TRAVEL ORDER RESOLVER_

< NATURAL LANGUAGE PROCESSING />

TRAVEL ORDER RESOLVER

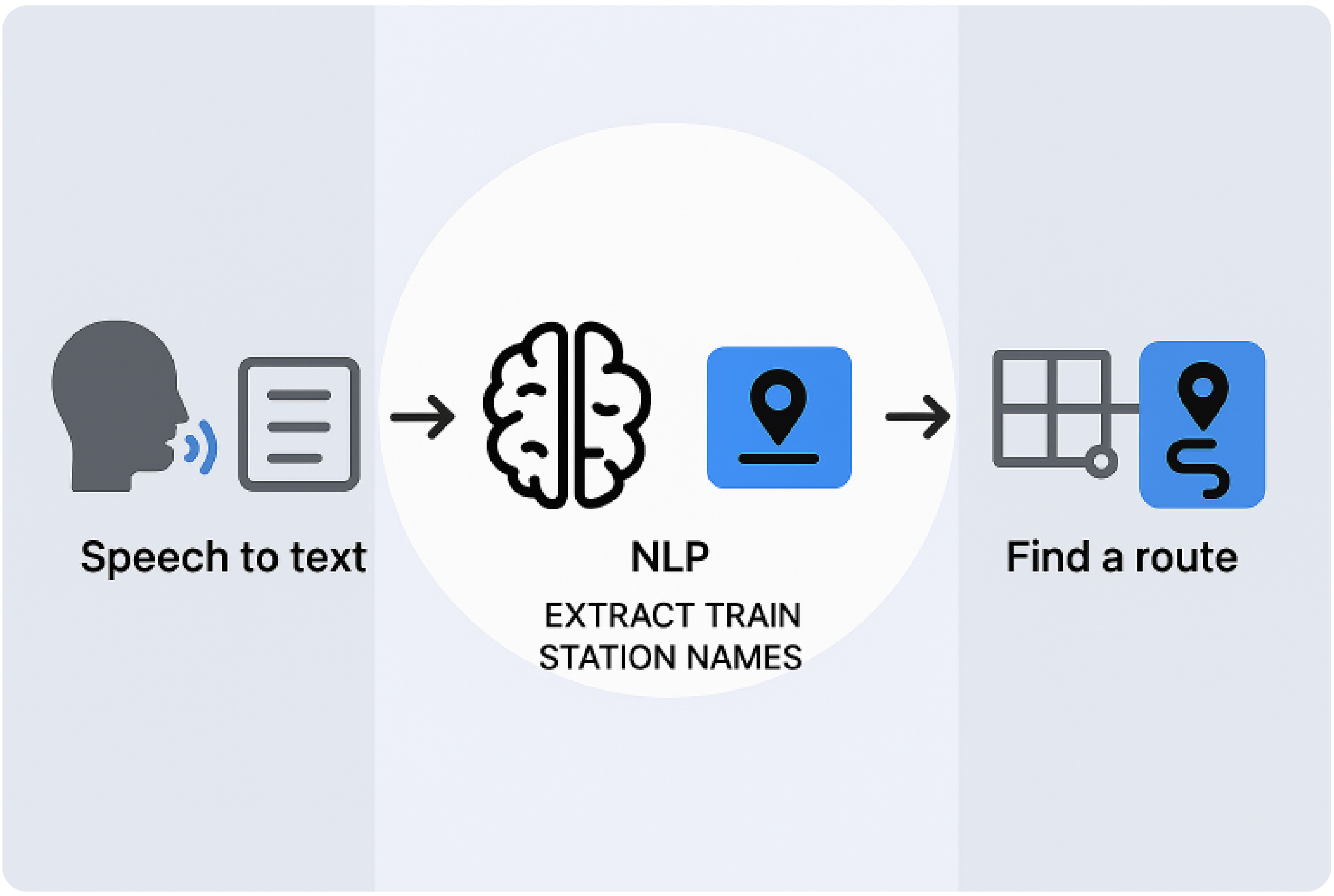
Build a program that processes text commands to generate an appropriate train itinerary.

More specifically, your software will receive trip orders as text (for instance a chatbot), and will output at least one appropriate travel line that fits the expectations of the customer, using French SNCF schedules.

It distinguishes between valid and invalid orders, and it identifies departure and destination, which can be a train station or a city. Although this is not the core of the project, it should then provide a corresponding train route and schedule.

Your NLP program must handle **French** sentences (as a bonus, it may additionally handle other languages)

As a bonus your program may for instance recognize intermediate stops, handle speech input...



