

Fiche ressources

TRAVEL ORDER RESOLVER

Présentation du projet

Ce projet consiste à construire un système capable d'interpréter une commande **textuelle** (principalement) ou vocale afin de déterminer **si une phrase est un ordre de voyage valide**, puis d'en extraire **au moins** une *ville/gares de départ* et une *ville/gares d'arrivée*. Le système doit ensuite générer un **itinéraire ferroviaire valide** en utilisant les données SNCF officielles.

⚠ Le PDF du projet précise clairement :

- Le **NLP est le cœur** du projet.
 - Le **STT est facultatif**.
 - Le **pathfinding est simple** et doit être livré en premier.
 - Le programme doit **obligatoirement traiter des phrases françaises** (autres langues = bonus).
 - Les commandes seront **évaluées en texte**, jamais uniquement en audio.
 - Votre programme doit être capable de distinguer **phrases valides et invalides**.
-

Bootstrap

- Découvrir les NER
- Spacy / hugging face
- metrics (F1, Precision, Recall)
- annotation manuelle d'un mini-dataset
- test sur phrases avec villes / direction

Méthodologie générale

REGARDER les données et les comprendre avant de faire quoi que ce soit.

1. Analyse du besoin

Objectif : Comprendre les attendues et la complexité du projet.

- La partie pathfinding consiste **uniquement** à produire une suite de villes.
- Le dataset doit être **massif** : environ **10 000 phrases**, issues d'environ **300 structures grammaticales**.

- Collaboration obligatoire entre groupes pour créer un dataset riche et varié.
 - Le projet ne nécessite pas un modèle STT multilingue géant → on peut viser **des modèles légers**.
 - Les informations nécessaires sont simple : villes, dates?, heures?, personne? → **pas besoin d'un ChatGPT**.
 - Path finding : réseau ferroviaire → **graphes pondérés**, potentiellement temporels. Complexité algorithmique.
-

2. Recherche d'algorithmes existants

Objectif: Trouver 2-3 algorithme pour chaque brique du projet

Pour pour chacun fournir:

- ressources informationnel
 - le papier scientifique
 - le blog
 - autre
 - ressources techniques
 - un code d'exemple
 - tuto
 - concepts clés à comprendre
 - ce que vous avez compris ou non
-

3. Sélection des métriques

- STT : WER, CER, Word Accuracy
- NLP : F1, Precision/Recall, Exact Match
- Pathfinding : temps de calcul, optimalité du chemin

Choisir 1–2 métriques de référence pour la comparaison.

4. Prototype et tests rapides

- Tester **sans entraîner** (modèles pré-entraînés)
 - Évaluer : vitesse, précision, difficulté d'usage
 - Retenir **un modèle baseline + un modèle cible**.
-

5. Analyse et création du dataset

Le PDF insiste fortement sur :

- La nécessité de **créer votre propre dataset** pour le NLP.
- Objectif recommandé : $\approx 10\ 000$ phrases, couvrant au moins **300 structures différentes**.
- Inclure :
 - fautes d'orthographe
 - absence de majuscules / accents
 - prénoms ressemblant à des villes (Paris, Albert, etc.)
 - villes composées de mots communs (Port-Boulet)
 - ordres inversés (destination avant départ)
 - formulations ambiguës
- Travailler en **collaboration** entre groupes (Teams ou autre) pour maximiser la diversité.

Pour les données SNCF :

- utiliser les fichiers open data (gares, horaires, lignes)
- représentation en graphe : nodes = gares, edges = trajets
- import simple dans NetworkX, Neo4j ou une base graphe.
- Examiner les données SNCF : gares, horaires \rightarrow représentation en graphe.
- Compléter un dataset :
 - pour le NLP : annotation manuelle / recherche de données annotées
 - pour le STT : enregistrement audio si nécessaire

6. Implémentation des modules

Pour chaque module :

- **choix du modèle**
- **pré-traitement**
- **entraînement (si besoin)**
- **validation**

7. Assemblage du pipeline complet

Pipeline final attendu :

(sentenceID, sentence) → NLP → {INVALID ou (departure, destination)} → PATHFINDING → résultat

⚠ Le PDF précise :

- Le module NLP doit être **isolé** pour évaluation (fichiers séparés).
- Pas de webapp nécessaire.
- Les fichiers doivent être lus par **stdin ou fichiers**.
- Tout doit être en **UTF-8**.

AUDIO → STT → NER → {départ, arrivée} → PATHFINDING → Résultat

Tests d'intégration + jeux de scénarios.

