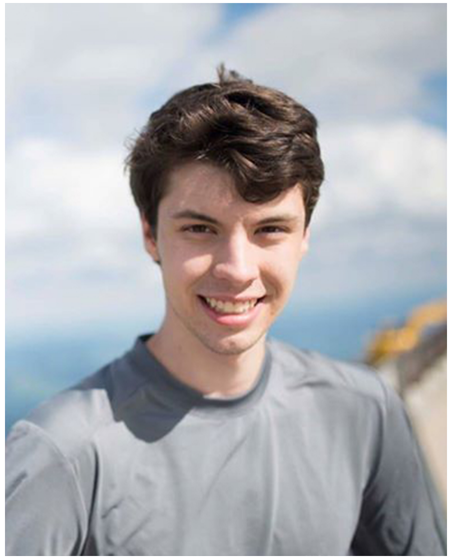
Understanding High-Level Properties of Low-Level Programs through Transformers

Zifan (Carl) Guo - St. Mark's School

Mentor: William S. Moses

MIT PRIMES Conference - May 22th, 2022

Carl Guo



William S. Moses

MIT

Susan Tan

Princeton

Yebin Chon

Princeton

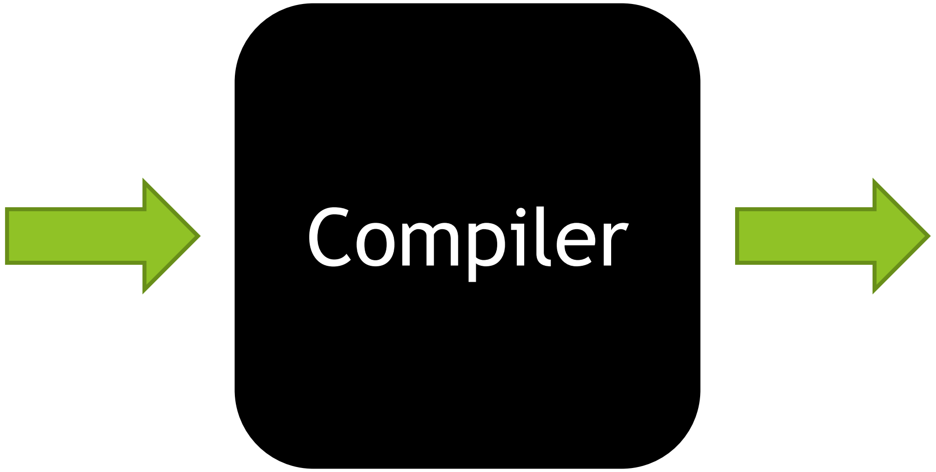
Johannes Doerfert

Argonne National Lab

Compilers

```c
double relu3(double x) {
  double result;
  if (x > 0)
    result = pow(x, 3);
  else
    result = 0;
  return result;
}
```

Relu3.c




```
011010010110110001101111
011101100110010101111001
011011110111010101110111
011001010110111001100100
011110010010000001101001
011011010110100101110011
011100110111010100001010
...
```

Relu3.o

- Programs need to go through compilers to be executed
- Compilers transform English-based programs to 1s and 0s that computer understand

Series of Transformation

- High Level Languages
- (C, Java, Python)
  - High abstraction
  - English-like

- LLVM-IR (Intermediate Representation)
  - Less abstract but still readable
  - Platform independent

- Assembly Language
  - Even less abstract and less readable
  - Computer platform dependent
    - X86_64
    - AArch64
    - RISC-V

- Machine Language
  - Not readable
  - 1s and 0s

C

```c
double relu3(double x) {
  double result;
  if (x > 0)
    result = pow(x, 3);
  else
    result = 0;
  return result;
}
```

LLVM-IR

```
define double @relu3(double %x){
entry:
    %cmp = %x > 0
    br %cmp, cond.true, cond.end
cond.true:
    %call = pow(%x, 3)
    br cond.end
cond.end:
    %result = phi [%call, cond.true],
    [0, entry]
    ret %result
}
```

