名古屋大学
NAGOYA UNIVERSITY

Internship at ALIFE-CORE

Activity report

Intern: Clément BARRIÈRE
Supervisor: Reiji SUZUKI
Duration: 7th of April - 10th of June 2025

Table of contents

Introduction

As a Bordeaux University computer science bachelor student, I had to achieve a two months internship to complete my second year. From the 4th of April to the 10th of June, I have been employed at ALIFE-CORE, artificial life laboratory at Nagoya's Graduate School of Informatics. This document is a factual description of my work during this internship.
I took part in research on the assessment of the recent Evolutionary Ecology of Words experiment model. The aim was either to identify the relevant parameters for open-endedness, and also to adapt the experiment's code and visualization tools to the new analysis requirements.

Research

To investigate the parameters influencing open-endedness, I chose to tackle a case study: the influence of prompt language on experiment results. I was still using the default parameters for the experiment, bartowski/gemma-2-9b-it-GGUF for the subject LLM, and sentence-transformer/all-MiniLM-L6-v2 for words embedding.

Hypothesis

A single word, translated into different languages, can have more or less different meanings or cultural representations. The theme of animal species is very demonstrative of this phenomenon. Taking legends and myths as exemples, the snake is a rather negative animal in Nordic and Christian folklore - symbolizing chaos and sin - but it is also a symbol of immortality and infinity in Buddhist, Mesoamerican and Australian cultures.
In the context of Evolutionary Ecology of Words, we were trying to maximize the emergence of novelties. These divergent representations could not only feed the experiment of rich folklore, but also provide a link between distant cultures. Consequently, it could renew the number of new possibilities and allow a sustained growth of emerging novelties along experiments.
The basic hypothesis of this study was therefore that a language should be imbued with cultural representations carried by the words that compose it.

Bias discovery

Four languages have been selected to compare experiment results: English (initial language of the experiment), Russian, Chinese, and French. Those languages implement different character mechanics, different ways to write, and come from more or less different cultures.
The first results showed dramatically different trajectory patterns. They were promising, as simply changing the language of the prompts produced something

different. By conducting these experiments again, we obtained the same constatations (cf. Fig. 1). Moreover, it is now clear that the model has difficulties dealing with other languages than English. It seems that every experiment with other languages tends to emerge at least English, or even more foreign languages. Which led to the development of multilingual ecosystems.

