# **Path to a Wordnet… Fulfilling Fulfulde**

|  |
| --- |
| Have you ever gone to the grocery store, found the aisle for your item…and couldn’t find it on the rack?  Sometimes the whole rack is empty or out of stock. **Sometimes that one item isn’t available in that store…**  The following is a real (if dramatized) experience I had of that latter ‘absence’ while exploring Huggingface’s courses. |
| *As an illustration of the challenges of facing NLP with Underresourced languages*  Huggingface has an NLP "course" with various NLP lessons.  **one such lesson is**… "[Training a causal language model from scratch](https://huggingface.co/course/chapter7/6?fw=pt)"   * (Thoughts I have as I'm reading it):*So Cool!... Wait, how is this "from scratch" if it imports a pretrained tokenizer? ... Oh, okay, I see another earlier lesson describes how to make a tokenizer! I'll go look there...!*   **next Lesson** - "[Building a tokenizer, block by block](https://huggingface.co/course/chapter6/8?fw=pt)"   * (Begins reading the first paragraph)... *Awesome, it begins by immediately mentioning under-resourced languages, this is perfect!...* * (The second paragraph)...  "*Our first task will be to****gather lots of data****in that language in a training corpus." ....*   (sigh) I suppose there really is no such thing as a "free lunch"  **(A message to a friend that evening)**  But even in this, I'm grateful for another chance to see through the eyes of people who do not have the privileges I do - privileges I too easily take for granted in this field. Had I read this NLP course prior to today, I would not have noticed the very subtle hidden requirements. I would have walked away with the conviction that this was a great resource to give to my friends across Africa. And it still is a great resource, but it **really**misrepresents how hard small languages have to work just to reach the lowest "rung" on this long, long ladder of NLP. |

# Introduction

So. When embarking on an NLP project, one for an under resourced language… where do you start?

Huggingface? That was how I ended up with the experience above.

Result? All **roads lead to tokens**.

Any good (sub-word) **tokenizer requires stemming**.

And tokens/**stemming requires “wn.morphy”**

And morphy? That requires a wordnet.

But … Hmmm… So how do I make a wordnet?

# Dictionaries: Rejected by Neural Networks …but… maybe welcomed by WordNets?

Hi. I am a data science masters student at the University of Michigan. I have the privilege of growing up in Guinea West Africa, from 6 weeks old until ~ 13 years old. The Fula people of Guinea are an incredible people, and I am so grateful for the opportunity to know that, experience that truth.

A little about me, I’m an engineer by … well, inability to not engineer things, for better or worse. I started in Chemistry as my undergrad, and ended up Graduating with Chemical Engineering. And I started a Data Science program here at UofM…and since there’s no Data Engineering program ‘per se’ / ‘as yet’, I am unable to deny its how I’m wired.

So when considering a project to give back to be people of my youth, the Fula people, I began with the science approach to creating a NLP solution, and have ended up with a very engineering approach. So, here is the process I went through, lightly (heavily?) seasoned with the lessons I learned, one I hope will help others who walk down this fairly untraveled road…

And Data Science needs Data. So that’s where we’ll start.

## Data Options

On personal review of the data available to me or available on request, the text comes in four general categories

* Free text
* Translational Data (Parallel Translated Text, Dictionaries)
* Linguistic notes and academic study.

There are potential benefits and uses for all of these resources.

### Free (unstructured) Text

The NLP industry generally refers to any text or document as “unstructured” text.

This can feel insulting to someone like me who is actively “structuring” this article, being careful with spacing and sentence structure. This feels inaccurate to call “unstructured. But that dissonance captures what might be the reason NLP exists as a field: the structure we see and use in language is almost completely missed by normal data science tools and machine learning architectures. And so even with a huge investment of resources and time, we still cannot call an article like this “structured” … not even close.

From past work exploring Yoruba (another West African language), “unstructured text” is hugely valuable, but most of the data it contains is lost, even in languages like English that are privileged to be extensively digitized and modeled. This text can be used without any pre-existing data on the language, but in my experience the information gained can be hard to leverage.

