble de données d'émotions pour en faire un modèle de classification des émotions. (Ensemble de données en anglais uniquement pour le fine tuning).

Modèle	Précision	Temps moyen	Langue évaluée
BERT (Google) pré-entraîné sur 1.6M de tweets	93.00%	0.022s	Anglais
Bernice (multi-langue) pré-entraîné sur 2.5Md de tweets	93.03%	0.021s	Français
Bernice (multi-langue) pré-entraîné sur 2.5Md de tweets	92.90%	0.021s	Anglais

Table 2: Performances comparatives des différents modèles BERT fine-tuned sur la tâche de classification des émotions

Performances des modèles BERT fine-tuned

On constate que les deux modèles BERT fine-tunés ont des performances très proches et satisfaisantes.

Tous ces modèles ont été fine-tunés sur un ensemble de données d'émotions en anglais de 16000 phrases, et testés sur un ensemble de validation de 2000 phrases. Le modèle Bernice a donc eu un score correct sur un ensemble de données de validation de 2000 phrases en français alors qu'il a été fine-tuné sur un ensemble de données d'émotions en anglais, ce qui montre que ce modèle est capable de généraliser sur d'autres langues.

Création du modèle customisé DeepFeel

Architecture du modèle

Pour concevoir la première version du modèle DeepFeel, nous avons utilisé une architecture de réseau de neurones récurrents (RNN), et plus spécifiquement une variante appelée LSTM (Long Short-Term Memory). Ce choix repose sur leur efficacité à traiter des données séquentielles comme le texte, où l'ordre des mots est essentiel pour en comprendre le sens émotionnel.

Qu'est-ce qu'un RNN ?

Un Réseau de Neurones Récurrent (RNN) est un type de réseau de neurones spécialement conçu pour le traitement de séquences. Contrairement aux réseaux de neurones classiques (feedforward), qui considèrent chaque donnée de manière indépendante, les RNN sont capables de mémoriser les informations passées pour influencer la prédiction actuelle. Concrètement, lorsqu'un RNN lit une phrase mot par mot, il conserve à chaque étape une sorte de mémoire de ce qu'il a lu précédemment, appelée état caché (hidden state). Cela lui permet de comprendre le contexte global d'une phrase et d'en extraire le ton ou l'émotion. Cependant, les RNN classiques rencontrent des difficultés lorsqu'il s'agit de retenir des informations sur de longues séquences à cause du problème du vanishing gradient. C'est là qu'intervient le LSTM.

Pourquoi une LSTM ?

Le LSTM est une amélioration du RNN classique. Il introduit un mécanisme de mémoire plus sophistiqué, grâce à l'utilisation de cellules de mémoire et de portes (d'entrée, d'oubli et de sortie), qui régulent le flux d'information à chaque étape de la séquence. Ces mécanismes permettent au modèle de :

- Conserver les informations importantes plus longtemps,
- Oublier les informations inutiles ou obsolètes,
- Se concentrer sur ce qui est pertinent pour la tâche, ici la détection de l'émotion dans un texte.

Cela en fait un choix particulièrement pertinent pour la compréhension du langage naturel, où le sens d'un mot peut dépendre de ceux qui le précèdent ou le suivent.

Implémentation avec PyTorch

Nous avons utilisé PyTorch, une bibliothèque de deep learning flexible et puissante, pour implémenter DeepFeel. L'architecture du modèle comprend :

- Une couche d'embedding, qui transforme chaque mot en un vecteur dense représentant son sens,
- Une ou plusieurs couches LSTM, qui traitent la séquence de mots et en extraient le contexte émotionnel,
- Une couche de dropout pour réduire le sur-apprentissage,
- Une couche linéaire finale, suivie d'un softmax, qui prédit la probabilité que le texte exprime l'une des six émotions cibles.

Données d'entraînement

Nous avons entraîné le modèle sur un jeu de données de phrases annotés avec six émotions : joie, tristesse, peru, colère, surprise et amour. Ce dataset, récupéré sur Kaggle, est en anglais et offre une base solide pour entraîner un premier modèle de détection d'émotions. Les phrases ont été prétraités (nettoyage, tokenisation), puis organisés en batches pour l'entraînement. La fonction de perte utilisée est la cross-entropy, adaptée aux tâches de classification multiclasse.

