Clément CHABAN, Pacôme DHISER, Andréas LEROUX, Baptiste PÉGNÉ

Rapport AIA-901

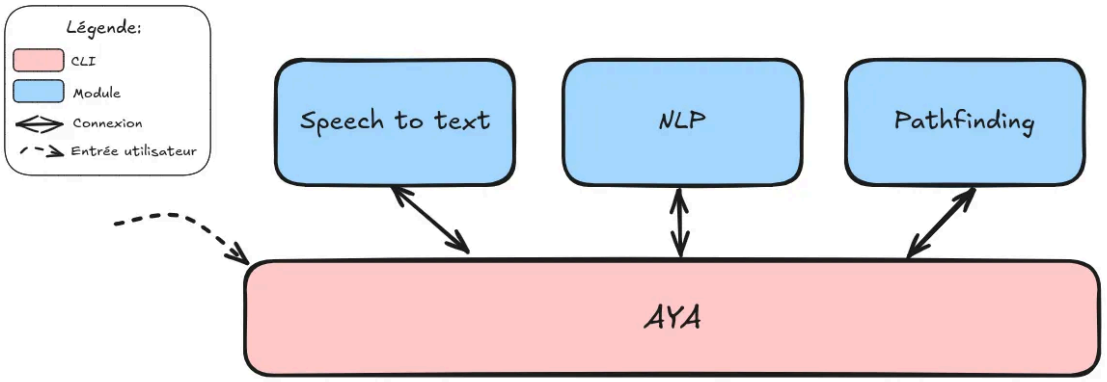
Groupe: NAN_2

Plan :

1. Architecture de l'application..... 3
1. AYA : L'Interface Centrale (CLI)..... 3
2. Modules Fonctionnels..... 3
3. Communication et Flexibilité..... 4
4. Avantages de cette Architecture..... 4
2. Création des jeux de données..... 4
1. Collecte et Préparation des Données.....5
2. Génération de Phrases à l'Aide de Templates..... 5
3. Annotation des Données pour le Modèle NER.....5
4. Constitution des Jeux de Données d'Entraînement et de Validation.....6
6. Conclusion.....6
3 Exemple détaillé de traitement d'un texte NLP..... 6
3.1 BERT (Bidirectional Encoder Representations from Transformers).....6
1. Pré-traitement des données..... 7
1.2. Tokenisation.....7
1.3. Annotation initiale en IOB2.....7
2. Alignement des étiquettes avec la tokenisation des Transformeurs..... 7
3. Extraction des informations et annotation NER.....8
4. Classification de la phrase et résultat.....8
3.2 SpaCy Model.....9
1. Pré-traitement des données..... 9
2. Extraction des informations (NER)..... 9
3. Classification de la phrase..... 10
4. Résultat final.....10
4. Résultats et Comparaison..... 11
1. Résultat de SpaCy sur la détection des phrases de voyage..... 11
2. Comparaison détaillée entre SpaCy et BERT pour l'extraction des informations de voyage.....13
2.1 Résultats chiffrés..... 13
2.2 Analyse des performances..... 13
5. Conclusion et Recommandations..... 14

1. Architecture de l'application

- **Diagrammes d'architecture** :



L'architecture du système AYA repose sur une structure modulaire qui permet d'interconnecter plusieurs composants tout en laissant la possibilité de les utiliser indépendamment. L'objectif est de transformer une entrée utilisateur en une commande exploitable en suivant un processus clair et organisé.

1. AYA : L'Interface Centrale (CLI)

AYA est une interface en ligne de commande (CLI) qui centralise les interactions entre l'utilisateur et les différents modules. Il joue le rôle d'orchestrateur en recevant les entrées utilisateurs, en appelant les modules nécessaires et en fournissant un retour pertinent.

L'utilisateur peut interagir avec AYA de deux façons :

- **Par texte**, en saisissant une commande directement.
- **Par la voix**, en utilisant le module de reconnaissance vocale (Speech to Text).