For example, after training a [Word2Vec](https://www.tensorflow.org/tutorials/text/word2vec) model, the data learned (called “word embeddings” can be examined on a word-by-word basis. This data can then be clustered (grouped) with linear algebra like Non-Negative Matrix Factorization (NMF). A great deal of language information can be learned this way. But in my experience, this is most useful in experiencing how pre-existing labels and known groups of words connect and relate to each other. Examples might be parts of speech, positive/negative sentiment, or location words. Each known groups (labeled data) might interact differently when a NMF clustering is applied, and that information is valuable. So, if a wordnet existed, these sorts of analysis can be very effective… But this likely is not the best place to start creating a Wordnet.

Conclusion? Not a preferred source to start a WordNet

### Translational Data (Parallel Translated Text, Dictionaries)

With translational data, there seems to be two varieties of language resources: Multilingual Dictionaries and Translated Parallel Sentences. These two mediums represent one of (arguably "the") most dense forms of human labeled data. For the translator/writer first interprets the grammatical and lexical data (text), "projecting" that data into the semantic domain. This is then reprojected back into new lexical and grammatical domain, one governed by a separate set of rules and latent feature representations. As a result, the "data" is not only doubly encoded in the rich feature spaces of multiple languages, but a graph-to-graph style identity relationship is constructed between.

Other disciplines professionals dedicate themselves to may require equally complex mental models and transformations, yet few if any can be as richly encoded as language translations. As the data industry is grappling with the limits of big data, we much pursue reservoirs of deep knowledge is we seek to address deep challenges in society.

Parallel sentences are an extremely interesting field of study, and are the best data for training a Neural Machine Translation (NMT) model. However, the information learned by these models is difficult to interpret, and may not align with the sorts of information needed for a wordnet.

Translational Dictionaries are another interesting topic. However NMT industry has not yet found a reliable and effective way to use this data for Machine Learning. However these documents were the pinnacle of academic and linguistic research until digitization, and represent an extremely carefully labeled dataset. For readers familiar with Python, the fact Python has a data type called “dictionary” seems to suggest this type of data is accessible to machines.

This intuition is correct. And conveniently to this project, WordNets were “born” out of the digitization of dictionaries and matured with decades of layered research and data. So, this seems the most promising candidate for creating a WordNet.

Conclusion? Translation data enriches purely internal language data. Parallel translated text is complex, and data learned from it is difficult to interpret directly into rules or connections needed. However, Dictionaries are similar in structure to a WordNet. Therefor, translation dictionaries seem a good starting point

### Linguistic notes and academic study

Linguistic notes are arguable the most intensively labeled data available. However in my experience, despite having taken a linguistics course and years of cross-lingual study, I personally struggle to interpret these notes, and applying them in a scalable way seems well beyond my skills.

I suspect, however, that the experts who create these notes could use them to enrich a WordNet, and augment the data and links it contains.

Conclusion? Extremely useful after a WordNet is begun.

## Test Language: Pular

Pular (aka Fula, Fulfulde, Fulani, Peul, Pulaar, Adamawa, Masina) is one of the principal languages of the Sahel regions of West and Central Africa, spoken by over 65 million people. However, as are many African languages it is extremely under-resourced (minimal or missing the digital presence and language tools needed for technical support or initiatives). And while "Data" is often seen as the panacea for any challenge, modernization, Language as "data", as "code", has proven to be one of the most challenging frontiers for new data science work.

In response to the profound needs in this space in general, this project aims to not only develop digital NLP resources, but to explore and create open-source tools that might assist other developers or linguists. These efforts can be further scaled by working within existing communities, and contributing to existing solutions, which is often the path followed by many open-source projects.

Combining these goals into an overarching plan: The initial object will be developing a pipeline for processing language data sources, specifically Pular at this time. The data science outcomes will serve/supply as the unit tests or benchmarks for ongoing development of new methods, iterative improvement, as well as abstraction to create modular tools or functions. For the second objective will be the ongoing pursuit of lessons and insights learned while developing language solutions in West Africa.