Vectorisation des mots et création du vocabulaire

Avant de pouvoir entraîner un modèle de deep learning sur des textes, il est nécessaire de transformer les mots en une représentation numérique que le réseau peut comprendre. Cette étape est appelée vectorisation des mots.

La première étape consiste à créer un vocabulaire, c'est-à-dire une liste de tous les mots uniques présents dans notre jeu de données. Chaque mot se voit attribuer un indice unique.Cette conversion des mots en indices entiers permet ensuite de traiter les textes comme des séquences de nombres.

Une fois les mots encodés, nous utilisons une couche d'embedding pour transformer chaque indice en un vecteur dense de dimension fixe. Ces vecteurs permettent de capturer la signification sémantique des mots : deux mots ayant un sens similaire auront des vecteurs proches dans l'espace. Cette représentation continue et dense est bien plus informative que de simples entiers, car elle permet au modèle de mieux généraliser et de comprendre les relations entre les mots.

Nous avons également appliqué un padding sur les séquences pour qu'elles aient toutes la même longueur, ce qui est nécessaire pour un traitement efficace en batch avec des LSTM. Cette phase de vectorisation est donc essentielle pour faire le lien entre le texte brut et l'entrée du modèle.

Processus d'entraînement

Le modèle à donc été entrainé sur le dataset cité précédemment. Le processus d'entraînement comprend:

1. La tokenisation des textes
2. La création des embeddings/vocabulaires
3. L'entraînement du modèle
4. L'évaluation des performances

Évaluation et optimisation

L'évaluation du modèle a été réalisée à l'aide d'un jeu de données de test distinct de celui utilisé pour l'entraînement, afin de prévenir tout risque de surapprentissage et de mesurer la capacité réelle du modèle à généraliser.Le premier modèle a obtenu des résultats satisfaisants, atteignant une précision de 88.05%, sur le dataset de validation. Ce score montre une bonne capacité du modèle à capturer les émotions exprimées dans les textes, malgré la complexité du langage naturel.

L'entraînement du modèle a été effectué sur une machine dotée de la configuration suivante :

1. Processeur (CPU) : AMD Ryzen 9 7900X
2. Carte graphique (GPU) : Nvidia GeForce RTX 4070 Ti
3. Mémoire vive (RAM) : 48 Go
4. Système d'exploitation (OS) : Windows 11

Améliorations

Afin d'améliorer les performances du modèle DeepFeel, plusieurs optimisations architecturales et stratégiques ont été mises en œuvre au cours du développement. Dans un premier temps, nous avons remplacé le modèle LSTM unidirectionnel par un bi-LSTM (LSTM bidirectionnel). Cette architecture permet de prendre en compte à la fois le contexte précédent et suivant d'un mot dans une phrase, contrairement au LSTM classique qui ne traite les séquences que dans un seul sens. Cela renforce considérablement la compréhension du sens global, en particulier dans des textes ambigus ou complexes. Pour accompagner ce changement, le modèle a été agrandi en ajoutant une seconde couche LSTM ainsi qu'en augmentant le nombre de neurones dans chacune des couches récurrentes. Ces ajustements ont permis d'atteindre une précision de 89.40%, soit une amélioration notable par rapport au modèle initial. Par la suite, nous avons élargi le jeu de données d'entraînement, dans le but d'améliorer la capacité de généralisation du modèle. Malgré cela, la précision a légèrement fluctué, atteignant environ 89.15%, suggérant un début de saturation du modèle par rapport à sa complexité. Pour lutter contre ce phénomène, nous avons introduit une couche de dropout, afin de limiter le surapprentissage (overfitting) en forçant le réseau à ne pas trop dépendre de certains neurones pendant l'entraînement. Cette modification a eu un effet positif, portant la précision à 90.8%. Enfin, le dernier modèle DeepFeel a été entraîné sur un dataset étendu à 420 000 phrases, toutes annotées avec l'une des six émotions cibles. Cette augmentation massive du corpus a permis d'apporter une grande flexibilité au modèle, notamment face à des formulations variées ou à des données bruitées. Ce modèle final a atteint une précision de 94.25%, illustrant l'efficacité des différentes améliorations mises en œuvre au fil du projet.