X86-64 Intel

```
.LCPI0_0:
        .quad 4613937818241073152
relu3:
        push rbp
        mov rbp, rsp
        sub rsp, 16
        movsd qword ptr [rbp - 8], xmm0
        movsd xmm0, qword ptr [rbp - 8]
        xorps xmm1, xmm1
        ucomisd xmm0, xmm1
        jbe .LBB0_2
        movsd xmm1, qword ptr [rip + .LCPI0_0]
        movsd xmm0, qword ptr [rbp - 8]
        call pow
        movsd qword ptr [rbp - 16], xmm0
        jmp .LBB0_3
.LBB0_2:
        xorps xmm0, xmm0
        movsd qword ptr [rbp - 16], xmm0
.LBB0_3:
        movsd xmm0, qword ptr [rbp - 16]
        add rsp, 16
        pop rbp
        ret
```

- As the language becomes more low-level,
  it becomes more complicated, less readable, and more precise

Compiler Optimization

- Code transformation to make the program run faster (under the hood)

```
//Compute magnitude in O(n)
double mag(double[] x);

//Compute norm in O(n^2)
void norm(double[] out, double[] in) {
  for (int i=0; i<n; i++) {
    out[i] = in[i] / mag(in);
  }
}
```

```
//Compute magnitude in O(n)
double mag(double[] x);

//Compute norm in O(n)
void norm(double[] out, double[] in) {
  double res = mag(in);
  for (int i=0; i<n; i++) {
    out[i] = in[i] / res;
  }
}
```

Unoptimized: $\Theta(n^2)$ → Loop invariant code motion (LICM) → Optimized: $\Theta(n)$

This series of transformation is similar to … language translations

Transformers

- Attention is all you need (Vaswani et al. 2017)
- Unlike RNN or LSTM, not sequential → no locality bias
  - Better performance for long-distance context
- Allow parallel computation to save time
  - Process sequences as a whole instead of word by word

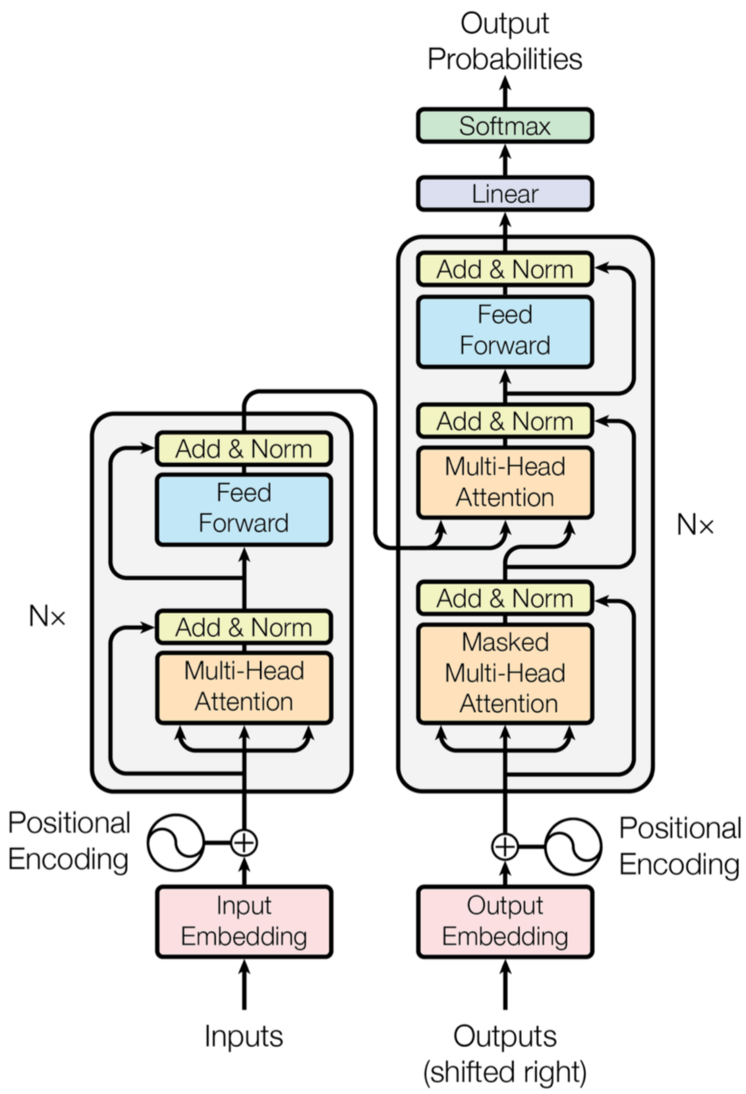


Figure 1: The Transformer - model architecture.

