LSLlama: Fine-Tuned LLaMA for Lexical Simplification

Anthony Baez
Massachusetts Institute of Technology
Cambridge, MA, USA
acbaez@mit.edu

Horacio Saggion
LaSTUS / TALN / DTIC
Universitat Pompeu Fabra
Barcelona, Spain
horacio.saggion@upf.edu

Abstract

Generative Large Language Models (LLMs), such as GPT-3, have become increasingly effective and versatile in natural language processing (NLP) tasks. One such task is Lexical Simplification, where state-of-the-art methods involve complex, multi-step processes which can use both deep learning and non-deep learning processes (Sheang et al., 2022). LLaMA, an LLM with full research access, holds unique potential for the adaption of the entire LS pipeline. This paper details the process of fine-tuning LLaMA to create LSLlama, which performs comparably to previous LS baseline models LSBert and UniHD.

1 Introduction

Lexical Simplification (LS) is a sub-task within the field of Text Simplification (Saggion, 2017) in which complex words are substituted with simpler words while maintaining the meaning of the surrounding sentence (Shardlow, 2014). This is done to improve the comprehension of text for those who do not have sufficient reading proficiency, such as a language learner, young child, or someone with a learning disability (Saggion et al., 2022). Current LS models usually involve a multi-step process, including 1) the identification of complex words; 2) the generation of substitution words; 3) the selection of the substitutes based on context; 4) ranking substitutes by their simplicity; and 5) further context adaptation (Saggion et al., 2022). Recently, deep learning has been incorporated into some of these steps, such as a method that adapted BERT (Devlin et al., 2019), a bidirectional encoder, to generate substitution words (Qiang et al., 2020).

However, the development of generative LLMs, such as GPT-3 (Brown et al., 2020), presents an opportunity to drastically simplify this multi-step process by utilizing their ability to process and evaluate natural language. While these LLMs have significantly more parameters (110M for BERT vs. 175B for GPT3), and their training requires a substantially greater amount of computation than BERT-based models, this cost could be mitigated by fine-tuning a pre-trained LLM on a selected task. This fine-tuning could enable the model size to be drastically reduced while maintaining similar performance.

Large Language Model Meta AI (LLaMA) is a recently released generative LLM (Touvron et al., 2023). The size of its smallest variant, at 7B parameters, and its full research access provide a unique potential for adapting it to perform the LS task.

This paper details the novel fine-tuning of LLaMA on a Lexical Simplification task to create LSLlama. In order to determine if LSLlama is effective at the task, its performance was evaluated and compared to previously existing benchmark LS models that use deep learning, such as LSBert (Qiang et al., 2020) and UniHD (Aumiller and Gertz, 2022) using three testing datasets [1].

The structure of the following sections is as follows: Section 2 details related work in LS and deep learning. Section 3 details the method used to fine-tune LSLlama and evaluate the models. Section 4 details the results of this evaluation, and Section 5 discusses the implications of these results, error analysis, and limitations of the method. Section 6 concludes the paper and points to possible future work.

2 Related Work

The first method to incorporate deep learning into LS involved the use of neural networks to rank substitution candidates after being generated by a word embeddings model (Paetzold and Specia, 2017).

[1]The data and code are available at https://github.com/acbaez9/LSLlama/tree/main

A later implementation (Qiang et al., 2020) used BERT, a bidirectional encoder transformer model. BERT was used as a masked language model to predict the masked word in a sample sentence for simplification. The proposed substitution candidates were then ranked using a combination of LSBert, semantic similarity, and a frequency feature.

Recently developed generative LLMs such as GPT-3 (Brown et al., 2020) can be trained on billions of tokens and can process and respond in natural language. In the TSAR-2022 Shared Task on Lexical Simplification (Saggion et al., 2022), the highest scoring model on the English language task, called UniHD, used GPT-3 inference with six prompt variations to generate substitution candidates and a ranking algorithm that combined these candidate lists (Aumiller and Gertz, 2022). While this method outperformed all other BERT-based models, GPT-3 is not fully publicly available, and inference had to be done using paid API requests.

