EvaCun2025 Paper

Anonymous submission to EvaCun2025

Abstract

Through the use of a hierarchical clustering algorithm and a prompting-based method for the few-shot model, existing LLMs can be trained to better perform the task of lemmatizing ancient cuneiform languages. In our submission, the clustering algorithm was implemented using Levenshtein distance, also known as edit distance. Batches were passed through first with prompts telling the model what the lemma of each word was, then passed through again with prompts asking it what the lemma of each word is. To measure our accuracies, new batches were passed through with only prompts asking for the lemmas, and the model's output was then compared to the actual lemmas to calculate our accuracy. Our accuracies were lower than we had hoped, with highly performing tests ranging from 5-20% of prompted words being correctly guessed. Despite a low accuracy, we demonstrated that various prompt engineering strategies as well as different ways of analyzing the initial data improved the model's ability to accurately lemmatize cuneiform languages.

1 Introduction

Our goal was to use the few-shot prompting model with established LLMs. In focusing on the prompts we use, we can better interact with the model to convey the relevant information without providing confusing details. We created a pipeline based on hierarchical clustering in which initial, inner clusters were formed according to distance from the lemma. Then, the LLM formed outer clusters by grouping various inner clusters together as a way of forming 'grammar rules'. When a prompted clean value is given to the LLM, it determines which clean values inside the inner clusters are closest to the prompted clean value via Levenshtein distance. It then derives the transformation rules and applies them to output the generated lemma.

To implement this, the training data was analyzed, processed, and fed to an LLM model through the prompts we engineered. Lemmas were compiled along with their associated clean values. Using batch training, we familiarized our model with the lemmas of a few clean values following a few-shot model. Then, we asked the lemmas of other clean values that the model was not trained on to test its accuracy during the development process and to create our submission in the final testing process.

2 Few-Shot Prompting

The goals of this system are based on few-shot prompting. The ideas of this concept are two-fold: to design properly-formatted and meaningful prompts and to select a small amount of important examples that accurately train the model on the data (Wang 2022). We implemented the template prompt engineering pattern cataloged by White et al as a method proven to get better results when interacting with LLMs, especially OpenAI models (White 2023).

The template pattern involves prompts telling the model what information to expect, as well as what format it can expect to find this information in. We implemented this in prompts such as 3, 4, 10, and 12. Prompt 3 informs the model what kind of information it can expect to receive (a word to identify and possible background information), and prompt 4 tells it how to respond (with a single word representing the lemma of the word). Prompt 10 feeds it the word and identifies the correct lemma, keeping details to a minimum so that the model can focus on the important information. Prompt 13 was used to include the provided example sentences before asking for each lemma. We observed that dropping the example sentence led to a decrease in accuracy. When forming our prompts, we aimed to be specific and concise so that there was no ambiguity to confuse the model. Appendix A lists the set of prompts that we created for use in this task.

The second part of the prompting few-shot model method is selecting meaningful examples to train the model on. Our implementation focused primarily on number of occurrences of the lemmas, or lemma frequency, in the process of choosing examples. By selecting a larger amount of the most frequently occurring lemmas, we train the model to recognize lemmas of words that come up frequently. By choosing a smaller amount of infrequently occurring lemmas, we train it to be aware that there are other, less common words out there to be aware of.

3 System Description

The data we were given at the start of the task was split into an 80-20 partition. The first, larger portion was used as the training set while the second, smaller portion became the dev set. The training set was used to get lemmas to train the model on, and clean values were taken from the dev set to test the model's accuracy in predicting the lemma of a clean value.

During the training process, batches were created by sorting the data by properties such as the total number of occurrences of the lemma. This method of selecting examples on which to train our model allowed us to focus on common words while diversifying our example set so that the model does not become biased. Batch sizes ranged from 4 to 30 lemmas, with batch counts ranging from 2 to 24. Clean values per lemma ranged from 1 to all per lemma. The distribution of lemmas within the batches also varied, including distributions using more infrequently occurring lemmas than frequent ones, where lemmas from specific languages were selected for or ignored, and where sorting by occurrences pertained to the frequency of the clean value rather than the lemma.

During training, each clean value in the batch could be sent as a statement and/or question. Statement prompts were sent to the model in the format "[L] is the lemma of [CV]" where a lemma and clean value replaced the [L] and [CV] tokens. Question prompts were in the format "What is the lemma of [CV]?" which would require an answer from the model. We collected data on our training accuracy by evaluating the accuracy of the responses given to these question prompts.

