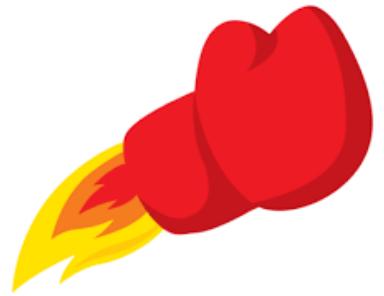




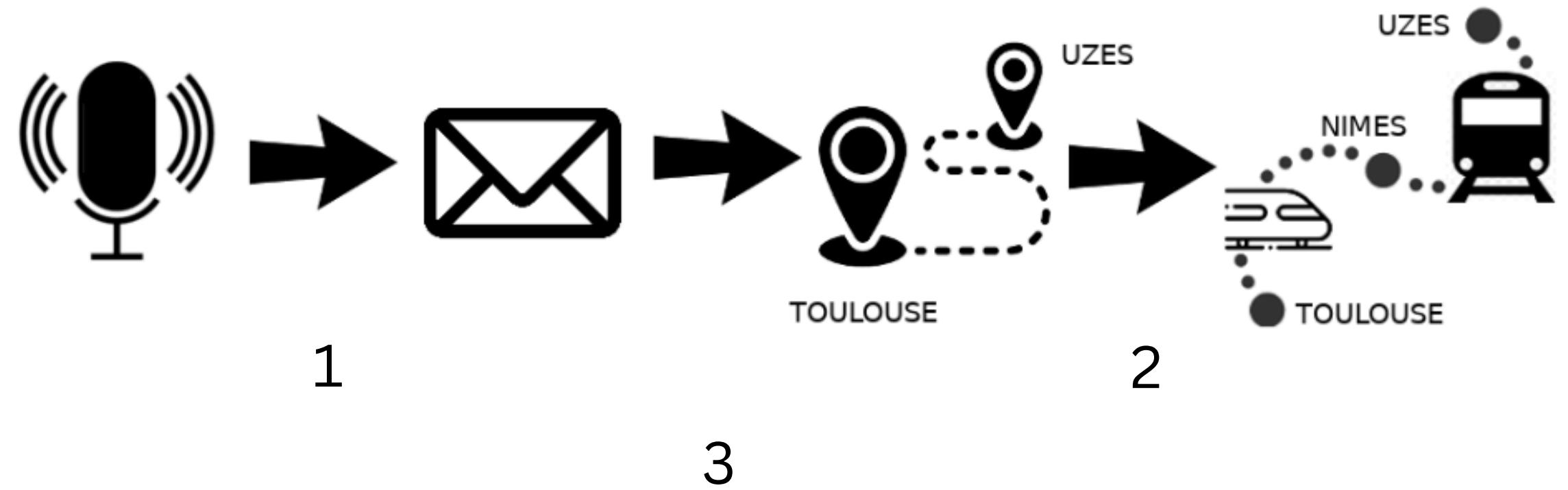
Théorie/KO

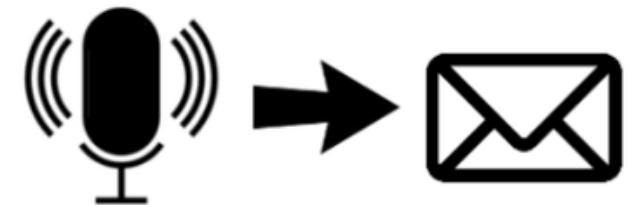
NLP

Travel Order Resolver



Travel Order Resolver





Travel Order Resolver

- Chaînes de Markov → avant 2010
 - Probabilités - Transition
- HMM, GMM → avant 2012
 - HMM: États cachés
 - GMM: Mélange gaussiennes
- RNN/LSTM → avant 2018
 - Récurrence
- Transformer → aujourd'hui
 - Attention



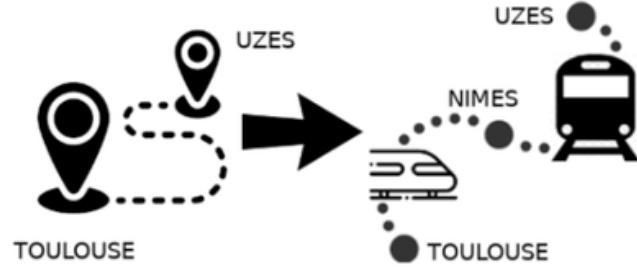
Doit fonctionner offline
(téléchargement auto de modèle : OK)

Prévoir de le déconnecter (texte seulement)



test_speech2text.py

- HMM → Hidden Markov Model
- GMM → Gaussian Markov Model



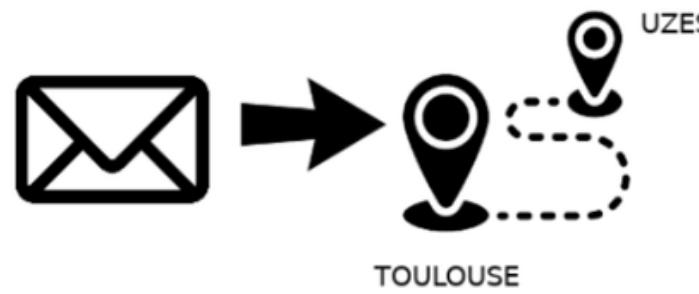
Travel Order Resolver

- Vous avez un CSV avec les gares et les distances
- Objectif → Trouver le trajet le plus court entre ces deux gares
- Transformation du CSV en données exploitables
- → Trouver l'algorithme adapté
- Complexité acceptable
- Résultat : chemin optimal + distance totale

- Autres datasets disponibles, possibilité d'en exploiter d'autres



A coder manuellement
Comprendre l'algorithme



Travel Order Resolver

- **Objectif NLP** : analyser chaque phrase et identifier les commandes de trajet en français
- Deux cas possibles :
 - **Phrase valide** → extraire Gare Départ & Gare Arrivée
 - **Phrase invalide** → rejeter avec un code d'erreur



Cas limites, erreurs, etc.