2. Modules Fonctionnels

Trois modules principaux interagissent avec AYA, chacun ayant un rôle bien défini :

- **Speech to Text** : Convertit la voix en texte pour permettre une interaction orale avec AYA.
- **NLP (Traitement du Langage Naturel)** : Analyse et comprend le texte afin d'en extraire des intentions et des informations clés. Il se divise en deux parties :
    - **Détection de l'intention** : Identifie l'objectif de la phrase (demande d'itinéraire, question, commande, etc.).
    - **Détection des villes concernées** : Extrait les noms des villes mentionnées pour les utiliser dans le module **Pathfinding**.

- **Pathfinding** : Effectue des calculs d'itinéraires et de navigation en fonction des besoins identifiés.

Chaque module est conçu pour être utilisé **de manière indépendante**. Cela signifie que l'on peut, par exemple, utiliser **le module NLP seul**, sans passer par AYA, pour analyser du texte dans un autre contexte. De la même manière, le module **Pathfinding** peut être utilisé pour du calcul de trajectoire, indépendamment du reste du système.

3. Communication et Flexibilité

L'architecture repose sur une communication fluide entre les composants :

1. L'utilisateur interagit avec **AYA** via une entrée textuelle ou vocale.
2. Si l'entrée est vocale, **le module Speech to Text** la transforme en texte avant de l'envoyer à AYA.
3. AYA transmet ensuite le texte au **module NLP**, qui en extrait l'intention.
4. Si un itinéraire est requis, **le module Pathfinding** est sollicité pour générer une réponse.
5. Enfin, le résultat est retourné à l'utilisateur via une visualisation.

Cependant, chaque module pouvant être utilisé **indépendamment**, un autre programme pourrait directement appeler **Speech to Text**, **NLP** ou **Pathfinding**, sans passer par AYA, selon les besoins du projet.

4. Avantages de cette Architecture

✅ **Modularité** : Chaque composant peut être amélioré ou remplacé sans impacter le reste du système.
✅ **Réutilisabilité** : Les modules peuvent être intégrés dans d'autres applications indépendamment.
✅ **Flexibilité** : AYA sert d'orchestrateur, mais son utilisation n'est pas obligatoire pour exploiter les modules.

Cette approche garantit une meilleure évolutivité et une plus grande souplesse dans l'utilisation des différentes fonctionnalités.

---

2. Création des jeux de données

Dans le cadre de notre projet, nous avons développé un jeu de données destiné à entraîner un modèle de traitement du langage naturel (NLP) spécialisé dans la reconnaissance d'entités nommées (NER). L'objectif est d'identifier les villes de départ et d'arrivée dans des phrases en français. Ce jeu de données est constitué de phrases valides contenant des

villes réelles et fictives, ainsi que de phrases invalides servant à améliorer la robustesse du modèle.

Nous avons distingué deux ensembles de données : un jeu d'entraînement et un jeu de validation, chacun ayant des objectifs spécifiques dans le processus de développement du modèle.

1. Collecte et Préparation des Données

Nous avons tout d'abord collecté une liste de villes réelles à partir d'un fichier GeoJSON contenant les gares françaises. Chaque ville a été extraite des propriétés du fichier et stockée dans une liste. En complément, nous avons ajouté une liste de villes fictives afin d'élargir la diversité des données et d'éviter un sur-apprentissage sur les seules villes réelles.

2. Génération de Phrases à l'Aide de Templates

Pour assurer la cohérence et la diversité du jeu de données, nous avons utilisé des modèles de phrases (ou templates) préétablis, stockés dans des fichiers CSV distincts pour les phrases valides et invalides. Ces modèles contiennent des espaces réservés pour les villes de départ et d'arrivée, qui sont ensuite remplacés aléatoirement par des villes issues de notre liste.

Exemple de template valide :

> "Je veux aller de [ville_depart] à [ville_destination] demain matin."

Exemple de template invalide :

> "Est-ce que [ville_depart] est plus loin que [ville_destination] ?"

3. Annotation des Données pour le Modèle NER

Chaque phrase générée est ensuite analysée pour identifier les positions des villes dans la phrase, ce qui permet de les annoter avec des entités spécifiques :

- `VILLE_DEPART` pour la ville d'origine du trajet.
- `VILLE_DESTINATION` pour la destination.

