Unitex 4.0 alpha and Prompt Engineering

Hanna Brinkmann

July 26, 2024

Contents

1 Introduction

This report is an overview what I did during my internship at the Institut Gaspard Monge of Université Gustave Eiffel from May 13th until August 2nd 2024. The main tasks were:

1. Comparing Unitex with ChatGPT and a Neuronal Network
2. Compare a ChatGPT translation with locally run Llama3 and Google Translate
3. Test the new functionalities of Unitex 4.0 alpha version
4. Write prompts to generate language exercises for a learning platform.

2 Named Entity Recognition - Comparison

2.1 Data Collection

The input text from which I wanted to collect the Named Entities is an extract of the novel "Around the World in Eighty Days" written by Jules Vernes. I chose to use the two first chapters for this test. The first step in data preparation was to annotate the Named Entities of the two chapters by hand. This annotation serves as gold annotation to which I can compare the results obtained by ChatGPT and Unitex. Step number two consisted in building Graphs in Unitex to recognize Named Entities and extract its predicted entities. In step number three, ChatGPT was prompted with the corpus in various portions. Iprompted the model with the full text at once, with both halfs of the text, four quarters and finally with eight eighth of the text (see table 1 below). For every portion Iasked ChatGPT to extract the Named Entities.

Portion	Number of Words	Number of Tokens
Full text	2864	5054
Half	1489	2627
Quarter	746	1303
Eighth	392	715

Table 1: ChatGPT prompt - Word and token distribution per portion.

2.2 Unitex

Unitex is a tool for semi-automatic text analysis developed by researchers at Unitversité Pars Est Marne-la-Vallée now called Université Gustave Eiffel. This tool allows us to build Grammars, Dictionaries and Patterns that are then applied to a text. For my purposes, I used the feature that can locate patterns

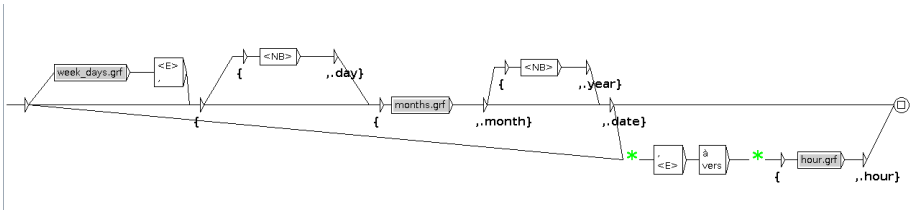

Figure 1: Graph searching for dates (and hours). Using sub graphs to define the weekdays and month as well as the pattern to recognize time.

in the text. Here, the first step is to build graphs that describe the pattern. In my case, I built one graph for every category in Named Entities. The easiest part was to build a graph for Dates and times as there is always the same preposition used in French and it also includes numbers. This graph is shown in figure 1 as an example.

For locations, names and organization, the recognition is more difficult. Unitex is based on prior selected dictionaries that ideally provide information on every word in the text. One of the dictionaries includes toponyms so I can use the property "city" or "country" in my graph. The problem I encountered doing this is that also the city or country specific adjectives or inhabitants are marked as city or country. But these words are not Named Entities therefore I do not want to extract them. Another dictionary includes first names with the feature "prenom" (first name in French). I therefore can extract the first names in the corpus but last names are not included. This challenge is met by adding another word after the first name that starts with a capital letter and that is included in none of the selected dictionaries. Another problem is that some of the first names used in my corpus are not part of the dictionary. Here it is useful to include the context. As the context leading to names changes every time, relying on titles helps recognizing names. For this event, I specified a list of possible titles including not only French titles like "Monsieur" and "Madame" but also English ones like "Sir", "Mister" and "Misses". If one title is followed either by one or two words not present in any dictionary and starting with a capital letter the sequence is recognized as a name. The most difficult part of Named Entities for Unitex are organizations because no dictionary includes organization names. Trying to resolve the problem, I built a graph that recognizes all the words that are not defined in any dictionary and that start with a capital letter. This gives us many occurrences that do not belong to organizations if applied alone. However, Unitex has a feature in which I can apply several graphs at the same time in a specific order. The occurrences already recognized by one graph, will not be recognized by the following. Therefore, I apply first the graph for dates and times, second the graph for names, third the graph for locations and lastly the graph recognizing nearly everything that is not included in any dictionary as graph for organizations. With this feature, the patterns recognized by the last graph are more pertinent than if applied alone. Nevertheless, the recognition is not perfect.

```
Please extract the named enities of the following text in a csv style format. Take also into
account dates and times:
"Chapitre I
 DANS LEQUEL PHILEAS FOGG ET PASSEPARTOUT S'ACCEPTENT RÉCIPROQUEMENT L'UN COMME MAÎTRE, L'AUTRE
COMME DOMESTIQUE
 En l'année 1872, la maison portant le numéro 7 de Saville-row, Burlington Gardens - maison dans
laquelle Sheridan mourut en 1814 - , était habitée par Phileas Fogg, esq., l'un des membres les
plus singuliers et les plus remarqués du Reform-Club de Londres, bien qu'il semblât prendre à tâche
de ne rien faire qui pût attirer l'attention.
 A l'un des plus grands orateurs qui honorent l'Angleterre, succédait donc ce Phileas Fogg,
personnage énigmatique, dont on ne savait rien, sinon que c'était un fort galant homme et l'un des
plus beaux gentlemen de la haute société anglaise.
 On disait qu'il ressemblait à Byron - par la tête, car il était irréprochable quant aux pieds - ,
mais un Byron à moustaches et à favoris, un Byron impassible, qui aurait vécu mille ans sans
vieillir.
 Anglais, à coup sûr, Phileas Fogg n'était peut-être pas Londonner.
 On ne l'avait jamais vu ni à la Bourse, ni à la Banque, ni dans aucun des comptoirs de la Cité.
 Ni les bassins ni les docks de Londres n'avaient jamais reçu un navire ayant pour armateur Phileas
Fogg.
 Ce gentleman ne figurait dans aucun comité d'administration.
 Son nom n'avait jamais retenti dans un collège d'avocats, ni au Temple, ni à Lincoln's-inn, ni à
Gray's-inn.
 Jamais il ne plaida ni à la Cour du chancelier, ni au Banc de la Reine, ni à l'Échiquier, ni en
Cour ecclésiastique.
 Il n'était ni industriel, ni négociant, ni marchand, ni agriculteur.
 Il ne faisait partie ni de l'Institution royale de la Grande-Bretagne, ni de l'Institution de
Londres, ni de l'Institution des Artisans, ni de l'Institution Russell, ni de l'Institution
littéraire de l'Ouest, ni de l'Institution du Droit, ni de cette Institution des Arts et des
Sciences réunis, qui est placée sous le patronage direct de Sa Gracieuse Majesté.
 Il n'appartenait enfin à aucune des nombreuses sociétés qui pullulent dans la capitale de
l'Angleterre, depuis la Société de l'Armonica jusqu'à la Société entomologique, fondée
principalement dans le but de détruire les insectes nuisibles.
 Phileas Fogg était membre du Reform-Club, et voilà tout.
 A qui s'étonnerait de ce qu'un gentleman aussi mystérieux comptât parmi les membres de cette
honorable association, on répondra qu'il passa sur la recommandation de MM. Baring frères, chez
lesquels il avait un crédit ouvert."
```