Version multilingue

Dans le but d'élargir les capacités du modèle DeepFeel, une version multilingue a été développée, en commençant par l'intégration du français en complément de l'anglais. Pour cela, nous avons utilisé les mêmes datasets annotés émotionnellement que ceux utilisés pour le modèle anglais, mais traduits en français. Après l'entraînement, le modèle français a montré des performances équivalentes à celles du modèle en anglais, confirmant la robustesse de l'architecture bi-LSTM face à un changement de langue, à condition que la qualité de traduction soit maintenue.

Afin de permettre une analyse automatisée de textes en plusieurs langues, nous avons ensuite entraîné un modèle de détection de langue, capable d'identifier si un texte est en français ou en anglais. Ce modèle repose lui aussi sur une architecture bi-LSTM, et a été entraîné sur un jeu de données de 5 000 phrases équilibrées entre les deux langues. Grâce à cette étape de prédiction de langue en amont, DeepFeel est désormais capable de détecter automatiquement la langue d'un texte, puis d'appliquer le modèle d'analyse émotionnelle approprié. Cette approche rend le système plus flexible et ouvre la voie à une prise en charge de langues supplémentaires à l'avenir.

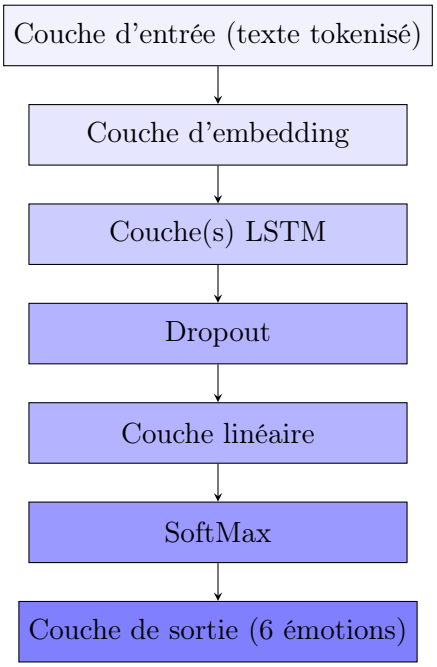


Figure 3: Architecture du modèle DeepFeel

Interactions humain-modèle

Proxy pour filtrer les posts sur X

Parmis les potentiels cas d'usage de notre projet, nous avons décidé de nous concentrer sur la modération en direct du contenu sur les réseaux sociaux pour protéger les enfants du contenu potentiellement inapproprié.
Nous avons décidé de développer une solution pour X (ex Twitter) qui permet de filtrer les posts en temps réel et de les censurer si nécessaire.
Pour ce faire, nous avons développé un proxy qui intercepte les réponses API entre l'application et le serveur de X. Après rétro-ingénierie du contenu des réponses, nous avons pu identifier le contenu des posts et le remplacer par des étoiles au besoin, cf. Figure 4. et Figure 8.

Le mode de fonctionnement de notre solution s'assimile donc à une attaque Man in the Middle de l'API de X, mais avec des intentions bienveillantes.

Ce choix nous permet de mettre en place une solution efficace mais qui s'expose à des blocages, notamment puisque nous devons déchiffrer les requettes HTTPS et qu'il y a donc une rupture de certificat.
Ces problèmes se sont particulièrement fait ressentir sur la version android du proxy. En effet, l'application vérifie l'authenticité du certificat (mécanisme de certificate pinning) et nous avons donc du corrompre l'APK pour intégrer notre propre certificat (voir apk-mitm), ce qui est inenvisageable pour une application en production, cf. Figure 5.

Pour une solution moins complexe pour l'utilisateur, nous avons développé une application Flutter intégrant Twitter en version web pour contourner les problèmes de certificats.
L'installation de nos certificats pour le proxy est ainsi automatique et quasi-transparente pour l'utilisateur

Nous en avons profité pour rajouter une collecte de statistiques qui permettent à l'utilisateur/aux parents de l'utilisateur de voir les émotions prédominantes et la proportion de contenu censuré, cf. Figure 6.