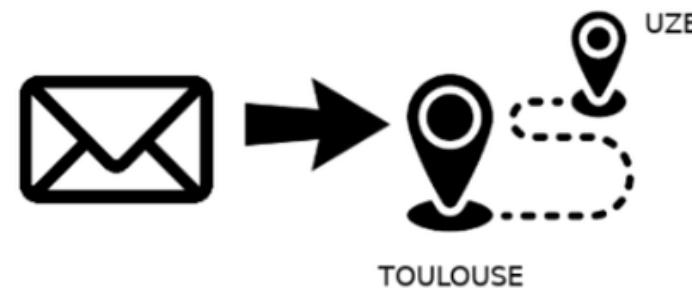
Données d'entrée : phrases libres

Exemples :

- “Je veux aller de Paris à Lyon”
- “Réserve-moi un billet Marseille-Nice”
- “Quel temps fait-il demain ?” → à rejeter

Sortie normalisée :

- sentenceID,**Départ,Destination**
- sentenceID,**Code**
 - Codes possibles : NOT_FRENCH | UNKNOWN | NOT_TRIP



Travel Order Resolver

1. Règles manuelles + regex + dictionnaire de gares
→ *Rapide à coder, fragile*
2. Modèle encodeur only (type BERT) + classification / NER
→ *Besoin de post-processing*
3. Modèles séquence-à-séquence (=encodeur-décodeur) léger + prompt + instruct
→ *Zéro-shot ou few-shot*
4. Fine-tuning classique de modèle séquence-à-séquence
→ *Besoin d'exemples (quelques milliers ?)*
5. Fine-tuning LoRA / QLoRA sur des modèles instruct plus gros
→ *Devrait tourner sur un laptop*

Quelques approches possibles / raisonnables pour ce projet (parmi d'autres)



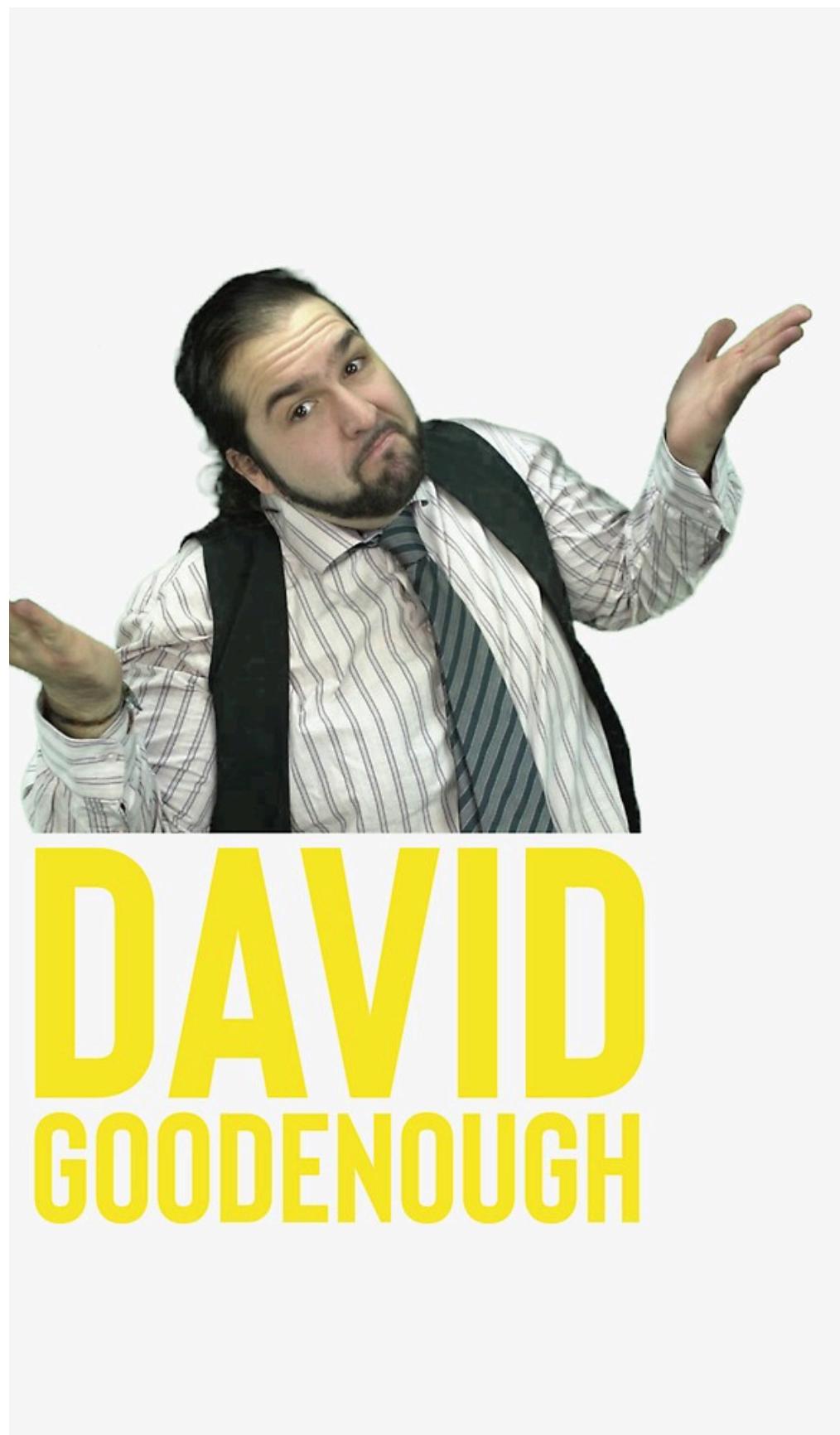
En tester plusieurs
Les comparer

SpaCy (+ pre/post traitement)



« Pourquoi on devrait faire du fine-tuning alors qu'on a déjà fr_dep_news_trf qui est basé sur CamemBERT et qui est censé être le meilleur modèle spaCy français ? »

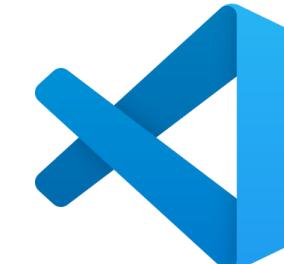
SpaCy (+ pre/post traitement)



