# Cross-Architectural Assembly Code Translation Based on Natural Language Processing

**Chun-Sheng Wu**
832001192
CEEN, Texas A&M University
jinsonwu@tamu.edu

## Abstract

Accompanying with the incredible performance of novel ARM structure processors, binary translation (BT) grabs the attention in the industry. Plenty of approaches have been nominated targeting at reinforcing the translation efficiency. Rosetta2, which was published by Apple Inc., gave us a convenient method to execute cross-architectural translation from x86 to ARM structure. The procedure of translation usually takes hours of work though, much room of improvement was left. In this point, people proposed neural machine translation (NMT) aiming to facilitate the translation with the assistance of recurrent neural network (RNN). This project was conducted based on this concept. Optimizations and comparisons employed to improve the training performance, therefore enhancing the translation efficiency.

## 1 Introduction

Binary Translation is a cross-architectural translation strategy implementing on different structures, e.g. x86 and ARM. Conventional BT includes multi-level history tables (MHT), which is composed of entries with instructions, as reference to translate binary codes. There are two types of binary translation, static BT (SBT) and dynamic BT (DBT) respectively. The first one adopts the general method to translate the binary codes without following data dependency inspection and scheduling optimization. To efficiently operate the translation tasks, DBT comes out with various adjustments. Code cache[1][2], sliding window[3], and basic block[4] utilization are approaches augmenting data fetching. Machine manipulates the hardware resources based on previous collected information. Accordingly, improves the translation performance with balanced computing loading on processors. However, machine should request the MHT in countless times to fetch the corresponding binary codes line by line in the execution stage. Continuous data loading costs plenty of time accessing the reference. Therefore, I attempted to resolve this problem with the help of natural language processing (NLP).

There was a critical issue here that sequences of binary codes could not exhibit the semantics inside the logic. More specifically, machine could not easily fetch the correlation between instructions by tons of numbers. Additionally, Binary codes are the lowest level machine code in a hierarchical computer. It would waste time switching programming language and intermediate assembly code to the binary codes at first, then initiating the translation. Hence, I decided to translate the assembly codes to keep semantic equivalence between different instruction set architectures (ISAs) and avoid the unnecessary translation simultaneously.

Followed by was the NLP structure after deciding the input and target codes. I firstly did a comparison between three NLP units, SimpleRNN[5], long short-term memory (LSTM)[6], and gated recurrent unit (GRU)[7]. A small batch of training was adopted to select which unit attained the best loss and cost time per epoch. The selected unit would be used to implement the encoder and decoder layers[8]. I chose the "encoder-attention-decoder" structure with above-mentioned NLP units as the overall processing structure[9]. This structure had the capability to tolerate various sequence length since

instructions in different structures have discrepancies. Utilizing the attention as the intermediate layer could enhance the model dealing with inherent sequences. Thorough comparison between conventioal BT with MHT and neural machine translator with NLP carried out subsequently.

In this report, the motivation and related work concerning with BT and NMT exhibits in 2. In the following is the methodology I used in this project. Firstly, I will introduce the data preprocessing about the methods handling the raw assembly code into tuples suitable for training in tensorflow. Next, three different NLP units and the ultimate NLP structure are shown in 3.2, 3.3. Section 4 contains the details of the experiment. Conclusion and future directions are placed in 5 , 6.

## 2 Background and Related Work

With the iteration of computer architecture, ARM structure has grown its importance in nowadays processor design. Fewer power consumption and more flexible instruction invoking achieves significant performance enhancement without technical node advancements. Nonetheless, the universally-adopted structure is still the complex instruction set computer (CISC) structure, mainly the x86 structure, which is employed in Intel and AMD's processors mostly. Therefore, translating the software in the aspect of low-level structuret with the least harm to fully exploit the benefits brought by ARM structure. There exists several solutions to deal with different types of structure translation. Among them, binary translation is the most popular strategy.