Several multilingual dictionaries have been collected (why dictionaries? See below). The primary dictionary is a tri-lingual dictionary with over 10,000 entries. Each entry contains lexical and grammatical information (root word, part of speech, conjugation class, etc) as well as a "gloss" in both English and French (a gloss captures a distinct "sense" of its meaning, and can contain a sentence fragment, multi-word definition sufficient to encode the semantics and usage). Other Multilingual dictionaries have also been collected but are only bi-lingual to either French or English, not both.

# Inspirations for using Dictionaries

After deciding on what data to use, and specifically what data would benefit most if it had a pipeline, the next step is finding the dictionaries and determining a more detailed project plan.

Through connections in the [Masakhane](https://www.masakhane.io/) community, I learned of a small organization named [Kamusi](https://kamusi.org/), which is developing a new style of WordNet that is natively multilingual. Their expertise and years spent in Africa have been extremely helpful. And between Kamusi and Masakhane I learned the current pain-points in the global under-resourced translation community today. Kamusi is a non-profit translation organization that has created a variety of language tools while curating a multi-lingual database created initially from the multi-lingual wordnet, but iteratively refined and expanded by native speakers and linguists.

Kamusi shared an unpublished paper they had begun that used a tri lingual Pular dictionary (Pular to English and French) as a starting point. After parsing it, the paper describes the process of linking it to the Open Multi Lingual WordNet. With they permission, I decided to use their process as the prototype around which to begin building supporting tools.

# Reviewing Dictionary Structure

Kamusi provided two formats of the dictionary. A machine readable plain txt file, and a Microsoft DOC file with styled font and formatting. A reference PDF of the Dictionary may be readily found in online journals.

### Abbreviated Dataset “Card”:

**Source:**

* + Donald W. Osborn, David J. Dwyer, and Joseph I. Donohoe Jr. 1993. *A Fulfulde (Maasina)-English-French Lexicon: A Root-based Compilation Drawn from Extant Sources Followed by English-Fulfulde and French-Fulfulde Listings*. Michigan State University Press.

**Details:**

* + The dictionary is for Pular (the language of the West African Fula people (aka Fulani, Fulfulde))
  + It was published in 1993 and is directly accessible. It is ~400 pages (10,000+ entries)
  + Each entry is a lemma (fundamental “core” of a word) in Pular, for which there is a definition into both English and French. This is followed by an example Pular sentence translated directly into both English and French.
  + Each entry also contains additional Linguistic information such as Parts of Speech, gender, conjugation class, and sometimes mentions of highly related words.
  + The entries are structured exactingly to a specific set of programmatic rules, which are detailed explicitly in a 4-page instructional section at the beginning of the dictionary.
  + This allows for a robust ability to parse this document programmatically with a high degree of confidence.

**File Format**

**--TXT--**  Text, letter

Description automatically generated

**--MS DOC--** A screenshot of a computer

Description automatically generated with low confidence

**Examples of Content Variations**

Text, letter

Description automatically generated

**Annotated Markup of Dictionary Preface**

# Foundational Decisions for the Pipeline … DOC files for data labeling?

Visually, comparing the “machine readable” TXT to the “human readable” DOC file, I made an early decision for the parsing approach. I wanted to use the DOC version of the file, and leverage the font and styling the authors provided there, and relied on in their Preface to the document. Why?

Recall the problem of under resourced languages.

* Minimal labeled data is available
  + Limited numbers and access to people to provide labels
* Minimal data in general
* Little to no existing language processing
* Often, limited digital connectivity options

That’s a lot of limits. And most of those are contrary to the direction of modern Tech which focuses on big data and cloud native methods.

But conveniently, word processing applications are nearly universally available, limited mainly by economics and community demographics. And very clearly from the detailed Preface the authors created, Word Processors are routinely used as data labeling tools.