Nous utilisons des expressions régulières pour retrouver avec précision la position de ces entités dans la phrase. Cette étape est essentielle pour générer des annotations correctes, qui serviront ensuite à entraîner le modèle de reconnaissance d'entités nommées.

4. Constitution des Jeux de Données d'Entraînement et de Validation

Nous avons généré deux jeux de données distincts :

- **Jeu d'entraînement** : 7 500 phrases, comprenant 3 500 phrases valides et 3 500 phrases invalides. Cet ensemble est utilisé pour ajuster les paramètres du modèle.
- **Jeu de validation** : Ce jeu distinct est utilisé pour évaluer la performance du modèle après son entraînement. Il permet de détecter d'éventuels biais et d'améliorer la généralisation du modèle. Il est créé avec des phrases qui ne figurent pas dans le jeu d'entraînement.

Les jeux de données finaux sont sauvegardés au format JSON, avec une structure claire incluant la phrase et ses annotations associées. Ce format est adapté à une utilisation avec des frameworks de NLP comme SpaCy.

6. Conclusion

Grâce à cette approche méthodique de génération et d'annotation des données, nous avons obtenu des jeux de données équilibrés et structurés, idéaux pour entraîner et valider un modèle de reconnaissance d'entités nommées. La diversité des phrases et l'inclusion de données fictives permettent d'éviter le sur-apprentissage et d'améliorer la capacité du modèle à généraliser sur de nouvelles phrases.

---

3 Exemple détaillé de traitement d'un texte NLP

3.1 BERT (Bidirectional Encoder Representations from Transformers)

BERT est un modèle de deep learning basé sur les **transformers**, qui analyse une phrase dans son contexte global grâce à une attention bidirectionnelle. Il est particulièrement performant pour capturer des relations complexes entre les mots.

- **Avantages :**
    - Excellente compréhension contextuelle, même pour des phrases ambiguës.
    - Capable de traiter des structures complexes.
    - Performances supérieures sur des tâches nécessitant une interprétation fine.
- **Inconvénients :**
    - Plus gourmand en ressources (GPU/TPU recommandé).
    - Moins rapide pour le traitement en temps réel.
    - Nécessite un fine-tuning sur des données spécifiques pour être optimal.

L’exemple suivant va illustrer la pipeline du NLP sur un texte donné.
L’objectif est de montrer étape par étape, comment le pipeline effectue la reconnaissance des entités nommées (NER) ainsi que les transformations depuis l'entrée brute jusqu'au résultat final structuré.

Texte d’entrée :
- "Je veux aller de Paris à Marseille"
Texte attendu :
- Paris,Marseille

1. Pré-traitement des données

Le texte original utilisé en entrée est :
"Je veux aller de Paris à Marseille"

1.2. Tokenisation

À l’aide de la méthode _data_to_iob2, le texte est divisé en unités minimales (ou tokens) pour une analyse plus fine.

Exemple de résultat après tokenisation : ['Je', 'veux', 'aller', 'de', 'Paris', 'à', 'Marseille']

1.3. Annotation initiale en IOB2

Chaque jeton est annoté selon son rôle :

- **O** : Autre élément textuel, sans importance pour la tâche NER,
- **B-VILLE_DEPART** : Début d'une ville de départ,
- **I-VILLE_DEPART** : Intérieur ou continuation de la ville de départ,
- **B-VILLE_DESTINATION** : Début d'une ville de destination,
- **I-VILLE_DESTINATION** : Intérieur ou continuation de la ville de destination.

  Les annotations issues du texte :

  Tokens : ['Je', 'veux', 'aller', 'de', 'Paris', 'à', 'Marseille']
  Tags : ['O', 'O', 'O', 'O', 'B-VILLE_DEPART', 'O', 'B-VILLE_DESTINATION']

2. Alignement des étiquettes avec la tokenisation des Transformeurs

Les modèles de transformeurs tels que CamemBERT tokenisent un peu différemment le texte, générant des sous-tokens. Un alignement est réalisé pour lier chaque token d'entrée à son étiquette correspondante.