DAVID GOODENOUGH

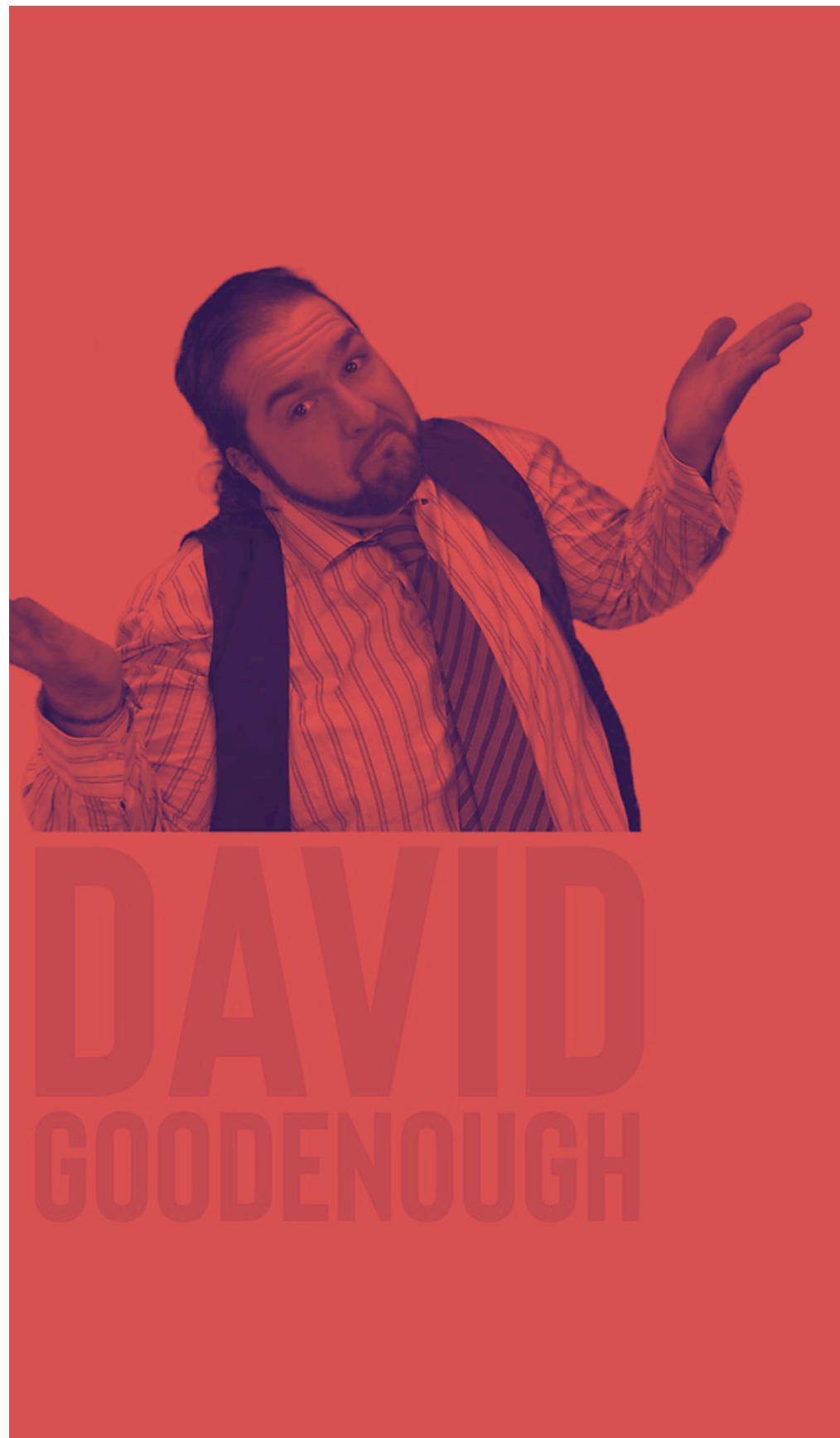
fr_dep_news_trf est effectivement le meilleur pipeline spaCy français actuel :

- il utilise CamemBERT
- il est très bon sur du texte journalistique
- sur des phrases bien écrites avec des grandes villes, bon pour la détection des LOC



test_spacy.py

SpaCy (+ pre/post traitement)



Mais il a des limites bloquantes ici :

1. Il ne connaît presque aucune des 3000+ gares SNCF
2. Il ne fait pas la différence entre départ et arrivée. Il faudra rajouter toute une couche de règles très fragile.
3. Il n'a pas de classification d'intention. Il ne sait pas dire si la phrase est une vraie demande de trajet ou pas. Il faut un autre modèle à côté.
4. Gère mal le français approximatif
5. Trop difficile de fine-tuner : spaCy verrouille le transformer



Les Transformers

- un peu de théorie

- Avant 2017, on traitait le texte séquentiellement : RNN, LSTM, GRU - chaque mot attendait le précédent (lent et limité en contexte).
- Les Transformers (Attention is All You Need, 2017)
 - Tout se passe en parallèle, et chaque mot regarde directement tous les autres mots de la phrase en une seule opération.
- Comment ?



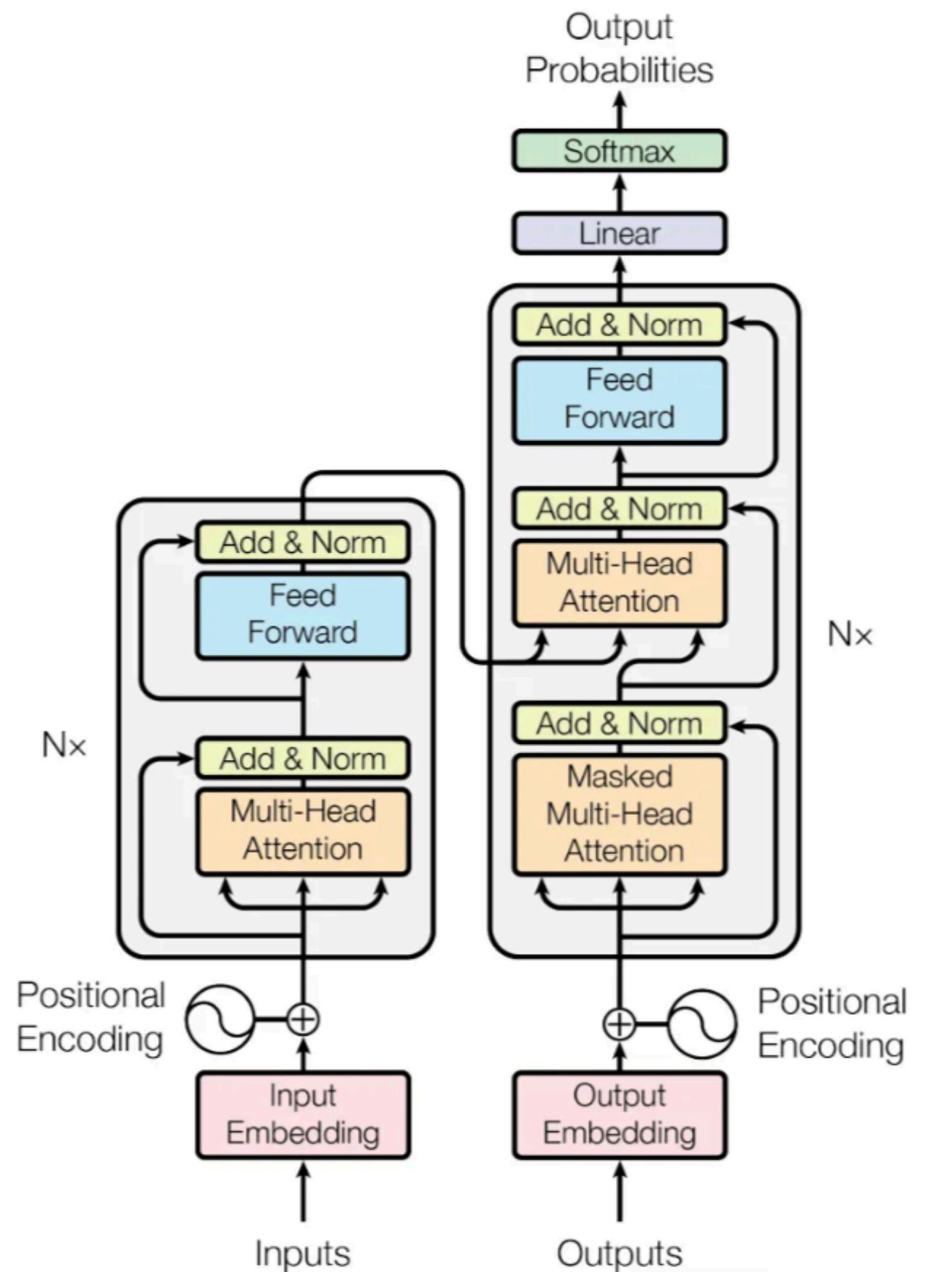
Les Transformers - comment ?