This feature was implemented after we noticed a pattern in our training, in which our accuracy when asking the lemma of each clean value would start low and climb as our model was being trained. It would peak about three-quarters of the way through and then begin to decrease. Thinking that we were likely seeing overfitting, we began to alternate between sending statement prompts and question prompts within the training section, a process shown to reduce overfitting. This resulted in improvements to our performance.

4 Refining the System Using the Dev Set

To refine the system, we ran our pipeline and performed error analysis on the dev set. The factors we implemented during this stage included variations in prompt wording, alternating between prompts stating rules and prompts asking questions, positive/negative reinforcement, and using mask tokens to note spaces in the example sentences that were missing words.

Positive reinforcement meant sending a prompt indicating that the model's prediction was correct, while negative reinforcement meant sending a prompt indicating that it was incorrect. Tests with positive reinforcement did not result in increased mean accuracy, but implementing negative reinforcement was effective in the general form seen in prompt 14. After some error analysis, we tried to take it a step further by implementing various degrees of negative reinforcement feedback. This included 'small' corrections, which sent an additional negative response telling the model its answer was close in order to address the common case in which the lemma was mostly accurate but a few letters off (see prompt 15). Commonly mistaken lemmas were addressed by keeping track of how often an incorrect lemma was guessed within each section of clean values within a given lemma and prompting it to avoid making those guesses (see prompt 16). Both of these options seemed promising in theory but ultimately caused worse results, so they were discarded before final testing began.

We used the data from the dev set to test the accuracy of our model post-training. Batches were created using the dev set, with only the question flag set to true. They were each passed through as question prompts. The responses were collected and evaluated to get our output accuracies.

Batch Properties							Features			Reinforcement				Accuracy
Size	Count				Question	Statement	Sentence		Lang.	Negative			Positive	
	High	Low	Med	Rand			Mask	No Mask		G	S	C		
15	1	2			X	X		X		X				6.26
30	1	2			X	X		X		X				8.09
4	8	12			X	X		X		X				7.56
4	4	6			X	X		X		X				6.57
4	16	12			X	X	X			X				9.42
4	16	12			X	X	X			X	X	X		4.01
4	16	12			X	X	X			X	X	X	X	1.91
10	3	1	2		X	X	X		X	X				6.47
5	7	1	2	2	X	X	X		X	X				6.40

Table 1: Mean accuracy comes from 2 tests per test configuration (with the exception of the highlighted test, which was tested 4 times) with test batches of 30 randomly selected lemmas from the dev set. Highlighted test resulted in highest scores and parameters were used for final testing. High frequency: total appearances > 100; Medium frequency: 50 <= total appearances <= 100; Low frequency: total appearances < 50. Mask refers to mask tokens [MASK] used in place of missing words in example sentences. Abbreviations used: Rand (random lemmas not sorted by frequency), Lang. (language), G (generic negative reinforcement prompts), S (small correction negative reinforcement prompts), and C (common mistake negative reinforcement prompts).

Table 1 visualizes the various strategies we used to filter and order the data in the batch-formation process ('Batch Properties' section) as well as some features we implemented through prompt engineering ('Features' and 'Reinforcement') and the resulting accuracies we obtained in the tests. Each accuracy is computed by averaging accuracies from two tests (with the exception of the highlighted test). All tests in the table are ran on OpenAI's ChatGPT-4 Mini model.

To create our submission, we ran the pipeline on the test data using the batch parameters that had the best results when testing on the dev set.

5 Results

Our results take the form of accuracies representing the portion of words that the model was accurately able to lemmatize. These values are shown in Table 1 as mean accuracies, with each test being run twice. The exception is the highlighted test. It was our highest scoring test, and was tested four times instead to confirm its performance before final testing. The performance of our model varied greatly between tests, which is an indicator that there is more work that can be done here to stabilize and increase our accuracies.

Despite the low accuracy, our work on this task still showed how different data analysis and prompt engineering strategies can improve LLM performance. For example, our tests demonstrated that performance dropped when the example sentences were not included, and increased when telling the model its answers were incorrect or alternating between sending statement and question prompts.

6 Resource Limitations

In the process of training our model and performing error analysis, we ran into several limitations. This included time constraints as well as not having access to a language expert to answer language-specific questions. An expert's guidance could lead to the realization of relevant features and context not implemented in our project. The biggest issue we ran into was pricing of various models. Our configurations display results from ChatGPT-4 Mini, but we also ran tests on OpenAI's ChatGPT-4, Anthropic's Claude 3.5 Sonnet, and DeepSeek's DeepSeek Chat. Our best results came from running tests with the Claude 3.5 Sonnet model, but this also ended up being the most expensive option. Since our final testing would need to send a large amount of tokens, Claude required resources beyond those allocated for this task.

