Grammar Inference of Mixed Notation Language Using ALBERT Model for String Validation

***Abstract*— Grammar inference is a machine learning branch that can be applied in a variety of contexts. One popular approach for inferring grammars is using deep neural networks. The recurrent neural network architecture and its derivatives have been used in the past for this task due to their sequential nature. However there have been limitations in the kinds of grammars they can learn such as context-free grammars. Meanwhile, the transformer architecture has demonstrated remarkable success in various language modeling tasks. For this reason, in this study, ALBERT, a lighter version of the BERT model, which is based on transformers, was fine-tuned for the classification of a simple context-free grammar which mixes infix and postfix arithmetic notation. The fine-tuned model displayed an accuracy of 0.954 and an F1 score of 0.954.**

Keywords— ALBERT, BERT, grammar inference, notation, prefix, postfix, context-free language

I. INTRODUCTION

Grammar inference or grammar induction is the process by which a language is learned from examples (which can be positive or negative) [1] in an automatic manner, after which tasks such as classification can be performed [2]. Such process is applied in a variety of fields, from software engineering to bioinformatics.

A grammar in the context of computer science is a set of rules by which valid sentences in a language are constructed [3]. For example, English grammar or the specifications of programming languages. More formally according to [4], a grammar "is a quadruple (Σ, V, S, P)", where:

- Σ is a finite nonempty set of elements called terminals. [4]
- "V is a finite nonempty set disjoint from Σ. The elements of V are called the non-terminals" [4]
- "S ∈ V is a distinguished nonterminal called the start symbol" [4]
- "P is a finite set of productions (or rules) of the form

$$\alpha \rightarrow \beta$$

where α is a string of terminals and non-terminals containing at least one nonterminal and β is a string of terminals and non-terminals" [4]. In this regard, in a context-free grammar (CFG), α can only be a non-terminal, while β is allowed to be either a non-terminal or a terminal [5].

In terms of applications of grammar inference, a technique have been developed perform reverse engineering of a programming language dialect from a set of programs and the language from which such dialect was derived with an iterative approach [6]. Additionally, a "memetic algorithm (MAGIc) for unsupervised learning of context-free grammars" [7] has been developed and applied in the context of domain specific languages (DSLs.) Similarly, as an extension of grammar inference, algorithms for sematic inference have also been developed [1]. Lastly, grammatical inference was also used in the classification of amyloidogenic hexapeptides [8], some of which are responsible for amyloidosis, a set of diseases such as Alzheimer's, Huntington's disease and type II diabetes [9].

Considering the rise in popularity of neural networks due to advances in computer hardware and software, it is important to mention that there has also been an interest in utilizing them in learning arbitrary grammars. Recurrent neural networks (RNNs), which were designed to model sequential data, such words, or letters [10] by using previous outputs as inputs [11].

One example [12] is using Long Short-Term Memory networks (a type of RNN designed to address some of the shortcomings of traditional RNNs) to learn the structure of URIs from HTTP access log files and classify them as "good" or "bad" with a remarkable degree of success. However, in another case [13], it was shown that LSTMs were unable to learn a Dyck language with two types of brackets. According to the authors, there was a substantial difference in the error rate between the training data and the test data was significant, which is why they concluded that LSTM has limited capacity to learn grammar rules.

In another study [14], the authors designed a system, Grammar Guru, featuring two LSTM which received the input in opposite directions, whose disagreement was important to locate syntax errors and offer fixes. Such LSTMs were trained on JavaScript code found on GitHub and a subset was modified

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

by inserting, deleting, or substituting tokens randomly. The authors remarked that Grammar Guru was able to suggest the correct line in its top-4 suggestions at a rate of 54.74%. However, the authors noted that a big limitation of this system was the speed of producing suggestions.

Since the introduction of the transformer architecture in 2017, it has become the state of the art in natural language processing. Instead of relying on recurrence as it is in the case of recurrent neural networks, it only uses an attention mechanism, which has allowed it to have more parallelization thereby having faster training times [15]. Additionally, this enables transformer based neural networks to capture long-range dependencies [16]. When it first was proposed, the original transformer architecture was utilized in the context of machine translation and featured an encoder-decoder design [15]. Since then, transformers have been found useful beyond natural language processing, in computer vision tasks, audio and speech, and signal processing [17].

Transformers are also the foundation of generative models like Generative Pretrained Transformers (GPT), which became popular with the launch of tools such as ChatGPT by OpenAI or Claude by Anthropic. GPTs "predict the next token of a sentence given all the previous tokens" [18] and utilize the decoder part of the transformer architecture.

On the other hand, there are other architectures such as BERT (Bidirectional Encoder Representations from Transformers) which is "designed to read texts in both directions at once" [19] which allows it to perform masked language modeling and next sentence prediction [19]. According to the authors, a pre-trained BERT model can be fine-tuned to create models for other tasks by attaching an additional output layer [20].

