**DOCS**

**Introduction**

In the beginning there was a Tensor. This is a documentation of ongoing study on different models and methodologies to obtain solutions for problems that relate to areas of artificial intelligence and machine learning in the team "Safety net". This documentation presently, covers our journey of understanding the problems at hand alongside gaining knowledge on the foundations of Artificial intelligence and machine learning to build a solution down the line. We are excited to embark on this journey, as this is a fresh start for us into the humungous world of Artificial Intelligence and machine learning. There is also an obvious feeling of making mistakes along the way, hence it is suggested to take this read with a pinch of salt. Have fun!

**Notes:**

1. We use the words application and the model interchangeably in this documentation. By "application" we mean the whole solution that may consist of different layers and different models/ embeddings / tools.

**Chapter 1: Problems at hand**

Below table contains overview of problems at hand.

|  |  |
| --- | --- |
| **Problem statements** | **Description** |
| 1. Automatically suggest solutions to issues. | --to be updated soon-- |
| 1. Automatically risk evaluate and suggest improvements for chemical module.( The chemical module) | Using environmental aspects of where and how a chemical is stored and other risks involving chemical hazards in proximity as the input, goal is to build a model that produces warning instructions.  Ex: If Sulfuric acid is stored in the vicinity of water there should be warning and safety measures generated by model as sulfuric acid reacts violently with water hence posing threat to humans. |
| 1. Powerful data insights and recommendations for questionnaires. | --to be updated soon-- |

Table 1.1 – Overview of Problem statements

The above three problems posed a challenge to us, the team, which has minimal experience in the field of Artificial intelligence as there could be subjective interpretations about the problem itself and the various directions one would explore in which we can find a solution. Without diving into details of each problem we decide to brainstorm about the chemical module as it appears that accessing public data for research is easier when it comes to chemicals.

**Section 1: Discussing the Chemical module**

The main take aways from our discussion regarding the chemical module are,

* We aim to work in the space of non-english languages ( ex: Danish)

This could be either achieved by using a model whose base is trained on the specific language itself or use a translation models as a layer in our application.

**Areas involved – Machine translation.**

* The solution focuses on alerting the user about the risks involved or that could be avoided due to proximity of different chemicals in the surrounding environment.

We assume the INPUT to be status of laboratory chemicals, their storage, chemicals in the vicinity etc. The OUTPUT should be a warning message if any possibilty of chemical accidents with instructions to identify and avoid the same. Let's call this as Hazard identification for now.

**Areas involved – Semantic search, Text generation.**

**Chapter 2: The Groundwork**

Our competence and background revolves around programming and web development. Hence, we have a bit of catch up to do here. When we started studying about research and development in artificial intelligence happening around the world we found that we are in the era of transformers which was addressed in the research paper [attention is all you need](https://arxiv.org/abs/1706.03762) in 2017 which brought great advancements in attention mechanism that was then adopted by models that performed NLP tasks ex: BERT ,GPT-2 etc . Now, in 2023 there are much more powerful models with billions of parameters and trillions of tokens used to train these huge models, perhaps the large language models. It may occur to the you that we have jumped directly from machine learning beginner to advanced engineering architectures, but in actual we are just drawing an overview about what the trends are outside since it has key role in choosing certain direction to proceed with.

**Section 1: Natural language processing (NLP).**

Our first step in reading about similar problems solved using AI/ML gave us understanding that our problem lies in the paradigm of Natural language processing (NLP). To give some examples,

* **Machine translation**: NLP is used to translate text from one language to another. This is a challenging problem because languages have different grammars, vocabularies, and meanings. However, NLP techniques have made significant progress in machine translation in recent years.
* **Semantic Search**: Semantic search is a type of search that uses natural language processing (NLP) techniques to understand the meaning of the search query and the documents that are being searched. This allows semantic search to return more relevant results than traditional keyword search, which only matches the search query against the keywords in the documents.

You will find the above two methods to be part of the requirements in the inference drawn in chemical module discussion.

**Section 2 : TRANSFORMERS**

Enter transformer models! Transformer models are powerful solutions by now in and has been a game changer for NLP too. Covering all the large applications from language translation , sentimental analysis to text generation.

Transformers are a type of deep learning architecture that has proven to be highly effective for various natural language processing tasks. They excel at handling sequential data, like text, by allowing information to be processed in parallel rather than sequentially. This parallel processing capability makes them ideal for tasks like machine translation, text generation, sentiment analysis, and more.

**Brief history:**

Recurrent neural networks (RNNs): RNNs were one of the earliest neural network architectures used for NLP. They were able to learn long-term dependencies in text, which was a major breakthrough at the time. However, RNNs were also computationally expensive and difficult to train.