Exemple :

Tokenisation par CamemBERT :

['_Je', '_veux', '_aller', '_de', '_Paris', '_à', '_Marseille']

*Le symbole "_" marque le début d'un mot entier dans la phrase d'origine.

- Si un token commence avec "_", cela signifie que ce token est le début d'un mot d'origine.
- Si un token ne commence pas par "_", cela signifie que ce token est une sous-unité associée au mot précédent.

Indices des mots originaux :

[0, 1, 2, 3, 4, 5, 6]

Étiquettes après alignement :

[-100, -100, -100, 'O', 'B-VILLE_DEPART', 'O', 'B-VILLE_DESTINATION']

Ici, -100 désigne les sous-tokens ou tokens spéciaux ignorés par le modèle lors de l'apprentissage.

3. Extraction des informations et annotation NER

Le modèle CamemBERT pour la classification des tokens est utilisé pour prédire les entités nommées. Les prédictions sont interprétées selon l'annotation IOB2.

Dans cet exemple :
Paris est détecté comme B-VILLE_DEPART → Ville de départ.
Marseille est détecté comme B-VILLE_DESTINATION → Ville de destination.

4. Classification de la phrase et résultat

À la suite de l'extraction des entités nommées, le contexte de la phrase est analysé automatiquement :

La phrase contient une VILLE_DEPART (Paris) et une VILLE_DESTINATION (Marseille), ce qui permet de classer la phrase comme une commande ou une requête de voyage valide.
En concluant donc le résultat : <id>Paris,Marseille

3.2 SpaCy Model

SpaCy est une bibliothèque NLP optimisée pour la rapidité et l'efficacité. Elle utilise une combinaison de **règles linguistiques** et de **réseaux neuronaux légers** pour identifier les entités nommées et comprendre la structure grammaticale des phrases.

- **Avantages :**
  - Très rapide et peu gourmand en ressources.
  - Facile à intégrer dans un pipeline de production.
  - Peut être amélioré avec des règles personnalisées.
- **Inconvénients :**
  - Moins performant sur des phrases ambiguës ou mal formulées.
  - Plus limité dans la compréhension profonde du contexte.

1. Pré-traitement des données

Le texte brut fourni en entrée reste tel quel :

"Je veux aller de Paris à Marseille"

Ensuite, on convertit en entité (spacy.tokens.Doc), car le texte passé au modèle Spacy doit être sous forme de document manipulable pour la normalisation.

Normalisation des phrases :

Lors de cette étape, si le texte est sous forme de chaîne brute, il sera converti en objet Spacy. La méthode utilisée : normalize_sentence().

2. Extraction des informations (NER)

Le modèle Spacy est entraîné pour reconnaître les entités nommées (NER). Lorsqu'il rencontre le texte donné, il prédit les entités présentes dans la phrase.

Méthode clé :
Les entités détectées sont extraites via get_entity(doc, entity_label) où doc est l'objet Spacy représentant la phrase, et entity_label indique le type d'entité recherché (par exemple, VILLE_DEPART ou VILLE_DESTINATION).

Résultat pour l'exemple :

- VILLE_DEPART (départ) :

  La méthode get_departure_city(doc) détecte l'entité "Paris".

- VILLE_DESTINATION (destination) :

  La méthode get_destination_city(doc) détecte l'entité "Marseille".

Résumé des entités détectées :
- Départ : Paris
- Destination : Marseille

3. Classification de la phrase

En complément de l'analyse des entités, une classification binaire est réalisée pour déterminer si la phrase correspond à une commande de voyage valide.
Dans cet exemple, le modèle SpacyIsTripModel analyse et vérifie que :

La phrase contient une entité de type VILLE_DEPART (Paris).
La phrase contient une entité de type VILLE_DESTINATION (Marseille).

Cela valide la phrase comme étant une commande de voyage valide.
Méthode clé :
is_true_sentence(sentence) retourne True pour indiquer qu'il s'agit d'une requête valide

4. Résultat final

Après toutes les étapes, le pipeline structuré fournit un résultat final prêt à être utilisé.