People have dedicated to the optimization of binary translation for years. Conventional BT is conducted statically with the help of MHT. Leveraging BT performance requires a thorough research from data processing to hardware modifications. Data-level parallelism (DLP) enables parallelization in the context searching during binary code decoding. Code cache[1][2], sliding window[3], and basic block[4] utilization are approaches augmenting data fetching. Fine-grain data partitioning is a great candidate handling tasks depending on data dependencies. Furthermore, weighted edge graphs are beneficial for balancing the loadings in processors[10]. Exploit extended single instruction multiple data (SIMD) with prolonged instruction window to better parallelize the binary code processing[11][12]. Operating binary translation dynamically is capable of reducing overhead of resource allocation and power consumption[13]. Hashed-like look-up table benefits the information searching processing without countless requests when browsing desired binary codes[14].

In case that conventional methods for algorithm optimization expects endeavors to achieve significant improvement. Including NLP into the optimization procedure of BT breaks through a milestone recently[15][16]. Machine itself has the capability of predicting targeting binary code segments and arranges the ordering of tasks according to data dependencies through semantics retrieving in RNN. Bilingual machine translation was first implemented in 1990s[17]. It employed a mechanism named seq2seq translation[18], which attracted attentions while NLP progressed in various domains. Specific topics, e.g. Voice detection[19], infant language translation[20], letters formation[21], and sign language Recognition[22] are prospective directions utilizing NLP algorithms.

To have a better classification of semantics inside assembly code, specialized structure is required. In [23], it introduced methods to augment feature extraction with small sample set. In addition to composing meaningful dataset, efficient training on various sequence length was of importance since there were usually different character numbers in instructions. Accordingly, encoder-attention-decoder structure became a great candidate in this project. Bidirectional lookahead encoder and decoder further facilitated the target generation by additionally providing two-way information[24]. [25] recommended a strategy to universally train a NLP with multi-structure input and output. Above-mentioned works all bring possibilities to this project.

## 3 Methodology

The initial part in the workflow is data preprocessing. I will introduce how raw data (input assembly codes) is processed into tuples, which are necessary for traninig in tensorflow. Three types of NLP units used for performance comparison (not the comparison with BT) are instigated in the following. The overall processing structure implementing the major translation learning shows up in the last.

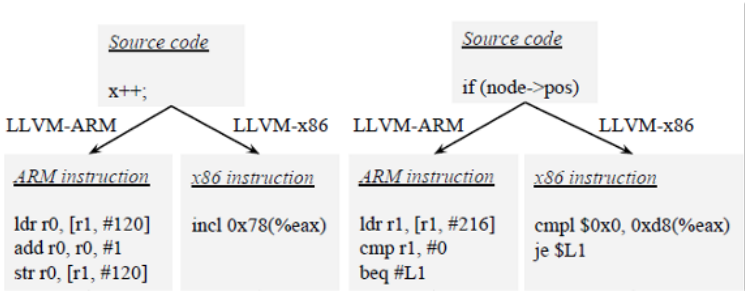


Fig. 1 exhibits the essential difference between two ISAs. Either the number of instructions or the content has huge discrepancies between x86 and ARM structures. This otherwise displays the potential problems about semantic equivalence[26].

### 3.1 Data Preprocessing

Fig. 1 presents raw assembly code respectively in ARM and x86 structure. Although these sequences point to identical operating logics, they put different appearances on their machines. This implies us that it requires additional work to maintain initial meanings while the tuples are formed. Preprocessing is comprised of four major sections. Following introductions will appear in order.

#### 3.1.1 Redundant Information Elimination

Raw assembly codes, which were transferred from original C/C++ codes through GCC/G++ compiler, were crossing with ineffective lines. Therefore, the first priority was to remove the redundant information in the very beginning. Unnecessary punctuations, extra spaces, machine information, and etc. were objects required to be erased from the original codes.

```
.txt .cstring .align 3
lC0: .ascii "w\0" .align 3
lC1: .ascii "q4input.txt\0" .align 3
lC2: .ascii "%d\12\0" .align 3
lC3: .ascii "%f\12\0" .align 3
lC4: .ascii "r\0" .align 3
lC5: .ascii "q4output.txt\0" .text .align 2 .globl _main
_main: LFB29: sub sp, sp, #176
LCFI0: stp x29, x30, [sp, 16]
LCFI1: add x29, sp, 16 str w0, [sp, 44] str x1, [sp, 32] mov w0, 20 str w0, [sp,
 164] adrp x0, lC0@GOTPAGE ldr x1, [x0, lC0@GOTPAGEOFF];momd adrp x0, lC1@GOTPAG
E ldr x0, [x0, lC1@GOTPAGEOFF];momd bl _fopen str x0, [sp, 152] ldr w0, [sp, 164
] str w0, [sp] adrp x0, lC2@GOTPAGE ldr x1, [x0, lC2@GOTPAGEOFF];momd ldr x0, [s
p, 152] bl _fprintf str wzr, [sp, 172] b
```

Fig. 2. Loops and branches were split with the semantics mentioned in the paragraph.

