LLM-based Assertion Writing for Verilog Designs

H Sriram

Supervisor: Dr.Binod Kumar

Abstract

Language Models (LMs) based on Large Language Models (LLMs) have revolutionized natural language processing tasks by capturing the statistical properties of language and generating coherent and contextually appropriate text. LLMs, such as GPT-3 and GPT-4, are pre-trained on vast amounts of text data and learn to predict the next word or token in a sequence. They leverage powerful transformer architectures to model complex dependencies and generate high-quality text.
In this report we use BERT to generate assertions for a simulation trace obtained from a verilog design.

Introduction

Large Language Models (LLMs) are foundational machine learning models that use deep learning algorithms to process and understand natural language. These models are trained on massive amounts of text data to learn patterns and entity relationships in the language. LLMs can perform many types of language tasks, such as translating languages, analyzing sentiments, chatbot conversations, and more.
One such LLM, BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. Section I focuses on how BERT models work, Section II and Section III discusses about the code and the results of the analysis.

Section I

BERT is an open source machine learning framework for natural language processing (NLP) developed by Google AI language [1].BERT was pre-trained using only an unlabeled, plain text corpus (namely the entirety of the English Wikipedia, and the Brown Corpus).

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. The key component of transformers is the self-attention mechanism. This mechanism allows the model to focus on different parts of the input sequence and assign importance weights to each word or token based on its relevance to others. By considering the relationships between all the words or tokens simultaneously, transformers can capture long-range dependencies and understand the context of the sequence.
Transformers consist of encoder and decoder layers, which process the input sequence and generate the desired output. Multi-head attention is used to capture different types of relationships by applying the self-attention mechanism multiple times in parallel. Positional encoding is used to incorporate the order information of the words or tokens in the sequence.
During training, transformers learn to predict the next word or token in a sequence based on the previous context. They are trained on large datasets with known input-output pairs, and their parameters are updated through backpropagation.In inference, transformers can generate text by predicting the next word or token given a prompt or partial sequence. This process is performed iteratively until the desired length or completion criteria are met.

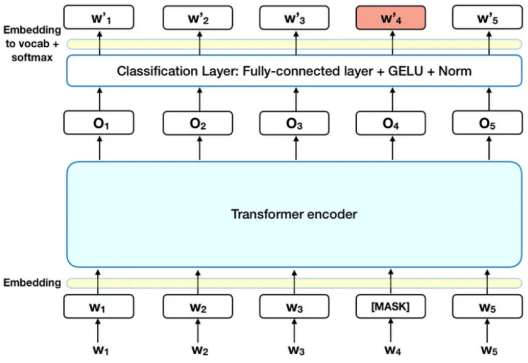


Figure 1: Architecture of Transformer

Section II

The code block to generate assertions is divided into four blocks

- Importing the necessary libraries and dataset
- Select a value k such that the true assertions generated has a relationship between k columns. Using this value of k we generate assertions of k columns from this dataset.
- Train the assertions on the BERT model.
- Using the BERT model predict if the assertions generated are actually true assertions or not. If yes, store it in a set.

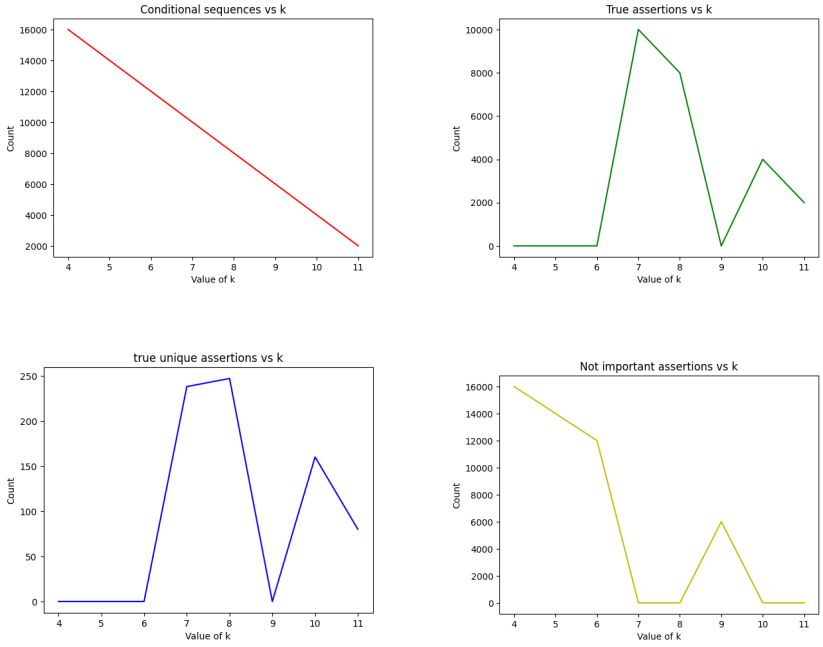
For the second step, chossing the value of k, it is important that K is selected in such a way that we are able to accurate get maximum amount of true assertions from the dataset. If a small value of K is chosen then then a large number of assertions would be flagged off as false prediction and the number of true assertions generated would be less. If a large value of k is chosen then we would not be able to train the model as efficiently as possible.
We use the model BertForSequenceClassification to train the dataset. For encoding, set the addspecialtokens paramter to true and set the maxlength depending upon the dataset. A set is used to store the true assertions so that repetitions are avoided.

```
for sequence in conditional_sequences:
      encoding = tokenizer.encode_plus(sequence,
    add_special_tokens=True, truncation=True, padding='
    max_length', max_length=256, return_tensors='pt')
      input_ids = encoding['input_ids'].to(device)
      attention_mask = encoding['attention_mask'].to(device)

      with torch.no_grad():
          outputs = model(input_ids, attention_mask=
    attention_mask)
          predicted_label = torch.argmax(outputs.logits).item()

      if predicted_label == 1:
          always_true_assertions.append(sequence)
      else:
         not_imp+=1
```

Section III

The model was tested over a simulation trace which contained 11 columns and 2000 entries. The model was run from k=4 to k=11 and four values were found out at the end of each iteration- total number of conditional assertions generated, total number of true assertions generated, total number of unique true assertions generated and total number of assertions that are not important. The results are shown in figure 2 and 3. From the figure we see that as k increases the number of conditional sequences generated decreases.

For k below 6 we see that true assertions are not being able to be generated as all of them are being flagged off as false predictions. We see that for k=6 to 8 and 10 to 11 we see that we are able to generate unique true assertions. We also see that the not imporatnt assertion count also decreases as k increases.



Conclusion

In conclusion, the use of Language Model-based (LLM) approaches for generating assertions in Verilog design holds great promise. By leveraging powerful language models such as BERT, we can automate the process of generating assertions from simulation traces or conditional sequences. These

models have the ability to understand the structure and semantics of the Verilog code, allowing them to generate meaningful and accurate assertions.

Using LLMs for assertion generation offers several benefits. First, it reduces the manual effort required in writing assertions, which can be a time-consuming and error-prone task. With LLMs, we can automate this process, saving valuable engineering time and resources. Additionally, LLMs have the potential to discover complex relationships between signals in the design, leading to more comprehensive assertions that cover various design scenarios.

References

[1] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

[2] Attention Is All You Need