Introduction to Large Language Models (LLMs) and Prompt Engineering

This whitepaper is a brief introduction to using a type of machine learning called Large Language Models (LLMs).[1] It is not intended to be comprehensive and does not represent a recommendation that readers deploy commercial versions of this technology in Bible Translation. It is intended for purely educational purposes.

What is Artificial Intelligence (AI)?

AI is an attempt to use computers to replicate aspects of human intelligence. Machine learning is a subfield of AI that uses data to learn patterns and calculate probabilities. Generative AI is a type of machine learning that uses these probabilities to provide outputs in various mediums, including natural language.

What is an LLM?

An LLM is a type of generative AI ('a transformer') that allows humans to solve knowledge problems and perform creative tasks using sequences of natural language data. Think of it as a virtual brain ('a neural network'): it takes your input (e.g. a request), changes it to a list of numbers (a 'vector'), uses that list to identify patterns, and then assigns probabilities to predict the next token (typically a word or part of a word) and produce an output. Under the hood, LLMs have billions of conjoined nodes called 'neurons' that use training data (typically hundreds of billions of tokens) to adjust their connections to accurately predict the next token in a string.

The key breakthrough behind LLMs is the self-attention mechanism. When the model reads a sentence, it breaks it down into tokens ('tokenization') and creates three different representations: a query (question), a key (identifier), and a value (meaning). For each token, the model compares its query with the keys of all other words to see how relevant they are to each other. The model then uses these comparisons to assign each token a

[1] Prepared by Dr Travis Wright for the AI Task Force. Version date: 8.16.24.

weight. It combines the values (meaning) of these tokens using these weights, and repeats this process several times in parallel, each time from a different perspective ('multi-head attention'), allowing the model to capture different aspects of the sentence. The result is that the model can represent context, and therefore meaning.

LLMs perform well because communication is logical. With enough data, it is possible to represent the rules and functions of language mathematically. However LLMs do not use language like humans, reason like humans, and cannot 'understand' you like a human. In fact, LLMs are probably not cognitively realistic. Google's LLM has *540 billion parameters*; ChatGPT was probably trained on 300 billion tokens. At the time of writing, children still outperform LLMs in key tasks, although they have been exposed to significantly less 'data'.

Do LLM make mistakes?

Yes. When an LLM makes a mistake, it's called a 'hallucination': the LLM did not understand the prompt, its context, or the data it was trained on. Researchers distinguish between intrinsic and extrinsic hallucinations. Intrinsic hallucinations occur when the prediction of an LLM generates an answer that contradicts its training data. Extrinsic hallucinations are information the model adds that it cannot verify (e.g. it makes things up).

The truth is that hallucination is probably an inherent property of these models. However, in the years to come we can expect LLMs to grow increasingly powerful and accurate as they receive more industry attention. LLMs have already sparked a new 'AI summer' (a period of renewed interest in AI). Because many LLMs are freely available on the internet, it is important to understand how to use them well and to educate our field partners about their use and liabilities.

Should I use an LLM?

Before you ask how to use an LLM, you should ask whether to use one at all. LLMs are not useful for solving certain problems. LLMs, surprisingly, are not good at advanced math, solving logical puzzles, determining causation, or tutoring in basic subjects like

reading. LLMs should not be used to translate into or from low-resource languages when they have not been trained on this data. LLMs cannot be trusted for value statements or theological instruction ('open domain' questions). LLMs outperform humans in certain domains but cannot reason like humans in others and should not be trusted to perform all tasks equally well.

However, LLMs are extremely good at automating tasks ('correct the spelling and format of this text using the Chicago Manual of Style, 17th ed.'), retrieving information from multiple sources ('explain a few major opinions on issue *x* in field *y*'), translating from one gateway language into another (French > English), creating basic computer code, performing creative tasks, explaining complex concepts from different fields, and analyzing labeled data. LLMs are popular because they democratize the potential of machine learning by allowing anyone to interact with a machine learning model.

What are the risks of LLMs?

There are three major risks with LLMs:

1. **Bias**

LLMs are trained on virtually all the data that architects can find, which unfortunately includes harmful and biased material (vulgar, obscene, racist, inappropriate, etc.). Most models have been manually adjusted ('aligned') to reduce the possibility of harmful responses. But this material can still influence the way the model responds to your input, even when the prompt has been well-designed. Bias is always a risk, and as models become larger and larger, their biases don't disappear, they simply become more covert. The moral (and political) values of the engineer are irreparably baked into the model.

2. **Interference**

Anything that a model is trained on can provide false context to your prompt. For example, if you give ChatGPT a Biblical Hebrew sentence, it might use Modern Hebrew to analyze it (the two are not the same). That is because it was trained on billions of words of Modern Hebrew but only thousands of words of Biblical Hebrew. Likewise, if you ask ChatGPT to translate a Bible verse into a gateway language, it may simply output an existing translation.

3. **Blackbox Problem**

The internal logic of an LLM is opaque, even to those who design and train it. Imagine you stand in one room and your friend stands in another. Between you is a room that neither of you can see into. You can pass messages under the door but neither of you know what happens in the room between you. That room is a black box. We do not know what is happening 'under the hood' between your input and the LLM output: it is a black box. We know how to change an input to manipulate the output, but we do not know what's in the middle room. You should never use an LLM for an unsupervised task (e.g. 'translate x into Nahuatl' and then copy and paste the result without checking it).

