An Exploration of Reinforcement Learning for Tokenization

Paul Mello
University of Nevada, Reno
pmello@unr.edu

Gino Nicosia
University of Nevada, Reno
gnicosia@unr.edu

Ankita Shulka
University of Nevada, Reno
ankitas@unr.edu

Abstract

With the widespread ubiquity of Large Language Models (LLMs), optimizing these models have become critical research directions, with tokenization being a significant, but under represented factor in model performance and efficiency. Traditional tokenization methods, such as Byte Pair Encoding (BPE), often rely on static rules that fail to generalize effectively across languages and tasks, particularly in capturing subword or morphological structures. In this work, we introduce a Reinforcement Learning (RL) framework for dynamic tokenizer optimization, leveraging a transformer encoder policy network trained using Proximal Policy Optimization (PPO). We define our tokenizer framework as a sequential decision-making process where the model, at each step, selects between splitting the text at the current position or continuing to form a longer token. This enables tokenization adaptation based on linguistic context. To address the computational challenges of training sequential RL models, we implement a vectorized environment, effectively parallelizing tokenization, improving both scalability and reducing required resources. We evaluate the embedding space of our model utilizing analogous word tasks. Through evaluations we find our framework fails to learn a relevant embedding space and subsequently detail points of failure and how one may improve this framework for future neural tokenizers. Our code can be found here: GitHub/RLToken

1 Introduction

Tokenization is a universally necessary and crucial component of natural language processing (NLP)(Ali et al., 2024), converting raw text to discrete tokens (Rajaraman et al., 2024). Tokenization directly contributes to how LLMs represent the data in an embedding space through defining data representation and processing. While traditional methods like BPE (Sennrich et al., 2016) and WordPiece (Devlin et al., 2019) have become widely adopted [(Radford et al., 2019), (Liu et al., 2019), (Lewis et al., 2020)] for their scalability and simplicity. Despite this, their rigid structure is a bottleneck that constrains performance .

Effective tokenization is a challenging, but important component of a models ability to generalize beyond training data. Recent work has show that without tokenization, models struggle to learn the appropriate distributions, especially when data is generated through Markov processes (Rajaraman et al., 2024), as we do in this work. Traditional tokenization strategies, such as BPE (Sennrich et al., 2016) and WordPiece (Devlin et al., 2019), reduce the generated vocabulary through reducing informational losses. The PathPiece tokenizer (Schmidt et al., 2024), compresses token counts by pruning informationally sparse tokens. They find that fewer tokens do not inherently lead to better performance, revealing that other factors, such as pre-tokenization and vocabulary construction, also play a significant role. These works highlights the present research gaps in our understanding of tokenizers and demonstrates the complexity of designing efficient tokenization strategies.

Inspired by BPE and (Rajaraman et al., 2024), we introduce a learnable RL tokenizer framework (TokenRL) which aims to leverage the power of RL to the task of tokenization. We leverage unsupervised learning to the task of generating token segmentation, allowing for dynamic tailoring of tokenization strategies for shifting contexts, ultimately allowing for semantically informed token generation. To do this, we define the generation of tokenization into a sequential decision making process. To tackle the computational inefficiencies introduced by sequential processing, we vectorize the environment, allowing for parallelization during training. This parallelization is possible in a Markov process since the sequence length means nothing for learning token distribution (Radford et al., 2019).

TokenRL utilizes a transformer encoder policy network guided by PPO to predict token segmentation through a reward system. This system rewards the creating of longer tokens and higher frequency tokens, while penalizing short and unused token predictions. In order to efficiently evaluate TokenRL's performance, we test the embedding space on analogy-based reasoning to which measures semantic and syntactic relationships between words. We find that, our model fails to learn the generation of high quality token sequences.

Our contributions are as follows:

1. We propose TokenRL, a novel framework which formulates tokenization as a RL task, enabling trainable tokenization strategies.
2. We design a simple reward shaping mechanism which rewards longer tokens and penalizes short ones
3. We find that our model fails to learn any relevant tokenization and conclude that our methodology is insufficient to develop a strong neural tokenizer.

The rest of this paper is organized as follows: In section two we cover related work by categorizing tokenization into three distinct categories defined by how semantically meaningful text is segmented. In section three, we define our methodology consisting of PPO, parameter settings and our model architecture. Section four then describes the experimental setup consisting of analogous words to evaluate our generated embedding space. Section five then details the results of section four illustrating the failure of our model to learn and providing some insights into potential reasoning for failure. In section six we detail discussions and future work to expand our research. Finally, section seven concludes our paper summarizing key takeaways in our conclusion.

