**ML Model Implementation**

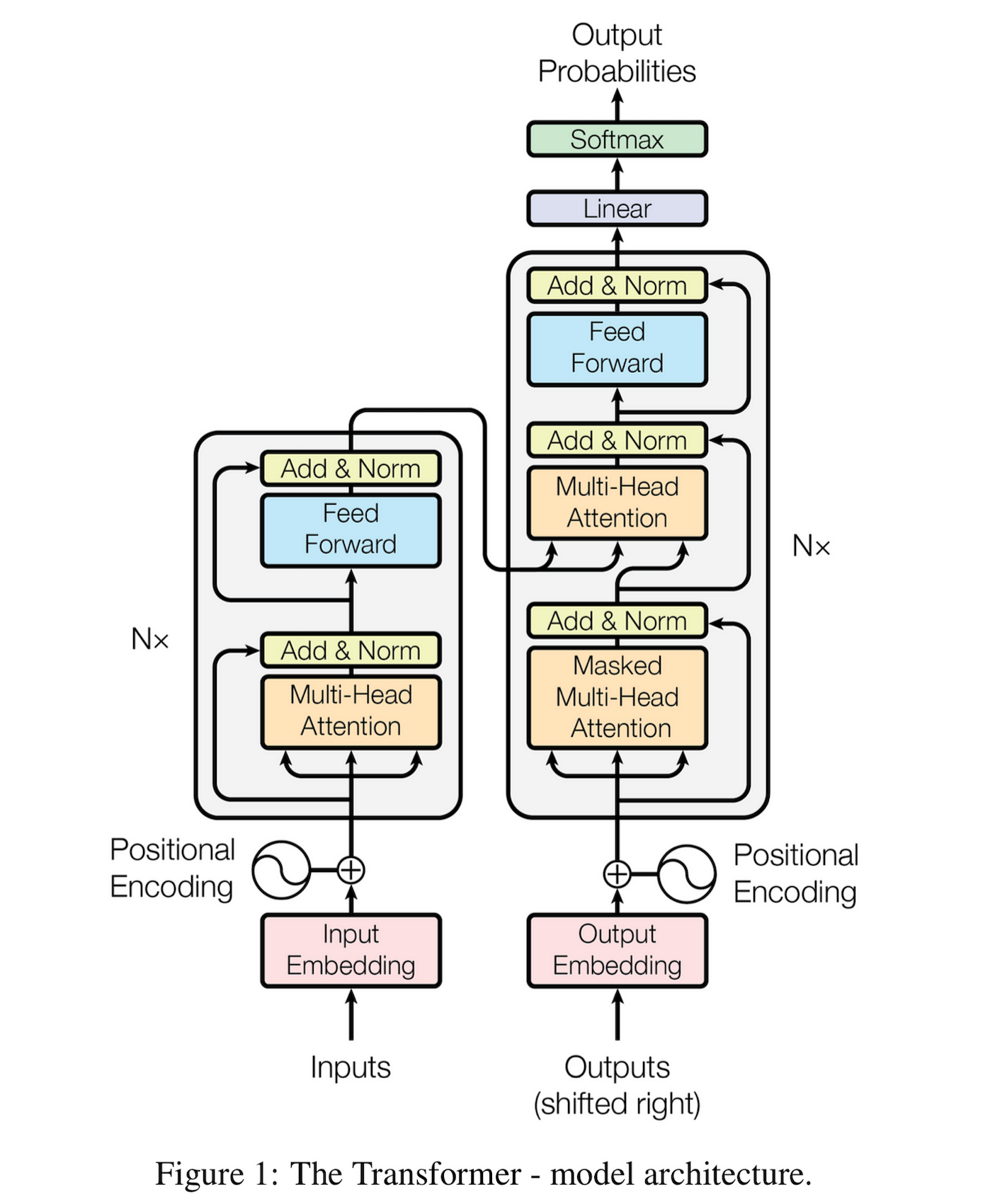
Initial idea was to use a Sequence Model like LSTM(Long Short-Term Memory). But due to the fact that LSTMs being sequential, take a long time to produce an output and still yield below average accuracy in Speech-to-Text conversion, Transformers became a better choice.