Long short-term memory (LSTM) networks: LSTMs were a more powerful variant of RNNs that were able to address the vanishing gradient problem that limited the performance of RNNs. LSTMs are now widely used for NLP tasks, such as machine translation and text summarization. .They can be computationally expensive to train and are susceptible to overfitting if data is not sufficient enough.

Transformers: Transformers were a new type of neural network architecture that was introduced in 2017. Transformers are not recurrent, which makes them more efficient to train and easier to parallelize. Transformers have quickly become the state-of-the-art for many NLP tasks, such as machine translation and question answering.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Model Recurrent? | Long-term dependencies | Efficiency |
| RNN | Yes | Yes | Low |
| LSTM | Yes | Yes | Medium |
| Transformer | No | Yes | High |

**Section 3: Large Language Models (LLMs):**

Large Language Models (LLMs) are a specific application of the Transformer architecture, where a massive number of parameters are trained on vast amounts of text data. These models, such as OpenAI's GPT-3, are capable of understanding and generating human-like text. They have demonstrated impressive performance across a wide range of language-related tasks, making them a powerful tool in the field of natural language processing.

Some of the advantages of LLMs are:

**Pre-trained Knowledge:** LLMs come pre-trained on vast amounts of text data. This means we can leverage this existing knowledge to quickly prototype and develop applications without needing to build extensive custom datasets from scratch.

**Few-shot and Zero-shot Learning:** LLMs, especially the more advanced ones like GPT-3, have the ability to perform tasks with minimal examples (few-shot learning) or even without any examples (zero-shot learning). This reduces the need for an extensive dataset and can save us a significant amount of time and effort.

**Rapid Prototyping:** LLMs enable us to rapidly prototype and iterate on ideas. We can quickly test out different approaches and refine our models without investing a substantial amount of time into training and fine-tuning.

**Community and Open Source:** There's a strong community around LLMs, which means we can tap into pre-built libraries, tools, and resources that have been developed and shared by others. This can save us development time and align with our limited budget.

**Reduced Training Overhead:** Training and fine-tuning traditional neural networks can be computationally expensive and time-consuming. LLMs eliminate much of this overhead by utilizing pre-trained weights.

**State-of-the-Art Performance:** LLMs often achieve or approach state-of-the-art performance on various language-related tasks, providing us with high-quality results even with limited resources.

**Section 4: Why have we decided to explore the path of large language models?**

This section mainly focuses on explaining concerns that arise about, viability of gaining knowledge and implementing LLM based solution. What are our plans to tackle the apparent obstacles?

It can be daunting for newcomers in this field to understand the underlying workings of LLM's but from the reading and browsing through enough of materials gave us a feeling that right level of "ABSTRACTION" would make our lives easier. Meaning to say we need not dig into each detail and process that have been used to develop LLMs. There are lot of platforms, hubs and educators that guide the newcomers in the industry to leverage the power without large amount of time being spent by us the developers. By now it is clear that we are not going to develop an LLM because that is not our core competence and objective here. Our objective is to leverage the state-of-the-art technology to integrate and fit into our business requirements.

One of the key points to note is OPEN-SOURCE models. The rapid advancement of generative AI has led to powerful models that are trained on billions of tokens and have billions of parameters. And these can be downloaded from model hubs or repositories for free and can be fine-tuned for our own tasks or use these models along with other tools to solve our problems. Model hubs like Hugging face, Pytorch and TensorFlow provide us APIs and libraries to interact with these models.

Apart from frameworks for working with transformers the expanding ecosystem has led to development of various LLM frameworks and vector databases that even allow us to chain different models, use vector databases to store and retrieve vector embeddings and build applications around them like Langchain. Just the idea of using vector embeddings along with a generative model opens the possibility of wide applications and solve large number of problems. Even the context size (context corresponds the input provided for prompting) have risen recently. The latest release of Llama 2, the autoregressive model released by meta has open access and is available for research and commercial use has a context length of 4032 tokens.

These developments in the ecosystem and flexibility without needing to breakdown mountains give us fair enough motivation to explore in this direction first. We also believe that our background in coding will also give us the acceleration to begin in understanding deep learning and working with LLMs.

**Chapter 3: Our need to learn deep learning.**

On a fine afternoon, we sit and open google collab and Hugging face tutorial on each tab. The tutorial starts with words like inference, parameters in the beginning and when we skip to the part where we can customize LLM according to our task by feeding own custom data we come across new words like fine tuning, embeddings, Pytorch , loss functions and others. It doesn't feel right. We were supposed to use these APIs and have a custom chatbot at our disposal as we had already witnessed people creating custom chatbots and what not. We could have followed some tutorial step by step to and copy pasted the code and it would have worked. But as coders we felt cheated . We needed more control and understanding of underlying workings, at least to a point where "ABSTRACTION" made us comfortable.