Transfer Learning (BERT)

Input For Pretraining Task

Pre-trained Model
(on massive corpus)

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)
- Utilize unlabeled data and need less labeled data
  - Creating labels is labor-intensive
  - Allow us to feed in more data

Knowledge Transfer

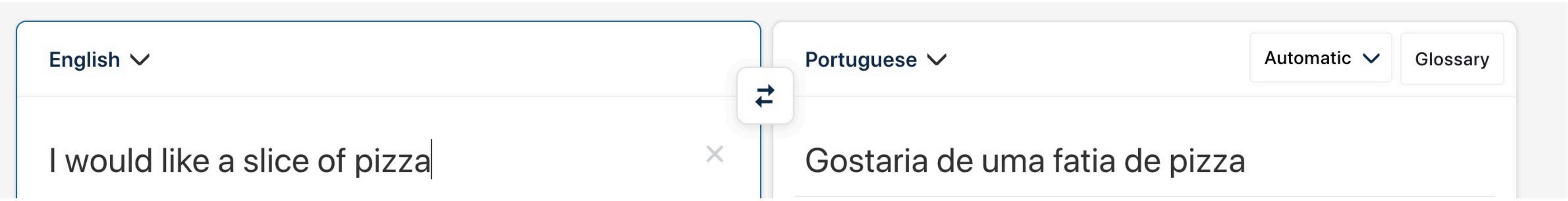
Input For Fine-tuning Task

Fine-tuned Model
(on smaller corpus)

Output For Fine-tuning Task

Research Context

- Success on natural languages (e.g. translating English to Portuguese)



DeepL Translator

- Success on high-level programming languages (e.g. translating Java to Python)

Java

```java
public static int max(int a, int b){
  return a > b ? a : b;
}

public static void createDirectory(Path path)
            throws IOException{
  if(!Files.exists(path)){
    Files.createDirectories(path);
  }
}
```

Python

```python
def max(a, b):
  return a if a > b else b

def create_directory(path):
  if not os.path.exists(path):
    os.makedirs(path)
```

Unsupervised Translation of Programming Languages (Roziere et al., 2020)

- Few research on compiler optimization has used Transformer models before

Our Goal

- Test whether Transformer language modeling can extract high-level properties of low-level programs and perform various downstream tasks on low-level programs
- Through transfer learning, Transformer models can be more effective
  - Unlabeled data
  - Code more subject to error than natural languages
- Such information would be able to better inform us where and how to apply compiler optimization

Three Levels

1. High-Level

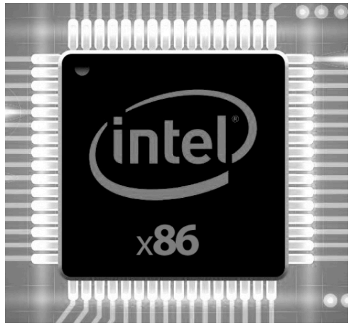
```c
double relu3(double x) {
  double result;
  if (x > 0)
    result = pow(x, 3);
  else
    result = 0;
  return result;
}
```

Relu3.c

3. Cross-Level

2. Low-Level

LLVM COMPILER INFRASTRUCTURE



Existing High-Level Work

TransCoder

Java

```java
public static int max(int a, int b){
  return a > b ? a : b;
}

public static void createDirectory(Path path)
            throws IOException{
  if(!Files.exists(path)){
    Files.createDirectories(path);
  }
}
```

Python

```python
def max(a, b):
  return a if a > b else b

def create_directory(path):
  if not os.path.exists(path):
    os.makedirs(path)
```

Unsupervised Translation of Programming Languages (Roziere et al., 2020)

DOBF (deobfuscation)

Input code

```python
def bfs(graph, root):
  visited = [root]
  queue = [root]
  while queue:
    node = queue.pop(0)
    for neighbor in graph[node]:
      if neighbor not in visited:
        visited.append(neighbor)
        queue.append(neighbor)
  return visited
```

DOBF

DOBF input

```python
def F0(V0, V1):
  V2 = [V1]
  V3 = [V1]
  while V3:
    V5 = V3.pop(0)
    for V4 in V0[V5]:
      if V4 not in V0:
        V2.append(V4)
        V3.append(V4)
  return V2
```