Figure 2: Prompt for ChatGPT version GPT-3.5 to extract Named Entities followed by 1/8th of the text.

2.3 ChatGPT

ChatGPT is a Large Language Model (LLM) developed by OpenAI using a decoder-only transformer. As the name indicates, it is a chat-bot designed to answer questions and to interact with user. Even if it was only trained to predict the following word, ChatGPT is able to analyse text and extract information from text. I used this ability to ask ChatGPT to return the Named Entities present in a text. Processing text is computationally expensive and the cost increases quadratically by input length. In addition, it has been proved that the information given in the middle of the input / context is less accessible than information at the beginning and at the end (paper, 2023). For this reason I decided to feed the model my corpus in various portions in order to see how the results change. For my tests I used the freely accessible version GPT-3.5 online. I provided it with always the same prompt followed by the text to look at between quotation marks (see figure 2). I asked the model to return the results in a csv-style format to ensure consistency in the notation. Nevertheless, the LLM was not always consistent including sometimes determiners or converting written numbers to actual numbers. I corrected these inconsistencies as the Named Entities stay the same.

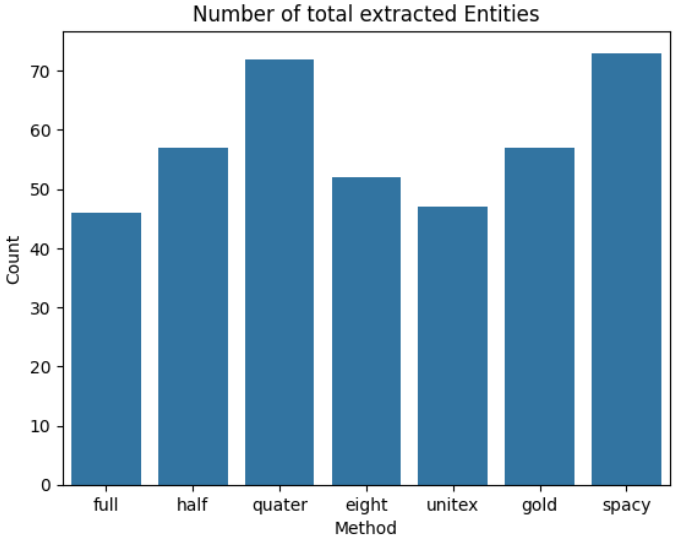

Figure 3: Total number of extracted named entities per extraction method.

2.4 SpaCy

SpaCy is a Python library that specializes in Natural Language Processing (NLP). It can perform several tasks from Part Of Speech (POS) tagging on to Named Entity Recognition (NER). The spaCy library uses pipelines to Neuronal Networks to perform the tasks at hand. There are pre-trained pipelines available for many languages but it is also possible to train a network from scratch if needed. For this test of NER I selected the corpus "fr_core_news_sm" which contains data of the French Sequoia Treebank and WikiNER. When initializing spaCy with this corpus, I already have a pre-trained NER pipeline for French. All what is left to do is the provide the model with my corpus and ask it to extract the Named Entities present in the corpus. My motivation to also test a Neuronal Network for this task is the following: NER is one of the less complex tasks in NLP. As it is possible achieve a high score with a rule based approach, a Neuronal Network would in theory perform as well as the rule based approach if not better. Even if LLMs can manage to extract those entities, the cost that it takes to process a request in such a large model is too much for as simple NLP task. A Neuronal Network is less computationally expensive and should be able to achieve as good results as the rule based extraction and ChatGPT.