#### 3.1.2 Code Block Construction

Since I chose to apply seq2seq neural machine translation, the codes should be replaced with sequences, named code blocks in this project. I utilized the intrinsic behavior that assembly code was split into loops or branches to accelerate this task. Code blocks were limited to contain the characters beforehand once it encountered specific loop signs. They were usually a combination of dashes and loop names with colons at the end showing in Fig. 2.

Nevertheless, the code block number in ARM (1402) and x86 (788) were not the same to perform tuples. I reorganized the codes with sufficient groups with greatest common denominator (GCD). The GCD of characters in ARM and x86 represented the final group number. Keep padding spaces in the tail of each ARM code block until the final group number is bigger than the root of characters in two structures. Then shuffling the reorganized code blocks could we obtain the desired training dataset constituted with one-to-one sequence pairs.

#### 3.1.3 Tokenization

The last step in data preprocessing was to tokenize the characters and concatenated them to form meaningful sentences. After establishing tuple sets, every character was transformed to an assigned,

unique number as a index determining the output. Spaces were padding between each word. 'Start' and 'End', these two words indicating the beginning and termination of sequences inserted in each code block afterwards.

### 3.2 NLP Unit

Recurrent Neurons are commonly used in natural language processing structures. Recursive learning benefits the convergence speed about text classification. In this section, I will elaborate three prevailing NLP units, respectively SimpleRNN, LSTM, and GRU. Encoder and decoder are both implemented by these units. Fig. 3 demonstrates the internal algorithms in these three NLP units.

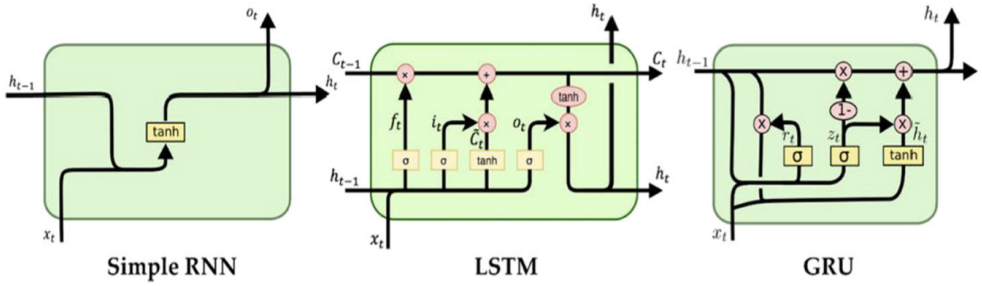


Fig. 3. Internal structure in SimpleRNN, LSTM, GRU from left to right[27].

#### 3.2.1 SimpleRNN

Recurrent neural network exploits recursive neurons between hidden layers to exhibit dynamic temporarily behaviors, e.g. grammar or causality in human languages. This leads to RNN becomes an adequate model for learning natural languages. Eq. 1, 2 point out the iterating behavior of hidden layer vector $h_t$ and output $y_t$.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{1}$$

$$o_t = \sigma_o(W_o h_t + b_o) \tag{2}$$

where $x, o$ are input and output vectors. $W, U, b$ are parameter matrices and vector (hidden weights and bias), $\sigma$ means the activation function. Vectors and output from previous stage (neurons) are fed into the next node as input. Parameters in intermediate hidden layers keep updating through continuous input invoking. Tanh is selected as the activation function in each node.