You may add an optional voice recognition step, to convert voice sentence into textual data.

However, keep in mind that the project is mainly focused on NLP, and will be mainly evaluated with written inputs.

The NLP part is the main component, and shall extract (at least) origin and destination cities (or stations)

After text processing, you may add an optimization process that finds the best (either shortest or fastest) path in a graph, in order to get the optimal train connections. Rely on libraries and tools (e.g. neo4j) or implement an existing algorithm from scratch, but keep in mind the core of the subject is NLP.

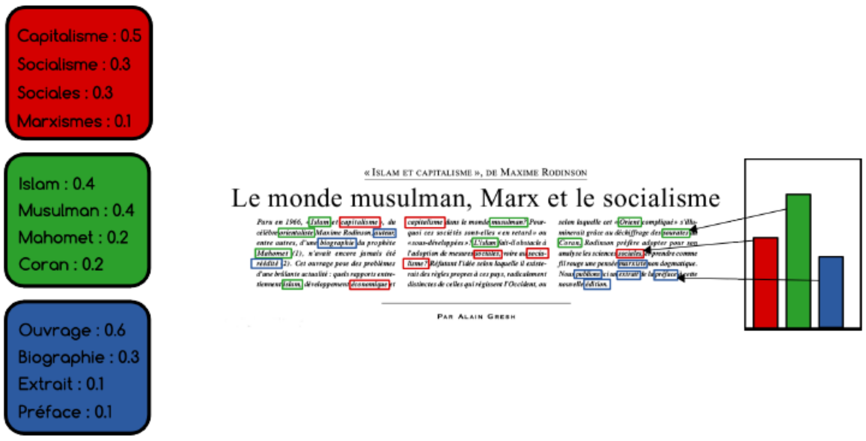
We assume that you already master the basics of Machine Learning and have some practice with related libraries.

Tools and techniques

Natural language processing

So you basically want to extract meaning from a text. NLP is an ever-expanding field of research of more than 50 years, with a wide range of techniques.
This means you have a lot of research to do and decisions to make before writing any code.

Some algorithms work with mere bags of words and use simple statistics on the data.
They are very useful for categorization or author identification, document context analysis, etc...



Some algorithms try to preserve the order of words or understand the grammar and context of specific sentences. They are often necessary for understanding fine information on small data.

For this, one must establish relations between words that may appear at the beginning or end of a sentence.
Transformers is one of the most effective architecture for handling this (the T of ChatGP**T** stands for Transformers).

Transformers were introduced in Attention is all you need.

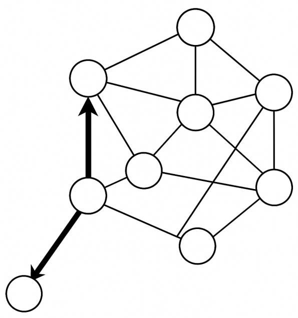
However, several simpler algorithms and libraries exist, and may be used at first to secure your project delivery

Such libraries, with simple processing, will also provide a base value for your metrics.

Path finding (Graph optimization)

When origin and destination are identified, you may find a train route, based on available schedules.

Train lines form a **graph** and many graph algorithms had been developed over the decades, and are a major field of algorithms and applications.



Finding an optimal path in a graph is a deterministic problem (which belongs to the P class), and several well-known algorithms exist, as well as library implementing them...

Take a look at the literature about algorithmic and complexity, before selecting an algorithm.

More generally, optimization algorithms are at the core of many ML solutions. In particular deep learning, which is based on backpropagation.

Graph are used to model numerous problems. For instance in IA, graph-based database are used for advanced RAG technique named GraphRAG. Take some time to check what Neo4j can do, an open source graph database that is widely used in these fields.

This is *not* a large task compared to NLP: some classical (and easy to implement) algorithms can be sufficient, and libraries can provide them.
However you are expected to understand the algorithm(s) and their complexity.

Voice recognition (bonus)

Since this is an isolated component of your architecture, and because it refers to a well-defined problem, implemented by many libraries, voice recognition should actually be the easiest part of the job here.

Nevertheless, the mathematics behind voice recognition is extremely interesting, and we strongly advise those who want to specialize in signal recognition to investigate how *Hidden Markov Models* work.

Most voice recognition libraries/services will correct misspellings and capitalize town names, thus providing sentences that are too simple to challenge your NLP implementation.

Orders recognition & sentence examples

We have high expectations regarding the sentence understanding. Finding town name and differentiating origin and destination is a complex task that should not be underestimated.

Some town names are made of common words. For instance *Port-Boulet* is a French town, but both "port" and "boulet" are common nouns.

