Shedding an OOV Light Into a Blackbox Model's Vocabulary

Anonymous ACL submission

Abstract

Large language models (LLMs) have recently entered the public spotlight as a powerful tool capable of generating fluent, relevant, and coherent text. We expect these models to have a significant societal impact as they are used for downstream tasks; however, research on these models to date has largely focused on English-language tasks, white-box approaches, or both.

In non-English or multilingual language models, one issue - OOV (out of vocabulary), arises frequently in character-diverse languages where tokenizers often do not capture the full range of possible inputs. In the black-box setting the lack of direct access to the LLM's internal representation makes it nontrivial to elicit useful responses to inputs with OOVs or even identify inputs where OOVs are interfering with understanding. In our work, we propose a method of prompt-directed probing to identify OOVs in a multilingual LLM (XGLM-7.5B), and assess a corresponding OOV patch method with a set of machine reading-comprehension (MRC) tasks. Through experiments, we demonstrate that it is possible to both probe and mitigate OOV without access to the internals.

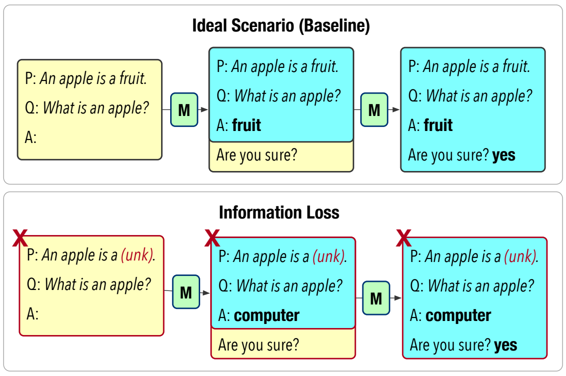

Figure 1: LLM OOV examples, with X indicating bad cases. P, Q, A, and M represent Passage, Question, Answer, and Model respectively. Bold is model generated.

1 Introduction

Large language models (LLMs), such as GPT-3 (Brown et al., 2020) and PaLM (Chowdhery et al., 2022), have been a popular topic of recent discussion not only in the research community but also in the wider public. Trained with incredibly large-scale corpora in a self-supervised manner, these models have enabled groundbreaking advancements in the capabilities of neural methods in natural language processing (NLP). This has become possible due to the highly scalable architecture for general-purpose sequence modeling originally proposed by Vaswani et al. (2017).

These models are trained using mass quantities of text. Two common methods of pre-training are either GPT-like, which generate text given past sequences (Radford et al., 2018; Dai et al., 2019), or BERT-like, filling masked portions of a sequence (Devlin et al., 2019). The product of either method is a model with many uses beyond the original pre-train task. These models can even perform tasks without any fine-tuning when conditioned with an appropriate task prompt (Brown et al., 2020).

While the developments are exciting, the potential of these advances is not yet realized equally across languages. Our work herein explores a multilingual model using a set of machine reading-comprehension (MRC) tasks, and evaluates ways that non-English performance can be improved without requiring direct access to model internals. In particular, we show that tokenization quality is an issue even in large models, and the vocabulary limits imposed by constrained tokenizer sizes harm reliability in character-diverse languages; however, these vocabulary limitations can be overcome with appropriate pre- and post-processing.

2 Motivation

2.1 Tokenization Information Loss

Beyond the general problems of limited data and sampling tradeoffs in a multilingual setting, the properties of the languages themselves may give rise to differences in performance. The specifics of these differences will depend on the particulars of the languages in question; here we explore an issue common to both Japanese and Korean.

Tokenization is the process of transforming input text into a sequence of tokens for a model to consume. In the context of recent neural models, this has shifted from a traditional word-level approach to subword-level tokenization (Sennrich et al., 2016). By operating at the subword level, the tokenizer enjoys both reduced computational cost (Yang et al., 2018) and robustness against unseen words; hence, this has become the standard.