#### 3.2.2 Long Short-Term Memory

SimpleRNN has a serious problem about gradient updating with small sample set. Learning is dominated by few features and difficult to accommodate sequences with versatile meanings. LSTM specializes the structure with additional input, forget, and output gates. Weights can decide the ratio of data is abandoned and transmitted to the next node. It has similar functionality with Dropout method in deep neural network training. LSTM correspondingly splits the semantics into pieces for better text recognition.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{3}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{4}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{5}$$

where $f, i, o$ represent result from forget, input, and output gates. Updating rule is akin to SimpleRNN, but in three separate parts (gates).

$$\tilde{c_t} = \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \tag{6}$$

$$c_t = f_t \circ c_{t-1} + i_t \cdot \tilde{c_t} \tag{7}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{8}$$

where $\tilde{c}, c$ are served as cell input activation vector and cell state vector. $\sigma_g, \sigma_h$ are activation function with sigmoid and tanh. One thing unique is that it activates hidden layer vector $h_{t-1}$ in advance, times it with updated input $i_t$, then concatenate with the Hadamard product (vector-wise multiplication) of previous cell state vector and updated forget ($f_t \circ c_{t-1}$). Pieceful information learning is thus enhanced through this mechanism. Ultimately, hidden layer vector remains revised with updated output and cell state vector after activation.

#### 3.2.3 Gated Recurrent Unit

LSTM augments learning behavior, but increasing considerable computing needs in the meantime. The total computing loading increases to three-fold more than the original RNN model. A lightweight version is devised recently with some internal integration. It fused gates into update and reset gates controlling data transmission. GRU model strikes a great balance between prediction accuracy and cost time.

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \tag{9}$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \tag{10}$$
$$\tilde{h_t} = \sigma_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{11}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h_t} \tag{12}$$

where $z, r$ are update and reset gate vectors. $\sigma_g, \sigma_h$ exhibit sigmoid and tanh activation functions. $\tilde{h_t}$ stands for candidate activation vector, which has likely function to $\tilde{c_t}$ in LSTM. Different to other two units, GRU exploits XNOR on candidate activation vector and following hidden layer vector iterating. Neuron decides whether which part of data should be evicted, others are broadcasted to neighboring nodes.

### 3.3 NLP Model Structure

Put our vision to the large processing structure on Fig. 4. I adopted encoder-attention-decoder structure in this project. One main reason was that intermediate attention enables to exploit all input hidden layers through concatenating produced context vectors. That causes better capitalization on recursive neuron learning. Decoder and encoder structure has been introduced in prior sections, the principal of attention layer will be the main character here. I utilized Bahdanau attention layer as the interface between encoder and decoder[28].

$$s_t = \sigma_h(D_1(o_t^e) + D_2(i_{t-1}^d)) \tag{13}$$

where $s$ performs as the score function determining initial attention layer weights. $D_1, D_2$ represent two preceding dense layer normalizing the input with self-defined units. $o^e, i^d$ is respectively output vector from encoder and input vector from decoder. $\sigma_h$ is tanh activation function. Score function calculates a score to relate all encoder's hidden layer states and the output from decoder in the previous stage. This score then gives feedback to current attention weights updating.

$$a_t = \sigma_s(s_t) \times i_{t-1}^d \tag{14}$$

where $a$ is the current attention weight vectors after softmax $\sigma_s$. Probability distribution for newly published attention weights are further calculated and influence the output context vector.

$$c_t = \sum a_t \times i_{t-1}^d \tag{15}$$

where $c$ exhibits output context vector in attention layer. In the last, apply $reduce_sum$ to reduce the vector dimensions to compose a updated context vector learning from previous outputs. By iterating updates of context vectors through attention layer, Model can keep grasping the dominant information and features.