DOBF output

```
F0 = bfs
V0 = graph
V1 = root
V2 = visited
V3 = queue
V4 = neighbor
V5 = node
```

A Deobfuscation Pre-Training Objective for Programming Languages (Roziere et al., 2021)

Training DOBF on C

- Original paper only implemented in Java and Python
- Pretraining objective → actual objective
- Crucial to recover lost information when one tries to recover LLVM-IR control-flow back to beautified C
- Constructed our own obfuscator through clang-tidy

```c
int main(void) {
  uint32_t llvm_cbe_sum_2e_0;
  uint32_t llvm_cbe_sum_2e_0__PHI_TEMPORARY;
  uint32_t llvm_cbe_i_2e_0;
  uint32_t llvm_cbe_i_2e_0__PHI_TEMPORARY;
  uint32_t _1;
  uint32_t _2;
  uint32_t llvm_cbe_add;
  uint32_t sum;
  uint32_t sum__PHI_TEMPORARY;
  uint32_t i;
  uint32_t llvm_cbe_call;

  ;
  ;
  llvm_cbe_sum_2e_0__PHI_TEMPORARY = 0;   /* for PHI node */
  llvm_cbe_i_2e_0__PHI_TEMPORARY = 0;   /* for PHI node */

  llvm_cbe_i_2e_0 = llvm_cbe_i_2e_0__PHI_TEMPORARY;
while (((int32_t)llvm_cbe_i_2e_0) < ((int32_t)10u)){
  _1 = *((&x.array[((int64_t)(((int64_t)(int32_t)llvm_cbe_i_2e_0)))]));
  if (((((int32_t)_1) > ((int32_t)3u))&1))) {
  _2 = *((&x.array[((int64_t)(((int64_t)(int32_t)llvm_cbe_i_2e_0)))]));
  llvm_cbe_add = llvm_add_u32(llvm_cbe_sum_2e_0, _2);
  ;
  sum__PHI_TEMPORARY = llvm_cbe_add;   /* for PHI node */

  } else {
    sum__PHI_TEMPORARY = llvm_cbe_sum_2e_0;   /* for PHI node */
}
  sum = sum__PHI_TEMPORARY;
  ;

  i = llvm_add_u32(llvm_cbe_i_2e_0, 1);
  ;
  llvm_cbe_sum_2e_0__PHI_TEMPORARY = sum;   /* for PHI node */
  llvm_cbe_i_2e_0__PHI_TEMPORARY = i;   /* for PHI node */

  llvm_cbe_i_2e_0 = llvm_cbe_i_2e_0__PHI_TEMPORARY;
  llvm_cbe_sum_2e_0 = llvm_cbe_sum_2e_0__PHI_TEMPORARY;
  llvm_cbe_i_2e_0 = llvm_cbe_i_2e_0__PHI_TEMPORARY;
}
  llvm_cbe_call = printf(((&_OC_str.array[((int64_t)0)])), llvm_cbe_sum_2e_0
  return 0;
}
```

Training DOBF on C

- Original paper only implemented in Java and Python
- Pretraining objective → actual objective
- Crucial to recover lost information when one tries to recover LLVM-IR control-flow back to beautified C
- Constructed our own obfuscator through clang-tidy

	Eval p_{obf} = 0		Eval p_{obf} = 1	
	Acc	F1	Acc	F1
MLM + 12 layers DOBF	32.89	30.46	30.40	27.88
MLM + 6 layers DOBF	29.35	26.94	28.17	25.72

- Results here are not satisfactory most likely because we chose a smaller, cleaned C dataset and will try a bigger dataset on all GitHub C code

2nd: Transformer on Low-level Programs
Existing work

- Ithemal (Mendis et al., 2018) uses a hierarchical LSTM to estimate throughput given x86_64 assembly basic blocks
  - Basic block = chunks of assembly code without branches
  - Throughput = clock cycles for executing a basic block in steady state
- Shypula et al. 2021 use a Transformer to superoptimize programs with a Self Imitation Learning for Optimization (SILO) approach
- Jayatilaka et al. 2021 focuses on automatically choosing between -01, -02, and -03 pipeline based on code structure with ML
- …