2.5 Results

For the results, I did several comparisons. First of all, I wanted to know how many Named Entities were extracted for the document. As figure 3 shows,

spaCy and ChatGPT did extract the most entities when prompted with the corpus divided into four parts. It is followed by ChatGPT with the corpus divided in half and the gold labels. ChatGPT prompted with the whole corpus at once and Unitex found the least entities.
But these number do not show us, if the extracted entities are the Named Entities I searched for. For this reason, I will compare the data found by ChatGPT and Unitex to the hand-annotated gold data. Table 2 shows the percentage of the Named Entities in the gold annotation this is also present in the data that were extracted with the different methods described earlier. This comparison is called "recall". SpaCy has the lowest result. Out of the ChatGPT tests, it presents the lowest results when prompted with the full text. Out of the 57 extracted entities, 21 entities have also been extracted by this method. That represents 36 %. The highest result achieved ChatGPT when prompted with a quarter of the text: 41 out of 57 gold entities can be found in the extracted data. This is closely followed by the half text prompt and Unitex.

Method	Percentage Recall
Full	36%
Half	70%
Quarter	72%
Eighth	60%
Unitex	68%
SpaCy	35%

Method	Percentage Precision
Full	46%
Half	70%
Quarter	57%
Eighth	65%
Unitex	83%
SpaCy	27%

Table 2: Number of Named Entities of the gold annotation also present in the extracted entities by ChatGPT and Unitex (left table). Proportion of gold terms in extracted terms per method (right table).

This comparison helps us understand how many of the extracted terms are also gold terms. With this data, the method to split the corpus into four parts is most favorable one. But I only looked at the data from on side. With this comparison I checked the "silence" which means how many terms were not found.
In order to have a complete vision of the data, I also have to check the "noise" or how many terms have been extracted that do not belong to the field of Named Entities. This comparison is called "precision". For this comparison, I need to look at how many terms of to total terms extracted by the method are also in the gold data. As you can see in the right table of table (table 2), 82 % of the entities extracted by Unitex are also part of the gold labels. If I look at my winner from above, I see that only 57 % of the from ChatGPT with on fourth of the corpus extracted data is also in the gold data. This method was also the one that extracted the highest number of terms. This means that it found most of the gold annotated entities but also many other that are in fact no Named Entities. Therefore, it has a low score for silence but produced a high score in noise. The constant score of 70 % is produced by ChatGPT when

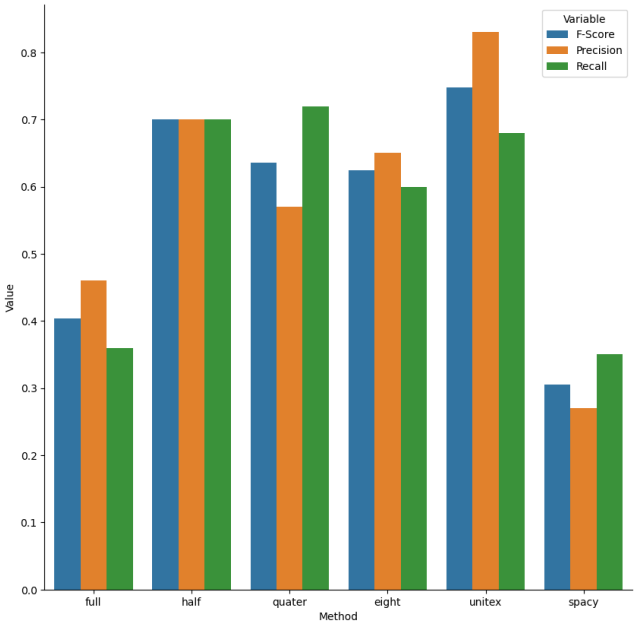

Figure 4: Graph showing precision, recall and f-score to compare

prompted with half of the text. SpaCy has the worst results with only 27 % of its extracted terms matching the gold data.
In order to better evaluate the overall score, I need the so called F-score which balances precision and recall. The formula that is used to calculate the F-score is the following:

$$F\text{–}score = 2 * \frac{precision * recall}{precision + recall} \tag{1}$$

This equation produces a score between 0 and 1 where a result close to 1 means a good performance and a result close to 0 a poor performance. As I can see in table 3, ChatGPT gives the poorest result overall when prompted with the whole text at once. The best result is achieved by Unitex what might be surprising given the rule-based nature of the program.

Method	Full text	Half	Quarter	Eighth	Unitex	SpaCy
F-Score	40%	70%	64%	62%	75%	30%

Table 3: F-Score per method. Unitex turns out to have the best performance and spaCy the worst.

We now have a closer look on the best performing method for ChatGPT. When prompted with around 2627 tokens or in my case with half of the corpus,

ChatGPT achieves a f-score of 70 %.
Figure 4 shows for each of the methods the f-score, the precision and the recall for better comparison. As I already remarked, the quarter and half method are almost equal in terms of recall. But what this figure shows more is that the precision for the quarter method is significantly lower than for the half method. This fact has an impact on the f-score making it with 70 % higher for the prompt including half of the text than for the prompt with a quarter of the text. It is remarkable that this method achieves consistent 70 % both in precision and recall. Given that I did zero-shot prompting which means not telling the model what Named Entities are and how to recognize them, the model performs very well. In this figure, we can also see that Unitex, the best performing model according the the f-score, has a 15 point discrepancy between precision and recall meaning that most of the found terms are in fact Named Entities but this method did not find as many as it should.