7 Conclusion

Our team began this task with the goal of applying few-shot learning techniques and theories to the lemmatization of cuneiform languages. The success in this project comes from the support of those theories in our results. Positive reinforcement proved ineffective at increasing accuracy while negative reinforcement, relevant context, and a balance of explicit rules and testing the model during training was effective at increasing accuracy. The highest accuracy configuration, along with template prompting, demonstrates these findings. Accuracy could increase under the same configuration applied to different AI models such as Claude 3.5 Sonnet or OpenAI's GPT-4 as well as with relevant context provided by a language expert to implement into our prompts and pipeline.

8 Future Work

With a better understanding of the lemmatization task and obstacles encountered, we feel that we have ideas that could improve the accuracy of our model greatly. Ideally, we would like to acquire the necessary funding to run more tests using the Claude 3.5 Sonnet model. Additionally, we would like to implement other features that we predict would increase accuracy as well.

Reflection prompts represent a form of chain-of-thought prompting, which would encourage the model to state its 'reasoning' for the response it gave. This would demonstrate its ability to extract lemmatization rules from latent space, hidden features, and rules that are present but cannot be directly observed in the data. With this data to analyze further, we could implement another layer of positive and negative reinforcement that addresses chain-of-thought prompting (Wei 2022). This would allow us to improve accuracy by supporting the formation of outer clusters in the model's hierarchal clustering of cuneiform language grammar rules. The model can then apply these rules to new and untested clean values in order to more accurately determine their lemmas.

Another idea we wanted to implement is soft prompting where the model is trained on prompts produced by other LLMs based on clean values and other features. This application could lead to better prompts that convey the data to the model without the inclusion of extraneous and potentially misleading information due to risk of human error. Soft prompting has not been tested in this context and could has the possibility to lead to higher accuracy compared to human produced prompts (Bulat 2024).

A Appendix

A.1 Prompt Rules

Purpose	Symbol	Description
Param	[P]	Establishes a field that requires input
Instruct	I	Gives instructions to the LLM
Training	RP	Instills rules to the LLM via training
PosConf	PC	Sends positive reinforcement
NegConf	NC	Sends negative reinforcement
Testing	EP	Asks the LLM to perform a task

A.2 Prompts

Rule	ID	Prompt
I	1	The following is a conversation between two Akkadian language experts. One guesses the lemma of a provided clean value while the other indicates whether they are correct or not. Using your knowledge of linguistic analysis and the information shared in this conversation, you will perform the task of identifying the lemmas of words from Akkadian.
I	2	A lemma is defined as the root form of a word without conjugation. Also known as one that would be listed in a dictionary entry for the word.
I	3	You will be given the word which you need to identify. Sometimes you will be given contextual information such as the language the word is found in as well as an example of its use in a sentence.
I	4	Return a single word without explanation nor formatting when asked for the lemma of a word.
RP	10	The lemma of [P] is [P].
EP	11	What is the lemma of [P]?
RP	12	This word is found in the language of [P].
RP	13	An example sentence using this word is [P].
NC	14	Your guess is incorrect. The lemma of [P] is [P].
NC	15	The correct lemma is slightly different.
NC	16	When given words whose lemma is [P], you commonly guess the lemma [P] instead.
PC	17	Your guess is correct.

References

Adhikari, Tapomoy. The Impact of Positive Reinforcement on AI Decision-Making Processes, 15 Oct. 2024, https://doi.org/10.20944/preprints202410.1031.v1.

Bulat, A. and G. Tzimiropoulos. (2024). Language-Aware Soft Prompting: Text-to-Text Optimization for Few- and Zero-Shot Adaptation of V &L Models. International Journal of Computer Vision, 132(4), 1108–1125. https://doi.org/10.1007/s11263-023-01904-9

Luo, Liang, et al. "Unsupervised Sumerian Personal Name Recognition." 193 Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference.

Sahala, A.J Aleksi, and Kristen Lindén. A Neural Pipeline for POS-Tagging and Lemmatizing Cuneiform Languages, aclanthology.org/2023.alp-1.23.pdf. Accessed 12 Dec. 2024.

Wang, Guanghai, and Yudong Liu and James Hearne. "Few-shot learning for Sumerian named entity recognition." Proceedings of the Third Workshop on Deep Learning for Low-Resource Natural Language Processing, 2022, pp. 136–145, https://doi.org/10.18653/v1/2022.deeplo-1.15.

Wei, J., et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. 2022. https://doi.org/10.48550/arxiv.2201.11903

White, Jules, et al. "A Prompt Pattern Catalog to Enhance Prompt Engineering with Chat gpt." arXiv.Org, 21 Feb. 2023, arxiv.org/abs/2302.11382.