Le format du résultat suit la structure suivante :

<ID>,<VILLE_DEPART>,<VILLE_DESTINATION>

Pour notre exemple avec SpaCy :

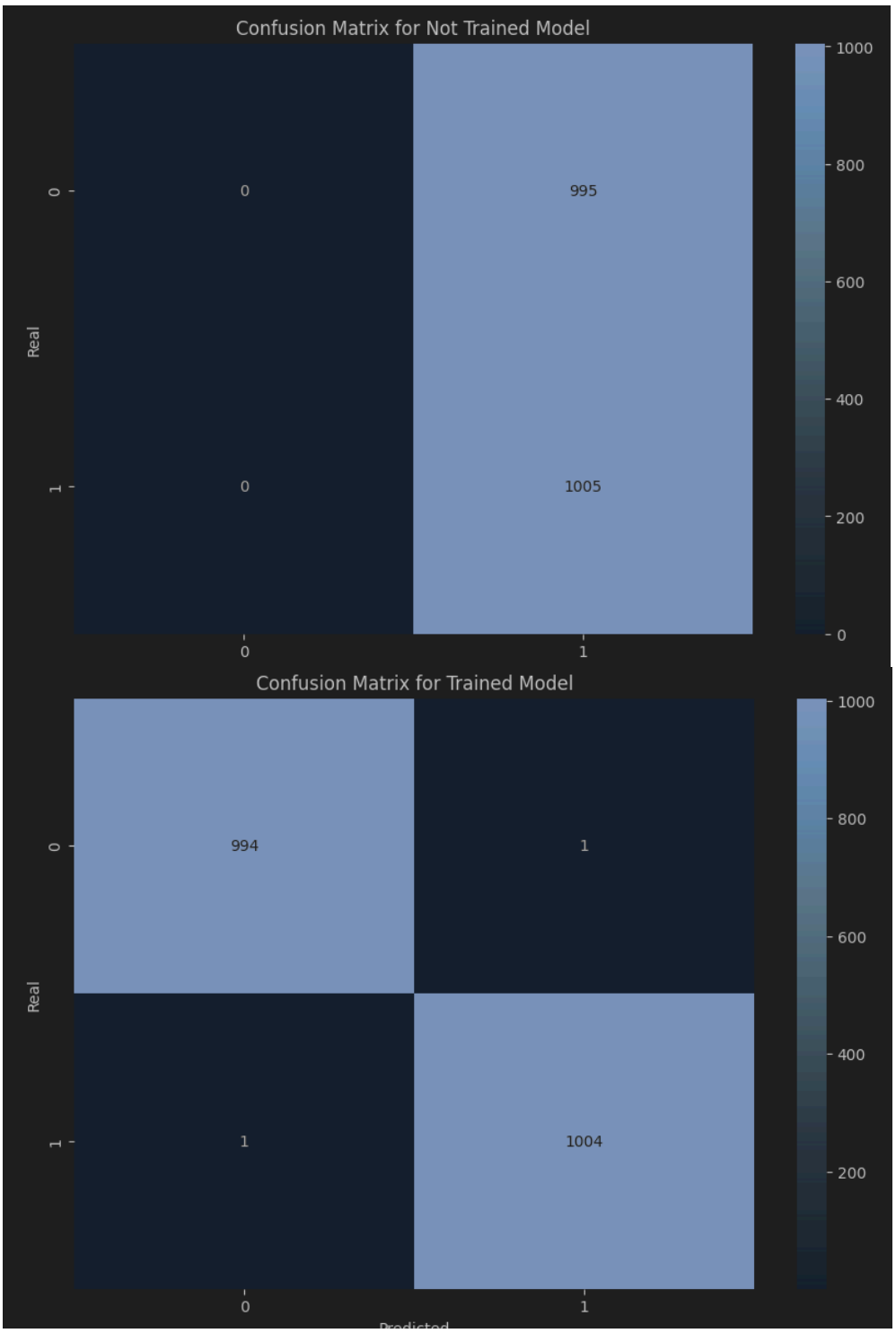
- <id>Paris,Marseille

4. Résultats et Comparaison

1. Résultat de SpaCy sur la détection des phrases de voyage