BERT

Encoder

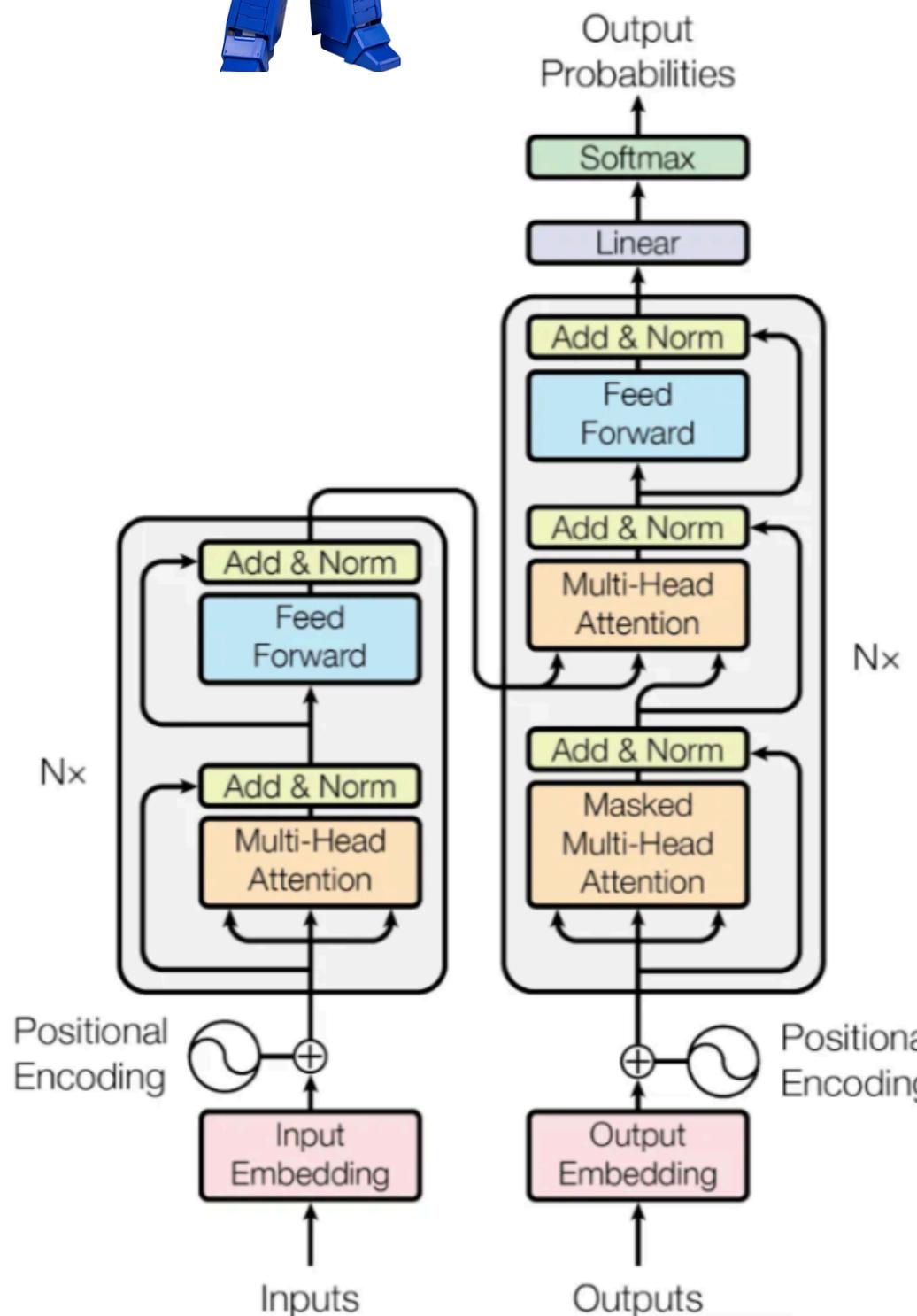
GPT

Decoder





Les Transformers - comment ?



- Embedding → mot(token) → vecteur
- Positional Encoding → + sinusoïdes fixes ou RoPE (donne l'ordre)
- Encoder (N=6-12 couches) → comprend la phrase entière
- Decoder (N=6-12 couches) → génère la sortie mot par mot
- Chaque couche
 - Multi-Head Attention
 - 8-16 têtes parallèles
 - $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{d_k})V$
 - Self-Attn (encoder) / Masked Self-Attn + Cross-Attn (decoder)
 - Feed-Forward
 - Add & Norm (Connexion résiduelle & stabilisation)
- Linear → projection finale
- Softmax → “probabilité” sur le prochain mot



Les Transformers - attention

1. On transforme chaque token en 3 vecteurs de même taille (ex : 512 dimensions) :

- Q = Query (ce que le mot “cherche”)
- K = Key (ce que les autres mots “offrent”)
- V = Value (le vrai contenu des autres mots)



Les Transformers - attention

2. On calcule une matrice d'attention :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(Q \times K^T / \sqrt{d_k} \right) \times V$$

Concrètement :

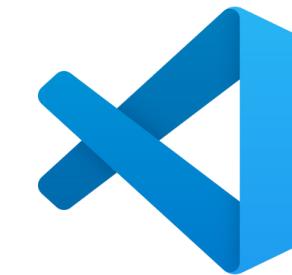
- $Q \times K^T$: matrice ($\text{seq_len} \times \text{seq_len}$) qui donne un score d'importance entre chaque paire de mots
- $/ \sqrt{d_k}$: stabilisation numérique (d_k = dimension de K)
- softmax : on normalise pour que la ligne somme à 1 (poids d'attention)
- $\times V \rightarrow$ on pondère les vrais vecteurs des mots par leur importance



Les Transformers - attention

3. Résultat :

"Je vais à Albert, je viens de Lyon,"



test_heatmap_attention.py
test_heatmap_q_k_v.py

- "Je viens de Lyon," → Lyon = départ
- "je vais à Albert" → Albert = arrivée

Un modèle classique (spacy ?) risque de se perdre dans l'ordre inversé.