This realization was huge for me personally. Under resourced language communities must rely on existing infrastructure and distribution channels. Furthermore, given the limited data available, it is a priority to make use of all the data available. And so, though it is not the “techy” or “cool” thing to do… I want to build a pipeline around modern Word Processing standards is the most scalable solution, most direct solution, and one of the most expressive solutions to providing tools for language communities.

To elaborate briefly on the impacts on this dictionary in particular:

The article’s approach to parsing the file was iterating over the txt lines, and some regex, accumulate each line into the previous “lemma” word entry, or else begin a new entry. Given the size of the document, this elegant method may not handle well exceptions or formatting issues in the document. Furthermore, relying just on regex, some of the content of the document may not be reliably extractable.

However the docx file has a large amount of metadata in the styles. Leveraging this to augment the regex/structural parsing of the object should be more robust. A quick review of dictionaries online shows visual font formatting is nearly universal in any language. Including these in the parser should make the pipeline more adaptable to other dictionaries with less consistent structure but strong visual cues easy for a person to interpret with styles.

# Making a Parser … and iterating… a lot

This project has taken many iterations and will certainly take many more. But with each iteration, even if it was a dead end, we learn something. There are many pitfalls in software development, and I am sure I haven’t yet plumbed the depths, or even stepped in half of them… But I like to remember that I have it better, MUCH better, than a Machine Learning routing. Because (hopefully!) I don’t need 500-10,000 Epochs before having something moderately presentable!

Another algorithm I am encouraged by is Gradient Decent (though in my case I’d *prefer* think of it as Gradient *Ascent*!). Gradient decent is an algorithm that iteratively “walks” through a multi-dimensional “surface” and can be tuned to find the minimum point in that space. These are used, for example, to find the parameters of a model that minimize some loss function. And it is the same in software development and data science: we choose our next steps as well as we can with the information. And so long as we keep our minds open, as well as continue to make reasonable next steps, we’ll keep making progress.

Now. To specifics.

I have captured several “phases” of this project, which represent critical decisions or outcomes.

* Each will list the Objective I began that Phase with.
* Each will have summaries of the actions taken.
* Each will include at least one representative image of code. As the images are necessarily small, they may be animated to highlight the structure, such as collapsing or expanding functions or classes. A future embedded web-version of this article will allow zooming and more crafted interactions. I will rely on the Github repo to convey the details that may be lost in this medium.
* Each will include lessons I learned that shaped my decision making process.

## A screenshot of a computer Description automatically generated with medium confidencePhase 1

### Objectives

* Make a Parser similar to routine used in the unpublished Kamusi Article
* Get a sense of content of the Dictionary to inform decisions
* Get a final benchmark to compare with the Article

### Actions

 This was my first exploration of the ‘python-docx’ and ‘docx’ packages, and the learning curve was surprising.

These docx packages are build on top of Microsoft API’s for its applications and files, and they attempt to directly align with the official documentation Microsoft provides. However Microsoft seems to inconsistently publish details on these, and many ‘current’ documents still date from over a decade ago. I therefore have even more respect for the volunteer developers who created the ‘docx’ and ‘python-docx’ packages.

But there are challenges with these packages.

The current version of ‘docx’ seems to be incompletely updated from python 2 to python 3. The ‘python-docx’ package modifies and wraps portions the ‘docx’ package to mitigate these conflicts. But even so, getting these installed and functioning is a challenge, even after doing it 3 times on 3 different virtual machines.

### Lessons – *Hint Hint*… Use Type Hints

It was during this process I really saw the value of VS\_Code’s completion hints. With the official Intellisense extension, it is straightforward to explore the data structure of an arbitrary package. Typing an objects name followed by a period pops up a window showing the methods and attributes declared in that object types source code.

Text

Description automatically generated

The ‘document’ object has many attributes, but the ‘.paragraphs’ generates a list of ‘paragraph’ objects. The following plot is the attribute data structure of the paragraph object.