3 Translation

Named Entities are a source of difficulty on several domains. Translation is one of them. As Unitex as best extraction method is not capable of translation texts, I turn to the second best result for Named Entity extraction: ChatGPT prompted with around 2600 tokens. I asked the model to translate a 2050 token long extract of the corpus into English and then checked if the Named Entities have been translated correctly. It turns out that the model was capable of translating all Named Entities correctly. Before I continue with the evaluation it should be said that the story takes place in London in 1872. Therefore, the names and road names are mostly in English but general places were translated into French. The names, English and French names, were kept as they were as well as road names which were completely in English in the French corpus. Dates were adapted to American English format and places were translated into their original names.
If I consider the whole translation, it was a good translation in general. The LLM misunderstood only one word, the word "retarder" in French. This word can mean either "to be slow" or "to be delayed". In this context, the translation with "to be delayed" would be appropriate. But the model chose to translate the word with "to be slow" which does not make sense in the context.

3.1 Testing different translation tools

3.1.1 Corpus and Models

In this section, I are going to present different models and how they perform in translation. For this test, I used the first 20 examples of the challenge set from the 2017 published article "A Challenge Set Approach to Evaluating Machine Translation". The proposed translation is then evaluated by a common machine translation metric, the bleu score. It ranges from 0 to 100 with 0 being the

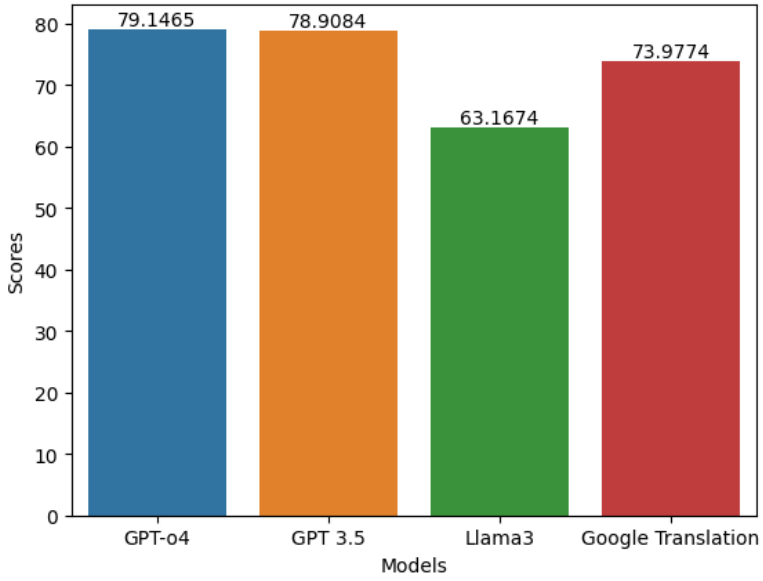

Figure 5: Bleu scores for the tested models. GPT-o4 turns out to perform best.

worst and 100 being the best possible result. The four models I tested were two versions of ChatGPT (GPT 3.5 and GPT-o4), the local running llama3 model implemented by Ollama and the translation pipeline for google translate.

3.1.2 Results

Figure 5 shows the obtained results. As a specifically for machine translation trained model, I expected Google Translate to perform best. The lowest score were expected form the locally running Llama3 model. For GPT models, I expected GPT-o4 to outperform GPT 3.5 only because the model is more recent.

I was right with my expectations about Llama3 achieving the lowest results. However, the other models performed differently from what I imagined. Even if GPT-o4 performs better than GPT 3.5, the scores of those two models are nearly identical. The trained machine translation model turned out to perform worse than the two LLM models by OpenAI.

4 UNITEX 4.0 alpha

Unitex is still being updated with more features with the aim to increase its relevance in the NLP field. Mid June, the new version Unitex 4.0 alpha has been made available. The aim of this section is to describe what has changed since the last stable version 3.3.

4.1 FST-Text

During the text's pre-treatment, there is a possibility to automatically construct a graph for each sentence of the corpus. As the dictionaries contain ambiguous entries, the sentence graph contains for some words several boxes with the same inflected word but coming from different lemmas. What has change since the last version is that the user can now select by hand the box that contains the real lemma of the sentence. This can be done by pressing Ctrl and do a left click on the box. It is also possible to add boxes or to delete useless ones. The software verifies automatically if the newly added box is a possible solution.

4.2 DELA

Another new feature is the possibility to edit the DELA dictionaries. This feature relies on a software developed by the university of Belgrade named LeXimir. It has been integrated into Unitex so that the predefined dictionaries could be modified either to correct mistakes or to add other entries by hand. This feature is accessible from the DELA menu by clicking on "edit DELA".

4.3 Search Graph Paths

The "Search Path Graph" feature already exists in the 3.3 version. It gives the user all the possible word combinations created by a certain graph. With the new version, it is now possible to also search by lexical mask (<N>/ <V>/ <A>/ ...). The program then prints all possible dictionary entries that match the lexical mask. It is worth noting that this features is only accessible if the graph only has one possible path.