Lyon regarde "de" → 0.8 → sait que c'est le départ

Albert regarde "à" → 0.8 → sait que c'est l'arrivée

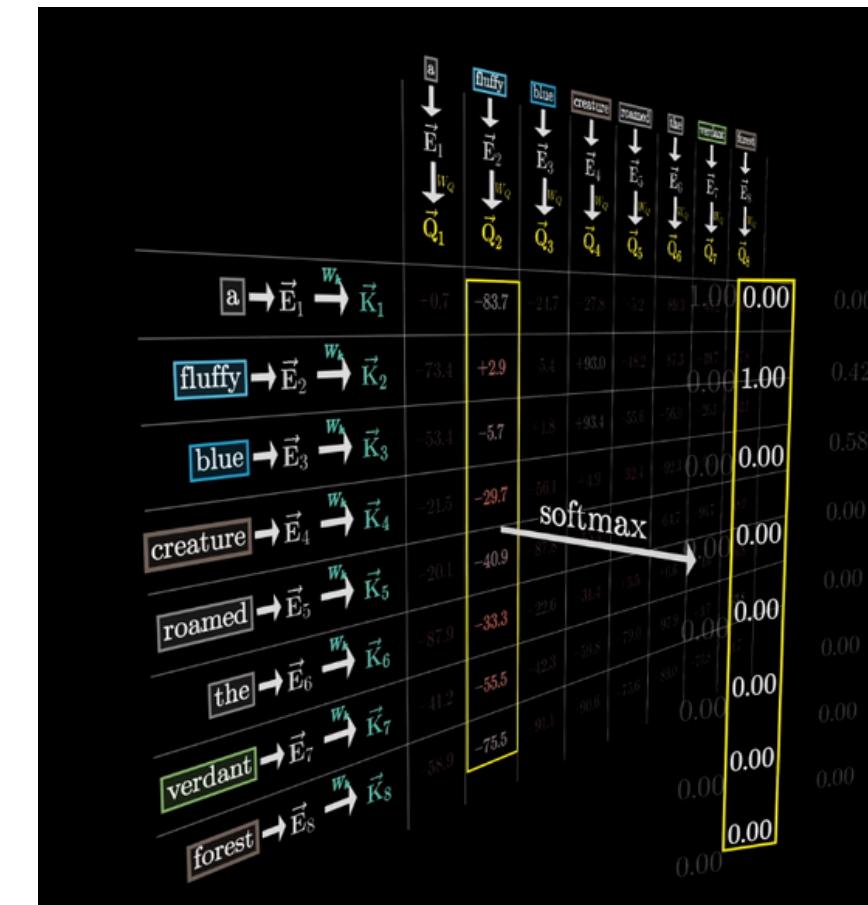
→ OUI | Lyon | Albert

Aucune règle, juste $Q \times K^T / \sqrt{d} \rightarrow \text{softmax} \rightarrow V \gg$

Les Transformers

Explication visuelle de qualité

- chaîne youtube : 3Blue1Brown



Neural networks

3Blue1Brown - 1 / 9



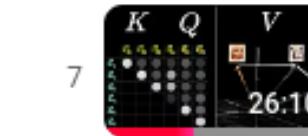
Large Language Models
explained briefly

3Blue1Brown



Transformers, the tech behind
LLMs | Deep Learning Chapte...

3Blue1Brown



Attention in transformers, step-
by-step | Deep Learning...

3Blue1Brown



How might LLMs store facts |
Deep Learning Chapter 7

3Blue1Brown



CamemBERT



(un modèle à tester parmi d'autres)

- CamemBERT-base = modèle Transformer de 110 millions de paramètres
 - pré-entraîné sur 138 Go de texte français brut (Wikipedia + CommonCrawl + livres + forums)
 - connaît déjà la grammaire, les expressions, 95 % du vocabulaire
 - ... mais il ne connaît pas toutes les gares, et peine sur la différence départ/arrivée.

Pré-entraînement → Fine-tuning

Pour pouvoir l'utiliser quand même



- **Pré-entraînement**

- Objectif : apprentissage général massif
- Dataset : 100Go - 10 To de texte brut
- Durée / coût : milliers de GPU sur des mois
- Résultat : Modèle polyvalent

- **Fine-tuning**

- Objectif : spécialisation sur le projet
- Dataset : Quelques milliers d'exemples
- Durée / coût : ordi perso sur des heures / jours
- Résultat : modèle qui comprend les demandes de trajets



Grandes époques NLP

- Avant 2016 : TF-IDF + SVM / CRF
 - Limite : pas de contexte
- 2016-2018 : FastText + BI-LSTM-CRF
 - Limite : contexte court
- 2018-2020 : BERT / CamemBERT Transformers
 - Limite : Encodage seulement
- 2021-2025 : Flan-T5, Mistral-7B, Llama-3-Instruct
 - Limit : coût