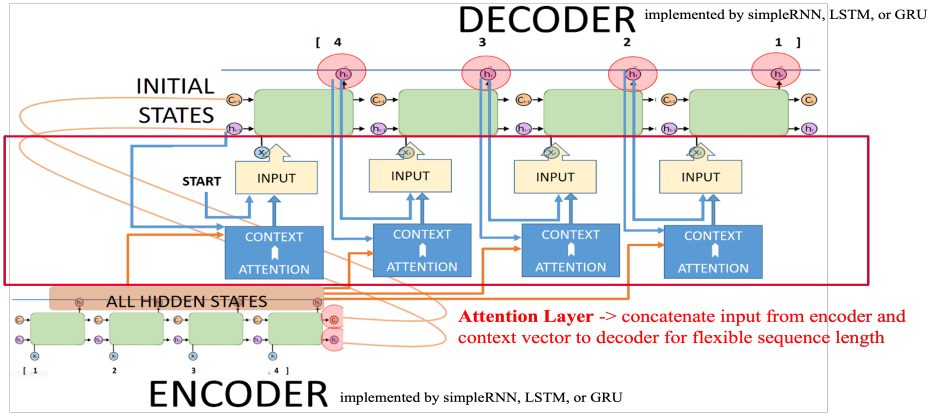


Fig. 4. Diagram about workflow in encoder-attention-decoder structure. At the outset, encoder processes the input sequence and passes it to the attention layer. Attention layer produces context vectors and concatenates them with the input from encoder. Decoder subsequently outputs the target sequences through the input vectors and dominant weights from attention layer.

# 4 Experiment

Performance and training cost time comparison between three NLP units operated in first. A-hundred-epoch training used to accelerate the evaluation. More training was later performed in order to achieve desired performance of neural machine translator. Then, inspected the translation efficiency of NMT and conventional BT including history tables. Discussion about the pros and cons was held thereafter.

## 4.1 Experimental Setting

Total characters in training dataset was 376809 with 2967 code blocks. Training was executed in personal computer, Macbook Pro M1 MAX with 24-core GPU core and 32GB RAM. MacOS Monterey 12.3.1 was the operating system. Due to the computing ability limit, batch size was constrained to 64. Hidden layers' unit all set to 512, embedding layer dimension was 256. Steps per epoch was changed according to code block number that designated to 60. 100-epoch-training was used for NLP unit evaluation, 500-epoch-training for ultimate comparison.

## 4.2 Evaluation betweeen NLP Units

Performance evaluation between three NLP units, SimpleRNN, LSTM, and GRU displays in Fig.5 after training under identical conditions mentioned in the earlier section. Loss function was categorical sparse cross entropy with mask. Fewer loss generally means the model achieving relatively superior prediction accuracy. Model always stored checkpoints reaching the lowest loss to preserve locally optimal model without overlapping. All three models obtained exceptional loss reduction in 100 epochs. SimpleRNN-based and GRU-based NLP model performed slightly better than LSTM-based model.

Except loss evaluation, training cost time was another crucial standard distinguishing neural machine translator since I aimed to improve the translation efficiency. There were many ups and downs in the graphs in Fig.6. It was caused by the property of recurrent neural networks. As we know, context vectors and weights vary in each epoch. These factors impact the portion of data existence. If weights determine to keep more information internally, it requires longer computing time to finish completing data updating and vice versa. Thus, I averaged the cost time to acquire an approximate translation time.

Cost time with RNN-based model exceeded LSTM-based and GRU-based models. Combined the behavior of loss, the GRU-based encoder-attention-decoder model became the final candidate to compete with conventional binary translator.

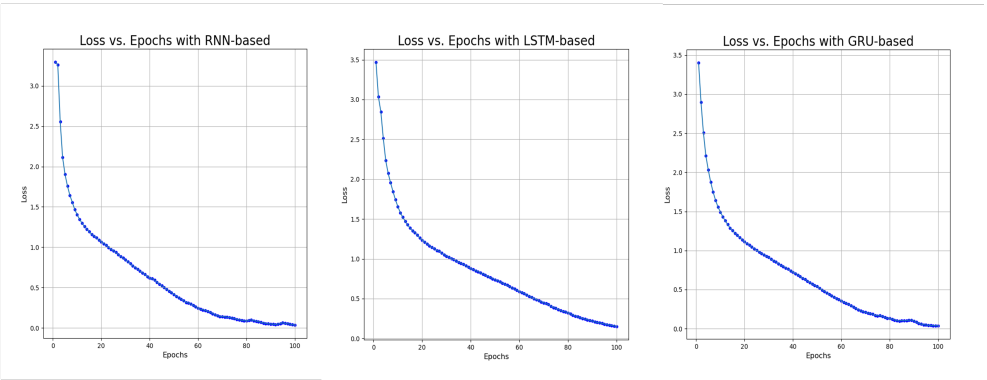


Fig. 5. Categorical sparse entropy loss versus epochs with three NLP units based models.