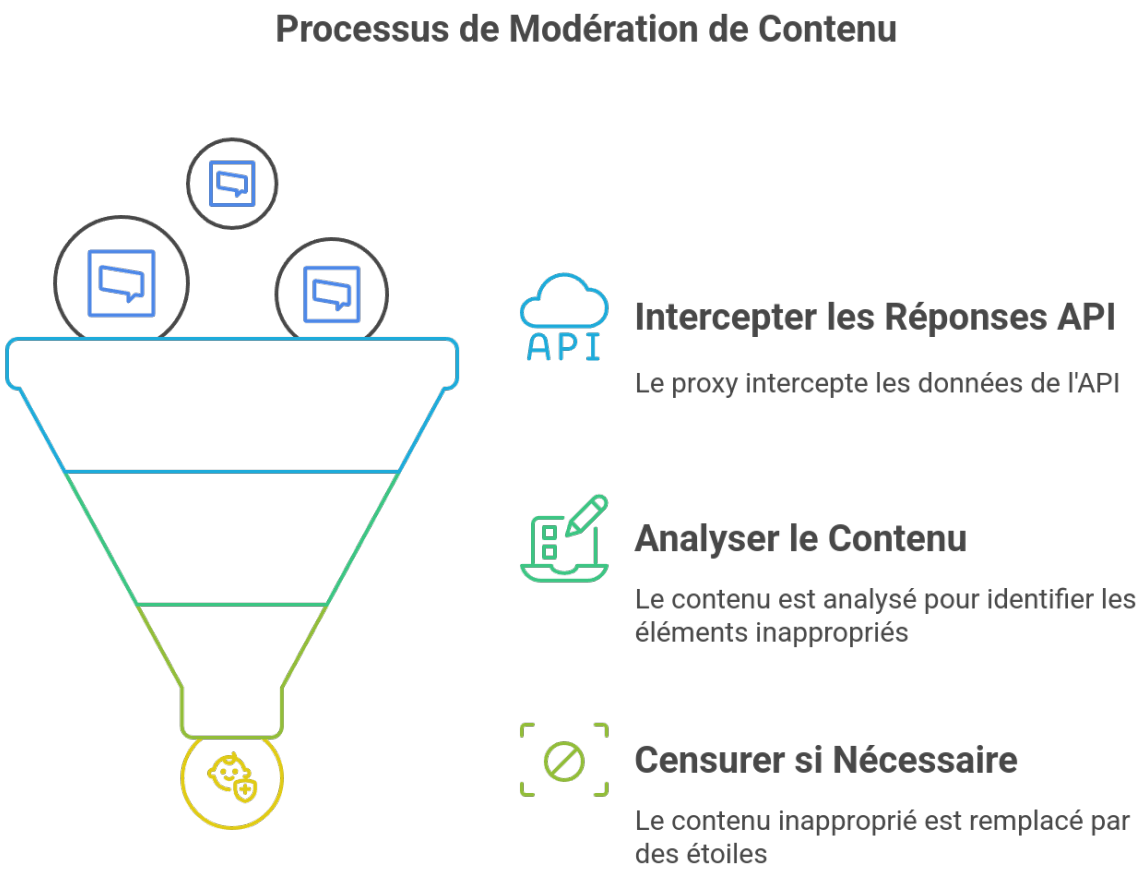


Figure 4: Schéma de fonctionnement du proxy pour filtrer les posts sur X

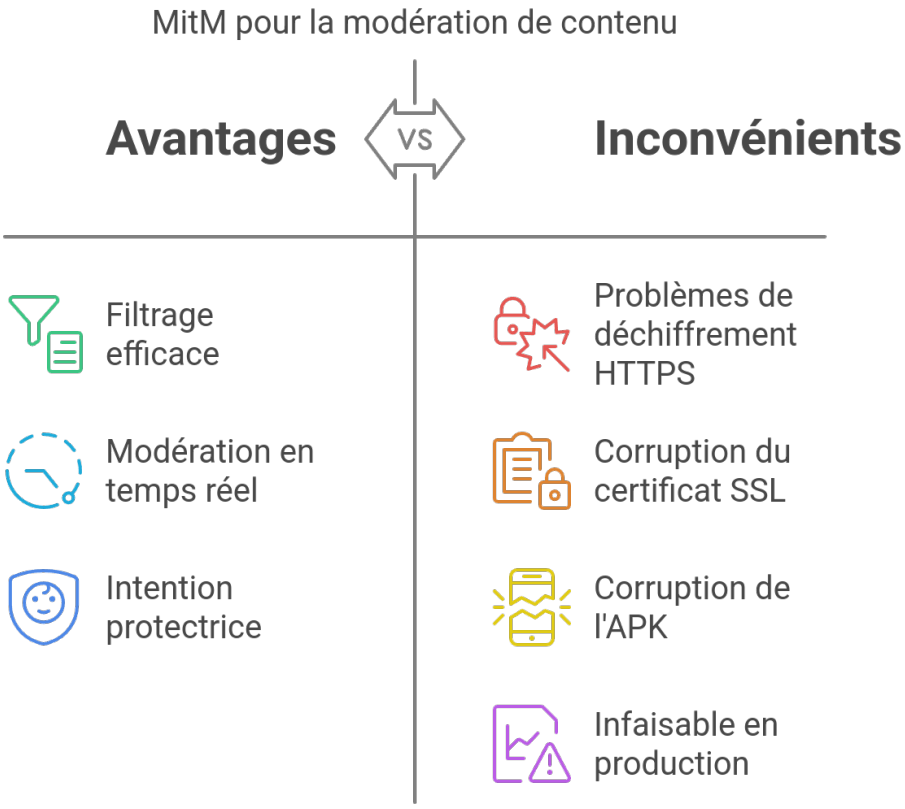


Figure 5: Avantages et inconvénients du concept Man in the Middle pour filtrer les posts sur X

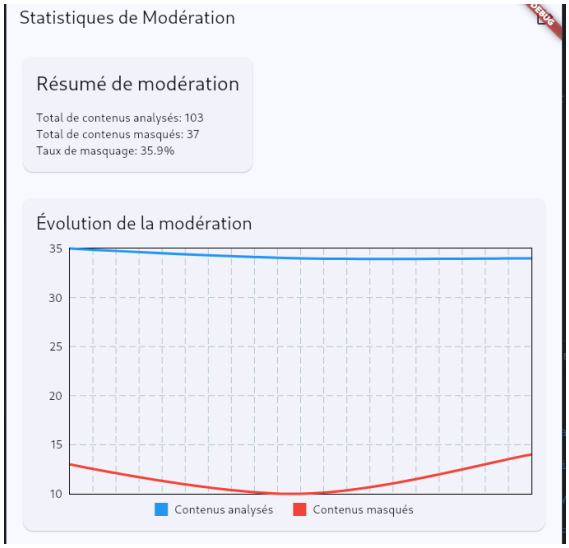


Figure 6: Proportion de contenu censuré

Figure 7: Statistiques des émotions détectées

Exemple de masquage d'un tweet

Voici un exemple de tweet filtré par notre proxy, cf. Figure 8.
Le contenu du tweet est remplacé par des étoiles.

Figure 8: Proportion de contenu censuré

Figure 9: Tweet original

Difficultés rencontrées

Le développement de cette solution a été particulièrement complexe, notamment à cause de la complexité des réseaux sociaux et de la difficulté à intercepter les requêtes HTTPS.

Comme mentionné, nous avons rencontré des blocages liés au mode de filtrage choisi assimilable à une cyber attaque.

Nous avons aussi eu des problèmes sur le format des requettes API. En plus d'être peu optimisées (le même contenu est chargé plusieurs fois), elles diffèrent aussi de l'application web à l'application mobile.
En effet, le site web effectue des requettes sur les serveurs de X, tandis que l'application mobile continue d'utiliser ceux de twitter.com. Les réponses sont similaires mais diffèrent légèrement (CamelCase contre snake_case notamment), ce qui nous a semblé un peu incompréhensible

Conclusion

En conclusion ce projet a été une expérience enrichissante pour nous tous. Ce projet avait pour objectif de développer un logiciel capable d'identifier les émotions dans des textes avec l'aide de l'intelligence artificielle. Nous avons pu découvrir des nouvelles API d'agents intelligents: Mistral, DeepSeek, Gemma2, et les utiliser pour réaliser des analyses d'émotions sur des textes. Nous avons également pu créer notre propre modèle d'analyse d'émotions, DeepFeel, qui a montré des performances très intéressantes sur des textes. Puis nous avons réussi à faire des tests de performance sur ces modèles. Nous avons aussi entraîné un LSTM performant en français puis multi-langues: BERT. Enfin, nous avons découverts de nouveaux languages et bibliothèques de développement: Dart/Flutter et Qt6 pour réaliser des interfaces graphiques pour le retour utilisateur.