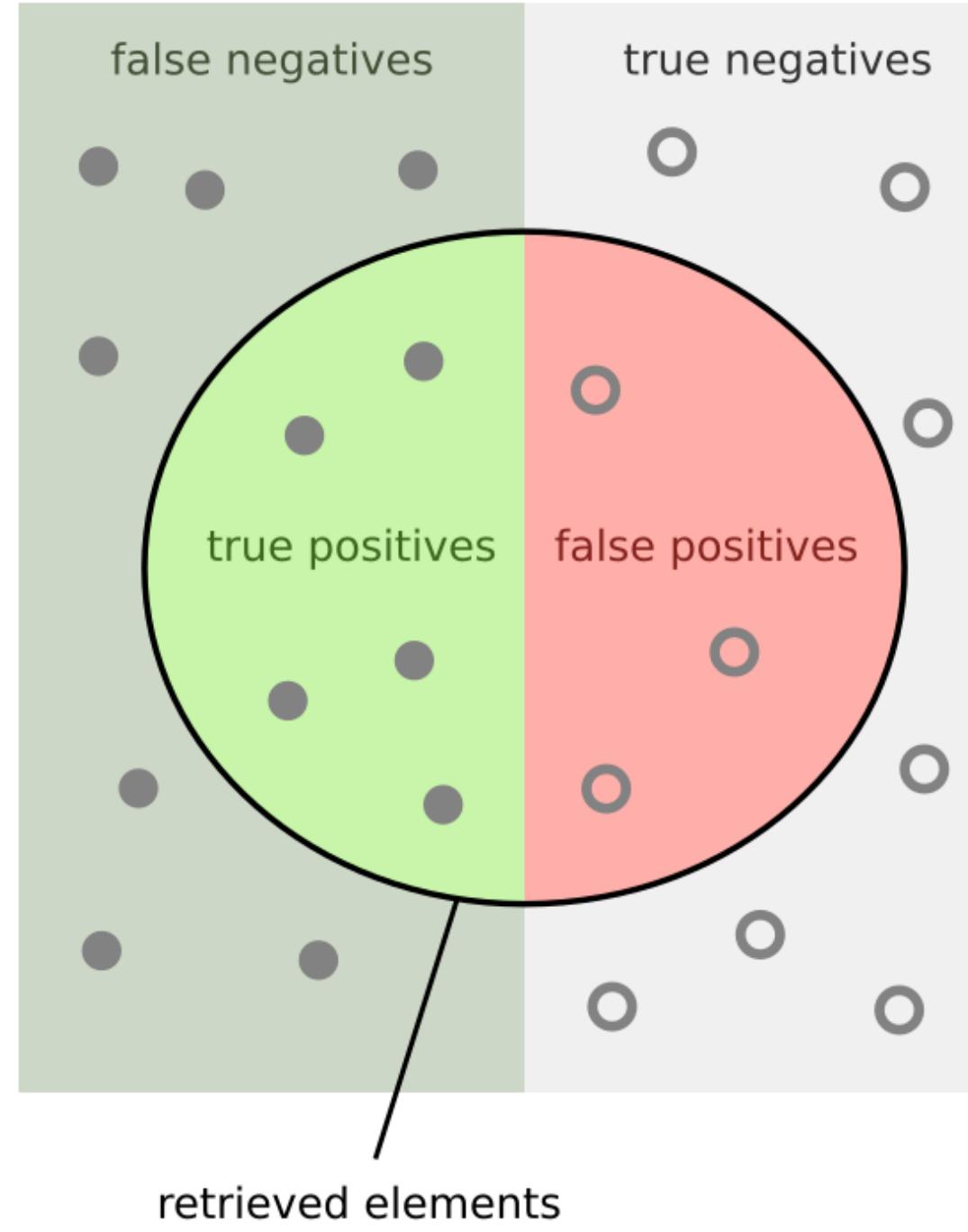


Pause - Vue d'ensemble

- Modèles pour speech-to-text
- Modèles pour NLP
- Modèle/algo pour path finding
- Interface / application (web ?)
- Métriques, rapport
- Planification, travail en groupe (normes, git, etc.)
- Architecture code
- Création de datasets
- Autre chose ?
- Des questions ?



relevant elements



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{How many retrieved items are relevant?}}{\text{How many retrieved items are retrieved?}}$$

How many relevant items are retrieved?

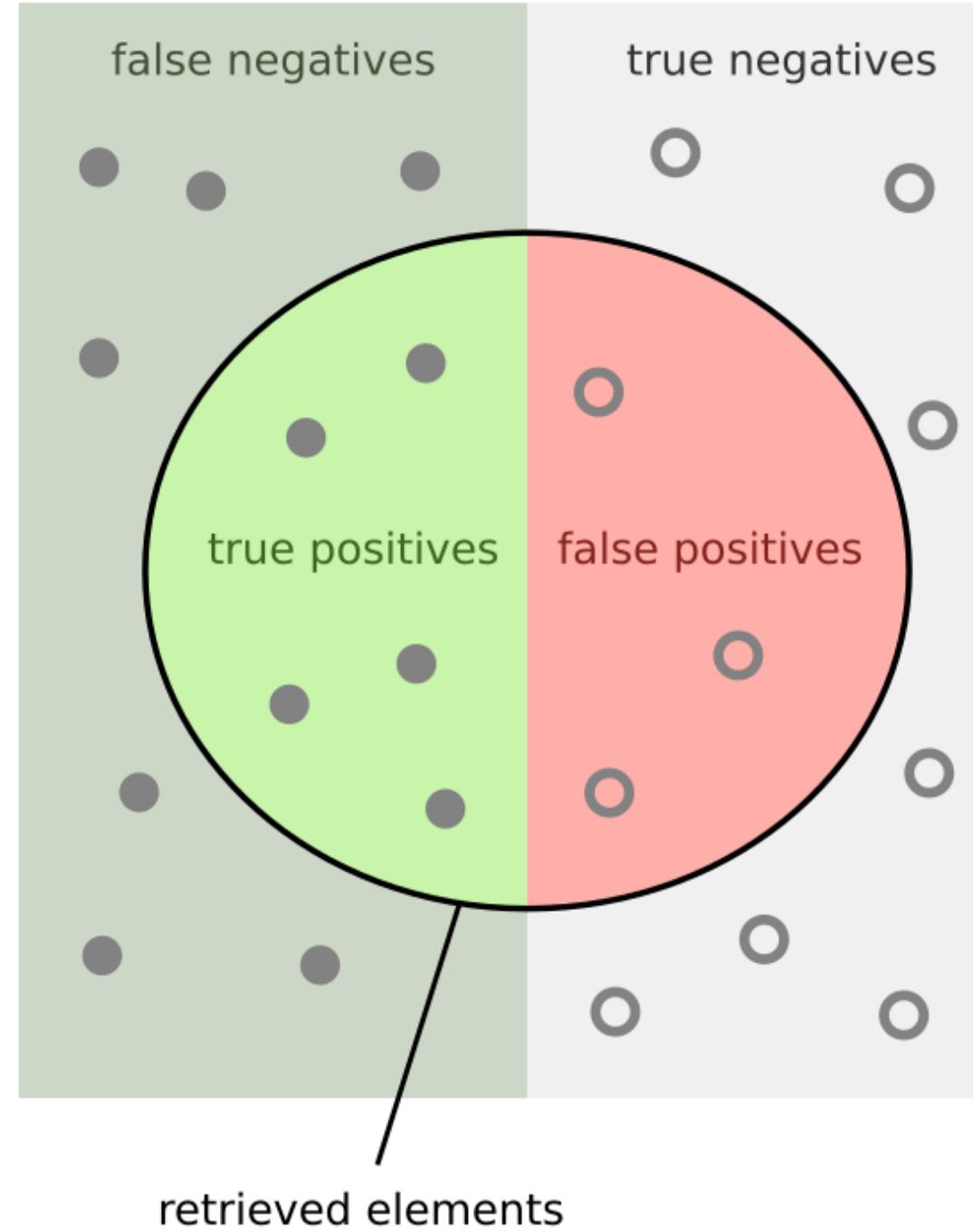
$$\text{Recall} = \frac{\text{How many relevant items are retrieved?}}{\text{How many relevant elements?}}$$

Si besoin : rappel métriques

Ex: Phrase correcte ?