**Transformer Model Details:**

A transformer is a deep learning architecture developed by Google and based on the multi-head attention mechanism, proposed in a 2017 paper "Attention Is All You Need". Text is converted to numerical representations called tokens, and each token is converted into a vector via looking up from a word embedding table. At each layer, each token is then contextualised within the scope of the context window with other (unmasked) tokens via a parallel multi-head attention mechanism allowing the signal for key tokens to be amplified and less important tokens to be diminished. The transformer paper, published in 2017, is based on the softmax-based attention mechanism proposed by Bahdanau et. al. in 2014 for machine translation, and the Fast Weight Controller, similar to a transformer, proposed in 1992.

Transformers have the advantage of having no recurrent units, and thus requires less training time than previous recurrent neural architectures, such as long short-term memory (LSTM), and its later variation has been prevalently adopted for training large language models (LLM) on large (language) datasets, such as the Wikipedia corpus and Common Crawl.

This architecture is now used not only in natural language processing and computer vision, but also in audio and multi-modal processing. It has also led to the development of pre-trained systems, such as generative pre-trained transformers (GPTs) and BERT (Bidirectional Encoder Representations from Transformers).



Transformer models are very taxing on the system and creating our own custom transformer model will lead to poor accuracy due to the lack of sufficient speech data in different accents and languages to train the model. Hence, the first approach was to try the pre-trained transformer models available on packages like Hugging Face or OpenAI. Two transformer models were tried out namely: ***Wav2Vec2*** and ***Whisper***.

The ***Wav2Vec2*** model produced an output in nearly the same time as the input recording(on Google Colab CPU) but had a WER(Word Error Rate) of around 40% - 60% which was simply not reliable enough for it to be mapped to the system call commands directly without excessive error correction.

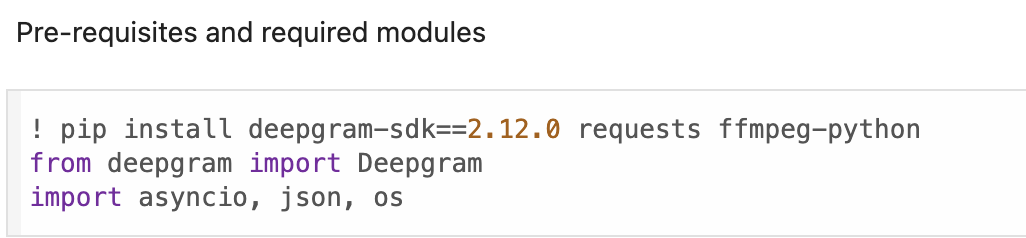
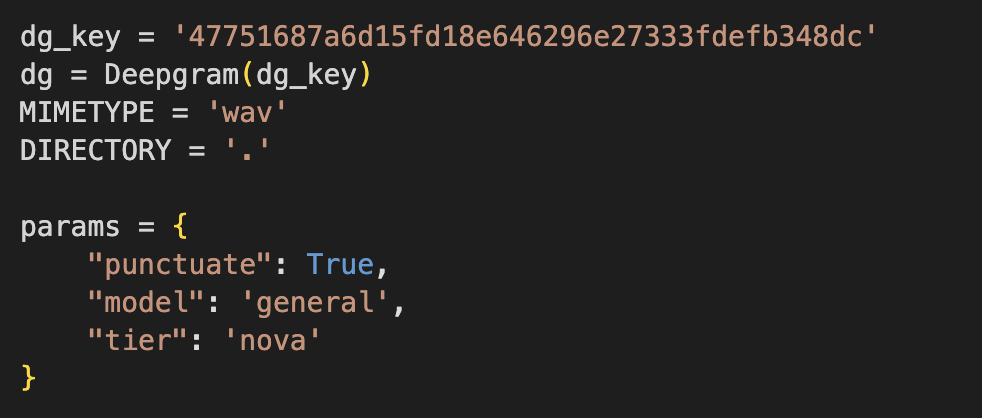
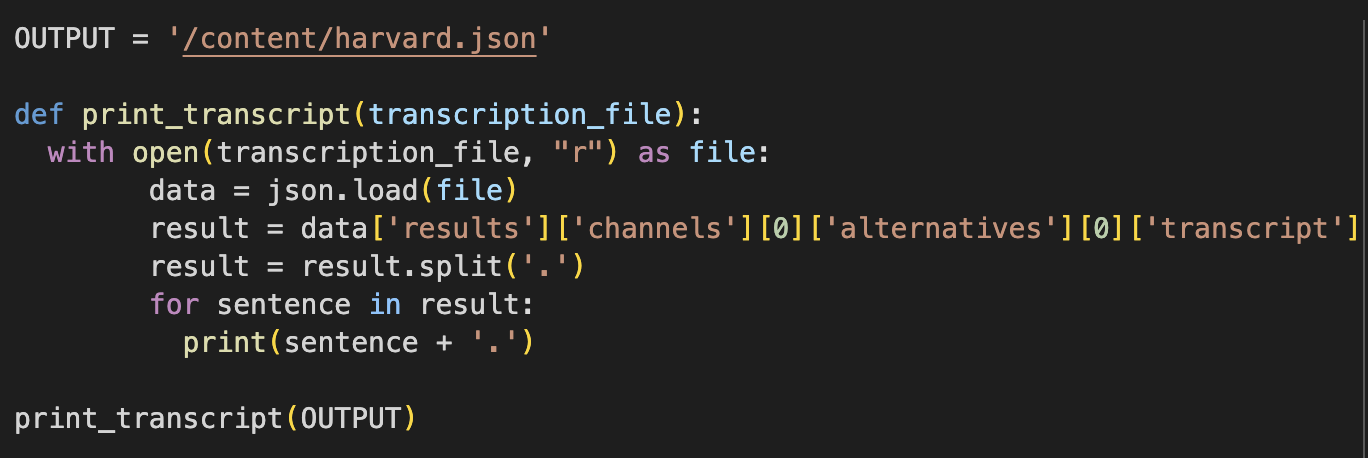
The ***Whisper*** model on the other hand, had a respectable WER of 10% - 15%. But it takes nearly 3 times the duration of the input audio file(on Google Colab CPU) to produce the transcript. Given that an average audio input will be around 10 seconds, this was too long a delay to realistically implement in our system, especially when we have no clue about the computing capacity of the machine on which the program will be executed, which may lead to even higher delays. One option was to enable the use of GPU to predict the model output but that is not ideal since that will be machine-specific leading to vastly different efficiency on different machines.