One important application of BERT is text classification, which has been found to outperform other approaches such as bag-of-words in a study where the goal was to classify Yelp shopping reviews as helpful or unhelpful [21]. In other cases, it was used in the classification of fake news [22], and in legal contexts [23].

Given the success of transformers and their derivatives, the goal of this study is to define a context-free grammar and evaluate the performance of a fine-tuned BERT-based model for classification, specifically for determining if a given string is correct according to the rules of such CFG.

II. Methods

A. Choice of model

Considering the size and computational requirements of the original BERT base model, which contains 110 million parameters [24], smaller versions with similar performance have been developed. For example, ALBERT (A Lighter BERT) used factorized embedding parametrization and cross-parameter sharing to reduce the number of parameters to 12 million, which is why it was chosen as the model for fine-tuning in this paper.

There is also CodeBERT which was trained on a variety of programming languages from GitHub repositories, which could be useful given that the proposed grammar in this paper describes a subset of the grammar that describes most modern programming languages, but it was considerably large than ALBERT and even larger than BERT base, with 125 million parameters.

B. Context-Free Grammar

The context-free grammar utilized in this study is described by the following rules:

prefix_expr → prefix_expr + prefix_expr

prefix_expr → prefix_expr - prefix_expr

prefix_expr → prefix_expr * prefix_expr

prefix_expr → prefix_expr / prefix_expr

prefix_expr → (prefix_expr)

prefix_expr → [postfix_expr]

prefix_expr → variable

prefix_expr → number

postfix_expr → postfix_expr postfix_expr +

postfix_expr → postfix_expr postfix_expr -

postfix_expr → postfix_expr postfix_expr *

postfix_expr → postfix_expr postfix_expr /

postfix_expr → variable

postfix_expr → number

This grammar describes the language of arithmetic expressions both in infix and postfix notation. In infix notation, the operator appears between the operands and parentheses can be used to determine the order of operations. On the other hand, postfix notation, also known as reverse Polish notation, the operator is placed after the two operands, it does not need parentheses and it is computationally more efficient. [25]

To keep the grammar simple, unary operators were not considered in its design.

C. Dataset

Since the above grammar is custom-made there are no publicly available datasets that feature it. For that reason, the training and test data had to be generated synthetically. One simple approach was using the "generate" function of the NLTK platform which contains "libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning." [26] However, we found that such a function was programmed to generate all the possible strings of a language up to a certain level of depth. This had the main disadvantage of being computationally expensive and given more complex context-free grammar the number of tokens to be produced would be fairly limited for each string.

The other approach was to take advantage of the rules of the grammar, beginning with the starting symbol and randomly choosing production rules [27] to replace each non-terminal symbol.

In order to generate strings with different amounts of tokens and prevent the sentence generator from generating infinite strings. The following formula was used for determining the likelihood of choosing a production rule containing a terminal in its right-hand side:

$$P(choosing\ terminal) = 1 - \frac{1}{\frac{L}{k} + 1}$$

Where L is the sentence length in tokens and k is the sentence length at which the probability of choosing a rule that leads to a terminal is 50%.

In order to generate negative examples for the model, the same approach as in [14]. Some sentences were modified by randomly inserting, deleting, or substituting a token. During the generation process performing these operations sometimes would result in valid sentences, which is why when generating incorrect sentences, a parser written with PLY [28] (Python Lex-Yacc), an implementation of lex and yacc parsing tools, was utilized to validate that a sentence was grammatically incorrect. The sentence generating function would keep altering a given sentence until the parser classified it as incorrect.

D. Model and training setup

With the aim of setting up the ALBERT model for fine tuning, an existing example of text classification for misleading titles with ALBERT was used [29]. This example was adapted to accept our dataset. The model consists of the ALBERT model with a classification head which consists of a linear layer on top of the output from the pooled output [30]. An L4 machine from Google Collab was used to train and run the model.

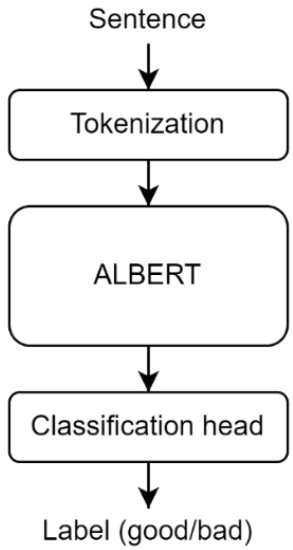

Fig. 1. First approach (the watch determines the user state)

Additionally, a dataset of 40,000 examples was randomly generated, half of which were good examples, and the other half were bad examples. Afterwards the dataset was split into training and testing subsets with a ratio of 80:20.