- Precision = parmi celles que vous avez classées OK, combien étaient vraiment OK ?
- Recall = parmi toutes les vraies demandes OK, combien en avez-vous trouvées ?
- F1 = moyenne harmonique des deux

relevant elements



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{Number of true positives}}{\text{Number of retrieved items}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{Number of true positives}}{\text{Number of relevant items}}$$

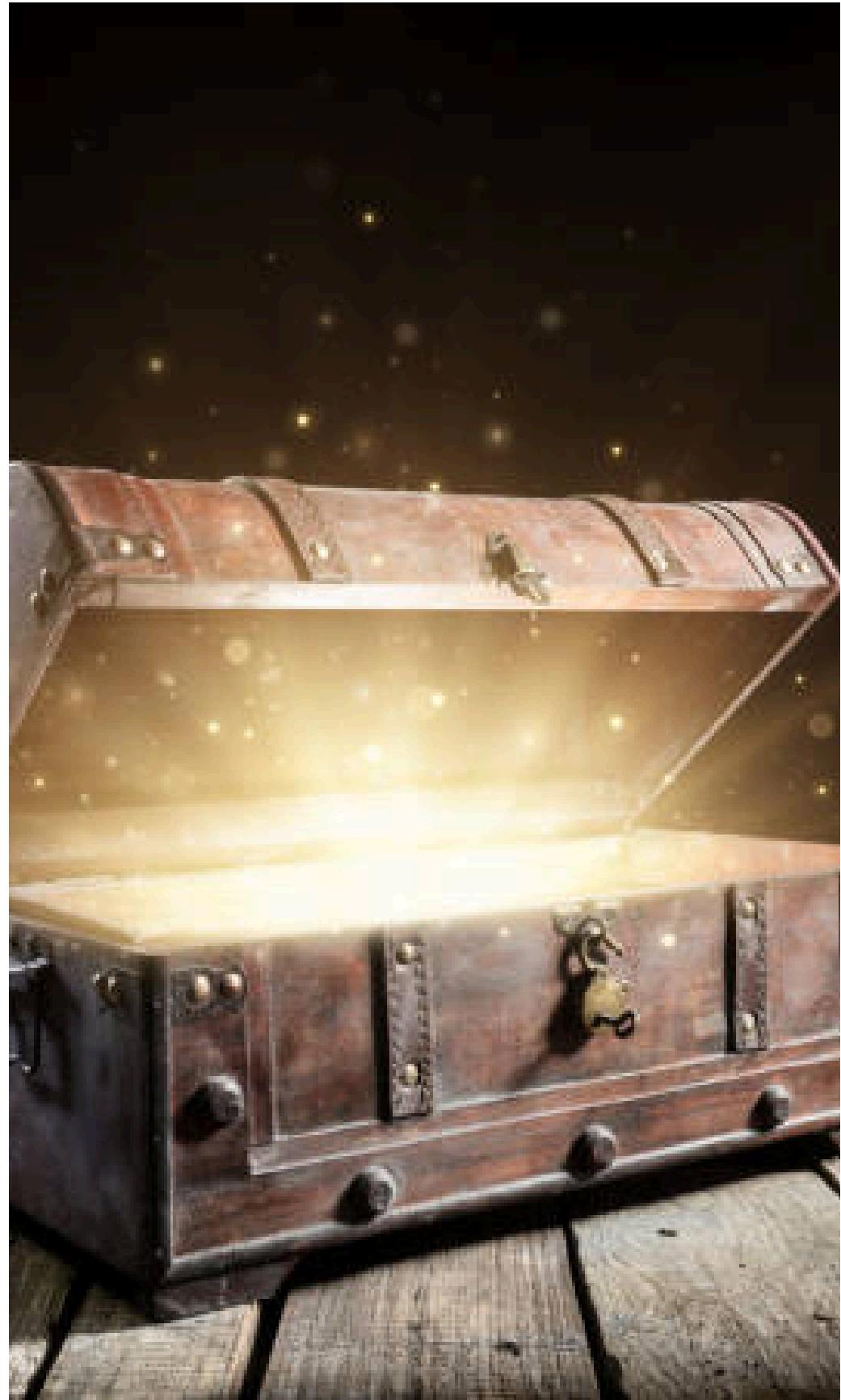
Si besoin : rappel métriques

Différentes métriques à trouver :

- taux de phrases où les 3 champs sont parfaitement corrects : intention + départ + arrivée
- % de phrases où OK / NOT_FRENCH / etc. est correct
- Parmi les phrases où les deux gares sont détectées, % où l'ordre est bon
- etc.

Garder les différents modèles pour les comparer

Ressources clés



Hugging Face



Exemple : flan-t5-base sur HF → Colab

Google Colab



Suggestion d'autres modules/outils (en vrac)

gradio, streamlit, transformers, peft, mlflow (ou wandb, si toujours illimité pour les étudiants), seaborn, tqdm, pandas, typing, etc.



Bootstrap

Travel Order Resolver

Bootstrap (suggestions)



Exploration

- Aller sur HuggingFace, commencer à regarder les différents modèles (NLP, speech-to-text) - comprendre les différentes familles / catégories
- Chercher les modèles/algos pour le path finding

Groupe

- Se mettre d'accord sur l'organisation, l'architecture, les modules, la communication, etc.