Even though there is lot of hand holding out there for developers to build on LLMs, in our understanding it is essential to understand neural networks while working with these models in order to come up with a solution that fits within our time and cost constraints but at the same time it should be as efficient as possible. Having said that, we are also open to learning new ideas outside LLM if that suffices our needs and compensates well enough in terms of time and cost . We believe if there is any alternative approach that we should keep into consideration that would be deep learning techniques. It could also be a possible case that we will use a hybrid approach.

**Chapter 4: Wrangling with dataset.**

At this point, we decide to keep two main tasks going hand in hand.

1. **Preparation of Dataset** – Taking Safety Data Sheets as the core information for training, preparing SDS like dataset for various chemicals could be useful to work with along the lines of NLP. Potential use case of this data :

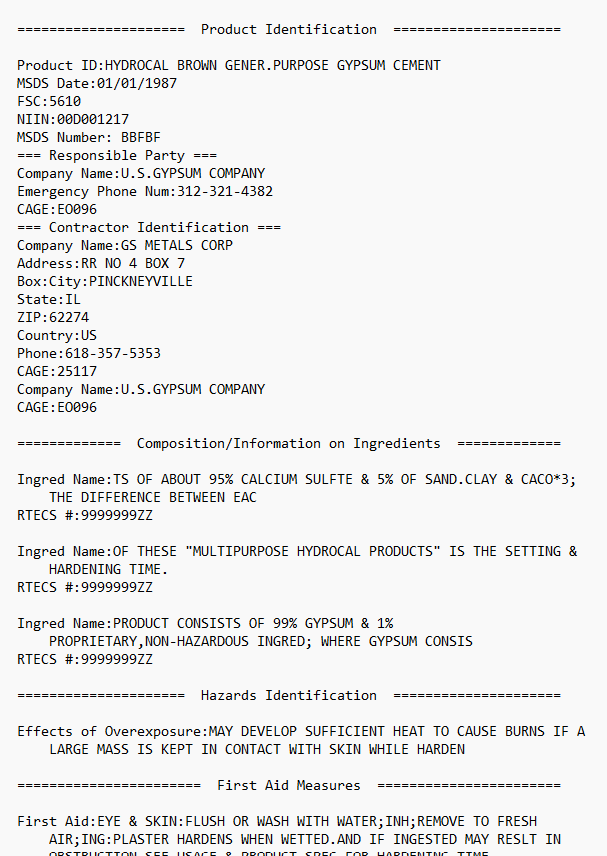
* Creating word embeddings and chain them as input to another LLM along with the prompt for inference.
* Use the dataset to train a semantic search algorithm.

1. **Finding an open source text generating LLM** – This task involves study of various open source LLMs. A pretrained LLM that can have a commercial license could be a potential solution.

**Material Safety Data Sheets (MSDS):**

Our useful encounter with a raw safety data sheets was with [Material Safety Data Sheets (MSDS)](https://www.kaggle.com/datasets/eliseu10/material-safety-data-sheets) on Kaggle. The MSDS dataset contains Material Safety Data Sheet files in the .txt format. The data was collected from [hazard.com](https://hazard.com/msds/). The dataset contains are 2 types of files with different structure: 1) **f1 type**, with 17454 files (155.4MB); and 2) **f2 type**, with 236507 files (1.3GB).

This dataset contains raw information about materials(products) of not just chemicals but of various materials that are used across industries like construction, household, transportation etc. These data sets contain folders having txt formatted files. These files have subset of standard specifications for safety data sheets as mentioned by [Globally Harmonized System of Classification and Labelling of Chemicals](https://en.wikipedia.org/wiki/Globally_Harmonized_System_of_Classification_and_Labelling_of_Chemicals).

A screenshot of a computer

Description automatically generated

Fig 1.2 – example txt file containing SDS information for Formaldehyde on left and Cement on the right .

Since the dataset from the MSDS contains unnecessary samples too, we cleaned up, filtered and separated chemical SDS information. We played around with the dataset and resorted to store the data in a single CSV formatted file with the header containing standard specifications of the SDS. We have retrieved about 200 chemicals' data from the entire dataset presently.