⚙️ Algorithmes et ressources

1. Speech-to-Text (STT)

Niveau 1 — Simple

- **Vosk**- Repo : <https://github.com/alphacep/vosk-api>- Comprendre : modèles acoustiques + LM

Niveau 2 — Intermédiaire

- **Wav2Vec2 (CTC)**- Paper : <https://arxiv.org/abs/2006.11477>- Tuto HF :-
[<https://huggingface.co/learn/audio-course/chapter5/introduction>]
(<https://huggingface.co/learn/audio-course/chapter5/introduction>-
[<https://huggingface.co/blog/fine-tune-w2v2-bert>] (<https://huggingface.co/blog/fine-tune-w2v2-bert>- [<https://huggingface.co/learn/audio-course/en/chapter3/ctc>]
(<https://huggingface.co/learn/audio-course/en/chapter3/ctc>- Points clés : CTC, embeddings audio

Niveau 3 — Avancé

- **Whisper**- Paper : <https://arxiv.org/abs/2212.04356>- Implémentation rapide : <https://github.com/guillaumekln/faster-whisper>- Concepts : transformer audio, multitask

2. NLP — Named Entity Recognition (NER)

Niveau 1 — Simple

- **Rule-based + Regex**- Utile pour reconnaître les villes un dictionnaire- Baseline rapide

Niveau 2 — Intermédiaire

- **CRF** (Conditional Random Fields)- Paper : <https://www.cs.columbia.edu/~mcollins/crf.pdf>- Ressource: <https://www.geeksforgeeks.org/nlp/conditional-random-fields-crfs-for-pos-tagging-in-nlp/>- Exemple Python : <https://sklearn-crfsuite.readthedocs.io/en/latest/>

Niveau 3 — Avancé

- **CamemBERT NER** (transformers)- HF model : <https://huggingface.co/Jean-Baptiste/camembert-ner>- Concepts : subwords, attention, token classification
-

3. Path Finding

Outils recommandés (graphes / bases orientées graphes)

- **NetworkX** (Python)- Idéal pour prototyper rapidement les graphes SNCF.- Implémente Dijkstra, A*, Floyd-Warshall, etc.- <https://networkx.org>
 - **Neo4j** (base de données orientée graphes)- Import facile des CSV SNCF (gares, lignes, horaires).- Permet de faire des requêtes Cypher pour trouver des chemins.- Très utile si vous souhaitez visualiser votre graphe.- <https://neo4j.com>
 - **TypeDB** (anciennement Grakn)- Base de données orientée logique + graphes.- Très puissante pour représenter des relations complexes entre entités.- Pas nécessaire pour ce projet mais utile si vous voulez modéliser un réseau ferroviaire riche (types de lignes, contraintes temporelles, etc.).- <https://typedb.com>
-

Niveau 1 — Simple

- **Dijkstra**- Concepts : files de priorité, distances minimales- Visualisation : [youtube](#)

Niveau 2 — Intermédiaire

- **A***- Heuristique géographique- Ressource : [redblobgames](#)

Niveau 3 — Avancé

- **Graphes temporels (Time-dependent Dijkstra)**- Paper : [epubs.siam](#)- Idée : chaque train = une arête avec heure de départ/arrivée

Organisation et versionning

Outils recommandés

Pour un projet en équipe, la gestion des données, des modèles et des expérimentations est critique. Voici des outils conçus pour faciliter cette partie souvent difficile.

1. DVC (Data Version Control)

- Permet de versionner datasets, modèles, artefacts.
- Fonctionne comme Git mais pour les fichiers lourds.
- Indispensable pour suivre les versions d'entraînement.
- Site : <https://dvc.org/>

2. Weights & Biases (W&B) ou MLflow

- Suivi d'expériences : métriques, hyperparamètres, courbes.
- Permet de comparer facilement plusieurs modèles.
- W&B : <https://wandb.ai>
- MLflow : <https://mlflow.org>

3. Git LFS (Large File Storage)

- Pour versionner des fichiers volumineux directement dans Git.
- Moins structuré que DVC mais simple à installer.

4. Gestion de versions

- Nommer systématiquement :- model-STT-v1/ → baseline- model-STT-v2/ → fine-tuning- dataset-v1/ → brut- dataset-v2/ → corrigé / annoté
- Indiquer **comment une version a été produite** : paramètres, date, personne responsable.
- stocker les modèles : /models/STT/v1/ , /NER/v1/ ...
- enregistrer les notebooks d'expérimentation
- tenir un LOG.md avec les choix + résultats des tests
- documenter chaque module avec : input, output, métriques, exemples