Learning C to x86 Translation: An Experiment in

Neural Compilation

Dionis Xhaferi

Machine learning has been explored for a long amount of time, while recently deep learning models have been built to translate or compile code. The purpose of the paper is to find a method to completely automate compilation, meaning given a program, generate the equivalent assembler code. The study focuses on the use of translating small C functions and turning them into x86 assembler code. With the use of neural translation, a user can complete unsupervised compilation from a language to an instruction set architecture (ISA). Given this, it is suggested that there could be a way to create new compilers for new systems and programming languages automatically.

To complete this study, the authors use existing function-level C corpus, which is from a study called Anghabench, [da Silva et al., 2021] and create the equivalent x86 assembler code. The different variables in the experiment include different sizes of training data, the model size, and the number of epochs. In the past, it has been easy to syntactically translate code. This experiment focuses on attempting to semantically translate the equivalent code. In other studies, it has been shown that given a program, we can syntactically generate an equivalent assembler over 80 percent of the time. Currently, the best model can only semantically generate an accurate assembler based on a given program only 33 percent of the time (Estapé 2022). In the experiment, the dataset that is used is Anghabench, which is a dataset of 1 million C functions that comes with the C code that is used to compile them. The C code is accessed from GitHub repositories. Rather than whole programs, the experiment uses C functions since they are shorter and easier to unit test. Because headers and type definitions only serve to complicate the machine translation job, the Anghabench was filtered to only include programs that were necessary for the experiment. They called this filtered group of programs “Angha-Par.” For the syntactic evaluation, GCC is used to check if the generated assembler by generating object code from the assembler. This is used because the GCC metric is up to date and easy to compute. Semantic evaluation involves determining the equivalency between the output produced by the model and the GCC assembler. For instance, the outputs of the assembler and the GCC compiler are compared given a set of inputs. The models used for testing are called the Transformer architecture. This is a deep learning model that has had a significant impact in natural language processing (NLP). The Transformer model is known for its effectiveness in handling sequential data and has become a foundation for many models used in machine translation. In the experiment, the Transformer model is split into four parts, Trans-Small, Trans-Med, Trans-Big, and Trans-Big+. The bigger the Transformer model, the larger the data size. With the Trans-Big model, different vocabulary, data sizes, and number of epochs are used. This is split into 5 different models, -50 percent data, -1/2x vocab, +1/2x vocab, +1e2x w.-decay, and +1/2x epochs. All the models are trained with the same data.

Based of the observational equivalency, the experiment’s results demonstrated that the Transformer-Big was the most effective model. It was also the best model based on syntactical accuracy using the BLEU score. Considering the model with the highest syntax accuracy, the Transformer-Small had a syntactic accuracy of 98.50, which is the highest of the models. The other models were averaging a score of about 80. (Estapé 2022) This was surprising to see since the outputs of the data were very long. The Transformer-Small model underfit the task of the compilation, and all the average sized models had similar results. There was one model that was trained with half of the data and that was seen to perform a lot worse than the rest of the models.

The experiment concludes with proving that the neural compilation approach can show that the deep learning models can learn to compile assembler code. Although the results of the data were somewhat weak, based on the experiment, the study is a step in the right direction for future research in code translation. There were many obstacles in the tasks such as output length, grammatical, and accuracy requirements that were not addressed. The impact of the article can be seen as this was one of the first steps into learning an end-to-end machine compiler using a neural machine translation. Using the information from this article, the future works can improve experiments by scaling up the amount of data, computation time, and the model’s parameter. The experiment creates an improvement in the system software field because it shows a potential for an advance in machine translation and code generation techniques. This improvement and optimization can lead to a more efficient compiler.

1. Armengol-Estapé, J., & O’Boyle, M. (2022, December 16). Learning C to x86 translation: An experiment in neural compilation. https://arxiv.org/pdf/2108.07639.pdf