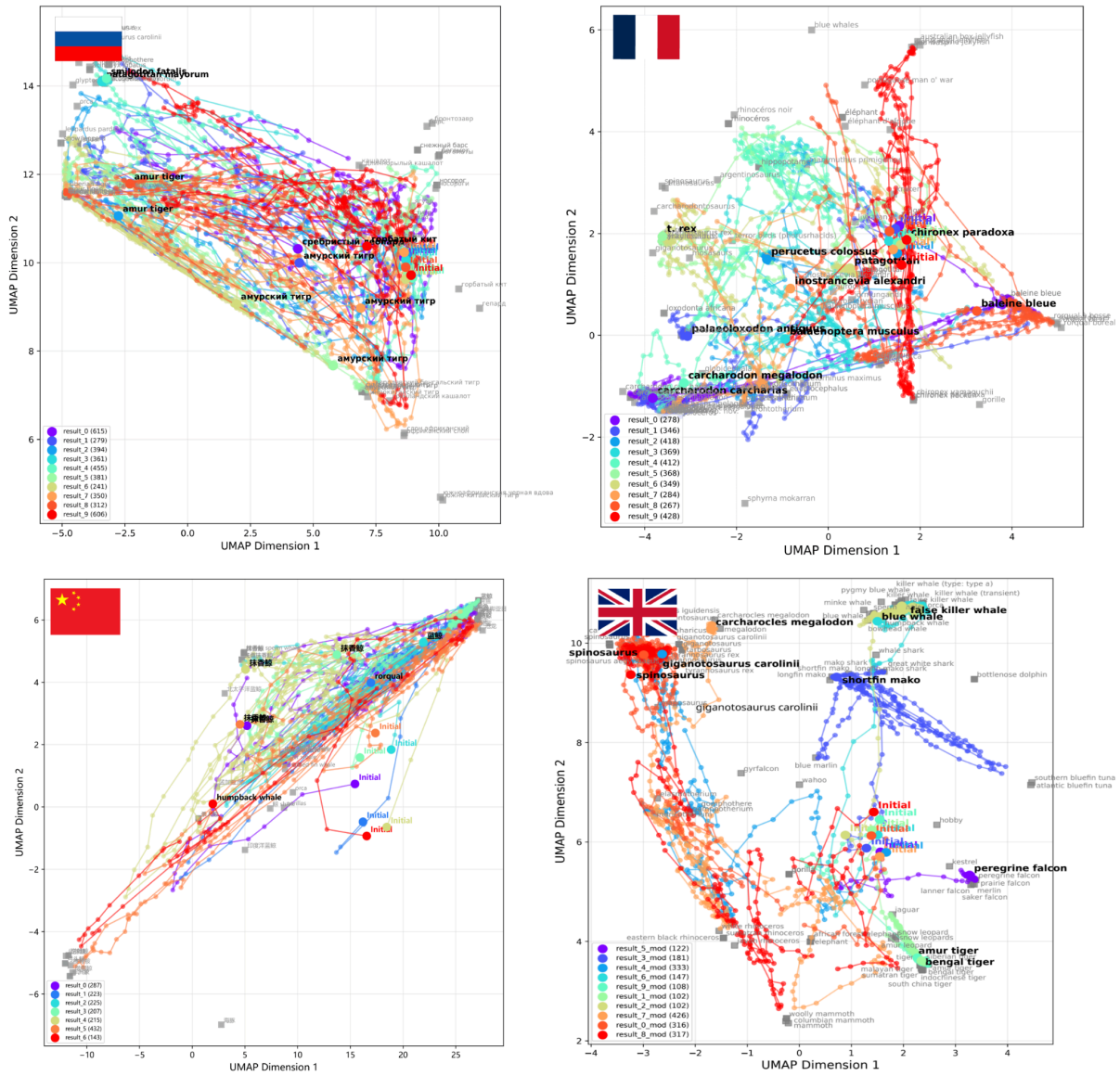

Fig. 1 - Results from experiments conducted with different languages.

Interestingly, we could not only locate semantic areas for species categories, but also areas relative to the languages themselves (cf. Fig. 2). If we look at a single case, we observe that two translations of a single word have different locations, even though the population at these generations were dominated by the same specie category (cf. Fig. 3). This phenomenon can also be observed on a larger scale, with the presence of entire species categories areas located according to the language. At this point, we assume that the language bias is visible: languages themselves have strong semantic values.

