Large Language Models for Compiler Optimization

Meta AI

2023 年 10 月 19 日

中山大學 SUN YAT-SEN UNIVERSITY

主要内容

LLMs

- Increasing interest in Large Language Models (LLMs) for software engineering domains.
  - code generation
  - code translation
  - code testing
- code optimization?

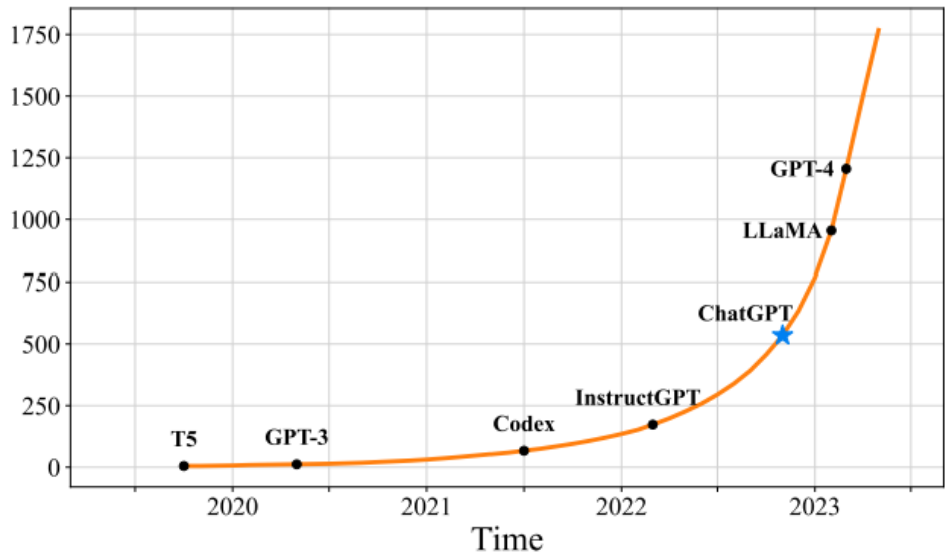

Figure: The trends of the cumulative numbers of arXiv papers that contain the ”large language model” (since October 2019)

ML-guided Code Optimization

- hand-built features: MLGO
- GNN: PrograML
- However, in all cases, the way the input program is represented to the machine learning algorithm is incomplete, losing some information along the way.