However, subword-level tokenization has its shortcomings. This work focuses on one such limitation: the lack of robustness to out-of-vocabulary tokens in character-diverse languages. In languages that use compact alphabets, such as English, subword tokenization can reliably encode and decode most text without any issues, as the vocabulary-level cost of considering every character in the alphabet is low. This advantage does not hold in character-diverse languages such as Chinese, Japanese, and Korean (CJK), as even at a character level the tokenizer must still consider thousands of tokens. Even considering only character-level bigrams, the resulting combinatorial explosion renders the vocabulary challenging to fully cover in a neural language model. For these reasons, it is common for models to lossily capture the spectrum of possible text. When the tokenizer encounters text outside of the supported spectrum, it will capture an OOV, which results in information loss. This imposes additional challenges in an autoregressive setup, as when generating text, the model will emit OOV tokens in place of any subword it cannot represent. At the point of output, it is no longer possible to discern what the original token was. As an example, a moderately large model such as XGLM 7.5B (Lin et al., 2022) is incapable of outputting the Japanese word for the organ "heart" or the Korean word for "wear out".[1]

This "OOV problem" is further exacerbated in the multilingual setting by the restrictions training imposes on the distribution and sampling of data. Training a multilingual model is not as simple as repurposing existing large-scale data as-is (Conneau et al., 2020); rather, it requires careful planning

[1]This is because the Japanese suffix subword for U+81D3 (organ) and the Korean prefix subword for U+B2F3 (to wear) are missing in the vocabulary.

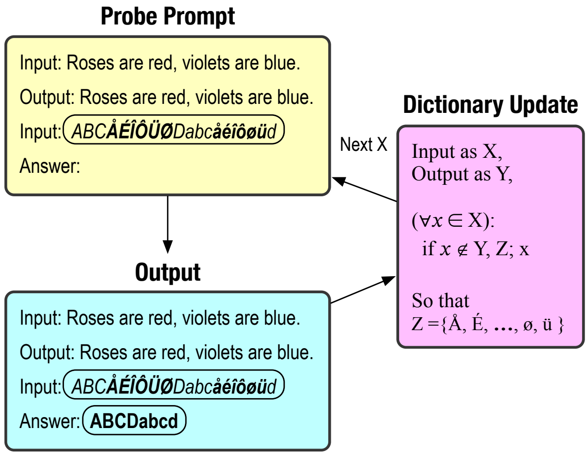

Figure 2: Proposed scheme for probing OOV (3.1) using a text autoencoder prompt against the model. **Bold** characters denote OOV characters for this model.

and allocation of model capacity to ensure that the resulting model's multilingual proficiency inherits the proper imbalance from the training corpus. This is further amplified due to current tokenizer training practices, where it is common to sample the corpus and fix the vocabulary in advance as tokenizer training schemes are less scalable than the language models themselves. In character-diverse languages, this sampling can result in certain characters present in the larger corpus missing in training (Wang et al., 2019; Moon and Okazaki, 2020), leading to additional OOVs and further loss of interpretive and expressive capacity.

2.2 Zero-shot Methods and Black-box Models

Traditionally, the most common use of a language model was for autocompletion of sentences. Recent findings (Brown et al., 2020) have shown that with a carefully engineered **prompt**, it is possible to use a language model for many downstream applications without task-specific training (which would require access to the model weights and significant computational resources). This opens up an opportunity to provide a well-trained general-use model through an abstraction, such as through an API.

As a side effect, many recent model releases are not openly available for vocabulary investigation. In light of this trend, methods we hope to apply to such systems must be viable even without visibility into model internals (e.g., the tokenizer or intermediate layer activations). Instead, the behavior of these models must be treated as a black-box transformation from an input to an output. In this work, we focus on methods that can be applied at the prompt level – by inspecting and adjusting the inputs to or outputs from the model.