A recent generative LLM, LLaMA, (Touvron et al., 2023) was shown to achieve similar performance to GPT-3 in various NLP tasks at a fraction of the size with architecture improvements and a 1T token training set. Alpaca, a model that was created by fine-tuning LLaMA on 52K question-answer prompts, was also found to often behave similarly to ChatGPT in answering broad sets of questions (Taori et al., 2023).

3 Method

In order to evaluate LSLlama and compare it to LSBert and UniHD, LLaMA was first fine-tuned on an LS task to produce LSLlama. The specific version of LSBert used was adapted from its original version to act as a benchmark of an LS task (Štajner et al., 2022). All three models were then used to propose substitution candidates on three different datasets and evaluated on performance metrics.

3.1 Datasets

The dataset used to fine-tune LLaMA was the TSAR-2022 English gold standard dataset (TSAR) from Saggion et al. (2022), a multilingual shared LS task. There were three test datasets used to compare the three models: NNSeval (Paetzold and Specia, 2016b), BenchLS (Paetzold and Specia, 2016a), and LexMTurk (Horn et al., 2014). All datasets each contained hundreds of instances of a sentence, target word, and list of substitution candidates created using human annotators. While LexMTurk was sourced from Wikipedia, BenchLS was created by combining two other datasets, and NNSeval is a refined version of BenchLS.

All datasets were processed slightly to create ranked lists of candidate substitutes. For the TSAR dataset, repeated words were removed from its lists of candidates. In BenchLS and NNSeval, each candidate substitute had a number denoting the frequency that it was chosen by annotators, and this was removed.

3.2 Training

The 7B parameter variant of LLaMA was fine-tuned on the TSAR dataset using a modified version of the fine-tuning method of Alpaca (Taori et al., 2023). The fine-tuning involved feeding a prompt to the model which instructs it to respond with a list of synonyms that fit the context of the sentence. The target given was the corresponding list of substitution candidates, ranked by frequency, for that instance. This was done so the model could directly respond with ranked lists. The exact wording of the prompt went through multiple alterations to improve performance. The final version of the prompt used for fine-tuning, along with a example instance of an target word, sentence, and candidate list is in Table 1.

Prompt:
Respond with a list of different, simpler synonyms of the complex word in the given context. ### Complex Word: ***prototype*** ### Sentence: *This discovery helped to establish yet another spectral class even cooler than L dwarfs, known as ”T dwarfs”, for which Gliese 229B is the **prototype**.* ### Response:
Ranked Candidate List:
[’model’, ’sample’, ’original’, ’example’, ’template’, ’base’, ’archetype’, ’test’, ’first’]

Table 1: An example instance of the prompt and corresponding substitution candidate list used to fine-tune LSLlama

3.3 Inference

The generation parameters of the LSLlama inference were manually tuned. The most extensively tuned parameter was repetition penalty, which had considerable impact on the quality and nature of the output of LSLlama. When the repetition penalty

Dataset	Model	ACC@1	ACC@1@Top1	ACC@2@Top1	ACC@3@Top1
NNSeval	LSBert	0.4310	0.2469	0.3766	0.4519
	UniHD	**0.5732**	0.2803	**0.3849**	0.4435
	LSLlama	0.4519	**0.3096**	0.3808	**0.4686**
BenchLS	LSBert	0.6631	0.3703	0.5016	0.5748
	UniHD	0.7234	0.3057	0.4564	0.5436
	LSLlama	**0.7820**	**0.4700**	**0.5700**	**0.6460**
LexMTurk	LSBert	0.8300	0.3200	0.4300	0.4920
	UniHD	**0.8480**	0.4060	0.5560	0.6260
	LSLlama	0.8060	**0.4680**	**0.5740**	**0.6440**