Some town names may also represent people name, such as: Albert, Paris, Lourdes, etc...

> Comment me rendre à *Port Boulet* depuis *Tours* ?
> Je veux aller à *Tours* voir mon ami Albert en partant de *Bordeaux*.

Recognizing origin and destination should not be underestimated, as neither the order of the names (usually origin first, destination second) nor the adverbs ("de", "depuis", "à", "vers") are sufficient for this task.

> Je voudrais un billet Toulouse Paris.
> Je souhaite me rendre à Paris depuis Toulouse
> A quelle heure y a-t-il des trains vers Paris en partance de Toulouse ?
> Avec mes amis florence et paris, je voudrais aller de paris a florence.

Keep in mind that people may not use capitals, accents, hyphens, ...
This must be handled.

Obviously a model that handles misspelled sentence is more valuable than one that does not. This shall appear in your **metrics**.

Specifications

Input/output

The main **NLP** component should be able to:

✓ receive some text file in input:

- To be evaluated (and possibly compared to the work of other groups), your NLP part shall read sentences (one per line) from any file or URL.
- A simple option is to build a command line program that reads sentences on stdin (and use, for instance, cat or curl to get the sentences).
- Each line shall start have the following format : *sentenceID,sentence*
- The *sentenceID* will not be directly used by your program, but simply printed with the result, for validation.

✓ for each sentence, optionally check if the text is a valid trip order:

- return a negative answer if it isn't, using the format : *sentenceID,Code,* where code can be 'INVALID' or more precise
- return a triplet *sentenceID,Departure,Destination* if it is.

All input and output files **MUST** use UTF-8 encoding.

The final pathfinder component must be able to:

✓ receive as input a triplet *sentenceID,Departure,Destination* as defined above;

✓ return a **train route** as a sequence of cities: *sentenceID,Departure,Step1,Step2,...,Destination*.

Datasets

Some models, such as *CamemBERT* are pre-trained. However they have been trained with generic texts, and they need to be **fine-tuned** (by doing additional training) for a given problem

You will probably not find any existing dataset of sentences for this specific problem, and it is your job to build a training set for the NLP component (as well as a test set).

Be careful to include various kinds of trash texts, and that real orders explore the variety of grammatical structures.

As every person write differently, you will get a much wider dataset with more diversity if you do it collectively, on any collaborative media (Teams for instance).

In order to have a dataset with a wide variety of sentences, you must collaborate with other groups !

You probably need around 10000 different sentences, based on approximately 300 different sentence structures

Concerning train schedule, many information are available as SNCF open data, for instance station list information to have town names instead of station names.

It should be relatively easy to import these CSV files to a graph oriented database.

Deliveries

The main delivery consists of the NLP module, and the integration of the other parts (pathfinding and possibly voice to text), as well as **documentation**.

The NLP part **MUST** be isolated for testing and evaluation purpose.

We do **NOT** expect an integrated web application!!!

We obviously also expect a strong documentation, specifically on the NLP part (please provide PDF files), which includes:

- ✓ the full architecture of the various layers in your application ;
- ✓ a description of the training process (including the sets delivery) ;
- ✓ an identification of the final parameters (after training + fine-tuning) ;
- ✓ a detailed example of what happens to a given text all through the process ;
- ✓ explanation of different experiments and results.

You must **evaluate** your NLP processing!
And you must document the progress you've done on the NLP, for instance with the fine tuning.

You obviously need **metrics** to measure it!

Bonus

You can improve this project in many ways, including:

- ✓ speech to text module
- ✓ benchmarking different models, according to performance criteria,
- ✓ handling intermediate stops in the route (*Je voudrais aller de Toulouse à Paris en passant par Limoges*),
- ✓ considering waiting times in intermediary stations. Be careful: computational complexity may be a problem here,
- ✓ hosting your solution on cloud platform (e.g. Microsoft AI platform),
- ✓ monitoring CPU/RAM (and therefore cost or carbon footprint) for each request (and for training),
- ✓ handling destination not in the timetable file,
- ✓ searching through other means of transport,
- ✓ etc...

Proposed steps

It may be a good idea to follow this guideline:

1/ Start with generating sentence dataset. Try to include all possible grammatical structures and also include some misspellings. Also define how you will measure the quality of your model (metrics).

2/ Secure your delivery with a simple solution. Measure the quality of this simple model.

3/ Try a more complex model (possibly BERT), and try to fine-tune it. Check your metrics and their evolution when you fine-tune.

Don't forget to document your work.

v 3.1