3 Detecting and Repairing OOV

3.1 Vocabulary Limits

Given that a model is effectively a function which takes input and produces output, if the input is faulty, it is highly likely that the output will also be faulty. We hypothesize that **information loss** at the input level can result in faulty output. Following this logic, and continuing the discussion from Section 2.1, OOV can pose a problem in any character-diverse language. We hypothesize that even in a moderately large multilingual model such as XGLM 7.5B (Lin et al., 2022), we can expect OOV due to the number of languages the model supports, along with pre-train corpus sampling.

In a conventional setup, OOV is detectable by directly applying the model's tokenizer to the input and searching for the reserved OOV token. In a black-box setup, the tokenizer is not directly accessible, and the full model output will likely remove OOV tokens as part of the output post-processing pipeline. We propose a prompt-based method to work around this limitation and probe missing tokens in the vocabulary indirectly. Our method uses a prompt that conditions the model as a text autoencoder. The prompt up to p_k includes few-shot examples to condition the model, and the last i tokens are an OOV probe $(p_{k-i+1}, ...p_k)$. We expect to reproduce the probe such that:

$$(p_{k-i+1}, ...p_k) \approx (g_{k+c+1}, ..., g_n)$$

(c is a buffer reserved for the sequence of tokens representing the "Answer:" portion of the prompt.)

In our method, we use sequences of size s across a subset of the Unicode pages of the target OOV probe languages, where the sequence $X_i = (u_i, ...u_{i+s-1})$ is generated by incrementing the Unicode ordinal i. By observing the missing tokens in the output compared to the input, e.g., $Z = \{p_{k-s+1}, ...p_k\} - \{g_{k+c+1}, ..., g_n\}$, we can approximate the OOV tokens and build an OOV vocabulary, as demonstrated in Figure 2.

For our experiments, we set the probe sequence length to $k = 2$ to prevent the model from omitting non-OOV characters in probe sequences with a high proportion of OOVs. This is an expensive process, but it only needs to be run once per target language as the results can be reused across any task for this model and language pair.

3.2 Pre-patching Input OOV

Using the OOV tokens previously acquired through this probing process, we can preprocess the corpus to replace the OOV with a similar token at task generation. Given that the k-th token of an input sequence $(p_1, ...p_k)$ is OOV, we replace the OOV p_k with a substitute token $\hat{p}_k$, constrained such that $(p_{k-1}, \hat{p}_k) \notin p$ and $(\hat{p}_k, p_{k+1}) \notin p$. This substitution allows us to reliably reconstruct $\hat{p}_k$ back to p_k, without the risk of introducing ambiguity in the output by duplicating bi-grams already present elsewhere in the input. We then perform an analogous post-processing step on the output by replacing occurrences of $(p_{k-1}, \hat{p}_k)$ or $(\hat{p}_k, p_{k+1})$ with the original (p_{k-1}, p_k) or $(\hat{p}_k, p_{k+1})$, respectively. A conceptual illustration of the overall process can be seen in Figure 3. We hypothesize that this method can mitigate the information loss.

When selecting the substitute token, we constrain to in-vocabulary tokens and prioritize those roughly from the same Unicode page. If a substitute was not found on the same page, the algorithm will search from the first page of the target language. Abrupt code-switching is disruptive to the overall output distribution, as the conditional probability of p_k being of a different language than the context $(p_1, ...p_{k-1})$ is low; ensuring we select an in-language substitute minimizes this disruption.

4 Experiments

All of the tasks for the different languages were run against an XGLM 7.5B (Lin et al., 2022) model in a zero-shot or few-shot setup, with no fine-tuning. The model was encapsulated in an API server to simulate a black-box environment, only providing a mechanism for prompting, along with an interface for setting the temperature, top-p, and repetition penalty. We constructed the OOV probe using a few-shot setup: two exemplars randomly selected from the task language's Wikipedia, fixed for all runs, followed by the character(s) we wished to probe. We evaluated the value of OOV as a predictor of correctness, and the efficacy of the OOV patch procedure, using zero-shot tasks with the prompt in Figure 1. Each MRC item was evaluated 5 times, and the metrics were averaged.