4.4 Extended Local Grammars

The greatest innovation in this version the the principle of Extended Local Grammars (ELG). The difference between ELG and Local Grammars (LG) is, that ELG can access functions to transform or verify the matched pattern. A simple function included in the installation is the calculate function. The graph recognizes the numbers and the operation and the calculate function returns the result. It is possible to launch the ELGs by clicking on Text/Locate Pattern. If the option "Merge with input text" is selected, the result of the function is displayed along with the matched pattern in the text. With the "Replace matched sequences" option, only the returned result of the function will be displayed. It is worth noting that a returned boolean will determine if the match will be included in the results or not. So the function for a certain input returns false, the list of matched sequences will not include this particular input. Otherwise, if the result is true the input will be displayed amongst the other results.

The functions are defined in the programming language Lua. They are stored by default in the folder UnitexGramLab/App/elg in .upp files. In Info/Preferences/

Directories it is possible to change this default directory where the .upp files are stored. If a file contains more than one function, it has to be called in the graph with fileName.functionName(args). In many cases, the function takes an argument that has been recognized in the graph before. For this, the recognized sequence has to be stored in a input (red brackets) or output variable (blue brackets). The syntax to call the function and pass those variables as arguments is the following:

$$<E>/\$@file.function(\${variable})\$ \tag{2}$$

4.4.1 Created functions

The new version comes with a already written function which calculates the result for two numbers and an operator. It is called calculate.upp and can be called with:

$$\$@calculate(\${nb1}, \${nb2}, \${operator})\$ \tag{3}$$

The PhD thesis in which this functionality is described introduces the example of measuring the closeness of a word to another for dates. Only one part of the code was published in this thesis that is why I decided to complete it so that it can also be included in Unitex. In the available code, the similarity is measured by Levenshtein Algorithm. The implementation I chose is the one defined in this GitHub repository [4]. As in my case, I wanted to check a table of words, a for loop was added around the function so it served my purposes. I modified the published code, dividing it in two functions: one function for the months and one for the days. The result for the month function is shown in Listing 1.

In order to create a link from what has been studied above, my mission was to improve the created graphs for NER with functions. Firstly, I had to identify problems that could be resolved with those functions before secondly writing the functions and including them in my graphs.

The first problem I identified is tied to the recognition of locations. Unitex recognizes country and city adjectives as well as words for the inhabitants. As those are not part of NER, I want to eliminate them. I tried to do it without using ELG by searching for <N+Territoire> or <N+Ville> but with the ambiguities in the dictionaries, there was always an entry where the word I wanted to eliminate was defined as exactly what I searched for. The new graph, with the added function looks like figure 6. The variable c stores the sequences that match the search for countries and the function country at the end takes the found sequence as input and verifies if it ends with specific suffix. The code in Listing 2 defines the different suffixes with which the matched sequence is

```
function m_llike(input,threshold)
    -- input : string for the word to verify
    -- threshold : int
    -- returns true if perfect match or closest word
    -- returns false if difference greater than threshold

    threshold = tonumber(threshold)

    if input == nil then return true end

    dist, word = levenshtein(mois, input)

    if dist <= threshold then
        return word
    end

    return false
end
```

Listing 1: Function to find the closest match in month names.

ignored. If it has no such suffix, the sequence is considered a valid location. This approach will not show a perfect result as it does not on the one hand include all suffixes used for adjectives and on the other hand excludes words like "Vienne" because they finish with a defined suffix.

The second problem I identified relates to the category of Person and Organization. As mentioned above, the Organization recognition is difficult as no Organization name is included in the French Unitex dictionaries. The same difficulty exists for names that are not included in the given resources. I observed that some unknown names are recognized as such by the context e.g when preceded by a title. But when they are not in this specific context, they are included

Figure 6: Unitex graph that recognizes countries and eliminates country related words (like adjectives and inhabitants).

```
function country(c)

    -- function to eliminate adjectives and inhabitants of countries
    -- c : string was matched as a country or city
    -- returns boolean

    -- singluar masculin
    if (c:match"ais$" or c:match"ain$" or c:match"ien$") then
        return false
    end

    -- singular feminin
    if (c:match"aise$" or c:match"ainne$" or c:match"ienne$") then
     return false
     end

    -- plural masculin
    if (c:match"ains$" or c:match"iens$") then
                return false
    end

    -- plural feminin
    if (c:match"aises$" or c:match"ainnes$" or c:match"iennes$") then
        return false
    end

    return true

end
```

Listing 2: Country function to eliminate country related words.

in the Organization part or NEs. To prevent that a name is included in both parts, the idea was to store the matched names in a file so that the ELG for Organization recognition can verify if the matched sequence is already present in the names. If it is the case, the function will return false and the name will not be shown in the results. The original idea was to identify the names and then store them in a file with a function. Outside of Unitex, the code works as expected. But when applying the function at the end of the graph, it creates an error. Next, the "Save matched sequences" option that is available in "Locate Pattern" was tested. As the name indicates, it stores the matched sequences in a file. The problem with this option is, that is stores not only the sequence but the whole sentence in which the sequence was matched. Therefore, it has no

use for my purposes. Finally, the html file created by "Build concordances" was used. This file is not optimal because it includes links to the place in the text where the matched sequence is located. Therefore it contains a lot of useless information. But as this information is mostly composed of numbers, it does not interfere with my task. Listing 3 shows the code used to verify if a matched Organization has already been recognized as a name.

```
function new_element(e)

    -- checks if the matched sequence is already present as name
    -- e : string, the word to check
    -- returns boolean

    -- read the names stored in the file
    r_file = io.open(path, "r")
    dec_names = r_file:read("a")
    r_file:close()

    -- returns false if a match is found -> the element is not new
    if dec_names:lower():match(e:lower()) then
        return false
    end

    -- returns true if no match is found --> the element is new
    return true

end
```

Listing 3: Function checking if the sequence has already been matched.