Diagram

Description automatically generated

It’s worth pointing out that the “main” pieces of DOCX objects is the ‘paragraph’ object (contains all the above) and the ‘run’ object underneath it. The paragraph object is defined by lines, specifically a ‘carriage return’ or “\n” will mark the end of a paragraph and beginning of a new one. These contain indentation information as seen vaguely in the paragraph format attribute above.

Runs are a little different. These all always associated within a paragraph, and one paragraph only. The text of the paragraph is broken into ‘runs’ based on several conditions. First, any contiguous string of text that all has the EXACT same format and style in all regards…that’s a run. Runs can also be broken up by characters like commas and periods. This makes Runs extremely useful for feature extraction, since a ‘run’ object will contain only text that is identically styled. This allows the **BOLD** text in a line to be directly extracted, and it will already be “pre-separated” into a run.

In addition to learning the DOCX data structure, I learned its limitations. The DOCX objects currently don’t have certain core functionality I have previous expected. They cannot be hashed and are difficult to serialize. Any given attribute my not be present, and the objects to not always conform to expected content.

Largely, these variations come from the underlying data being GIVEN to the package. But since the data for Pular is so rare, I cannot afford to “dropna()” like were used to doing in pandas. The exceptions are data cleaning issues need to be handled, not just omitted.

And so the final realization regarded the monolithic structure of the dictionary based Dictionary parser (geez this get confusing doesn’t it?). The “combined” nature of the code meant logically different processes were comingled. And so changing one was hrd to do without breaking the other.

## Phase 2

### Objectives

* Overcome the data cleaning issues by revealing the problems
* Have a system that made handling the problems easier
* Separate code into logical sections for better maintenance

### Actions

This being python, I decided to try Dataclasses. These are objects with attributes and methods that allow them to act much like string or lists: able to be passed around, as well has have type-specific functions/methods executed on them.

But for this to scale, the design of the dataclasses needs to be carefully done. I saw two approaches here. First was to create a class that describes a dictionary entry as a whole. The appeal to that is, in theory, I could pass the DOC object directly to the class and the dictionary objects, all grouped by lemma, would just, appear.

If that sounds too good to be true… it is. Or at least for me. I realized quickly that I needed to handle the complexity much lower if I wanted to actually get away from the monolithic structure I already had created.

So I decided to create dataclasses that “wrapped” the ‘python-docx’ paragraph and run objects. This class could have strict rules about attributes and I would be able to create guarantees of certain features.

This problem now is highly HIGHLY analogous to designing robust API handling in the Web, and other locations with few guarantees of what data will be provided. When researching “Data Schema Validation”, the main class that is mentioned is call “Pydantic”. This augments pythons Dataclasses, and adds a number of common needs, including validating data input. I need to pass a run object and not just know it’s a run object, I need to know that the ‘run.font.size’ is not “None” so I can safely extract “run.font.size.pt”. And this is where Pydantic excels.

So I created several wrapping Pydantic dataclasses to handle the raw ‘docx’ objects and control for exceptions.

Text

Description automatically generated

The above image shows 3 classes. A Paragraph class, a Run class, and a class for lists of Runs within pargraphs. Each class has declared attributes with required types. If they receive a value that is not the correct time, the class will raise an error.

That feels pretty harsh. But this is where Pydantic really shines: the ability to write custom validators that can handle exceptions, and make more complex checks, such as consistency between two fields that ‘should’ agree, etc. The “zero trust” default is wonderful (with code linters and highlighters!).

### Lessons - Data structures with Accountability enables… Outcome-informed Decisions?

Static typing, compilers, don’t normally give most people good feelings. But with the rise of dynamic code linters and tools like “Intellisense”, choosing self-enforced accountability might be the best way to productivity and effective reliable outcomes.

This was game changing for me, because it allows me to immediately “see” edge cases my code would have failed from, many of which I might never have realized. But with immediate code highlighting, this can be relied on.

## Phase 3-5

### Objectives

The objectives over these phases are similar. I had the dataclasses. Now I needed to begin to

* create methods and routines to parse the DOC file,
* handle exceptions
* begin creating order, then structure, within the data.