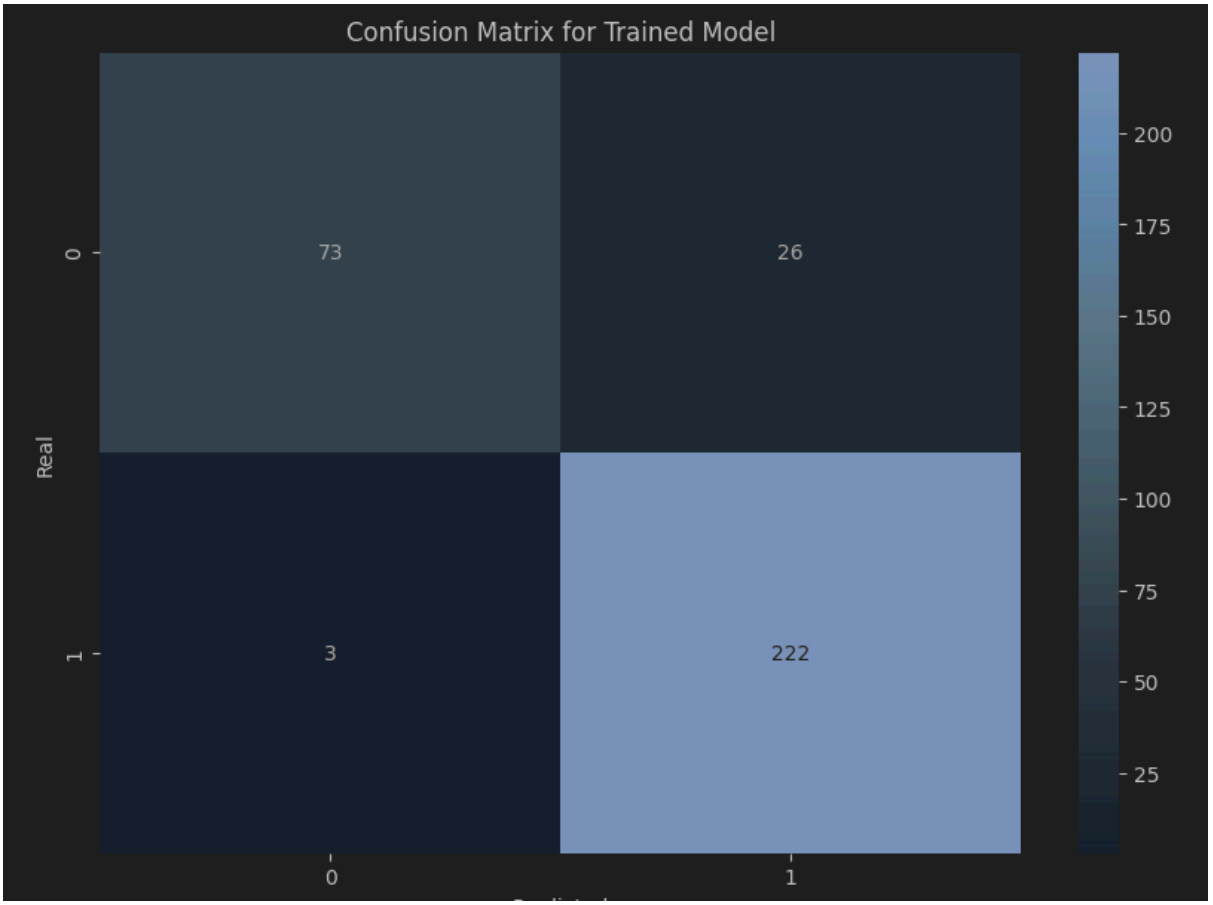
Pour l'évaluation des résultats de SpaCy sur la détection des phrases de voyage, nous utiliserons **les matrices de confusion** pour comparer les performances. Cela permettra d'analyser les **erreurs de classification**, en identifiant les faux positifs et les faux négatifs.