A major downside of Unitex is the handling of unknown words. It is not possible to include all the words in the dictionaries because NE are often unique to each text. For this reason, I tried to write a function that checks if the word is available on Wiktionary and returns the properties, if it is found. What I did is the following: First, the function stores the content of the website with a get request. If the content is empty or includes a specific string, the function returns false. Otherwise it continues with the second step: Lua patterns are used to identify the properties of the word. And lastly, those properties are put into a valid Unitex format. While writing the function, several problems occurred. First of all, the Wiktionary pages are not in a coherent format which means that is was difficult to identify patterns for the pattern matching. In the current version of the function, there are three different patterns to recognize the properties that work for most pages. However, there are pages where the information cannot be found with the defined patterns. Secondly, the programming language Lua does not include Regular Expressions but uses something

similar but less expressive called Lua Pattern. It would have been useful to check if a sequence in a certain context is repeated, for example if there are several lines in a table. But Lua Patterns does not allow the check for repeated sequences as RegEx does. Therefore, the function returns every occurrence of a less restrictive pattern and then decides if the extracted information is useful or not by comparing the word to a pre-defined list of words. The last problem I was not able to solve. Outside of Unitex, the function returns the information as expected. However, when used in a Unitex graph, it is not able to make the get request in order to access the content of the website. As it is a long function, it is available in the annex at listing 6. I created a GitHub repository for the lua files (already named .upp as required for the use in Unitex): https://github.com/hannabri/unitex_lua

5 Prompt Engineering

A professor of the Université Gustave Eiffel created a learning platform named PLaTon where professors can prepare exercises and tests for their students. For the moment, this platform is only used by maths and computer science professors but they want to extent the use also to language professors. Therefore, the PLaTon team started a collaboration with a Spanish professor in order to understand her needs and integrate her ideas. The general idea is to generate exercises and / or sentences with a LLM with a defined vocabulary. The exercise type a fill the blank exercise but in the future, also sorting word exercises in a Duolingo style are planned. In order to tell the LLM what it should generate, the prompt has to be adapted. For the first tries I used ChatGPT 4o and asked it to generate sentences only with the provided words and to translate them into French and Spanish. The initial prompt was the following:

> Build sentences ONLY with the following words. The sentences should have a main clause and a propositional clause. Translate the sentences into ['French, Spanish'] in a python dictionary style:
>
> nouns: ['house', 'garden', 'dog', 'cat']
> verbs: ['eat', 'sleep', 'fight']
> adjectives: ['small', 'big', 'tired']
> prepositions: ['in', 'behind', 'in front of']
> determinants: ['a', 'the']
> conjunctions: ['because', 'when', 'that']
> pronouns: ['I', 'you', 'he', 'she', 'it', 'we', 'they']

This prompt worked mostly well even if ChatGPT used some words that were not included in the dictionary.
During the interaction with the language professor, it became clear that basic

grammar exercises would be helpful for her. That is why, the second prompt I created was for her to ask the model herself but also as a first prompt that could be refined. It looked the following way.

> Please generate a completion exercise in Spanish with **ser y estar** *in complex sentences*. Return the following parts:
> 1. Instruction
> 2. Exercise
> 3. Answers

The prompt above is adaptable. The bold part can be changed to any grammar input and the italic part defines the difficulty of the exercises by specifying complex or simple sentences. It works well for our purposes and those of the language professor.

What follows now is the implementation in PLaTon to avoid the copy / paste step from ChatGPT to PLaTon. Therefore, the PLaTon team wanted to use an API for a LLM that should run locally. One freely available tool to run those models locally is called Ollama and it is compatible with several different LLM models. The team decided to use Mistral as it is a French model. In addition, the model should not output the three parts from above but only the answers with the word to fill in in curly braces. For example a sentence to study the two Spanish verbs to be (ser and estar) should look like this where the conjugated verb of estar is between curly braces:

> El perro {está} en el parque.

In order to have the best prompt possible, I tried to use a python library called dspy with which it is possible to train prompts. I defined the Input(s) and Output and an evaluation function. The evaluation function checks only the format if the outputted sentence contains curly braces assuming that the generated sentence is grammatically correct (see Listing 4).

Further, I defined a couple of training and validation examples for the training process. The problem I encountered is that the sentences are not always grammatically correct and also that the model did understand what to put between curly braces. Sometimes it puts the name of the grammar input sometimes it puts nothing between curly braces and rewrites the words at the end of the output. So the output even of the trained prompt is not consistent and does not satisfy the defined format.
As another solution, I tried to use the Ollama API directly without training the prompt. Here, the same issue occurred that not all sentences were grammatically correct. The format issue persisted as well but in another form.