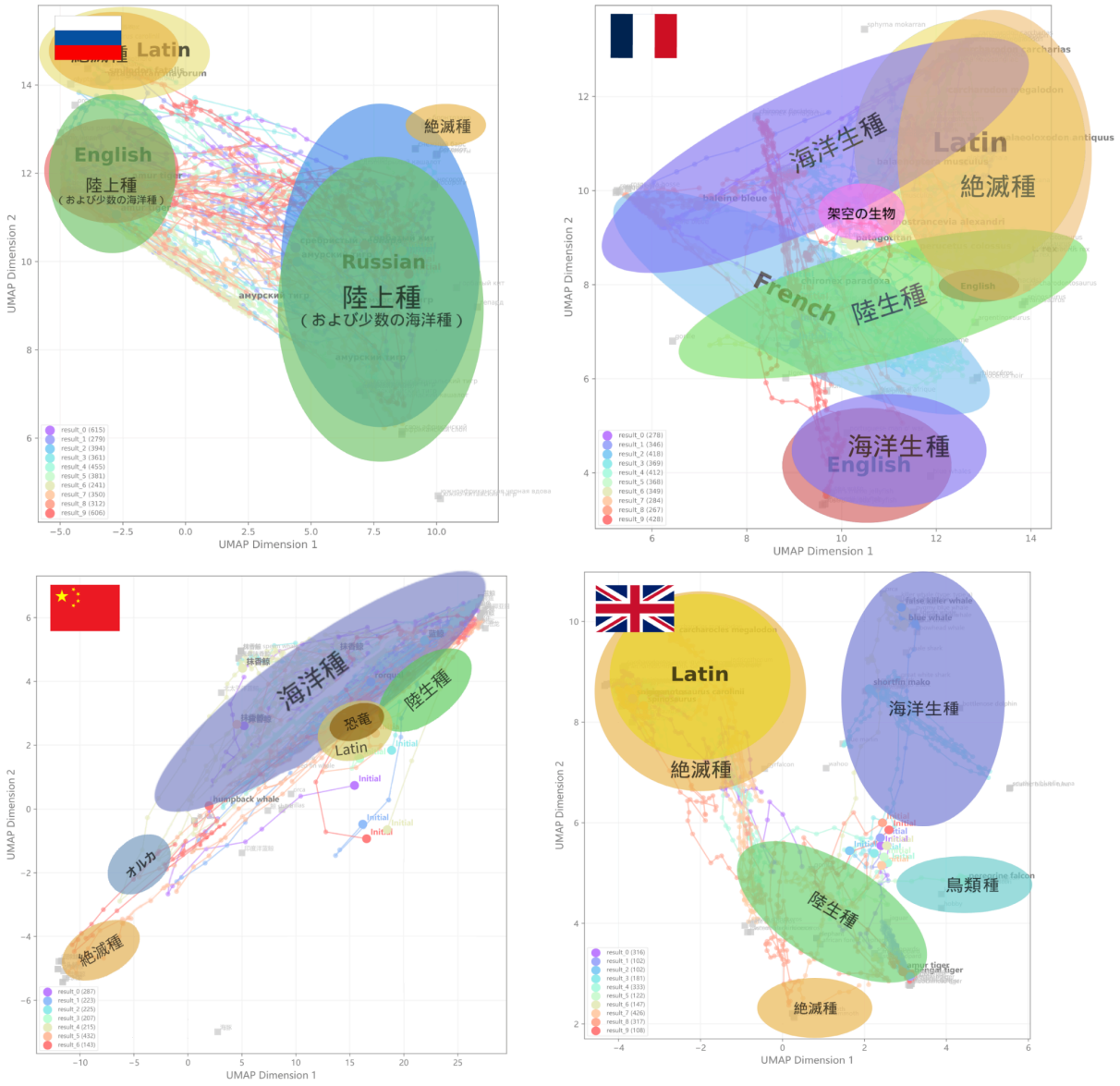

Fig. 2 - Languages semantic areas on UMAP graph.

2つのトップ10はそれほど違いません。

名前が単にフランス語または英語に翻訳されているだけです。

しかし、グラフはフランス語と英語の海洋種を分けるのに十分なほど大きな意味的な隔たりを示しているようです。

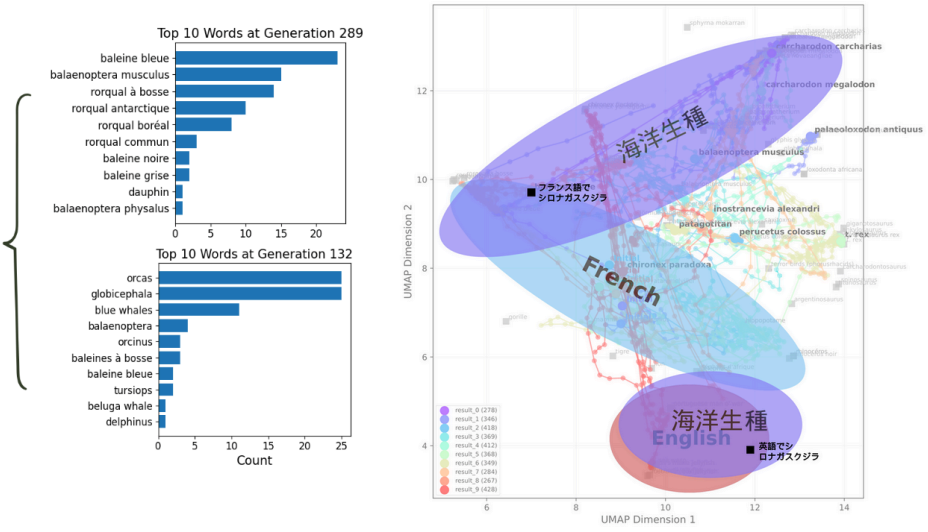

Fig. 3 - Same word, different locations of translations.