Pour plus de détails sur les données et les expérimentations, vous pouvez consulter les **notebooks fournis**.



La première matrice correspond au modèle SpaCy **non fine-tuné**, tandis que la deuxième représente les résultats après **fine-tuning**. Ces matrices montrent que le **fine-tuning est essentiel**, car il améliore considérablement les performances du modèle.

Les résultats du modèle SpaCy **entraîné** sont très prometteurs et démontrent l'importance de cette étape d'optimisation.



Avec le jeu de données de validation, les performances du modèle SpaCy **fine-tuné** sont moins bonnes, mais restent très convenables, ce qui est attendu étant donné que ce jeu de données est totalement différent de celui utilisé pour l'entraînement.

Le modèle rencontre plus de difficultés avec les **faux négatifs**, ce qui signifie qu'il a tendance à prédire davantage de phrases comme étant des voyages (**positifs**) plutôt que des non-voyages (**négatifs**). Cependant, dans notre situation actuelle, ce biais n'est pas réellement problématique.

2. Comparaison détaillée entre SpaCy et BERT pour l'extraction des informations de voyage

Dans cette section, nous allons approfondir la comparaison entre **SpaCy** et **BERT**, deux approches différentes de traitement du langage naturel, en nous basant sur nos résultats expérimentaux. Nous analyserons leurs performances sur la détection des intentions et l'extraction des villes de départ et d'arrivée, en mettant en évidence leurs forces et leurs limites..

2.1 Résultats chiffrés

Nous avons évalué les performances des deux modèles en termes de **précision**, **rappel** et **F1-score** sur l'**Extraction des villes de départ et d'arrivée**. :

Modèle	**Précision (Ville départ)**		**Rappel (Ville départ)**		**F1-score (Ville Départ)**		**Précision (Ville Destination)**		**Rappel (Ville Destination)**		**F1-score (Ville Destination)**	
SpaCy	90%		92%		91%		93%		87%		90%	
BERT (Begin; In)	95 %	100 %	100 %	96 %	100 %	95 %	76%	100%	70%	50%	73%	67%

2.2 Analyse des performances

Dans l'ensemble, BERT donne de meilleurs résultats, même après seulement 2 époques d'entraînement. En revanche, SpaCy, qui a été entraîné pendant 15 époques, présente des résultats légèrement moins bons.

Pour BERT, les résultats sont présentés avec les étiquettes **B (Begin)** pour le début de la ville et **I (Inside)** pour la suite du nom de la ville. On observe que la performance pour l'étiquette **I (Inside)** est moins bonne que pour **B (Begin)**.

L'étiquette **I (Inside)** représente la partie interne du nom de la ville, c'est-à-dire tous les mots qui font partie d'une entité nommée après le premier mot (étiqueté **B (Begin)**). Par exemple, dans le cas d'une ville comme "Paris Nord", **Paris** serait étiquetée **B (Begin)** et **Nord** serait étiquetée **I (Inside)**.

La performance moins bonne de l'étiquette **I (Inside)** par rapport à **B (Begin)** peut être expliquée par le fait que l'entraînement a été effectué pendant seulement 2 époques. Avec

un entraînement plus long, on aurait probablement observé de meilleures performances et moins de problèmes.

5. Conclusion et Recommandations

Critère	SpaCy	BERT
Rapidité	✅ Très rapide	❌ Plus lent (besoin en GPU gourmand)
Facilité d'intégration	✅ Facile à déployer	❌ Nécessite plus de configuration
Précision NLP	❌ Moins bon sur phrases ambiguës	✅ Très précis sur le contexte
Utilisation en production	✅ Léger et efficace	⚠️ Plus lourd en ressources

Quel modèle choisir ?

- Si **la rapidité et la légèreté** sont prioritaires → **SpaCy** est un bon choix.
- Si **la précision et la compréhension avancée du contexte** sont essentielles → **BERT** est préférable.
- Un **hybride** est possible : utiliser **SpaCy** pour le pré-traitement rapide et BERT pour les cas complexes.