2 Related Work

In this section, we discuss the three types of tokenization word-level (Guo, 1997), character-level (El Boukkouri et al., 2020), and subword-level (Devlin et al., 2019). In the following subsections, we detail these approaches and highlight some challenges.

2.1 Word Level Tokenization

Word-level tokenization segments text into complete words. Traditional word-level tokenizers include Penn Treebank tokenization (Marcus et al., 1993) which was the de-facto tokenizer for decades. These tokenization techniques are naive approaches which assume the full word unit contains sufficient and distinctive information. While this approach is straightforward and computationally inexpensive, word-level tokenization often fails to capture finer linguistic structures, especially when words are found as substrings to out-of-vocabulary words (Nayak et al., 2020).

2.2 Subword Level Tokenization

Subword tokenization is an NLP method which decomposes text into smaller subword units which offer granular details including meaningful units such as roots, prefixes, and suffixes. Popular subword algorithms include BPE (Sennrich et al., 2016), WordPiece (Devlin et al., 2019), and Unigram (Kudo, 2018) which develop a vocabulary from a corpus. These methods iteratively merge character sequences to build compact vocabularies which prioritize frequent patterns. Moreover, this process allows for out-of-vocabulary words to be understood by the tokenizers by treating them as recognizable patterns. Despite their widespread usage, these methods have been shown to result in suboptimal word representations (Rajaraman et al., 2024). These include data imbalance in multilingual datasets which contribute to tokenizers becoming biased towards larger languages.

2.3 Character Level Tokenization

Character-level tokenization, unlike subword tokenization, decomposes words into their smallest meaningful units defined by characters. These type of approaches allow for tokenization to learn text agnostic structures like that of CharacterBERT (Boukkouri et al., 2020), which replaces the WordPiece tokenizer with a character-level CNN module and significantly improves model performance. These character level approaches have been shown to handle noisy datasets particularly well (Mofijul Islam et al., 2022). Unfortunately, the nature of character-level tokenizers add increasingly more computational power and complexity to these systems.

3 Methodology

3.1 Settings

Relying on character-level tokenization alone is challenging due to target vocabulary sizes and in-

frequent word occurrences. For this reason, we set our ground truth vocabulary as the BPE tokenizer to compare to TokenRL. Both tokenizers are trained on WikiText103 (Merity et al., 2016) with additional preprocessing to remove hyperlinks, non-ascii characters, and more. Our model utlizes only English data due to limited resources and the nature of our RL model learning splits or continuations. We bound the generated vocabulary to be $55k$, which defines the optimal upper-bound of monolingual vocabulary sizes (Ali et al., 2024). This choice severely limits our models learning capabilities, but allows for simpler training dynamics.

3.2 Proximal Policy Optimization

RL is a framework defined by an agent action loop whereby, agents learn to interact with their environment through states, actions, rewards, and a policy network. PPO is a sample efficient variant of RL which balances exploration and exploitation, making it a prime candidate for tokenization where sample efficiency is crucial. Despite this, RL is known to be a computationally intensive learning task due to its sequential operations. To handle this we generate multiple vectorized environments to parallelize the sequential operations, significantly reducing the necessary training times per epoch.

We define our state S_t by the context window C_t surrounding the current token, the previous token that was generated T_{t-1}, and the current token position p_t. Our action space consists of two actions, splitting the current text at the current position, effectively finalizing the token, and continuing the token without splitting.

Reward shaping is comprised of four point, length bonus, frequency bonus, pair frequency bonus, and short token penalty which balances the length and frequency of generated tokens.

$$R_t = R_{\text{len}}(T) + R_{\text{freq}}(T) + R_{\text{pair}}(T) + R_{\text{short}}(T) \quad (1)$$

The losses are computed by the combined policy, value, and entropy losses.

$$L_{\text{total}} = L_{\text{policy}} + \lambda_{\text{value}} L_{\text{value}} - \lambda_{\text{entropy}} L_{\text{entropy}} \quad (2)$$

More loss and reward shaping details can be found in sections A.1 and B respectively.