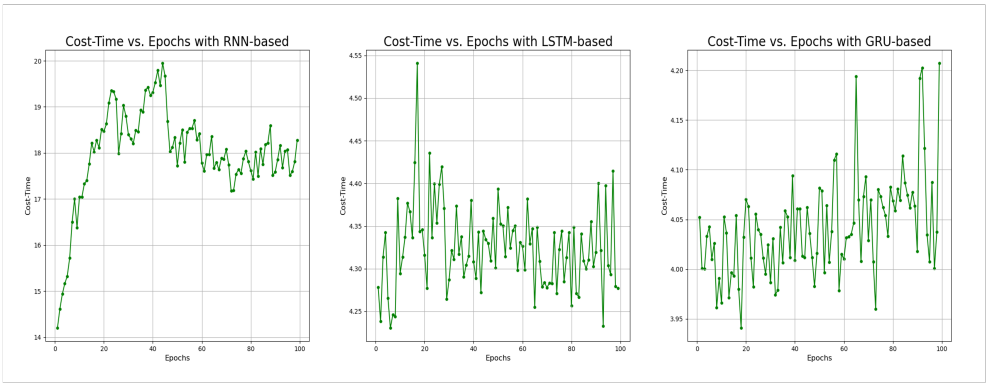


Fig. 6. Cost time while training versus epochs with three NLP units based models.

### 4.3 Comparison between NMT and conventional BT

The eventual neural machine translator utilized GRU as unit with 500-epoch training. It achieved minus 3 power of 10 in loss. Both training and testing accuracy was about 93%, implied there was no overfitting or underfitting happening.

As for the most important factor, translation efficiency, NMT cost 1.33secs to complete translation task with testing dataset consisted of 742 code blocks. Spent 2.09secs to process the whole dataset with 3709 code blocks. On the contrary, conventional BT with MHT required 2.33secs to finish translating the whole dataset. I compared five consecutive translation translation between these two translator in Fig.7. NMT outperformed BT 3.6% to 9.7% on execution time. This result indicated that robust neural machine translator model had the potential to surpassed industrial binary translator nowadays.

It would need some revisions or adjustments after the neural translation because the accuracy was not perfectly 100%. Compilers could handle some small mistakes (inaccurate instructions), but it still needed some else dataflow optimization to reduce this happening and make the processed target codes generally work in most situations.

In fact, I initially designed a following dynamic data-level parallelism (DLP) structure to optimize the produced codes from neural translation. In case that I only enhanced the translation efficiency statically by the machine itself, utilizes DLP on the execution stage could further leverage the performance. The idea was to inspect the instruction window beforehand and rearrange the schedule based on data dependencies and processors' loading. Instruction correction and optimization were also implemented while inspection to help us deal with this problem. In overall, it applies the neural network-based translation first and uses rule-based (conventional one) scheduling in the following. Accordingly, the machine can tolerate the imperfect accuracy by self-correction in the task optimization stage. This would be one possible direction to fulfill this project.

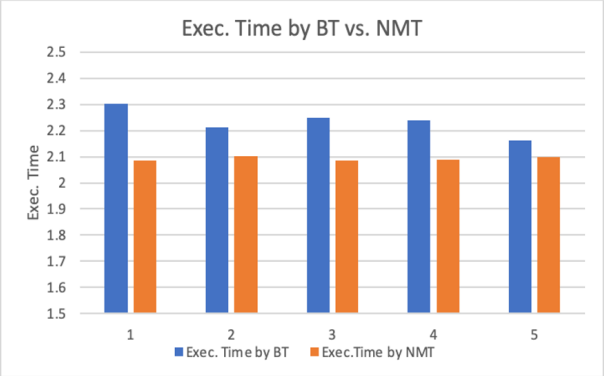


Fig. 7. Execution time of translating the whole dataset with Conventional BT versus with NMT in five consecutive times.

## 5 Conclusion

Although it demands substantial time to train a robust neural machine translation model with sufficient dataset, this experiment reveals that it has the power to eifficiently execute cross-architectural translation. I believe once the dataset is large enough, the accuracy of the model can reach close to 98 or 99%. In this view, we do not need endeavoring to adjust or revise the produced codes. As I mentioned in 4.3, following methods resolving incorrect instructions is urgent to research if being industrialized. Data-level parallelism optimization, dynamic scheduling, single instruction multiple data, and NLP algorithm optimizations are aspects could this project keep proceeding. In summary, the reduced time applying neural machine translator brought a promising future on cross-architectural assembly code translation.