Table 2: Results of models on the NNSeval, BenchLS, and LexMTurk datasets for Accuracy@1 and Accuracy@k@Top1

Dataset	Model	POT@3	POT@5	POT@10	MAP@3	MAP@5	MAP@10
NNSeval	LSBert	0.6946	0.7699	0.8619	0.2894	0.2180	0.1349
	UniHD	**0.7824**	**0.8619**	**0.9163**	**0.3661**	**0.2659**	**0.1629**
	LSLlama	0.7657	0.8536	0.8703	0.3233	0.2513	0.1425
BenchLS	LSBert	0.8396	0.8859	0.9225	0.4471	0.3341	0.2042
	UniHD	0.8751	0.9214	0.9483	0.4766	0.3552	0.2137
	LSLlama	**0.9420**	**0.9720**	**0.9760**	**0.5519**	**0.4329**	**0.2453**
LexMTurk	LSBert	0.9620	0.9680	0.9900	0.6044	0.4591	0.2865
	UniHD	**0.9700**	**0.9900**	**1.0000**	**0.6067**	**0.4638**	**0.2893**
	LSLlama	**0.9700**	0.9860	0.9880	0.5777	0.4556	0.2620

Table 3: Results of models on the NNSeval, BenchLS, and LexMTurk datasets for Potential@k and MAP@k

was too low, the model would respond with a long list of identical or very similar substitution candidates that would cause an error in the post-processing. When the repetition penalty was too high, the model would only output a few substitution candidates. Therefore, an optimal repetition penalty would not cause a post-processing error while allowing the responded list to reach a length of ten as frequently as possible. The optimal repetition penalty, to the nearest hundredth, was found by starting at a value of 1.00 and incrementally raising it until there was not an error in the post-processing after inference. If the repetition penalty for one dataset was found to be too low for another dataset, the tuning process was done again so the repetition penalty value did not cause an error for any dataset. The lowest repetition penalty value that did not cause an error was used for inference on all datasets, so that the same version of the model was used on all datasets. Regarding the prompt used for inference, it only differed from the prompt used in fine-tuning in that it asked specifically for a list of ten substitution candidates. LSLlama was eventually able to consistently output a single token that was a list of Python strings, so only minor post-processing was needed to correct occasional malformed strings and convert the output into a Python list. Other implementation details can be found in the Appendix.

3.4 Evaluation

Various metrics were calculated with the results of the tested models. These metrics were used in Saggion et al. (2022) to quantify the performance of models on the TSAR dataset.

Accuracy@1 (*ACC@1*): the percent of instances where the top-ranked substitute candidate is in the test dataset candidate list

Accuracy@k@Top1 (*ACC@k@Top1*): the percent of instances where at least one of the k top-ranked substitute candidates matched the top-ranked candidate of the test dataset

Potential@k (*POT@k*): the percent of instances where at least one of the k top-ranked substitute candidates is present in the test dataset candidate list

Mean Average Precision@k (*MAP@k*): a measure that incorporates the percent of k top-ranked substitute candidates that are present in the test dataset candidate list and the relative ranking of the proposed substitute candidate list

Values of k $\in \{1,2,3\}$ were used for ACC@k@top1, and values of k $\in \{3,5,10\}$ were used for Potential@k and MAP@k

4 Results

After evaluating LSBert, UniHD, and LSLlama on the test datasets, the results of the Accuracy@1 and Accuracy@k@Top1 metrics were compiled in Table 2, and the Potential@k and MAP@k metrics were compiled in Table 3.