3.3 Model Architecture

We utilize stacked flash attention transformer encoders from (Dao et al., 2022) using multi-headed self attention. The model inputs are defined by: token embeddings $\mathbf{E}(x)$ which are defined by a dimensionality of d_{model} and a vocabulary $\forall x$, and an attention mask. The attention mask applies infinite values to the locations of padded tokens for the model to ignore. A rotary position encoding (Su et al., 2023) is added to the generated embeddings, to allow the model to learn from the positional information of the token within the text. In the output layers, we pass the output through two linear layers for action probabilities and state values respectively. This produces the expected values and action to be taken by the model at any given step.

4 Experimental Setup

4.1 Model Training

We train our model on the salesforce/wikitext dataset version 103-v1. It consists of more than 100 million tokens in 1.8 million lines of text consisting of varying sequence lengths. We select this data set due to its cleanliness, depth, and monolingual properties which simplify the segmentation strategy we employ.

We train TokenRL for five epochs, two days of wall clock time on a single 4090 Nvidia graphics card. We perform a hyperparameter sweep and find the following settings to achieve the highest accuracy in our limited settings. We set our embedding dimensionality to 128, the layer count to 5, the context window size to 25, the head count to be 8, and the steps-per-episode to 50. These choices allow for training time to be used efficiently, as the reduced model size significantly improves training times and subsequently convergence.

4.2 Evaluation Method

The experiment carried out on our model was the word analogy task of (Pennington et al., 2014). It consists of question in the form of "a is to b as c is to <*prediction*>?". The experiment consisted of 19,544 questions that were further subdivided into semantic or syntactic groups. The semantic set questions, were phrased as follows "Sacramento is to California as Olympia is to <*prediction*>?". This comparison offers an evaluation of intrinsic meaning about the words. While the syntactic set questions focused more on the tense of a word, such as with the example "run is to running as walk is to <*prediction*>?". This research illustrates the syntactic understanding of individual parts of speech within the text.

To compare the predicted embedding and the ground truth embedding space, we test the vectorized differences of the first two word embeddings on the third embedding. This defines the following task Queen $\approx$ King $-$ Man $+$ Woman. This operation models the idea that the relationship between words in embedding space is preserved geometrically, as seen in word2vec. More broadly the equation is defined by eq. 3, where $\vec{e}$ defines an embedding vector.

$$\text{Predicted}_{emb} = \vec{e}_1 - \vec{e}_2 + \vec{e}_3 \tag{3}$$

We compare our TokenRL embedding space against word2vec on this analogous word task and demonstrate our findings in the following results section 5.

5 Results

We evaluate our models on a standard word analogy task and illustrate our findings below in Table 1.

Table 1: Evaluation of embedding spaces on CBOW, Skip Gram, and TokenRL. Given the incredibly low accuracies across all charts, we define our results as a failure to learn the embedding space or a poor testing suite.

Model	Analogies	Accuracy (%)
CBOW	18,966	38.31
Skip Gram	18,966	42.25
TokenRL	10,343	0.02

We find that none of the models, including TokenRL were able to define meaningful relationships in the embedding space. This result may be caused by an improper analogous words task implementation or various factors of our modeling. For example, our action space may not provide enough flexibility for model tractability, the state space may be too large to capture any semantically meaningful information, or even that our model learns to optimize for token segmentation rather than direct prediction of the tokens themselves to learn the resulting tokens generated by the spacing system. This could have also been caused by the use of the wikitext training data for training on all models including CBOW and Skip Gram as it did not provide proper relation between words for the analogy test.

6 Discussion and Future Work

There are many potential reasons as to why this approach did not work. These offer points of improvement which future work may iterate on. For example, our action space is defined by a binary system of split or continue while our state space is defined by dynamic context windows. Both spaces are different with one being incredibly sparse and one incredibly dense. While our model is optimized to allow for meaningful training runs on limited resources, a larger model and larger dataset are almost entirely necessary to capture more meaningful tokens. Despite being unsuccessful, further research may explore utilizing other structured vocabularies or customized vocabularies, or improving on the loss function and reward shaping techniques. While our model fails to learn, we do believe neural tokenizers offer an edge over static tokenizers like BPE and believe future work may find ways to rectify or improve tokenization strategies through black box deep learning methods.

7 Conclusion

In this paper, we introduce TokenRL an RL-based approach for neural tokenization. TokenRL utilizes states to idenitify whether to split or concatenate the current token. This framework is designed to be a rich tokenization strategy by adapting dynamic context windows. We utilize PPO to efficiently sample from the text corpus and generate a rich embedding space through reward shaping and dynamic states. While we failed to create meaningful results, we find the direction of this work promising. Dynamic contextualization of tokens, similar to that of an autoregressive model, may benefit from improved contextualization of local text from a greater corpus. TokenRL offers a framework for further developments in dynamic tokenization, expanding the potential of LLMs in a variety of applications including end-to-end task training.