## 6 Future Work

The first and most important one is how to preprocess the raw data and maintain the logic semantic equivalence in the meantime. Some instructions, take conditional branch for example, need register values to determine the following instructions. It will be a great lesson about how to keep them without harming the logics inside. As for the trade-off between accuracy and translation time, I attempt to simplify the training procedure (reduce dominant features) and make the accuracy to be a more important one. Therefore, it is crucial to maintain the semantic equivalence in the data preprocessing. I pay much attention on this one and there remains some logics required to modify. If constraints determining the output are strictly formed and always correct, we could just consider the accuracy and intuitively defined the model. Other techniques, redundant information elimination and padding space strategy are also methods to help make the codes semantically equivalent.

Except for the data preprocessing, there exists points can enhance the NLP model algorithm. Specialized bidirectional attention layer with GRU unit could further benefit the learning procedure[29]. [30] proposed solution for relatively insufficient sequence-level training in NMT. Additionally, there was a event under severe misprediction about process (ordering) that irregular instructions considered as other instructions in each translation, which is commonly happened in branch predictors. I inspected the generated codes and concluded some possible methods. The attention layer was conducted with tanh and softmax to vote for the most possible candidate. Replace the tanh activation function with Leaky Rectified Linear Unit (Leaky-ReLU) or Exponential Linear Unit (ELU) to impress the dominant features would be beneficial. An attached supervised register handling the instruction order could alleviate this problem as well. Additionally, utilizing and extending the bidirectional network could be another solution. I currently checked 10 characters ahead the input, bypassing more characters could give the model more information to take actions. I will focus on above-mentioned schemes to reinforce present model into a versatile one.

# 7 Reference


[1] Emilio G. Cota and Luca P. Carloni. "Cross-ISA Machine Instrumentation Using Fast and Scalable Dynamic Binary Translation". In: *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. VEE 2019. Providence, RI, USA: Association for Computing Machinery, 2019, pp. 74–87. ISBN: 9781450360203. DOI: 10.1145/3313808.3313811. URL: https://doi.org/10.1145/3313808.3313811.

[2] Xuan Guo and Robert D. Mullins. "Accelerate Cycle-Level Full-System Simulation of Multi-Core RISC-V Systems with Binary Translation". In: *CoRR* abs/2005.11357 (2020). arXiv: 2005.11357. URL: https://arxiv.org/abs/2005.11357.

[3] Gabriele Mencagli et al. "Raising the Parallel Abstraction Level for Streaming Analytics Applications". In: *IEEE Access* 7 (2019), pp. 131944–131961. DOI: 10.1109/ACCESS.2019.2941183.

[4] Richard Littin et al. "Block Based Execution and Task Level Parallelism". In: 1998.

[5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[6] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[7] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *ArXiv* 1409 (Sept. 2014).

[9] KyungHyun Cho, Aaron C. Courville, and Yoshua Bengio. "Describing Multimedia Content using Attention-based Encoder-Decoder Networks". In: *CoRR* abs/1507.01053 (2015). arXiv: 1507.01053. URL: http://arxiv.org/abs/1507.01053.

[10] Michael Chu, Rajiv Ravindran, and Scott Mahlke. "Data Access Partitioning for Fine-grain Parallelism on Multicore Architectures". In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 2007, pp. 369–380. DOI: 10.1109/MICRO.2007.15.

[11] Ding-Yong Hong et al. "Exploiting Longer SIMD Lanes in Dynamic Binary Translation". In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. 2016, pp. 853–860. DOI: 10.1109/ICPADS.2016.0115.

[12] Ding-Yong Hong et al. "Improving SIMD Parallelism via Dynamic Binary Translation". In: *ACM Trans. Embed. Comput. Syst.* 17.3 (Feb. 2018). ISSN: 1539-9087. DOI: 10.1145/3173456. URL: https://doi.org/10.1145/3173456.

[13] Tiago Knorst et al. "An energy efficient multi-target binary translator for instruction and data level parallelism exploitation". In: *Design Automation for Embedded Systems* (Jan. 2022), pp. 1–28. DOI: 10.1007/s10617-021-09258-6.