Case study: Throughput Estimation of X86_64 Basic Blocks

- Accurate throughput estimation is an essential tool that informs choosing the proper optimization passes
- Can a Transformer do better?
- DynamoRIO Tokenizer
  - ML needs fixed length input → need to tokenize
  - Programming Language specific
  - Fixed vocabulary

Throughput Estimation Experiment

- BHive benchmark dataset with 320,000+ basic blocks mapping to the throughput under Intel's Haswell microarchitecture
  - While the majority of data points fall under value between 20.0 and 1000.0, the maximum can go up to 1,600,450
- Pretrained on Masked Language Modeling and fine-tuned with mean squared error loss for regression on the same dataset

prediction

```
mov    rdx, qword ptr
[rbx+0x50]
xor    ecx, ecx
mov    esi, 0x01178629
mov    rdi, rbp
```

→ 110.00

Results & Observations

	Pearson Correlation	Spearman Correlation	Prediction Accuracy (<25%)
Reproduced Ithemal	91.8	96.0	85.39%
Transformer	94.95	90.04	56.7%
Transformer with Lab2id	93.69	95.74	76.06%

- Both Ithemal and Transformer struggle with large values
- Lab2id tries to mitigate the issue
- While Ithemal can be more exact for the small data points but is really far off for these big outliers, Transformer seems to model the big data points better but be less exact for all data points.

3rd: Cross-lingual Transformer model on both high-level and low-level

Case study: Translating C to LLVM-IR

```c
double relu3(double x) {
  double result;
  if (x > 0)
    result = pow(x, 3);
  else
    result = 0;
  return result;
}
```

```llvm
define double @relu3(double %0){
   %2 = fcmp ogt double %0, 0
   br %2, label %3 , label %5
3:
   %4 = tail call double @pow (%0, double
   3)
   br label %5
5:
   %6 = phi [%4, %3], [0, %1]
   ret %6
}
```

- Translating from C to LLVM-IR
- Preprocessing
  - Inherited TransCoder's C tokenizer and built my own LLVM-IR tokenizer
  - Performed Byte-Pair Encoding (BPE)
- Transfer Learning:
  - Pretrained first with Masked Language Modeling (MLM) on all data
  - Fine-tuned with Machine Translation instead of Back Translation on functions only

Data & Results

- Csmith (randomly generated compilable C programs) (Yang et al., 2011)
- Project CodeNet (web scrape of competitive programming online judging websites) (Puri et al., 2021)
- ~~GitHub Google BigQuery (all available GitHub C programs)~~
- AnghaBench (1 million selected and cleaned compilable GitHub C programs) (de Silva et al., 2021)

Model evaluation result on the 3 datasets

	Csmith	Project CodeNet	AnghaBench
Training Accuracy	90.73%	93.66%	99.03%
Reference Match	N/A	N/A	13.33%
BLEU Score (0~100)	43.39	51.01	69.21

Preprocessing Modification

- Removing unnecessary syntax while making sure it compiles
- Prefix Notation
  - A * B + C / D = + * A B / C D
  - Math Transformer proves prefix notation effective (Griffith & Kalita, 2019)
  - { 8, [3, 5.0, Carl] } = STRUCT2 8 ARR3 3 5.0 Carl
- Writing out definitions of global variables so they can be recoverable on the function level

Model evaluation result on AnghaBench with different preprocessing manipulation

	Original	Prefix	Prefix & Global
Training Accuracy	99.03%	98.69%	**99.60%**
Reference Match	13.33%	22.62%	**49.57%**
BLEU Score (0~100)	69.21	78.19	**87.68%**

- Note: Armengol-Estape & O'Boyle, 2021 attempted to translate C to x86_64 concurrently, concluding that machine learning itself can't be used as a compiler.

Discussion

- Cross-lingual Transformer models should be the direction
  - Includes lost information on the low-level
  - Shows preliminary successes in downstream tasks
- Future explorations:
  - Translating LLVM to C
  - Learn and reconstruct optimization passes and determine when to use
  - Automatic implementation of desired compiler passes

Acknowledgement

- My Mentor, Billy Moses, for his tireless support
- MIT & MIT PRIMES, for this incredible opportunity
- Additional collaborators (Susan, Yebin, Johannes)
- This research was supported in part by a DOE Computational Sciences Graduate Fellowship DESC0019323; in part by LANL grant 531711; and in by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government.
- My parents
- All of you, for listening

Questions?

Thank you!