4.1 Tasks and Datasets

We evaluated the correctness of patched and unpatched answers over five runs on Japanese-language JSQuAD from Kurihara et al. (2022) (val-

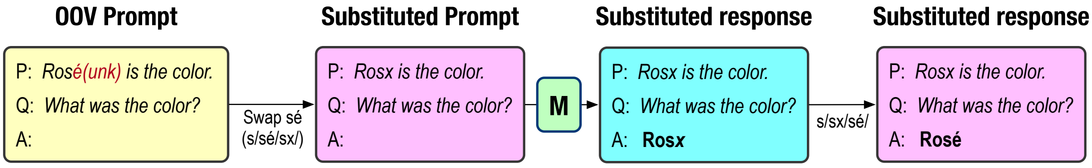

Figure 3: OOV substitution process. Given p_k =é, p_{k-1} =s, the bigram **sx** is chosen as it satisfies the condition $(p_{k-1}, \hat{p}_k) \notin p$, which prevents unintended substitution during post-process, when replacing back to **sé** in the output.

idation set), and Korean-language KorQuAD 1.0 (dev set). This particular task was chosen as it was one of the few tasks available in both languages, with roughly similar complexity. In these experiments, the model was presented with the context paragraph followed by a question through a prompt template, in a zero-shot setup with no training.

The standard evaluations for question-answering datasets, where available, are not well-suited to zero-shot evaluation. Strict evaluation runs the risk of eliminating semantically correct answers (such as "Normandy is located in France" when just "France" is expected), but overly permissive evaluation may allow a rambling LM to eventually "get lucky" and emit a correct answer as part of its output. To work around these limitations we evaluated the correctness of answers according to the following standardized rule: an answer is correct if it contains any one of the possible correct answers in its first N characters, where N is calculated as 2 $\times$ (max correct answer length). We found this to establish a balance between incorrectly rejected and incorrectly accepted answers.

5 Results

Probing for OOV was done using a two-shot translated autoencoder prompt. We observed this method to be extremely reliable, particularly with a short probe sequence length. With $k = 2$, used for all experiments below, we observed a vocabulary coverage accuracy $\frac{TP+TN}{|dataset|}$ of 99.999% in Korean and 100% in Japanese respectively. These results suggest that our method is highly effective at detecting OOV in a black-box model's vocabulary.

We identified specific cases where the model should be unlikely to produce an answer as the correct answer contains OOV: 348 in KorQuAD and 196 in JSQuAD. With no special modifications, the model scored an average of 0% on this KorQuAD subset and 9.76% on the corresponding JSQuAD subset. After OOV patching, the model produced a correct answer for an average of 42.8% of these "impossible" KorQuAD questions (44.1% for Korean prompts) and 44.8% of the "impossible" JSQuAD questions (49.2% for Japanese prompts), as seen in Table 1. While the ratio of impossible questions is fairly low (4% on JSQuAD, 5% on KorQuAD), we see modest macro-level improvements of 1.17% on JSQuAD and 2.21% on KorQuAD respectively with our method.

Variant	OOV	Correct	Wrong	Acc.
JSQuAD-en	Control	19.8	176.2	0.101
JSQuAD-en	Patched	87.2	108.8	0.445
JSQuAD-ja	Control	19.4	176.6	0.099
JSQuAD-ja	Patched	96.2	99.8	0.491
KorQuAD-en	Control	0	348	0.000
KorQuAD-en	Patched	148.8	199.2	0.428
KorQuAD-ko	Control	0	348	0.000
KorQuAD-ko	Patched	153.4	194.6	0.441

Table 1: Results for OOV samples. JSQuAD and KorQuAD had 196 (4.3%) and 348 (6.0%) samples respectively. Results are mean from 5 runs. OOV indicates ratio of samples with OOV in prompt/answer.