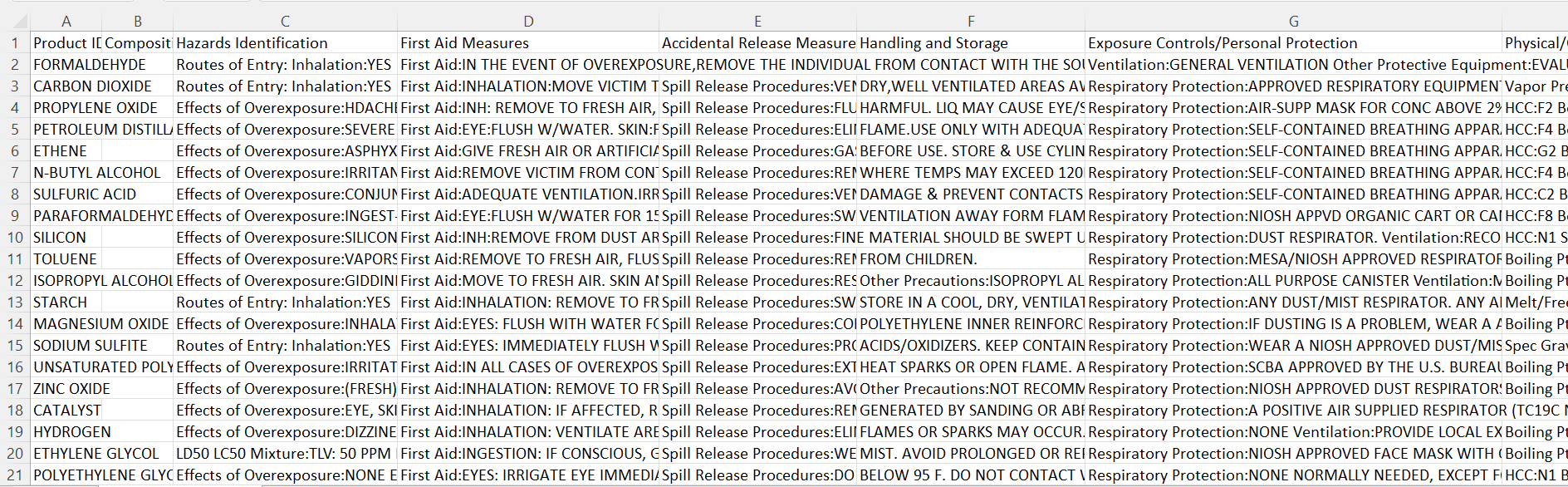
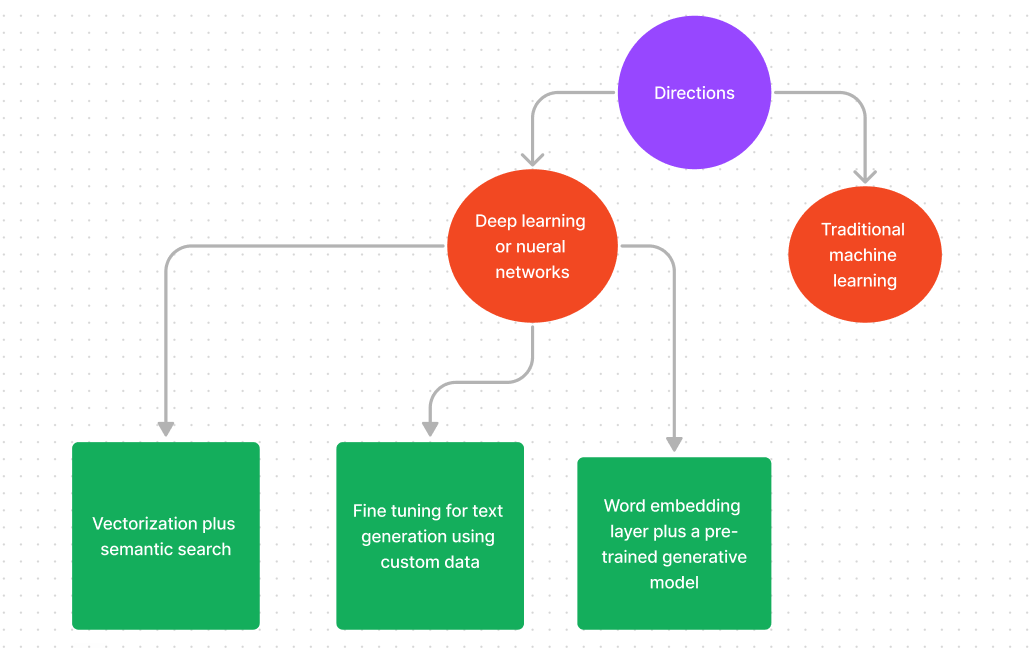


Fig 1.3- chemical dataset retrieved from MSDS and converted into single csv file using python.

A list of chemicals was used to compare and extract the relevant information and convert into csv format with SDS specifications as headers , each row representing information about particular chemical. This was automated using a [python script](https://github.com/Preetham-Saldanha-EG/Chemical-Module-AI-ML/blob/main/Scripts/Data-filtering/getSDSDataFromRaw/filter_chemical_data_from_MSDS_into_one_CSV.py).

**Chapter 5 : Mind Mapping.**

****

**Challenges we face.**

As they say, "it is all about data". One of the reason we picked chemical module is for easy public access of data. We found [Safety data sheets](https://en.wikipedia.org/wiki/Safety_data_sheet) and a raw dataset on Kaggle called Material safety data sheets (MSDS).

**Section 6: A potential solution.**