### Actions

My first ‘action’ was to add a 4th class to pull together my requirements of the three other Pydantic classes, and create a aggregated “paragraph” object that I could leverage for methods and functions, etc.

This is where things stopped being as…pretty. But I was able to add a series of class methods to the 4th DOCX-Pydantic class. The validator (collapsed below, but present in the github) grew fairly complex and nuanced. And by the 4th phase I added Logging functionality. This allowed me to port the data exceptions for a 400page document into a log. Adding a handler for the logger within the class allowed each exception to have auto populated text explanations, etc.

Text

Description automatically generated

To “run” this I created a general function that looped over the paragraphs in the document and evoked the Pydantic DOCX classes.

Early on I added the ability to pass configuration objects into the classes since modularity was a core goal of this pipeline project. The logger too could take a configuration file/dictionary, controlling what was logged, what was raised, and how. These functions are not yet as versatile as they will need to be, but that will come as I apply more documents through this fledgling package.

Text

Description automatically generated

### Lessons –Hear “optimization”? Think “trade-offs and sacrifices”

Choices of data structure are critical. And there seem to be endless optimizations.

My experience is that at a given skill level and experience, its normal to have a trade-off between flexibility, and clarity.

### Lessons -- Simplicity != brevity

Definitionally, ‘expressive brevity’ == Poetry… And that’s an art.

The art can be learned, but it is long in coming.

And poetry anybody can understand? That’s a masterpiece… just as with code, apparently.

## Phase 6

### Objectives

* Up until this point, I had be evoking the classes by iterating through the paragraphs in a for loop. This was effective, but as more was added the time slowly grew. It was rarely over 2 minutes for the whole document, but I wanted this to be robust for future scale.
* Add parallelization
* Add API to begin automating processes

### Actions

To solve the parallelization, there were a variety of options, none of which seemed easy or something I could implement safely without risking the guarantees of reliability I’d maintained in the package so far. However I realized Pandas has a truly robust parallelization routine, AND it has a great API most people are familiar with. And so if I could get the package to use Pandas as a ‘front end’ for the Pydantic-docx objects, I’d solve all my objectives at once!

I looked into registering a new dataclass to pandas.dtypes. That is DEFINITELY a dream now, but it was out of reach for a project as young as this, and I’d need a lot more infrastructure before being able to achieve that.

However, if I handed references to my object to Pandas, and used the df.Apply() function, I could get much of the benefits of parallelization and API with a fraction of the effort.

To test this utility, I needed a way to benchmarking the output, not just refining the experience (expressivity vs effectiveness, to borrow a visualization phase).

So after creating the pandas pipeline for processing the data, I created a 5th Pydantic Dataclass that contains all the Pular Dictionary specific structure, requirements, and data features. Its… a beast. I really hope there’s an easier way to do it, a simpler way. But the redeeming quality of Pydantic is the final processed outcome is very enjoyable regardless (mostly) of the pain of validating it.

The Following images show the Pular Dictionary Pydantic Dataclass

Text

Description automatically generatedThis first image above shows the (long) list of Attributes that each dictionary entry in the Pular Dictionary needs to be able to account for and record. Currently, not all of these have been implemented, specifically splitting the glosses into multiple sense for each word, as well as tracing those to their literary / dialect source where applicable.

But to populate those attributes, I need to validate the input DOCX-Pydantic paragraph dataclass objects. Those are only proven to be true representations of the document, not inherently any other structure. So, using the pandas pipeline I mentioned, each paragraph can be grouped together into discrete entries. These “proposed” Pular dictionary entries contain the DOCX paragraphs. But if we want to trust these entries enough to build a wordnet from them, they need to be validated and auditable.

The following image shows the abbreviated validation function for this object (somewhat abbreviated by collapsing sections for handling the components of the input.

Text

Description automatically generatedThe above image shows the overview of the validation code for this object (full code available on Github).