Table 2 shows that for ACC@1, on the NNSeval dataset, LSBert scored 0.4310, UniHD scored 0.5732, and LSLlama scored 0.4519. On the BenchLS dataset, LSBert scored 0.6631, UniHD scored 0.7234, and LSLlama scored 0.7820. On the LexMTurk dataset, LSBert scored 0.8300, UniHD scored 0.8480, and LSLlama scored 0.8060. For ACC@1, in the NNSeval and LexMTurk datasets, UniHD is the highest scoring model, and on the BenchLS dataset, LSLlama is the highest scoring model. This signifies that UniHD's top-ranked substitute candidate is better than those of LSBert and LSLlama for the NNSeval and LexMTurk datasets, while LSLlama's top-ranked substitute candidate is the best for the BenchLS dataset. However, for ACC@k@Top1, LSLlama outperforms LSBert and UniHD on eight of the nine trials over all datasets. This shows that among the first three top-ranked substitution candidates, LSLlama generated candidates more likely to match the top candidate in all test datasets nearly every time.

In the POT@k values in Table 3, UniHD performed better than LSBert and LSLlama for all k values in the NNSeval and LexMTurk datasets, except for one tie between UniHD and LSLlama in POT@3 in the LexMTurk dataset. In the BenchLS dataset, LSLlama performed better than LSBert and UniHD in all k values. This indicates that on the NNSeval and LexMTurk datasets, UniHD is almost always able to produce the best substitutes over the whole list, while for BenchLS, LSLlama is able to produce better substitutes over the whole candidate list.

With the MAP@k values in Table 3, UniHD performed better than LSBert and LSLlama in the NNSeval and LexMTurk datasets. In the BenchLS dataset, LSLlama performed better than LSBert and UniHD in all k values. This result shows that UniHD generally proposes more relevant and better ranked candidates on the NNSeval and LexMTurk datasets, while LSLlama does this best on the BenchLS dataset.

5 Discussion

On the BenchLS dataset, LSLlama achieved the best scores on all evaluation metrics, so LSLlama consistently outperformed LSBert and UniHD on this dataset. On NNSeval, there were more mixed results in the ACC@k@Top1 metrics, with both UniHD and LSLlama having the highest score in different trials. However, for all other metrics on NNSeval, UniHD performed the best. On LexMTurk, LSLlama performed the best in the ACC@k@Top1 metrics, while UniHD performed as well or better than LSLlama on the other metrics. LSBert was always outperformed by either UniHD or LSLlama.

UniHD was better able to identify the top-ranked substitution candidate, as evidenced in its highest Accuracy@1 value, have a better ranking of candidates, as evidenced by its highest MAP@k value, and propose more relevant candidates when having multiple attempts, as shown by its highest Potential@k values on the NNSeval and LexMTurk datasets. However, on these datasets, LSLlama usually best identifies the test dataset's top-ranked candidate in the first few substitution candidates, as evidenced by its highest ACC@k@Top1 values on all but one trial. On the BenchLS dataset, LSLlama was best able to identify the top-ranked substitution candidates, have a better ranking of candidates, and propose more relevant candidates when having multiple attempts.

When looking at results overall, LSLlama always outperformed UniHD on one dataset. With the other two datasets, LSLlama outperformed UniHD on some metrics, and UniHD outperformed LSLlama on some metrics, with UniHD outperforming LSLlama more often. This indicates that, overall, UniHD and LSLlama performed comparably when looking at all datasets as a whole. Additionally, for all trials, at least one of these two models, UniHD and LSLlama, always outperformed LSBert, as LSBert was never the top scoring model on any metric.

5.1 Error Analysis

While using a generative LLM allows for drastic simplification of the LS pipeline, it can also lead to difficulty getting the model to respond coherently and as intended. Some issues and errors that were encountered are detailed below.

Variation in Output Length Despite including the specification of a list of ten substitute words in the prompt used for inference, LSLlama did not respond with a consistent number of substitution candidates. This likely occurred due to the generation method used, in which the list of candidates was generated using a single token, so there was no parameter that could directly control the length of the substitute list. The fine-tuning dataset for LSLlama also did not have a consistent length of substitution candidate lists. However, the average length of the candidate lists was around ten, and the fine-tuning and generation parameters were able to regulate the length of the response enough so that there was a difference in the results between the metrics with k = 5 and k = 10.