8 BibTEX Files

Acknowledgments

References

Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Schulze Buschhoff, Charvi Jain, Alexander Arno Weber, Lena Jurkschat, Hammam Abdelwahab, Chelsea John, Pedro Ortiz Suarez, Malte Ostendorff, Samuel Weinbach, Rafet Sifa, Stefan

Kesselheim, and Nicolas Flores-Herr. 2024. Tokenizer choice for llm training: Negligible or crucial? *Preprint*, arXiv:2310.08754.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Junichi Tsujii. 2020. Characterbert: Reconciling elmo and bert for word-level open-vocabulary representations from characters. *Preprint*, arXiv:2010.10392.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Preprint*, arXiv:2205.14135.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Jun'ichi Tsujii. 2020. CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6903–6915, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Jin Guo. 1997. Critical tokenization and its properties. *Comput. Linguist.*, 23(4):569–596.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. 2024. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment. *Preprint*, arXiv:2410.01679.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. *Preprint*, arXiv:1804.10959.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Md Mofijul Islam, Gustavo Aguilar, Pragaash Ponnusamy, Clint Solomon Mathialagan, Chengyuan Ma, and Chenlei Guo. 2022. A vocabulary-free multilingual neural tokenizer for end-to-end task learning. In *Proceedings of the 7th Workshop on Representation Learning for NLP*, pages 91–99, Dublin, Ireland. Association for Computational Linguistics.

Anmol Nayak, Hariprasad Timmapathini, Karthikeyan Ponnalagu, and Vijendran Gopalan Venkoparao. 2020. Domain adaptation challenges of BERT in tokenization and sub-word representations of out-of-vocabulary words. In *Proceedings of the First Workshop on Insights from Negative Results in NLP*, pages 1–5, Online. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. 2024. Toward a theory of tokenization in llms. *ArXiv*, abs/2404.08335.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. *Preprint*, arXiv:1505.04597.

Craig W. Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. Tokenization is more than compression. *ArXiv*, abs/2402.18376.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding. *Preprint*, arXiv:2104.09864.

A Appendix

A.1 Losses

In this framework, loss is defined by three core components, the policy loss L_{policy}, the value loss L_{value}, and the entropy loss L_{entropy}. The value and entropy loss are weighted through λ values to manage their influences.

A.2 Total Loss

The complete loss function is defined below.

$$L_{\text{total}} = L_{\text{policy}} + \lambda_{\text{value}} L_{\text{value}} - \lambda_{\text{entropy}} L_{\text{entropy}} \quad (4)$$

where:

- L_{policy}: Seeks to maximize rewards from selected actions.
- L_{value}: Minimizes the value error estimation by penalizing errors in predicted future rewards.
- L_{entropy}: Penalization factor to reduce greedy model policies.
- λ_{value}: Weight for value loss.
- λ_{entropy}: Weight for entropy loss.

1. Policy Loss: For our choice in policy loss we use a clipped surrogate objective with small updates to our model to ensure stable training. L_{policy} is defined as:

$$L_{\text{policy}} = -E_t \left[\min(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t) \right] \quad (5)$$

where:

- $r_t = \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}$: Ratio of improvement between current and previous policies.
- $A_t = R_t - V(S_t)$: The advantage function which gives preference to actions with higher predicted values.
- $\text{clip}(r_t, 1-\epsilon, 1+\epsilon)$: Bounds r_t updates to the range $[1-\epsilon, 1+\epsilon]$,
- ϵ: A clipping threshold.

2. Value Loss: The value loss is designed to account for future state rewards during prediction.

$$L_{\text{value}} = \frac{1}{2} E_t \left[(V(S_t) - G_t)^2 \right] \quad (6)$$

where:

- $V(S_t)$: Predicted state S_t value from the value network.
- $G_t = R_t + \gamma G_{t+1}$: Defines the discounted rewards from cumulative future states.