A good thing or a bad thing?

The question was then to tell whether this bias was a result jammer or not. Can simple translations be considered as relevant emerging novelties ? Confronted with this question, we decided to return to the basics: counting the emergencies. In other words, what influence do languages, and by extent multilingual ecosystems, have on the emergence of novelties ?

Firstly, experiments with languages other than English got better emergence rates (cf. Fig. 2). But these numbers are inevitably inflated by “simple translations”, as each of those experiments counted at least two languages, and we would like to find out if the interest goes beyond that. I therefore developed a new metric: the emergence score. It is the ratio of the number of emergencies by the number of mutations at a certain generation. In the context of novelty's emergence maximization, it reflects the efficiency of this generation. Keep turning in the same semantic field, the possibilities of emergence are exhausted; there aren't an infinite number of new words. A high score therefore means that the possibilities range is wide, and conversely, a low score *may* mean that there are not many words left to emerge. For a single trial, we can so correlate the evolution of this score over generations, with the behavior of its semantic trajectory (cf. Fig. 4). The conclusion of such an observation is that zig-zagging between languages helps to open ecosystems and to explore new horizons. Moreover, the French experiment showed a rich multilingual ecosystem, which led to promising and more abstract emergences, such as mythological species (cf. Fig. 2).

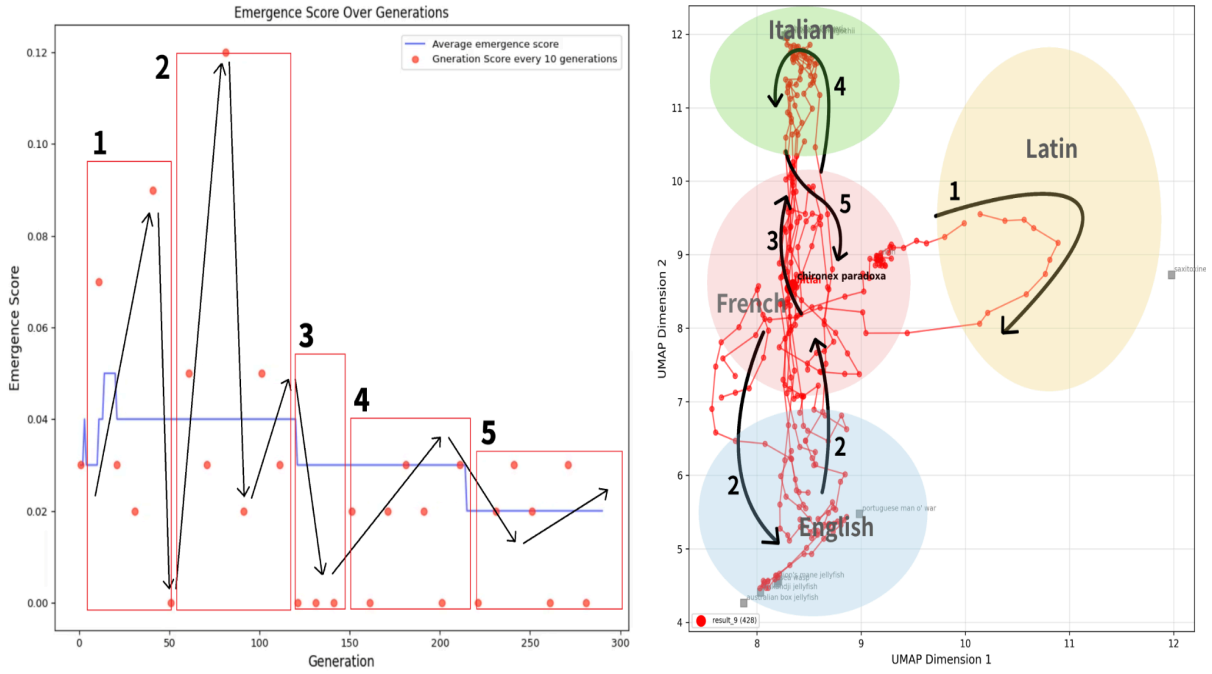

Fig. 4 - Emergence score and trajectory.