Prompt Stability A commonly-encountered weakness of generative LLMs is how subtle changes of the prompt can lead to dramatic changes in the nature of the output. A process of modifying the prompt and then performing inference to qualitatively gauge its effect was done in order to improve the efficiency of the prompt. Two of the changes that yielded the most improvement was specifying ”different” and ”simpler” synonyms in the prompt, which is reflected in the final prompt in Table 1. The lack of intuitiveness of this process made it time consuming and imprecise. For example, ”Respond with a list of words that can replace the complex word” was changed to ”Respond with a list of synonyms of the complex word”. In this specific LS task, using the first wording is more accurate, as depending on the context, the best substitution candidates are not necessarily exact synonyms to the complex word, and the best candidate to replace a word might be a phrase of two or more words. However, the second wording of the prompt noticeably outperformed the first wording. An explanation could be that asking for a ”synonym” is a more clear and direct command than asking for ”words to replace”. Every time such a change to the inference prompt was made, the fine-tuning prompt also needed to be changed, as altering only the inference prompt led to incomprehensible responses from the model. This resulted in needing to fine-tune the model again to get the results of each prompt change, which added a substantial amount of time to the process.

5.2 Limitations

In working with LLMs, a significant amount of computational resources were needed for fine-tuning and inference. This computational cost resulted in longer times for fine-tuning and inference, limiting the extent to which the fine-tuning hyperparameters and the inference parameters could be optimized.

Additionally, the TSAR dataset used for fine-tuning only contained 373 instances, a very small number when compared to other fine-tuning datasets, such as the 52K instance dataset used in Alpaca. Whereas ChatGPT was used to generate these examples, the specificity needed for the LS task necessitates human annotators in the creation of a dataset.

6 Conclusion and Future Work

This paper compared a fine-tuned, generative LLM, LSLlama, to previously existing LS baseline models LSBert and UniHD. At evaluation, LSLlama outperformed UniHD in all metrics on one dataset, while on the other two datasets, UniHD outperformed LSLlama on most, but not all, metrics. LSBert also never scored the highest on any metric. Regarding their architectures, generative LLMs require more computational resources than BERT-based models, however LSLlama is able to simplify the multi-step process of LSBert. LSLlama generates and ranks substitution candidates at inference, whereas a separate ranking algorithm is used after inference from LSBert. A separate ranking algorithm is also used in UniHD. Even though UniHD and LSLlama are both generative LLMs, LSLlama takes advantage of fine-tuning to significantly reduce its size, from 175B parameters for GPT-3 to 7B parameters for LSLlama, while maintaining comparable performance on evaluation metrics. Despite some recorded challenges posed by their architecture, this research demonstrates the potential for LLaMA and other LLMs that are fine-tuned on a LS task to improve upon existing benchmarks in Lexical Simplification.

For future work, fine-tuning could be done using multiple datasets. This could improve the model’s specificity, as it would increase the size of the training set used for fine-tuning. Testing then can be done on one dataset. In addition, further manipulation of the fine-tuning hyperparameters, inference parameters, and prompt wording could also be pursued to improve the performance of LSLlama.

7 Lay Summary

Lexical Simplification (LS) is a field which develops methods to simplify text by substituting complex words with simpler ones while maintaining the meaning of the surrounding sentence. This is done to improve the reading comprehension of text for those who do not have sufficient proficiency in a specific language.

Recently, methods involving deep learning, which use multi-layered neural networks, have improved upon the performance of previous methods that did not incorporate deep learning. There are two notable methods for LS that use deep learning: LSBert and UniHD. LSBert uses a combination of a deep learning model and other non-deep learning methods. UniHD uses a Large Language Model (LLM), a large neural network that runs on many powerful computer components called graphics processing units (GPUs), and ranks multiple outputs to propose candidates to substitute for a selected complex word.