For evaluating the model, metrics like accuracy, f1, recall were used. The parameters used in training the model were the following:

TABLE I. TRAINING PARAMETERS

Parameters	Value
num_train_epochs	3
per_device_train_batch_size	16
per_device_eval_batch_size	16
learning_rate	$2*10^{-5}$
gradient_accumulation_steps	2

III. RESULTS

The following table shows a summary of the performance of the model:

TABLE II. MODEL EVALUATION

Metric	Value
Accuracy	0.954
F1	0.954
Precision	0.958
Recall	0.954
Accuracy Bad Label	0.909
Accuracy Good Label	0.999

From the previous table, the model was able to achieve impressive performance, where all the evaluation metrics reached values over 0.9. Interestingly the accuracy for bad labels was worse than that of the good labels.

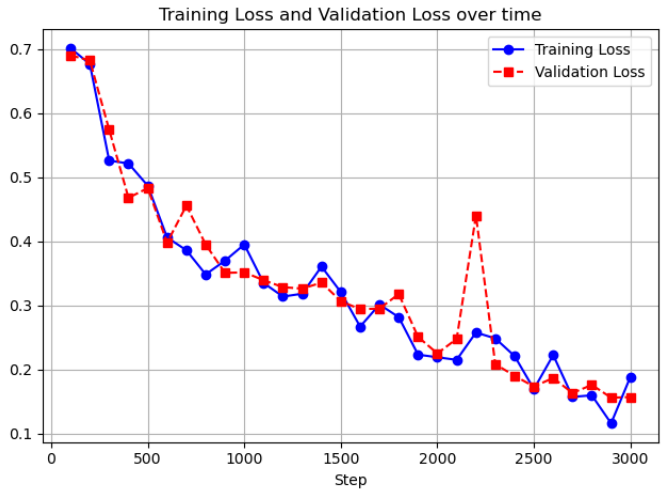

Fig. 2. Training Loss and Validation Loss over time

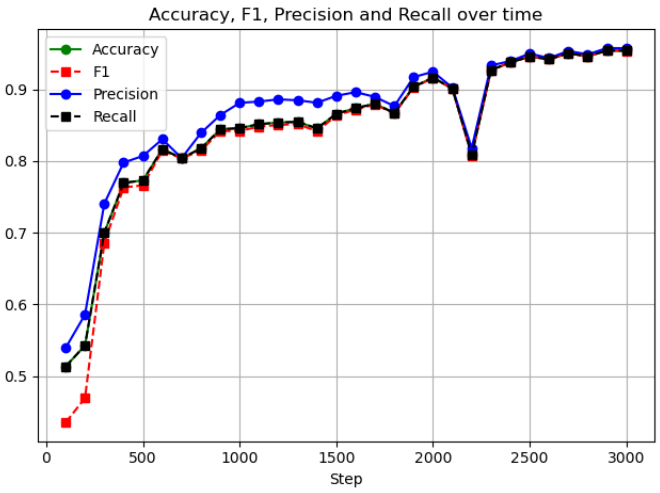

Fig. 3. Accuracy, F1, Precision and Recall over time

As can be seen from the previous graph, the accuracy, f1, precision and recall reached 0.8 around the 600th step. Since the effective batch size was thirty-two in order to reach good performance with ALBERT at least 19,200 samples would be needed.

IV. Conclusion

Grammar inference has been demonstrated to be an important field of study due to its wide range of applications, such as programming language reverse engineering and syntax validation. The approaches for performing grammar inference have shown to be diverse, from genetic algorithms to deep neural networks.

Given the limitations of recurrent neural networks in learning context-free grammars, transformers have emerged as a suitable alternative as shown during this study. An ALBERT model, a BERT-based model with reduced parameters, has been fine-tuned for the task of text classification for detecting sentences which are correct and incorrect given a grammar that describes arithmetic operations in both infix and postfix notation. After training the model, it achieved accuracy and an F1 score of 0.954.

Despite the exceptional results, the grammar used for the model was trivial. Future research could include the use of more complex grammars, such as the grammars that describe popular programming languages to test the capabilities of BERT-based models for detecting correctly formed sentences with arbitrarily complex grammars. In a similar vein, since we only displayed an approach to syntax validation, we propose syntax error location and correction using similar models as future research questions.

V. References

[1] Ž. Kovačević, M. Mernik, M. Ravber and M. Črepinšek, "From Grammar Inference to Semantic Inference—An Evolutionary Approach," *Mathematics*, vol. 8, no. 5, 2020.

[2] C. de la Higuera, "A bibliographical study of grammatical inference," *Pattern Recognition*, vol. 38, no. 9, pp. 1332-1348, 2005.

[3] M. Johnson and J. Zelenski, "Formal Grammars," 29 June 2012. [Online]. Available: https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/080%20Formal%20Grammars.pdf.