Specific characteristics of languages

In addition to the influence on emergencies, languages showed different and specific generation behaviors. In fact, each language seemed to follow patterns (cf. Fig. 5). For example, Russian has the best emergence rates but focuses on a single species category (terrestrial), which indicates a good exploitation of the semantic field. However, the major case stays the Latin one, specialized in extinct species and hard to out. We could also mention English, which seems to draw long and straight forward trajectories, indicating a good exploration of the surrounding fields. These constatations are in favor of the benefit to use multilingual ecosystems, as languages influence the LLM behavior, and could therefore leverage different generation dynamics to drive the populations.

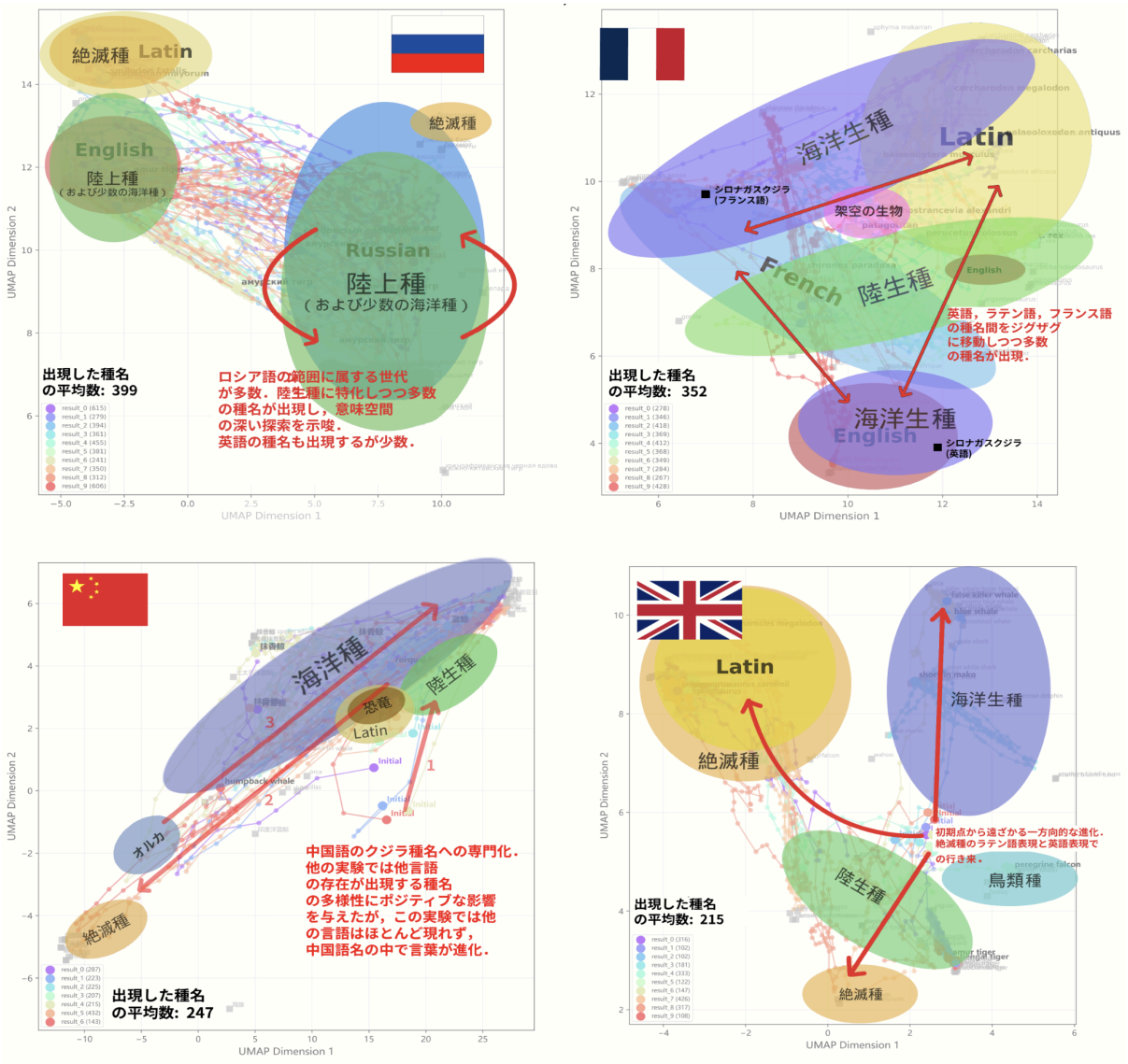

Fig. 5 - Languages and trajectory behavior.