Nous avons travaillé en équipe, majoritairement sur les séances dédiées au projet. Nous avons pu nous répartir les tâches et les responsabilités pour avancer efficacement, en utilisant GitLab et Discord.

Nous avons également pu bénéficier de l'expérience de nos encadrants: M. Tanguy Gernot et M. Emmanuel Giguet, qui nous ont proposé des pistes d'amélioration tout au long du projet. On les remercie pour leur disponibilité et leur aide.

Références bibliographiques

1. OLLAMA. *GitHub: Ollama*. 2025. Aussi disponible à l'adresse : `https://github.com/ollama/ollama`. Consulté le 23 mars 2025.
2. DOGARU, Ninel. *GitLab: Identification des émotions par IA*. 2025. Aussi disponible à l'adresse : `https://gitlab.ecole.ensicaen.fr/dogaru/identification-des-emotions-par-ia`. Références du projet.
3. DEVLIN, Jacob ; CHANG, Ming-Wei ; LEE, Kenton ; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. 2018, t. abs/1810.04805. Disp. à l'adr. arXiv : `1810.04805`. Consulté le 23 mars 2025.
4. SARKER, Md. Rasel. *Twitter Dataset*. 2023. Aussi disponible à l'adresse : `https://www.kaggle.com/datasets/mdraselsarker/sentiment140-dataset-with-1-6-million-tweets`. Dataset non annoté de tweets utilisé pour le pré-entraînement du modèle.
5. WOLF, Thomas ; DEBUT, Lysandre ; SANH, Victor ; CHAUMOND, Julien ; DELANGUE, Clement ; MOI, Anthony ; CISTAC, Pierric ; RAULT, Tim ; LOUF, Rémi ; FUNTOWICZ, Morgan ; DAVISON, Joe ; SHLEIFER, Sam ; PLATEN, Patrick von ; MA, Clara ; JERNITE, Yacine ; PLU, Julien ; XU, Canwen ; SCAO, Teven Le ; GUGGER, Sylvain ; DRAME, Mariama ; LHOEST, Quentin ; RUSH, Alexander M. Transformers: State-of-the-Art Natural Language Processing. In : *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online : Association for Computational Linguistics, 2020, p. 38-45. Aussi disponible à l'adresse : `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.
6. DELUCIA, Alexandra ; WU, Shijie ; MUELLER, Aaron ; AGUIRRE, Carlos ; RESNIK, Philip ; DREDZE, Mark. Bernice: A Multilingual Pre-trained Encoder for Twitter. In : *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates : Association for Computational Linguistics, 2022, p. 6191-6205. Aussi disponible à l'adresse : `https://aclanthology.org/2022.emnlp-main.415`. Modèle spécialisé pour l'analyse des tweets multilingues.
7. SUPABASE. *Supabase: The Open Source Firebase Alternative*. 2025. Aussi disponible à l'adresse : `https://supabase.com`. Utilisé pour la collecte des statistiques.
8. HIGI, Niklas. *apk-mitm: A CLI application to disable Android SSL certificate pinning*. 2025. Aussi disponible à l'adresse : `https://github.com/niklashigi/apk-mitm`. Utilisé pour autoriser notre proxy sur l'application X.
9. GOOGLE. *Flutter: Beautiful native apps in record time*. 2025. Aussi disponible à l'adresse : `https://flutter.dev`. Framework utilisé pour le développement de l'application.
10. MARSH, Joel. *UX for Beginners: A Crash Course in 100 Short Lessons*. O'Reilly Media, 2015. ISBN 978-1491912683. Guide d'introduction aux principes UX.
11. QT-COMPANY. *Qt: A cross-platform application framework*. 2025. Aussi disponible à l'adresse : `https://www.qt.io`. Framework utilisé pour le développement de l'interface graphique.

ENSICAEN
6 Boulevard Maréchal Juin
14000 Caen, France
https://www.ensicaen.fr