Adabot: Fault-Tolerant Java Decompiler (Student Abstract)

Zhiming Li,[1*] **Qing Wu,**[1*] **Kun Qian**[2*]
China University of Geosciences,Wuhan 430074,China
[1]{zhimingli, iwuqing}@cug.edu.cn; [2]kun_qian@foxmail.com

Abstract

Reverse Engineering has been an extremely important field in software engineering, it helps us to better understand and analyze the internal architecture and interrealtions of executables. Classical Java reverse engineering task includes disassembly and decompilation. Traditional Abstract Syntax Tree (AST) based disassemblers and decompilers are strictly rule defined and thus highly fault intolerant when bytecode obfuscation were introduced for safety concern. In this work, we view decompilation as a statistical machine translation task and propose a decompilation framework which is fully based on self-attention mechanism. Through better adaption to the linguistic uniqueness of bytecode, our model fully outperforms rule-based models and previous works based on recurrence mechanism.

Motivation

In order to build a fault-tolerant decompiler, we need to take both the lexical and structural (syntactic) information into consideration. Specifically, we need to combine the lexical information extracted from the bytecode with the corresponding structural information to form grammatically readable source code. In our work, we view reverse engineering as a statistical machine translation task instead of rule-based task and propose a fault-tolerant Java decompiler based on self-attention mechanism. Although decompilation of programming language appears similar to machine translation of natural language, they are actually different in many ways, which are as:

1. **Syntax Structure:** Programming language is rigorously structured (Hellendoorn and Devanbu 2017; Hu et al. 2018) that any obfuscation in source bytecode is significant enough to make an AST based decompiler invalid.
2. **Word Unit:** Programming language is less likely to suffer from out-of-vocabulary problem since its vocabulary size is relatively small and can be exhaustively learned.
3. **Vocabulary Distribution:** The frequency distribution of words in the vocabulary of programming language is large in variance. Specifically, it is comparatively lower in entropy, larger in redundancy and suffers from unbalanced distribution of information. The detail is illustrated in Tabel 1 .

Table 1: Comparison of languages in terms of entropy and redundancy

Language	Entropy (bit)	Redundancy (%)
Natural Language (English)	11.82	0.09
Source Code	6.41	0.40
Bytecode	5.31	0.28

It is apparent that the programming languages in our dataset, either source code or bytecode has an extremely unbalanced distribution of information in vocabulary. Concretely, structural information like keywords, which only account for 0.4% of the vocabulary, significantly contribute about 9% to the overall redundancy.

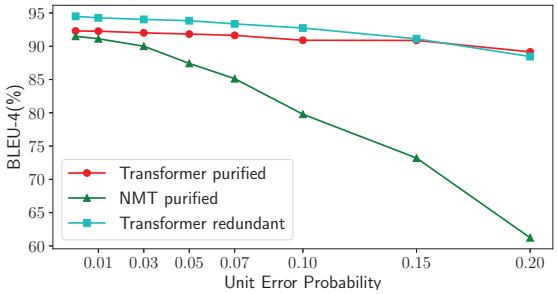
Methodology and Experiments

To handle the above mentioned problems, previous works have applied the recurrence and attention mechinasm in NMT model (Luong, Pham, and Manning 2015). While we propose fully basing our model on self-attention mechanism introduced by Transformer (Vaswani et al. 2017) model can better adapt to the linguistic uniqueness of programming language. The experiment mainly consists of three parts:

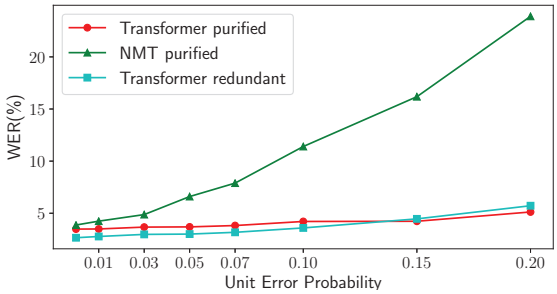
1. **Data Preprocessing:** We first crawl the original parallel corpus from Java 11 API offered by Oracle[1] and reflect them from the local packages with Java reflection mechanism. Finally compile them into corresponding bytecode with format templates.
2. **Purification:** In order to boost the potential of recurrence based model and better compare it with attention based model, we attach a manual purification step for it. Specifically, this step helps to acquire the purified dataset by removing a large proportion of structural information, which significantly alleviates the vanishing gradient problem of recurrence based model.

[*]Zhiming Li, Qing Wu and Kun Qian are co-first authors.

[1]https://docs.oracle.com/en/java/javase/11/docs/api/index.html



(a) Using BLEU-4 for evaluation



(b) Using WER for evaluation

Figure 1: Performance of attention-based NMT and Transformer models on the purified and redundant dataset with salt-and-pepper noise introduced. The unit error probability of the noise ranges from 1% to 20%.

3. **Training and Evaluation:** We evaluate our model on three tasks, including: overall performance of decompilation on purified and redundant datasets, fault tolerance and the impact of different word segmentation algorithms.

Table 2: Performance of attention-based NMT and Transformer models on the purified and redundant datasets

TASK	BLEU-4(%)	WER(%)
NMT purified	91.50	3.87
Transformer purified	92.30	3.48
NMT redundant	27.80	65.53
Transformer redundant	94.50	2.65

Overall Performance As Table 2 illustrated, the performance of attention-based NMT and Transformer are similar on the purified dataset. However, Transformer completely outweighs attention-based NMT on the redundant dataset since attention is only used as an auxiliary measure to alleviate the vanishing gradient problem of NMT and is only powerful enough to help it learn the structural information but not the lexical information. Therefore, we concluded that attention mechanism can better adapt to the linguistic uniqueness of programming language than recurrence mechanism.

Fault Tolerance We perform experiments on test set with salt-and-pepper noise introduced to demonstrate the fault-tolerance of our model. The unit error probability (UEP) of the noise ranges from 1% to 20%. As Figure 1 illustrated, Transformer presents strong fault-tolerance on both the purified and redundant datasets. Whereas, attention-based NMT compromised quickly with the increase of UEP. Conclusively, Transformer is much more robust and stabilized than attention-based NMT on the obfuscated decompilation task.

Impact of Word Segmentation We evaluate the performance of Transformer on purified dataset with different word segmentation algorithms, including space delimiter and subword model based on byte pair encoding (BPE). As Table 3 illustrated, using space delimiter for word segmentation performs better than using subword model by both metrics. It is because programming language has a relatively small and exhaustive vocabulary, thus less likely to encounter out-of-vocabulary problem.

Table 3: Impact of different word segmentation algorithms

ALGORITHM	BLEU-4(%)	WER(%)
space delimiter	92.30	3.48
subword model based on BPE	87.15	5.74

Conclusion and Future Work

In this paper, we propose a statistical, self-attention based Java decompiler and compare it with previous works based on recurrence mechanism. Experimental results demonstrate that our approach not only performs well on common decompilation task, but also presents strong fault-tolerance and robustness when bytecode obfuscation were introduced.

For the future work, we plan to perform experiments on our model with longer, more randomized code snippets from real-world scenarios with more complex obfuscations introduced in order to further verify its robustness and get better prepared for practical use.

References

Hellendoorn, V. J., and Devanbu, P. 2017. Are deep neural networks the best choice for modeling source code? In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 763–773. ACM.

Hu, X.; Li, G.; Xia, X.; Lo, D.; and Jin, Z. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*, 200–210. ACM.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.