Development

New way to integrate LLMs

At the beginning of the internship, I was asked to search for a more flexible way to integrate LLMs within the experiment model. The original code used the llama-cpp library, notably known for its optimisation and efficiency. But the current need wasn't to find the fastest tools but to be more flexible.

My attention therefore focused on the Transformers library from Hugging Face. It includes classes prefixed by 'Auto' that help resolve the configuration of models and tokenizer more or less automatically. The generation of responses uses a pipeline from this library to transfer parameters and components to a model very easily. In other words, the new LLM handling module for WEM allows the user to use a lot of parameters without the trouble of managing external requirements and system configuration. Everything goes by the 'LanguageModelHandler' class, and using a LLM now only requires to indicate the name of the wanted model on Hugging Face (cf. Fig. 6). However, it is important to note that specific model architecture may not be supported by Transformers 'Auto' classes yet; in such a case the module could need amends.

```
self._model: LanguageModelHandler = LanguageModelHandler(
    model_name=self.config["model"]["name"],
    log_event=self._log_event,
)


self._model.configure_model(
    local_offload= self.config["model"]["local_offload"],
    quantization= self.config["model"]["quantization"],
    temperature= self.config["model"]["temperature"],
    use_gpu= self.config["model"]["use_gpu"],
)


response = self._model.generate_response(
    prompt,
    min_new_tokens=n,
    max_new_tokens=self.config["model"]["max_tokens_per_word"]*n,
    rep_penalty=self.config["model"]["rep_penalty"],
)
```

Fig. 6 - Instantiate and use a LLM.

As for other critical parameters regarding this module, users can now also enable local offloading and quantization. When the model is too large to be loaded on the computation device and the RAM, the program can either use cache folders stored on the local disk and/or the model can be quantized in 8 or 4 bits. The computation device can rather be the GPU or the CPU, but local offload is only available when using GPU.

New dedicated application

More generally, I refactored the initial code into a WEM dedicated application. Every parameter from the experiment to the model and the workspace are manageable from a centralized JSON configuration file. This file is passed to the different actors of the experiment for them to use the parameters they respectively need.

The ‘simulation’ module allows to run a WEM based experiment (cf. Fig. 7):

- The ‘Simulation’ class is the main program of it, containing the loops, update methods, and the data on the ongoing experiment.
- The ‘Judge’ class is a wrapper for using the LLM, with all the actions that it is expected to take along the experiment, and the formatting of the responses.
- An instance of the ‘Agent’ class has an id, coords and a word. The class itself is used to keep track of the active agent and the state of the grid in which they evolve, thanks to static properties.

The app also includes a strong logging system. Each action taken along the experiment is logged (e.g trial, step, LLM working, warnings …) in a text file created into the experiment directory.
At the end of each step, the current data from the beginning of the experiment is also logged: experiment result, mutation history, competition history, and judgement history. This extended range of logged data allows not only to better keep track of ongoing experiments, but also to analyse the results in different ways. For instance, the mutation history is extremely useful for analysing the evolution of emergence possibilities and number of mutations, things that were not visible only from the result file.

Regarding details on application usage and other new parameters, I invite you to read the code documentation and the README file of the github repo.