Included below this validation are methods for this dataclass. One of the core benefits of a python Class is the ability to manipulate it and maintain and ongoing representation of it. This removes the need to parse all the features of the object, and only requires enough to prove this is a valid entry. After that, subsequent functions and methods can be called to further refine the entries, handle edge cases, etc.

### Lessons – There is no substitute for experience. So just…do, begin, try anyways

Implementing this is still under way, but the process is established, and fully working. The remaining work to finish this phase will be abstracting the pandas processing further away from the raw classes.

The current experience, of running a pandas pipeline over the document, checking for data cleaning, handling exceptions… this takes a full jupyter notebook with a few dozen cells. This is fairly ‘standard’ data science processing, but I’d like to further simplify it, and that will likely take a fair amount of infrastructure.

However, as a final outcome, being able to take a very niche and domain specific task like parsing a multilingual dictionary into a series of machine readable objects… and bring that out to a mostly conventional data processing experience? I feel that’s a pretty impressive result.

Granted, since I wrote it, and since parsing a dictionary no longer seems foreign to me, my personal perspective is going to be biased. So as I go further, or even BEFORE I go futher, I want to begin asking for people to use the tool. This will help solve bugs and usability issues. But more importantly, if I want this to become a python package (spoiler alert!), I really want to begin this project focusing on solving other peoples problems, not what I THINK other peoples problems are. So. That’s the true next step. And until then, this phase will wait.

Reflections after the Project Phases:

No substitute for experience. Being unable to say something simply usually means I don't understand it well enough yet.

To truly be able to reduce the complexity and the amount of controls and code required, will need several large things

1: Query engine and DAG-style control of process steps.

2: Adaptive routines able to handle whats given to them.

3: robust but abstract foundational data objects

4: feature rich aggregate classes.

I probably aggregated too early, having the largest aggregate really not be any more than a paragraph, but also not directly aligned with the true paragraph object it could have wrapped.

Multiple inheritance of objects and parent and child classes is something I've yet to build a solution with at scale, and that likely will have a couple specific use-cases going forward.

These would allow a lot less code for the foundational objects, and by having my workhorse classes be much more abstracted, I will not be tied to limits of the basic data structure.

The concept of "future-state requirements" is a powerful one. Requirements we will need in the future we don't even know we need yet, or do, but its not applicable until we solve the current problems. When working with a client, the needs that are expressed are critical. But often we need help seeing the things we will need next once our current reality is improve. The things we really want, but can't actually get to right now or even really know/believe we ever can ask for that exact thing.

Its odd being the driver of that, knowing I will not be able to ask better questions until I have learned more and gotten to a new position. There's no free lunch, and jumping to the end doesn't actually get you to the end. And when we try, when I try, its all to easy to spend time solving problems that don't exist yet, and may never exist. The challenge of decisions with limited information.

# Conclusions and Next Steps

So. To bring this back to the original objective.

But how to collect more data?

If I can prioritize certain words for extra labelling, or flag regions of the wordnet we want coverage in, how do I get more data, how can I crowdsource these tasks?

Kamusi is working on apps and tools to help crowdsourcing. But so far these seem to need the cloudnative architecture. Most of my contacts in Guinea have limited data, and its expensive to use. That friction poses a challenge.

But also this is a "dead end" usecase in that its not value add or a service people gain from in the process of training.

So I propose a DOCX related tool/plugin. Also, having python code people can run is less helpful since the hurdles to get there are so high. How can I bring tools to the everyman, and how can I get those tools to provide a service while having the option to allow training, to give control of those features?

If training can be incidental to the use, compliance and support will be easier to get

DOCX is a richer data format, and can more easily serve as a smart platform that can capture even more context and usage information, while having the same increase in utility options for users. Since this product is vertically integrated within microsoft, distribution, engineering, infrastructure, and scale are all largely handled by that ecosystem.

A .NET solution would allow the optimizations of MSWord search instead of creating a parallelizable search routine from scratch that would compete with MSWord for resources on the offline device.

Regex