The LLM understood that it should put something between curly braces but it put the word following the target word between those braces. The last idea I tested was to used two prompts. The first generates the sentences for

```python
# Evaluation metric
def respect_format(example, pred, trace=None):

    pattern = "([\w\s,¿]*{[\w]*}[\w\s]*[\.\?\!])"
    matches = re.findall(pattern, pred.exercise)
    if len(matches) != 0:
            print(matches)
            return True

    return False
```

Listing 4: Evaluation metric to check the outputted format.

the exercises and the second one adds the curly brackets around the our target words. This approach worked the best for me. In simple sentences, I got the exact format I want at the end. The python code I wrote for this task is displayed in Listing 5. The link to the GitHub repository GlotGuru is: https://github.com/PlatonOrg/GlotGuru

```python
import ollama

def generate_prompt(grammar, language):

    # first prompt to generate the sentences
    first_prompt = f"""Generate 5 sentences in {language} with the grammar {grammar}
    in a python-style list called 'spanish_sentences'.
    Only output the {language} sentences!"""
    instruction = f"The assistant should only output the {language} sentences!."
    first_response = ollama.chat(model='mistral', messages=[
    {
        'role': 'system',
        'content': instruction,
        'role': 'user',
        'content' : first_prompt
    },
    ])

    answer = first_response['message']['content']
    print(answer)

    # second prompt to add curly braces around the words of the grammar
    second_prompt = f"""In the given sentences, put words corresponding to
    the grammar {grammar} between curly brackets.
    \nSentences: {answer}"""
    second_response = ollama.chat(model="mistral", messages=[
        {

        'role': 'user',
        'content': second_prompt
        }
    ])

    final_answer = second_response['message']['content']
    print(final_answer)
    return final_answer
```

Listing 5: Prompt only using Ollama API for pyhton.

References

[1] Manish Chablani. Gpt and other llm's: decoder only v/s encoder-decoder models? *Medium*, 2023.

[2] Pierre Isabelle, Colin Cherry, and George F. Foster. A challenge set approach to evaluating machine translation. *CoRR*, abs/1704.07431, 2017.

[3] Douglas Johnson, Rachel Goodman, J Patrinely, Cosby Stone, Eli Zimmerman, Rebecca Donald, Sam Chang, Sean Berkowitz, Avni Finn, Eiman Jahangir, Elizabeth Scoville, Tyler Reese, Debra Friedman, Julie Bastarache, Yuri van der Heijden, Jordan Wright, Nicholas Carter, Matthew Alexander, Jennifer Choe, Cody Chastain, John Zic, Sara Horst, Isik Turker, Rajiv Agarwal, Evan Osmundson, Kamran Idrees, Colleen Kieman, Chandrasekhar Padmanabhan, Christina Bailey, Cameron Schlegel, Lola Chambless, Mike Gibson, Travis Osterman, and Lee Wheless. Assessing the accuracy and reliability of ai-generated medical responses: An evaluation of the chat-gpt model. *National Institute of Health*, 2023.

[4] Matthew Kelly, 2012. https://gist.github.com/Badgerati/3261142.

[5] Nelson F. Liu1, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Standford*, 2023.

[6] Cristian Martinez. *Grammaires locales étendues : principes, mise en œuvre et applications pour l'extraction de l'information.* PhD thesis, Université Paris-Est, 2017.

6 Annex

6.1 Translation

French	English
Chapitre I	Chapter I
DANS LEQUEL PHILEAS FOGG ET PASSEPARTOUT S'ACCEPTENT RÉCIPROQUEMENT L'UN COMME MAÎTRE, L'AUTRE COMME DOMESTIQUE	IN WHICH PHILEAS FOGG AND PASSEPARTOUT MUTUALLY ACCEPT EACH OTHER, ONE AS MASTER, THE OTHER AS SERVANT
En l'année 1872, la maison portant le numéro 7 de Saville-row, Burlington Gardens - maison dans laquelle Sheridan mourut en 1814 - , était habitée par Phileas Fogg, esq., l'un des membres les plus singuliers et les plus remarqués du Reform-Club de Londres, bien qu'il semblât prendre à tâche de ne rien faire qui pût attirer l'attention.	In the year 1872, the house at number 7 Saville Row, Burlington Gardens—where Sheridan died in 1814—was inhabited by Phileas Fogg, Esq., one of the most singular and noticed members of the Reform Club of London, although he seemed to make a point of doing nothing that could attract attention.
A l'un des plus grands orateurs qui honorent l'Angleterre, succédait donc ce Phileas Fogg, personnage énigmatique, dont on ne savait rien, sinon que c'était un fort galant homme et l'un des plus beaux gentlemen de la haute société anglaise.	To one of the greatest orators who honored England, succeeded this Phileas Fogg, an enigmatic character, of whom nothing was known, except that he was a very gallant man and one of the finest gentlemen of high English society.
On disait qu'il ressemblait à Byron - par la tête, car il était irréprochable quant aux pieds - , mais un Byron à moustaches et à favoris, un Byron impassible, qui aurait vécu mille ans sans vieillir.	It was said that he resembled Byron—by the head, for he was impeccable as to his feet—but a Byron with a mustache and sideburns, an impassive Byron, who would have lived a thousand years without growing old.

Table 4: Extract of the translation made by ChatGPT (in English) and the original text (in French).