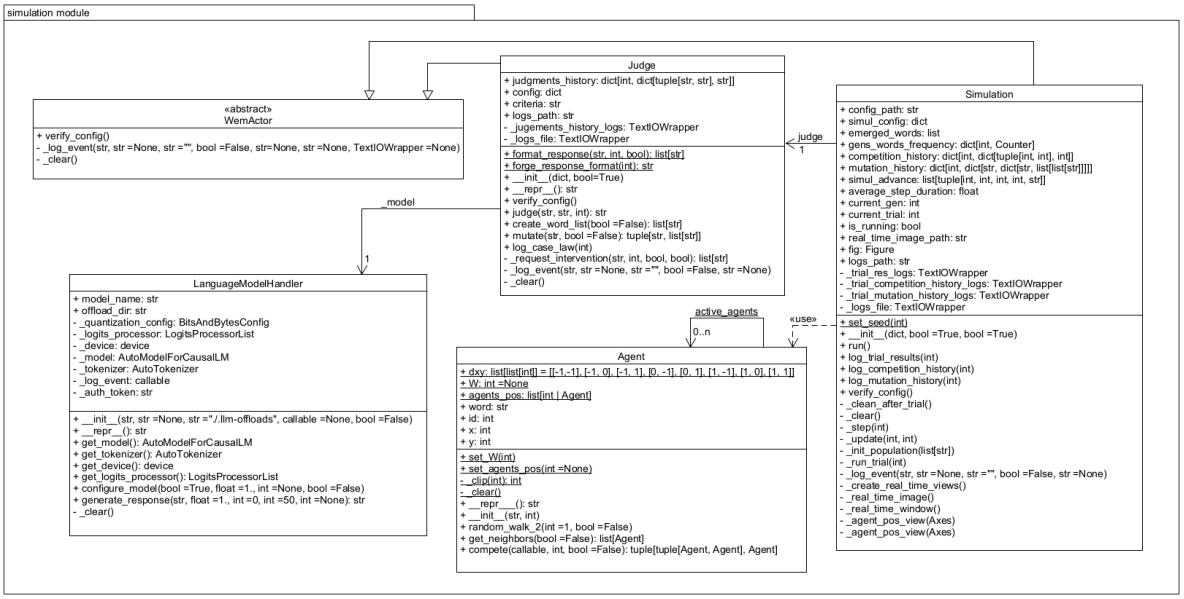

Fig. 7 - UML mockup of the ‘simulation’ module.

Visualization tools

The ‘makegraph’ module is the second part of the WEM dedicated application. It includes all the classes allowing to generate certain types of visuals from the resulting data.

Initially, the data was only used to draw the trajectory graph and the collection of animations to track the evolution of the top B names across experiments. This module is now counting 4 types of visuals (cf. Fig. 8):

- The trajectory analysis (‘ExpTrajectoryAnalyser’ class), including the trajectory graph for all or specific trial(s) and the animated version per trial.
- The top B analysis(‘ExpTopBAnalyser’ class), including the initial plots but that can be used now separately.
- The emergence analysis (‘ExpEmergenceAnalyser’ class), that only includes the computation and plot of the emergence score for now.
- The spatial analysis (‘ExpSpatialAnalyser’ class), including convex hull, percentiles contour areas, density plot, and numeric metrics, to quantify exploration, exploitation and redundancy behaviors.

Experiment data is now stored in dataclasses, that can be exported and passed to visual makers classes at instantiation, to reduce computation time and cost, or build upon read data otherwise.
The main pros for this new object oriented architecture is that we have now entire entities working on the computation of specific metrics and that the visuals can be created separately. To wit that the relative methods have a lot of parameters to allow

users to customize the plot, or even to export it. Moreover, the two modules of this application are communicating through result data files, to produce specialized metrics, which makes it a modular and adaptable system. It is also thought to be easily upgradable, as there are still many angles from which to study WEM.

For more details on the visuals that can be created, I invite you to read the code documentation and the README file of [the github repo](#).