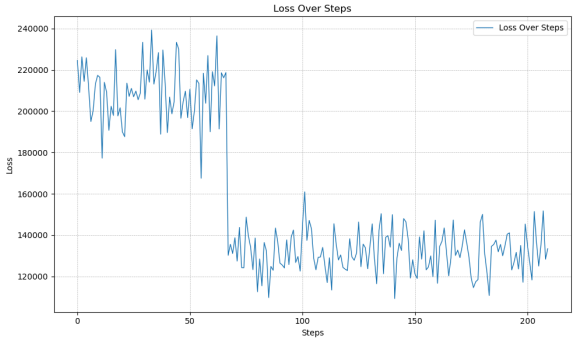

Figure 1: We include an example training run here illustrating losses reducing over the step counts. We set our model to train for 3 epochs, with 3 sub-batch epochs for PPO, a batch size of 512, and a sub-batch size of 256, embedding dimensions of 512, learning rate of $1e^{-}7$, 5 transformer encoder layers with 4 heads, a clip of 0.1, dropout 0.1, and a context window size of 15. For our dataset we train on the full wiki-103 dataset, training was conducted over 2 days of wall-clock-time on an Nvidia RTX4090. Over the course of training steps we can see a distinct drop in losses.

3. Entropy Loss: Since our action space is limited to a binary choice, we introduce an entropy loss which samples the action distribution to penalize the model for being overconfident in its prediction, effectively improving exploration of actions.

$$L_{\text{entropy}} = -E_t \left[H(\pi_\theta(\cdot|S_t)) \right] \quad (7)$$

where:

$$H(\pi_\theta) = -\sum_a \pi_\theta(a|S_t) \log \pi_\theta(a|S_t) \quad (8)$$

- $H(\pi_\theta)$: Entropy of action policy probability distributions.

These loss components contribute to the ability of PPO agents to learn by balancing strategies of exploration, exploitation, and future rewards.

A.3 Final Losses

B Reward Shaping

We design a heuristic reward shaping technique that aims to balance the linguistic properties of the English language with statistical efficiency. We define rewards as the following, which we repeat for clarity:

$$R_t = R_{\text{length}}(T) + R_{\text{freq}}(T) + R_{\text{pair}}(T) + R_{\text{short}}(T) \quad (9)$$

We define each component as follows, using the following notation:

- $w_{\text{freq}}, w_{\text{pair}}, w_{\text{len}}$: Scalars for token frequency, token pair frequency, and token length.
- p_{short}: Scalar that penalizes short tokens.
- $f(T)$: The frequency of token T in the training corpus.
- $f(T_{t-1} + T)$: The frequency of token pairs $T_{t-1} + T$ in the training corpus.

1. Length Bonus: Optimal length is left to the model, but we define reward length bonus to be between $3 - 10$ characters in length to discourage excessively short tokens and excessively long tokens:

$$R_{\text{len}}(T) = \begin{cases} w_{\text{len}} \cdot (|T| - 2), & 3 \leq |T| \leq 10, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

2. Frequency Bonus: Tokens which are frequently used are given more reward to encourage reuse by the model:

$$R_{\text{freq}}(T) = \begin{cases} w_{\text{freq}} \cdot \log(f(T) + 1), & f(T) > 0, \\ -0.1, & \text{otherwise.} \end{cases} \tag{11}$$

3. Pair Frequency Bonus: Here the model is rewarded by the frequency of pairs of tokens whereby the previous token influences the next most likely token. We effectively reward the model for learning the next most likely token below, where $g(x) = f(T_{t-1} + T)$:

$$R_{\text{pair}}(T) = \begin{cases} w_{\text{pair}} \cdot \log(g(x)) + 1), & g(x) > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

4. Short Token Penalty: As this is a character-level subword tokenization method, we add an additional reward to penalize the model for selecting a short token. This helps reduce the number of single character outputs the model has been known to produce.:

$$R_{\text{short}}(T) = \begin{cases} p_{\text{short}}, & |T| < 3, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

These rewards bound the goal of our model to aim for meaningful token sequences while penalizing trivial actions.

C Exploration Notes

Throughout this work, we explored various ablations from replacing our transformer policy network with U-Net (Ronneberger et al., 2015) and even Monte Carlo estimator from a recent paper called VinePPO (Kazemnejad et al., 2024). VinePPO replaces the value network with unbiased Monte Carlo estimator, however when we ran our own experiments, we failed to see any substantive results. Early on, we also utilized a neural network to shape our rewards, but these tests proved too inconsistent. Originally, our research aimed to include end-to-end training of a small LLM with our learnable tokenizer, but these tests failed due to mounting resource requirements and added complexity.