Each of these risks is well-known. As one AI expert writes:

- If you didn't test it, don't trust it.
- If you didn't test it in [your environment], don't trust it in [your environment].
- If you didn't test it with [user population], don't trust it with [user population].
- If you didn't test it with [data population], don't trust it with [data population].
- If an input is unusual, don't trust it to output something sensible.

How do I use an LLM?

To use an LLM, a user enters a prompt (an input). The principles of designing inputs are called 'prompt engineering'. Prompt engineering is a discipline dedicated to providing relevant context and preventing the issues of bias and interference described above. Designing a good prompt requires thinking very carefully not only about the output you desire, but also about the type of learning you want the model to perform. The prompt should be:

- **Clear**
  - Tell the model explicitly what you want to achieve.
  - Be specific. Be detailed. Leave nothing implied.
  - Separate instructions from other information in the prompt with ###.
- **Concise**
  - Every part of the prompt should be relevant to the output you want.
- **Contain Relevant Context**
  - Your prompt should be detailed.

- Your prompt should include a role (a character) for the LLM.
- Your prompt should include examples (where possible).

Different types of prompts include:

- **Zero-shot**
  - This is a prompt without prior context. Open up ChatGPT, and fire away.
  - LLMs don't typically perform as well on zero-shot prompts.
- **One-shot**
  - This is a prompt with an example whose data is labeled in some way: "Here's an example. Notice this example has *x*." You can also use labeled data from an Excel spreadsheet or attach a PDF. One-shot prompts perform better than zero-shot.
- **Few-shot**
  - Same as one-shot but with a few more examples. The model then generalizes to new data using these examples.
- **Chain-of-Thought**
  - Chain-of-Thought prompts are like few-shot prompts, but they provide step-by-step details that instruct the LLM how to solve the problem.

Prompt Engineering

While we don't fully understand LLMs, we do know that providing context is key. If you ask a general purpose LLM, "Who is Paul?", thinking of the apostle Paul, it will respond with a generic answer about the origins of the name Paul. It needs context. We have also learned an LLM doesn't perform well when it receives negative instructions ('don't do *x*, don't do *y*'). Negative instructions limit context but do not create it. An LLM needs explicit instructions that create context about how it should behave. LLMs perform better when you provide examples (instruct the LLM which example is good, *and* which is bad!). Here is a template:

> "You are [role]. You are great at [skills]. Your writing is [traits] and your audience is [audience]. I would like you to help me [goal]. I need [task]. Please remember to [caveats]. The output should look like [output]. Here are [examples].
>
> Before we start, do you have any questions to help you better prepare?

Let's use this template to build a prompt. Let's say you want to learn how to make authentic Neapolitan pizza. Believe it or not, Associazione Verace Pizza Napoletana has guidelines protected by law for what counts as authentic Neapolitan style pizza. A bad prompt would look like this:

> Teach me how to make Neapolitan pizza.

Since you want the LLM to teach you the steps of a recipe, and you want that recipe to be authentic, you need to design a good few-shot prompt:

> ###Teach a novice how to make authentic Neapolitan pizza###
>
> Pretend you are an experienced pizzaiolo at a worldclass pizza restaurant in Naples, certified by the Associazione Verace Pizza Napoletana. I have no culinary education, but I want to make amazing Neapolitan pizza. I have a good pizza oven at home. I want you to search the internet and explain to me the steps required to make my own authentic Neapolitan pizza. For example, I want to know which flour to use, whether a sourdough starter matters (and if so, how to make one), the importance of the crust (and which types count as Neapolitan style), and how to make an authentic tomato sauce. Tell me the process step by step, beginning with the qualities of authentic Neapolitan pizza, then step-by-step instructions for preparation, using bullet points that explain the process to me as if I were your apprentice. Use no more than 250 words. Before we start, do you have any questions to help you better prepare?

This prompt is clear and concise, detailed, leaves nothing implied, provides a role to the LLM, uses important key words, contains examples and steps, and instructs the LLM about the desired content and format of the output. In other words, it contains relevant context that can produce the intended output. ChatGPT responded to this prompt with a beautiful answer according to the guidelines of the Associazione Verace Pizza Napoletana. When I removed the context, however, ChatGPT simply gave a generic pizza recipe for a home oven, even though the prompt contained the word 'Neapolitan'.

Optimal results with an LLM can be difficult to obtain. You will need to tinker around with prompts, try different approaches, change key words, and alter context to get the output you desire. Which type of prompt you choose will depend on the type of problem you're trying to solve. Basic problems ("how many miles is Orlando from Jacksonville?") can be solved with zero-shot prompts. If you're writing a guide for a field partner and need to know the key differences between a signed and a spoken language variety, you can drop a link to an existing comparison, and then ask the LLM to do the same with another gateway language. Chain-of-Thought prompts are useful for solving logical

problems systematically. It's also possible to combine different types of prompts to solve complex problems.

Conclusion

The history of Wycliffe Associates is characterized by taking risks to accelerate the mission of Bible Translation. Missional AI is simply a new chapter in that story. As a new type of tehcnology, the Large Language Model has the potential to play a useful part in that mission.