6.2 Web Search Function

```
local http = require("socket.http")

-- variables with category, number, genre etc.
word_categories = {"nom", "adjectif", "adverbe", "verbe"}
genre = {"féminin", "masculin"}
number = {"singulier", "pluriel"}
verb_tags = {
  ["indicatif présent"]= "P",
  ["indicatif imparfait"] = "I",
  ["subjonctif présent"] = "S",
  ["subjonctif imparfait"] = "T",
  ["impératif présent"] = "Y",
  ["conditionnel présent"] = "C",
  ["passé simple"] = "J",
  ["infinitif"] = "W",
  ["participe présent"] = "G",
  ["partifipe passé"] = "K",
  ["futur"] = "F"
}
person = {
  ["je"] = "1s",
  ["tu"] = "2s",
  ["il"] = "3s",
  ["elle"] = "3s",
  ["nous"] = "1p",
  ["vous"] = "2p",
  ["ils"] = "3p",
  ["elles"] = "3p",
}



function get_number(text, word)
  -- returns the possible number(s) for the word : table.
  -- text : string is the content of the get request
  -- word : string is the word that was given

  possible_number = {}
  final_nb = {}

  -- look for singular / plural information in a table
  sp_contenxt = '<table class="flextable flextable%-fr%-mfsp">[%s\n]*<tbody><tr>'
  sp_match = '([<%w>\n%s/]*)</tr>'
```

```
  sp = sp_context..sp_match

  for m in text:gmatch(sp) do

    for _,n in pairs(number) do
      if m:lower():match(n) then
        possible_number[#possible_number+1] = n
      end
    end
  end

  -- depending on which of the matches equals the given word,
  --the number is set to singular or plural or both (for invariable words)
  mf_context = '<td><b[%w<>%s]*lang="fr" class="lang%-fr"><[%w""=<>/#%s%p]->'
  mf_match = '(%w+)</[%w<>/]*>[%s\n]*</td>'
  mf = mf_context..mf_match

  i = 1
  match = 1
  for m in text:gmatch(mf) do
    -- print("Word match: "..m)
    if match > #possible_number then break
    elseif m == word then
      match = match+1
      final_nb[#final_nb+1] = number[i]:sub(1,1)
    else
      match = match+1
      i = i+1
    end
  end

  return final_nb

end



function get_genre(text)
  -- returns possible genres for the given word : table
  -- text : string is the content of the get request


  infos = "<span class=\"ligne%-de%-forme\"><i>([%wéçèê%s]*)</i></span>"
  pattern_gn = infos

  possible_genre = {}
```

```
  for g in text:gmatch(pattern_gn) do

    for i=1,#genre,1 do

      if g:match(genre[i]) then
        possible_genre[genre[i]] = true
      end

    end

  end

  return possible_genre

end


function get_word_category(text)
  -- returns possible word categories for the given word : table
  -- text : string is the content of the get request


  pattern_cat = 'id="toc%-([%l%u_%ç]*)%-sublist"'
  possible_categories = {}


  for w in text:gmatch(pattern_cat) do

    for _,c in pairs(word_categories) do
      pattern_match_endstring = "(".."c.."")$"

      if w:lower():match(c.."_") then
        possible_categories[c] = true
      elseif w:lower():match(pattern_match_endstring) then
        possible_categories[c] = true
      end

    end

  end

  return possible_categories

end
```

```
function get_verb_tags(text)
  -- returns the final code for the verbe : table
  -- text : string is the content of the website

  -- looks for tense and aspect information in a table,
  -- the first group is the aspect and the second the tense
  pattern_conj_1 = "<b>([%l%u]*)</b>[\n]?</td>[\n]?<td"
  pattern_conj_2 = " [%l%=\"#%u%s%d%:]*>[%s]*<b>(%u[%a%é]*)[%s]*</b>[\n]?</td>"
  -- looks for person information
  pattern_pers = "<td>([%a%é]*)"

  for f,t  in text:gmatch(pattern_conj) do

    code = code..":"..verb_tags[f:lower().." "..t:lower()]

    -- checks if a person is given, if yes it appends it to the existing code
    for p in text:gmatch(pattern_pers) do
      if person[p] ~= nil then
        code = code..":"..person[p]
      end
    end

  end
  return (code)

end


-- main function to look up the word, the one to use with unitex
function easy_lookUp(word)
  -- main function for the lookup
  -- returns the possible codes with Unitex standards
  -- word : string is the unknown word

  print("Look Up for the word "..word)

  path = "https://fr.wiktionary.org/wiki/"..word
  print("Website: "..path.."\n")

  content = http.request(path)

  -- handle the case where Wiktionary has no results for the word
  if content:match("Pas de résultat pour <b>"..word.."</b>") then
    print("We couldn't find the word on Wiktionary")
    return nil
  end
```

```
-- build the final code with the pattern:
-- First Letter of the category + ":" + first letter of genre and number
final_code = {}
--search for possible word category
possible_categories = get_word_category(content)

for pc,_ in pairs(possible_categories) do
  -- print("Category: "..pc)

  code = pc:upper():sub(1,1)

  if pc == "nom" or pc == "adjectif" then

    -- search for possible genre
    possible_genre = get_genre(content)

    -- search for possible number
    possible_number = get_number(content, word)

    -- Count the number of keys in possible_number
    local genre_number = 0
    for _ in pairs(possible_genre) do
      genre_number = genre_number + 1
    end

    if genre_number < 1 then
    -- if no genre is found
      final_code[#final_code+1] = code
      break
    else

      for g in pairs(possible_genre) do
        -- if no number is found
        if #possible_number <1 then final_code[#final_code+1] = code.."":""..g:sub(1,1)
        else

          for _,n in pairs(possible_number) do
            code = code.."":""..g:sub(1,1)..n
          end
        end
      end
    end

  final_code[#final_code+1] = code
```

```
    elseif pc == "verbe" then
      final_code[#final_code+1] = get_verb_tags(content)
    end
  end

  -- print all final codes
  print("\n")
  for c,j in pairs(final_code) do
    print(c.."\t"..j)
  end

  return final_code

end
```

Listing 6: Function that searches information on Wiktionary.