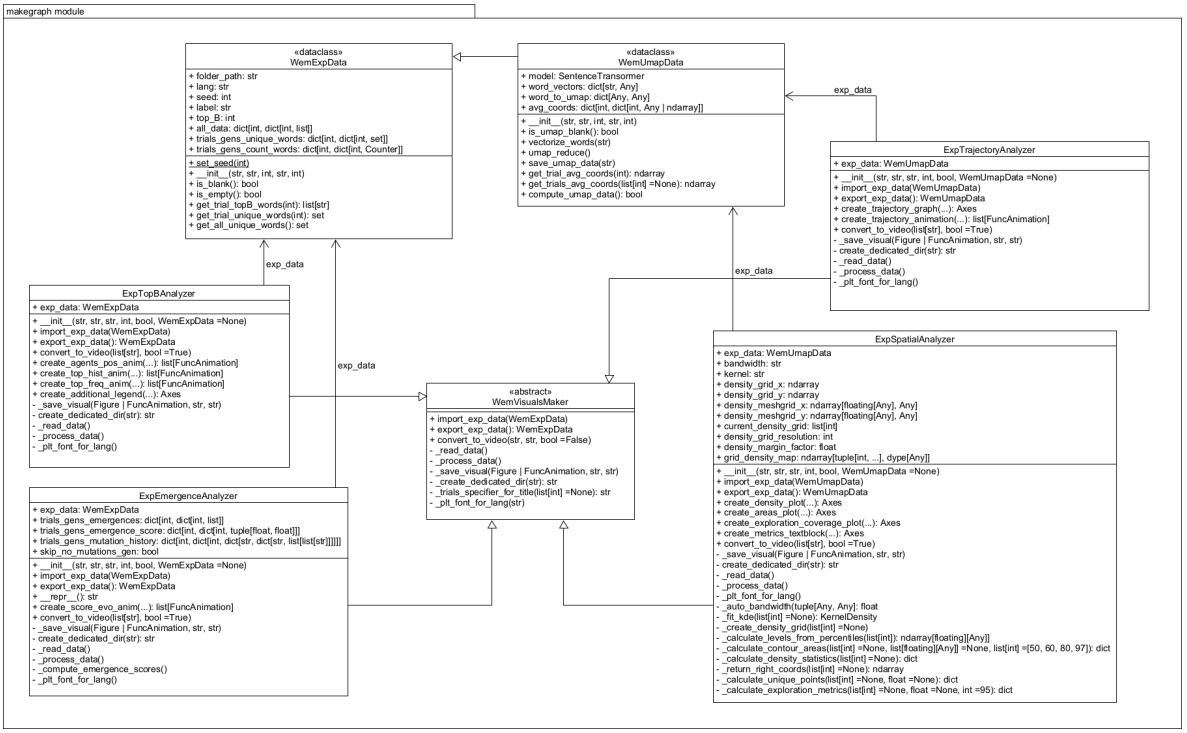

Fig. 8 - UML mockup of the ‘makegraph’ module.

Discussions and futur works

Discussions

Experiments were conducted with a limited number of languages. Other partially achieved experiments seemed to show the same results, but regarding the specific properties of languages, the observation of more dynamics could have been conducted. Additionally, the differences might be influenced by translation quality or specificities of LLM performance across different languages and not only by the language itself.
Before I left, I conducted experiments with other LLMs. They seem to have specific generations behaviors that could highlight other biases to explore. For instance, Gemma2 9b it GGUF often generated marine species, while Llama 9b it focused on african terrestrial species.

As for the application, the interfaces could be enhanced. Indeed the code is well modularized, dependencies aren't always correctly divided due to conception faults and the high flexibility that cost object permeability.
Moreover, the interactive graphic interface could not be implemented. The feature should have allowed users to edit the config file in a dedicated window, and even select the visuals to generate after the experiment.
The library choice for in-real-time views was also not the most relevant. Currently we are using matplotlib to regularly update an image file. But options like QtGraph exist and may fit better.

Future works

I personally think that the study of languages is not over yet, many interesting dynamics are still to be discovered. But one thing is sure: Evolutionary Ecology of Words is good at highlighting biases. A study that would compare LLMs generations tendencies could lead again to new adjustments to take into account more influencing dimensions.

The new parameter system is also a good opportunity to infer more and more abstraction. But in my opinion, the current visualization tools can not reveal too complex concepts yet. That is why I would find it interesting to develop a more or less automatic way to detect those. For language emergence we could use simple regex instructions, to compare alphabet and characters association. But for dynamics like epidemics, it would be more difficult. I propose to combine a 3D UMAP graph with the result analysis made by an hallucinations detection model (available on Hugging Face), or other text classification models. It could detect the difference between word vectors from a generation to another, and even labelize the detected emerged dimensions. On the graph, the Z axes could therefore represent those "major" emergencies. Such a new feature could fit in the 'ExpEmergenceAnalyser' class.

As for the application, the interactive graphic interface is an important feature. From my experience during the open doors of the laboratory, I must say that the current in situ observation tools don't fit well to share this work with uninitiated people. Samely, the real time views could be reimplemented with a better tool than matplotlib.
Anyways, the new architecture of the application allows us to develop new features and entire modules very easily. Therefore, it has the potential to become a real omni analysis and educative tool.