[4] T. Jiang, M. Li, B. Ravikumar and K. W. Regan, "University of California, Riverside," [Online]. Available: https://www.cs.ucr.edu/~jiang/cs215/tao-new.pdf.

[5] P. Linz, An Introduction to Formal Languages and Automata, Jones and Barlett Publishers, Inc., 2001.

[6] A. Dubey, P. Jalote and S. Aggarwal, "Learning context-free grammar rules from a set of program," *IET Software*, vol. 2, no. 3, pp. 223-240, 2008.

[7] D. Hrnčič, M. Mernik, B. R. Bryant and F. Javed, "A memetic grammar inference algorithm for language learning," *Applied Soft Computing*, vol. 12, no. 3, pp. 1006-1020, 2012.

[8] W. Wieczorek and O. Unold, "Use of a Novel Grammatical Inference Approach in Classification of Amyloidogenic Hexapeptides," *Computational and mathematical methods in medicine*, 2016.

[9] V. N. Uversky and A. L. Fink, "Conformational constraints for amyloid fibrillation: the importance of being unfolded," *Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics*, vol. 1698, no. 2, pp. 131-153, 2004.

[10] X.-Y. To, "Modern Approaches in Natural Language Processing," 2020. [Online]. Available: https://slds-lmu.github.io/seminar_nlp_ss20/book.pdf.

[11] A. Amidi and S. Amidi, "Recurrent Neural Networks cheatsheet," [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[12] S. S. Curley and R. E. Harang, "Grammatical Inference and Machine Learning Approaches to Post-Hoc LangSec," in *2016 IEEE Security and Privacy Workshops (SPW)*, San Jose, 2016.

[13] L. Sennhauser and R. Berwick, "Evaluating the Ability of LSTMs to Learn Context-Free Grammars," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Brussels, 2018.

[14] E. A. Santos, J. C. Campbell, A. Hindle and J. N. Amaral, "Finding and correcting syntax errors using recurrent neural networks," *Peerj*, 2017.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez and I. Polosukhin, "Attention Is All You Need," arXiv, 2017. [Online]. Available: https://arxiv.org/html/1706.03762v7.

[16] baeldung, "From RNNs to Transformers," Baeldung, 2024. [Online]. Available: https://www.baeldung.com/cs/rnns-transformers-nlp.

[17] S. Islam, H. Elmekki, A. Elsebai, J. Bentahar, N. Drawel, G. Rjoub and W. Pedrycz, "A Comprehensive Survey on Applications of Transformers for Deep Learning Tasks," arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2306.07303.

[18] X. Amatriain, A. Sankar, J. Bing, P. Kumar Bodigutla, T. J. Hazen and M. Kazi, "Transformer models: an introduction and catalog," arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2302.07730.

[19] C. Hashemi-Pour and B. Lutkevich, "BERT language model," TechTarget, 2024. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model.

[20] J. Devlin, C. Ming-Wei, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv, 2018. [Online]. Available: https://arxiv.org/abs/1810.04805.

[21] M. Bilal and A. A. Almazroi, "Effectiveness of Fine-tuned BERT Model in Classification of Helpful and Unhelpful Online Customer Reviews," *Electronic Commerce Research*, vol. 23, pp. 2737-2757, 2022.

[22] E. Shushkevich, M. Alexandrov and J. Cardiff, "Improving Multiclass Classification of Fake News Using BERT-Based Models and ChatGPT-Augmented Data," *Inventions* , vol. 8, no. 5, p. 112, 2023.

[23] J. A. F. Costa, N. C. D. Dantas and E. D. S. A. Silva, "Evaluating Text Classification in the Legal Domain Using BERT Embeddings," in *Intelligent Data Engineering and Automated Learning*, Évora, 2023.

[24] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv, 2018. [Online]. Available: https://arxiv.org/abs/1810.04805.

[25] baeldung, "Infix, Prefix, and Postfix Expressions," Baeldung, 2023. [Online]. Available: https://www.baeldung.com/cs/infix-prefix-postfix.

[26] NLTK, "NLTK," Documentation, [Online]. Available: https://www.nltk.org/index.html.

[27] G. Ilharco, "sentence-generator," GitHub, 2018. [Online]. Available: https://github.com/gabrielilharco/sentence-generator/tree/master.

[28] dabeaz, "PLY (Python Lex-Yacc)," dabeaz, [Online]. Available: https://www.dabeaz.com/ply/.

[29] I. Silfverskiöld, "Text Classification with Transformers (ALBERT)," GitHub, 2024. [Online]. Available: https://github.com/ilsilfverskiold/smaller-models-docs/blob/main/nlp/cook/fine-tune/albert_text_classification.ipynb.

[30] Hugging Face, "ALBERT," Hugging Face, [Online]. Available: https://huggingface.co/docs/transformers/v4.41.3/en/model_doc/albert#transformers.AlbertForSequenceClassification.