6 Conclusion

In this work, we demonstrate the impact of the OOV problem on a pre-trained, multilingual language model, and explore ways to work around those limitations exclusively by through prompts. Using a chain of simple methods, we show how it is possible to not only externally identify OOV tokens without access to the model's tokenizer, but also to mitigate their effects through pre- and post-processing. We share our findings and a path for downstream applications to mitigate OOV in similar setups.

While the findings in our work are still preliminary, we believe they provide insights into the limitations of current models. We hope that future research will apply and expand these techniques to both meaningfully inform users of potentially-incorrect output and also improve downstream performance when utilizing black-box models.

Limitations

One crucial detail is that our findings are not universally applicable to language models of all kinds. We have only evaluated our approach with a relatively small (7.5B) multilingual model.

In particular, the OOV problem we discuss in this work is common in smaller (Under 100B) models, but does not affect all multilingual language models. In particular, one counterexample is GPT-3 (Brown et al., 2020), which will fall back to operating at byte level (Gillick et al., 2016; Xue et al., 2022) if there is no corresponding subword token in the vocabulary. This is an effective method which other models do not commonly employ. There is a trade-off of compute cost with this approach, as it can increase the sequence length significantly.

Additionally, our experiments have yet to be validated across an exhaustive set of languages. In particular, one target language that was omitted was Chinese. This was a conscious decision, as the authors had limited knowledge of the language - which would have made it challenging to provide a fair setup with regard to the translated experiments. Another challenge was the lack of a task with meaningful amounts of OOV.

Ethical Statement

As with most topics surrounding the ethics of machine learning, this section is far from exhaustive. We will only discuss some key aspects and potential impacts of our work here.

First and foremost, our work proposes probing a black-box language model's limitations. As of the time of writing, even in a white-box setup, there is no reliable way to detect a model's limitations and, as a result, it requires validation of the output utilizing external knowledge or human validation. Given that our work is in an even further constrained setup, the robustness of our method is likely to be even more limited and should only be used with a clear understanding that it is only an approximation with no guarantees - and we have attempted to make this clear in the section discussing our limitations.

Additionally, from an environmental perspective, the approaches discussed in this work are likely positive; as the result of this is increasing the utility (indirectly, by making the generations usable through better validation) of smaller models. Popular models such as GPT-3 have already exceeded the 100B parameter milestone, which can never be as energy efficient as a model which is less than 5% of its size in terms of parameters.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context.

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.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. Multilingual language processing from bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1296–1306, San Diego, California. Association for Computational Linguistics.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation.

Kentaro Kurihara, Daisuke Kawahara, and Tomohide Shibata. 2022. JGLUE: Japanese general language understanding evaluation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2957–2966, Marseille, France. European Language Resources Association.

Xi Victoria Lin, Todor Mihaylov, Mikel Artetxe, Tianlu Wang, Shuohui Chen, Daniel Simig, Myle Ott, Naman Goyal, Shruti Bhosale, Jingfei Du, Ramakanth Pasunuru, Sam Shleifer, Punit Singh Koura, Vishrav Chaudhary, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Zornitsa Kozareva, Mona Diab, Veselin Stoyanov, and Xian Li. 2022. Few-shot learning with multilingual generative language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9019–9052, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Sangwhan Moon and Naoaki Okazaki. 2020. PatchBERT: Just-in-time, out-of-vocabulary patching. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7846–7852, Online. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

Hai Wang, Dian Yu, Kai Sun, Jianshu Chen, and Dong Yu. 2019. Improving pre-trained multilingual model with vocabulary expansion. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 316–327, Hong Kong, China. Association for Computational Linguistics.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2018. Breaking the softmax bottleneck: A high-rank rnn language model.

A Reproducibility Checklist

The following section discloses the required information to reproduce our experiments. Training-related details are not disclosed, as our work involves no training.