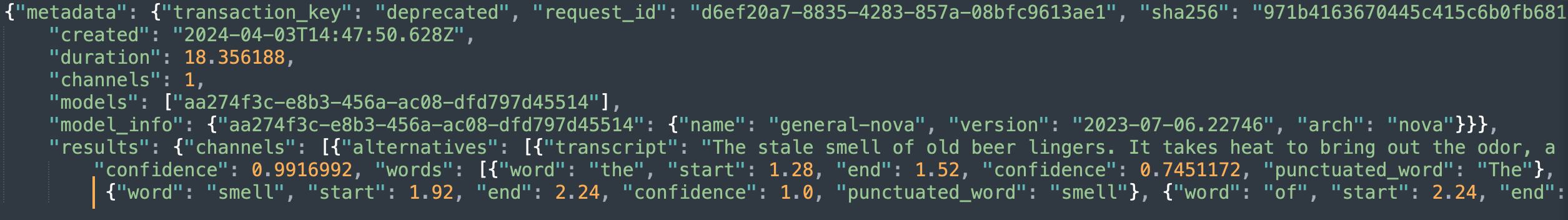
The objective now was to use a tool which can transcribe the audio in the shortest time possible with a respectable WER. The final model implemented is ***Deepgram API***.

**Tech Stack for Final ML Model:**

The libraries and SDKs required to be installed on the system for the Deepgram API are as follows:

* deepgram-sdk — pip version 2.12.0
* deepgram module in python
* asyncio module in python
* json module in python
* os module in python

1. We install and import the dependencies above first.  
   
2. We use an API key to connect to the Deepgram server and mention the file type and parameters of the model.  
   
3. The audio file in the specified directory is opened and a json file is generated containing all the details of the audio transcript like the words in the sample, the WER of each word, timestamp of each word, etc.  
   
4. Only the required transcribed text is extracted from the json file and displayed as output.  
   

The overview of the contents of the json file is as follows:  


Finally to implement this model, the required code was encapsulated into a function which takes the audio file path as input and returns the transcript text as output:  