Both of these models pose challenges. LSBert uses a multi-step process with multiple inputs to propose candidates, while UniHD uses a two-step process that needs a large network of GPUs. LSLlama, the proposed model, resolves these weaknesses by using a single-step process that can be run on four GPUs. The performance of LSBert, UniHD, and LSLlama on a LS task were compared to determine whether LSLlama is competitive with these previous baseline models.

After carrying out testing, LSLlama was found to perform comparably to LSBert and UniHD on the three test datasets. LSLlama outperformed UniHD on one dataset consistently, and UniHD outperformed LSLlama the majority of the time on the other two datasets. LSBert was never the highest performing model on any metric. This demonstrates that the improvements to LSLlama's design do not come at the cost of significant performance, indicating a promising direction for improvement in lexical simplification.

By simplifying text with LS, it becomes more accessible for readers such as new language learners, young children, or those with a learning disability. In order for someone to benefit from this research, LSLlama would need to be incorporated as a component in a text simplification system that can be used with natural language and that is available for free or commercial use.

References

Dennis Aumiller and Michael Gertz. 2022. UniHD at TSAR-2022 shared task: Is compute all we need for lexical simplification? In *Proceedings of the Workshop on Text Simplification, Accessibility, and Readability (TSAR-2022)*, pages 251–258, Abu Dhabi, United Arab Emirates (Virtual). Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Colby Horn, Cathryn Manduca, and David Kauchak. 2014. Learning a lexical simplifier using wikipedia. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 458–463.

Gustavo Paetzold and Lucia Specia. 2016a. Benchmarking lexical simplification systems. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3074–3080.

Gustavo Paetzold and Lucia Specia. 2016b. Unsupervised lexical simplification for non-native speakers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Gustavo Paetzold and Lucia Specia. 2017. Lexical simplification with neural ranking. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 34–40.

Jipeng Qiang, Yun Li, Yi Zhu, Yunhao Yuan, and Xindong Wu. 2020. Lexical simplification with pretrained encoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8649–8656.

Horacio Saggion. 2017. *Automatic Text Simplification*, volume 10 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers.

Horacio Saggion, Sanja Štajner, Daniel Ferrés, Kim Cheng Sheang, Matthew Shardlow, Kai North, and Marcos Zampieri. 2022. Findings of the TSAR-2022 shared task on multilingual lexical simplification. In *Proceedings of the Workshop on Text Simplification, Accessibility, and Readability (TSAR-2022)*, pages 271–283, Abu Dhabi, United Arab Emirates (Virtual). Association for Computational Linguistics.

Matthew Shardlow. 2014. A survey of automated text simplification. *International Journal of Advanced Computer Science and Applications*, 4(1):58–70.

Kim Cheng Sheang, Daniel Ferrés, and Horacio Saggion. 2022. Controllable lexical simplification for English. In *Proceedings of the Workshop on Text Simplification, Accessibility, and Readability (TSAR-2022)*, pages 199–206, Abu Dhabi, United Arab Emirates (Virtual). Association for Computational Linguistics.

Sanja Štajner, Daniel Ferrés, Matthew Shardlow, Kai North, Marcos Zampieri, and Horacio Saggion. 2022. Lexical simplification benchmarks for english, portuguese, and spanish. *Frontiers in Artificial Intelligence*, 5:991242.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

A Appendix

Huggingface Transformers and Pytorch were used for implementation of the model. DeepSpeed was also implemented to optimize fine-tuning, which used the cpu_adam optimizer. The model was fine-tuned for 4 epochs. All other fine-tuning hyperparameters are identical to Alpaca (Taori et al., 2023). The fine-tuning and inference was done on 4 V100-32G GPUs.

Inference on LSLlama was performed using greedy decoding with a temperature of 0.1, top_k of 0.75, and a repetition penalty of 1.11.

B Acknowledgements

We would like to thank Kim Cheng Sheang, Euan McGill, and Ella Tubbs for their contributions. We would also like to thank the MISTI program at MIT for making this collaboration possible.