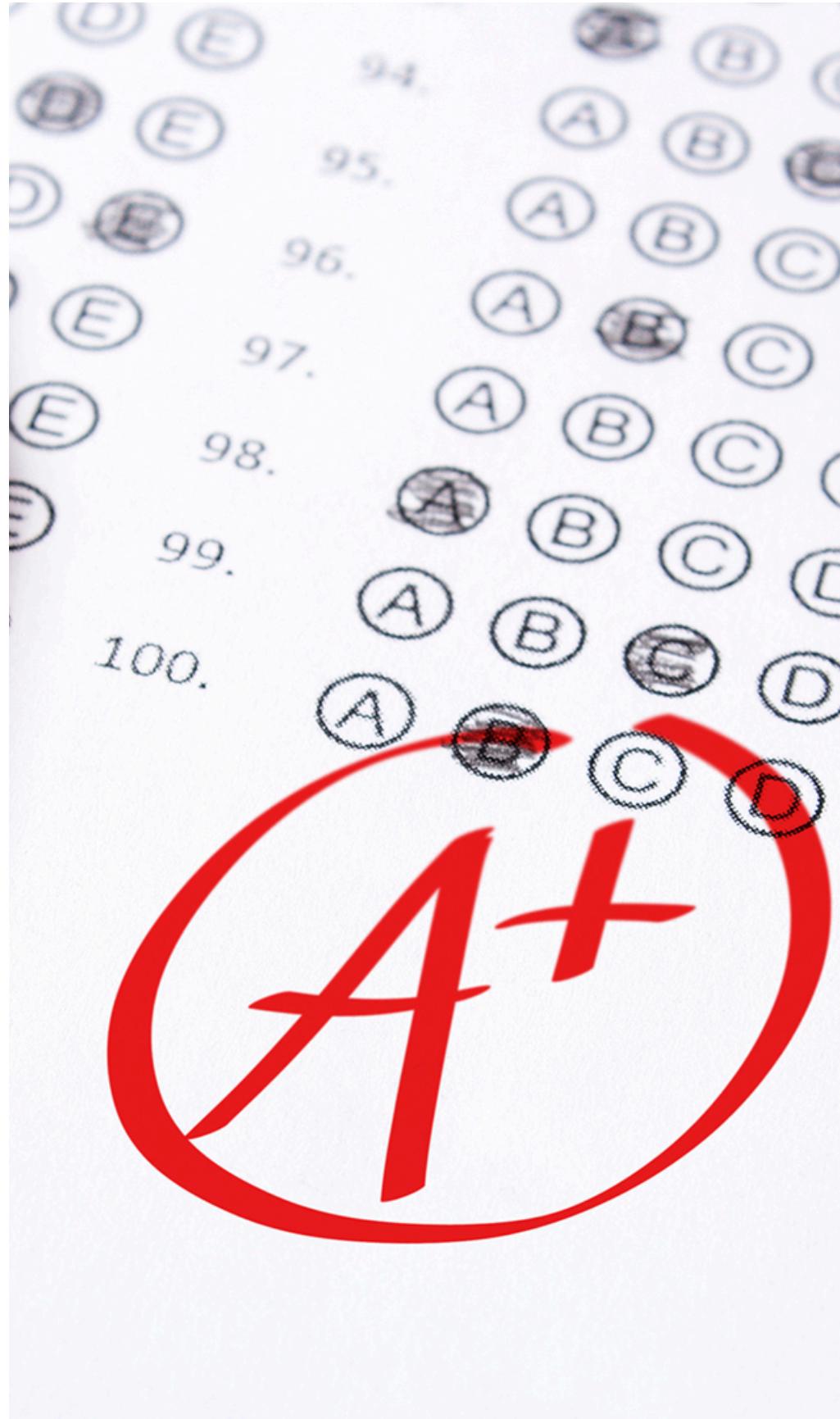
Code

- Commencer la partie speech-to-text. Quelles difficultés ? (“un train de Rennes à Lyon” ?)
- Commencer l’application ? → plus simple pour le speech-to-text
- Premier modèle NLP : spacy ?

Divers

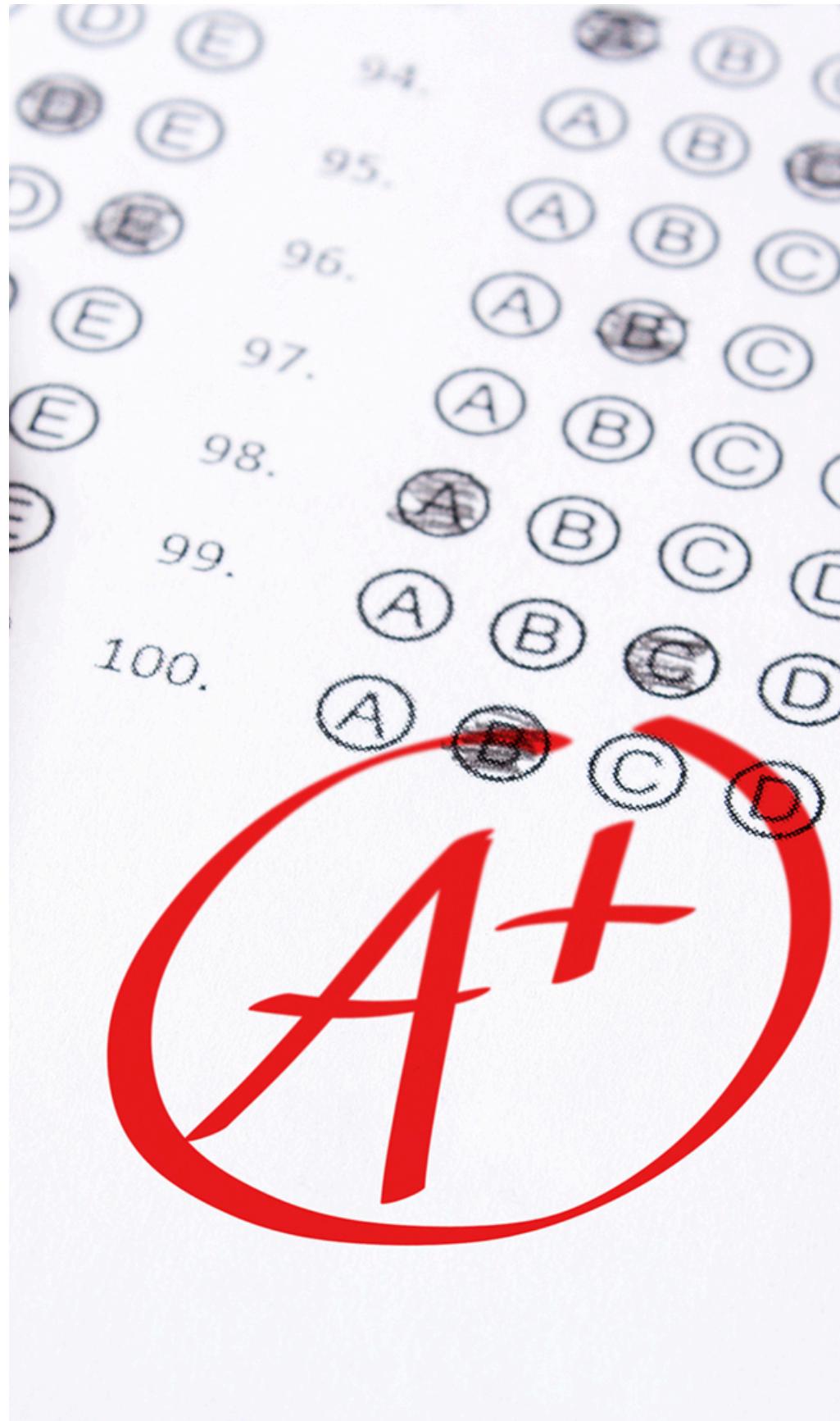
- Commencer à construire des datasets
- Bien relire le sujet

Conseils



1. Rapport ≥ 5 pages **utiles** (figures ++, code -)
2. **Benchmark** de plusieurs modèles NLP avec métriques
3. Ne perdez pas de temps sur l'appli
4. Documentez le processus de création des datasets
5. Gestion claire des cas invalides (langues, gares absurdes, etc.)
6. Code bien **découpé/organisé**
7. Git : branches + commits **réguliers de tous**, homogène, propre
8. Bien préparer l'oral, slides sobres
9. Soyez capables d'expliquer vos différents algos / modèles
10. Travail de **groupe** ! (\rightarrow pas chacun sa partie)
11. Prenez des initiatives
12. Communiquez

Dernière chose



- Considérez le projet comme un entraînement **proche de la réalité** du monde de l'entreprise, avec son lot de contraintes **arbitraires**.
- **Débrouillez-vous** avec un brouillon de cahier des charges plus ou moins clair. Le plus souvent, votre client ou votre employeur **ne sait pas lui-même exactement ce qu'il veut**. Ça peut évoluer en cours de route, et tout passe par la communication.
- Je ne suis pas là pour vous piéger, mais **pour vous aider** : n'hésitez surtout pas à me solliciter.