[14] Yuan Yao et al. "FPGA based hardware-software co-designed dynamic binary translation system". In: *2013 23rd International Conference on Field programmable Logic and Applications*. 2013, pp. 1–4. DOI: 10.1109/FPL.2013.6645554.

[15] Fei Zuo et al. "Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs". In: *Proceedings 2019 Network and Distributed System Security Symposium* (2019). DOI: 10.14722/ndss.2019.23492. URL: http://dx.doi.org/10.14722/ndss.2019.23492.

[16] Lucia Benková and L'ubomír Benko. "Neural Machine Translation as a Novel Approach to Machine Translation". In: Sept. 2020.

[17] T. Law, H. Itoh, and H. Seki. "A neural network-assisted Japanese-English machine translation system". In: *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. Vol. 3. 1993, 2905–2908 vol.3. DOI: 10.1109/IJCNN.1993.714330.

[18] Ilyas Cicekli and Halil Altay Güvenir. "Learning Translation Templates from Bilingual Translation Examples". In: *Appl. Intell.* 15 (July 2001), pp. 57–76. DOI: 10.1023/A:1011270708487.

[19] L. Leija, S. Santiago, and C. Alvarado. "A system of text reading and translation to voice for blind persons". In: *Proceedings of 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. Vol. 1. 1996, 405–406 vol.1. DOI: 10.1109/IEMBS.1996.657015.

[20] Chanjun Park et al. “Mimicking Infants’ Bilingual Language Acquisition for Domain Specialized Neural Machine Translation”. In: *IEEE Access* 10 (2022), pp. 38684–38693. DOI: 10.1109/ACCESS.2022.3165572.

[21] A. Bertallot. “Machine language translations - Letters”. In: *Computer* 37.5 (2004), pp. 7–7. DOI: 10.1109/MC.2004.1297223.

[22] Yung-Hui Lee and Cheng-Yueh Tsai. “Taiwan sign language (TSL) recognition based on 3D data and neural networks”. In: *Expert Systems with Applications* 36.2, Part 1 (2009), pp. 1123–1128. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2007.10.038. URL: https://www.sciencedirect.com/science/article/pii/S0957417407005210.

[23] Maxim Khalilov et al. “Neural Network Language Models for Translation with Limited Data”. In: *2008 20th IEEE International Conference on Tools with Artificial Intelligence*. Vol. 2. 2008, pp. 445–451. DOI: 10.1109/ICTAI.2008.35.

[24] Xuanwei Zhang et al. *Look Backward and Forward: Self-Knowledge Distillation with Bidirectional Decoder for Neural Machine Translation*. 2022. DOI: 10.48550/ARXIV.2203.05248. URL: https://arxiv.org/abs/2203.05248.

[25] Jing Liang and Minghui Du. “Two-Way Neural Network Chinese-English Machine Translation Model Fused with Attention Mechanism”. In: *Scientific Programming* 2022 (Jan. 2022), pp. 1–11. DOI: 10.1155/2022/1270700.

[26] Changheng Song et al. “Unleashing the Power of Learning: An Enhanced Learning-Based Approach for Dynamic Binary Translation”. In: *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC ’19. Renton, WA, USA: USENIX Association, 2019, pp. 77–89. ISBN: 9781939133038.

[27] Jitendra V. Tembhurne and Tausif Diwan. “Sentiment Analysis in Textual, Visual and Multimodal Inputs Using Recurrent Neural Networks”. In: *Multimedia Tools Appl.* 80.5 (Feb. 2021), pp. 6871–6910. ISSN: 1380-7501. DOI: 10.1007/s11042-020-10037-x. URL: https://doi.org/10.1007/s11042-020-10037-x.

[28] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. “Effective Approaches to Attention-based Neural Machine Translation”. In: *CoRR* abs/1508.04025 (2015). arXiv: 1508.04025. URL: http://arxiv.org/abs/1508.04025.

[29] Biao Zhang et al. “Neural Machine Translation With GRU-Gated Attention Model”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.11 (2020), pp. 4688–4698. DOI: 10.1109/TNNLS.2019.2957276.

[30] Chenze Shao et al. “Sequence-Level Training for Non-Autoregressive Neural Machine Translation”. In: *CoRR* abs/2106.08122 (2021). arXiv: 2106.08122. URL: https://arxiv.org/abs/2106.08122.