Sinead's note on LLM's - Large Language Models

They **predict the most likely word that comes next** and returns the top word in a list of predictions what have associated probabilities

The model will predict the word, then put that in the model, then predict the next word, put those 2 words in the model, predict word, put those 3 words in model etc etc

GPT was never trained to give correct answers, but it was trained for what word is most likely to come next, so SGD (Stochastic gradient descent) weights may not always be favoured for particularly correct answers

Not always in whole words, LLM's use tokens

An example could be unharmed
Where the tokens could be un har med
It was predicted after 'un' that 'har' was most likely, then med was most likely
Token can be whole words, part of a word, punctuation, a number etc

Deep neural networks typically have billions of parameters

GPT uses a library called tiktoken to get it's tokens
Tokens are numbers, then when decoded they become words
For the unharmed example, this would look like **[2990, 389, 117]** which when decoded would look like **['un', 'har', 'med']**

GPT model was based off a paper by Jeremy Howard published in 2017/18 about ULMFit

This system consists of **3 steps**

1. **Language model pre-training**
    a. This is the thing that predicts the next word in the sentence. If a word is guessed correctly, the network is rewarded, and if incorrect the network is penalised. The weights will update accordingly. They update using back propagation, and learns over time about the world

2. **Language model fine tuning**
    a. Feed the network a set of documents closer to the final task we want of the model. Nowadays, people do instruction tuning which is where you give the model sets of instructions and questions to learn from

3. **Classifier fine tuning (not always needed)**
    a. For example, reinforcement learning from human feedback (RLHF), where a human or a more complex model will pick which is the best from its output

You can print GPT to give higher quality answers by giving it custom instructions, they are pre-appended to all queries - you have to help LLM's to be 'good'

Some cons:

- Generally can't talk about itself, will just give what it thinks is the most likely answer
- It can hallucinate,
- Has info cut off in Sept 2021 for OpenAI
- Was not trained with URLs so does not know much about them
- Bad at pattern recognition

GPT-4 you can upload an image and it can perform OCR on the image to extract text, Gemini can perform OCR without code interpreter unlike GPT that has interpreter

When doing function calling, you **can't pass the function directly in, you need to convert the function to JSON schema**

Reasons you may want to use open source / in house LLM:

- Want to ask about your own documents
- Want more up to date info (post Sep 2021)
- Want to create a model that is particularly good at solving your problems

Deep learning = a NN with more than 3 layers, including input and output

LLM's are all about **memory speed rather than processing speed** so something like a 4090 is not necessarily better than a 3090. Memory size is very important too, around 48GB is a good start

People upload fine tuned models to **hugging face**
Fasteval has a good leaderboard for finding models

A lot of good models are based off Metas Llama-2 models, the smallest one they have is the **7b model which means it has 7 billion parameters**

How to improve open source model?

- **Fine tuning**
- **RAG (retrieval augmented generation)**
    - Takes the questions and try find documents that will help answer the question, i.e. Wikipedia, tell LLM about our found document to help it
    - Can use another model that will tell us what document is the most useful from a set of documents, using something like **SentenceTransformer**

A model in **SentenceTransformer** will take a document and turn it into a bunch of activations, 2 documents that are similar will then have similar activations

1. Turn question, doc 1, doc 2 into embeddings

2. Return a vector embedding of length n for each doc and question
3. Use vectors to calculate similarity i.e. cosine similarity of doc 1 and quest, then doc 2 and quest
4. Whichever of the cosines is higher is most useful to the question

When you have millions of documents use a vector database

Fine tuning is better than RAG as you can change the actual model based on documents available rather than just searching and retrieving