A.1 Experiment Setup

All of the experiments were run on one of the following computing environments:

- A: AMD Ryzen 7 3700X, Nvidia RTX A6000, 64GB RAM
- B: Intel Xeon Platinum 8360Y, Nvidia RTX A100, 64GB RAM

A.2 Compute Cost Estimation

Environment A is a workstation, with a power supply rated at 550W. Environment B is a shared node, with resources segregated between users. While the entire node's entire power is estimated at 2.2KW of thermal design power (TDP), our experiments only used 1/4 of the node's resources, as all of our experiments were executed on a single GPU. Following that logic, the peak energy use for both compute environments can both be estimated at an upper boundary of around 550W TDP. Only one model (XGLM 7.5B) was used for all experiments, running in half-precision during inference.

The experiments can be reproduced in a single run, translating to 12 hours, and the comparison requires 5 runs each across 4 variants, requiring 120 hours. [2].

A.3 Hyperparameters

Repetition penalty is as proposed in CTRL (Keskar et al., 2019). In this work, we used a temperature of 0.2, top_p of 1.0, and a repetition penalty of 1.0.

[2]This does not factor in failed experiments and is an approximation of reproducing the exact experiments disclosed in this paper.

A.4 Datasets and Models

All datasets and models we have used in this work are publicly available and downloadable through the following URLs.

- https://github.com/yahoojapan/JGLUE
- https://korquad.github.io/KorQuad 1.0/
- https://huggingface.co/facebook/xglm-7.5B

Our work does not involve the standard train-val-test flow due to it being a zero-shot or few-shot setup. Due to this, we did not use the standard evaluation. The scripts used to run the experiments against a subset of the dev sets. The programs and raw data used to subset the datasets, evaluate task performance, and compute results disclosed in the paper are in the experiment code package.

A.5 Prompts

The exact prompts, including the translated variants are available in https://pastebin.com/KbnY7zCw.[3]

B Negative Results

This section covers the more notable failed experiments performed as potential incremental enhancements to the methods proposed in our work.

B.1 Larger OOV Probes

We observed that in our initial probe experiments with $k = 30$, the error rate went up noticeably compared to $k = 2$, due to the model producing an erroneous output when the probe had a significantly higher ratio of OOV tokens compared to valid tokens. This resulted in tokens existing in the input sequence being omitted, resulting in false positives - in-vocabulary tokens incorrectly being detected as OOV. For example, in Korean, across the entire spectrum of 11,173 probe tokens, we observed 80 false positives with $k = 30$, while there was only one case in $k = 2$. Similarly, in 21,103 probe tokens for Japanese, we observed no false positives with $k = 2$; we terminated the $k = 30$ run after 2300 probe tokens, with 100+ false positives detected thus far. While this is not truly a negative result, it does suggest a trade-off between accuracy and probe computation cost – in light of the fact that the probe cost is incurred only once for a given model/language pair, the balance of this trade-off weighed in favor of shorter probe lengths for our use case, and we suspect this will be true for most others as well.

B.2 Contextual Trigram Substitutions

As we are operating at character level in the OOV experiments[4], there is a non-zero possibility that the generated output may contain a bigram that can be damaged by our bigram-based OOV post-processor. While qualitative analysis did not show any obvious cases where this is happening, as a safety measure we also extended the experiments to use trigrams instead of bigrams. The use of trigrams did not seem to eliminate any undesirable postprocessing, and it *did* eliminate some correct postprocessing (most notably where an OOV character appeared in the prompt next to a punctuation mark), so in the final analysis we consider this inferior to the bigram-based solution.

[3] This will be part of the final source release, and has only been disclosed in this format for review purposes during the anonymity period. The source release will have a permissive, 3-clause BSD license.

[4] This is likely a side effect of the Unicode-level alphabet for the corresponding language being overly diverse, and not intentional.