主要内容

Overview

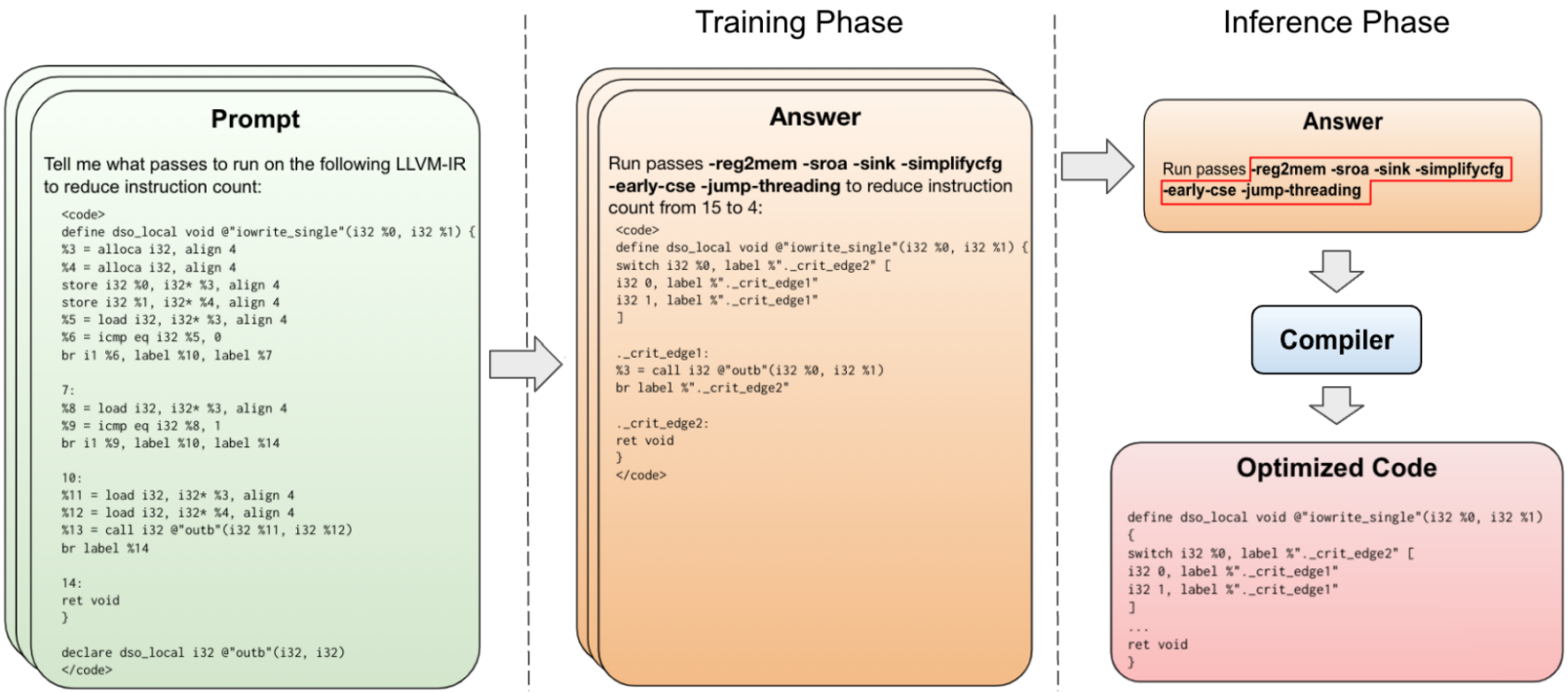

Figure: Overview

Prompt

- Deep Understanding
  - generating the instruction counts of the code before and after the optimizations are applied.
  - generating the output IR after the optimizations are applied.
- LLVM-IR Normalization
  - discard comments, debug metadata and attributes
  - ensure consistent whitespace
  - to reduce the length of the input

Model

- 7B-parameter LLaMa 2
- train model from scratch
- an approximate upper limit on the longest LLVM-IR: 2KB

Training Data: LLVM-IR

- functions from datasets of publicly available handwritten C/C++ code
- synthetic code generated by C/C++ compiler test generators

	n functions	unoptimized instruction count	size on disk	n tokens
Handwritten	610,610	8,417,799	653.5 MB	214,746,711
Synthetic	389,390	13,775,149	352.3 MB	158,435,151
Total	1,000,000	16,411,249	1.0 GB	373,181,862

Figure: Training data.

Training Data: pass order

- Autotuner
- 122 optimization passes+6 meta-flags (-O0, -O1, -O2, -O3, -Oz, and -Os)
- pass lists: typically up to 9 passes $\rightarrow$ search space: around 10^{18}
- compiled each training program an average of 37,424 times
- 5.8% improvement in instruction count reduction over -Oz

主要内容

Training & Validation

- 64 V100s
- 620 GPU days

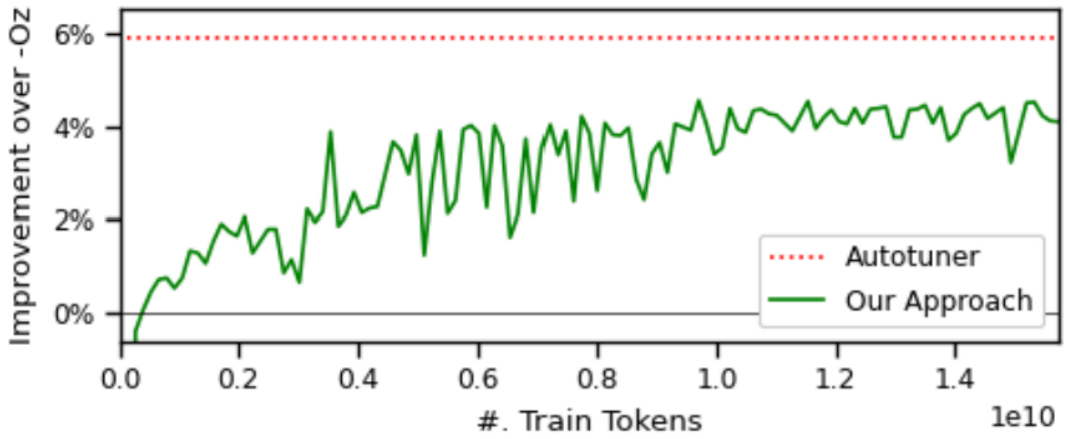

Figure: Performance of generated pass lists.

Training & Validation

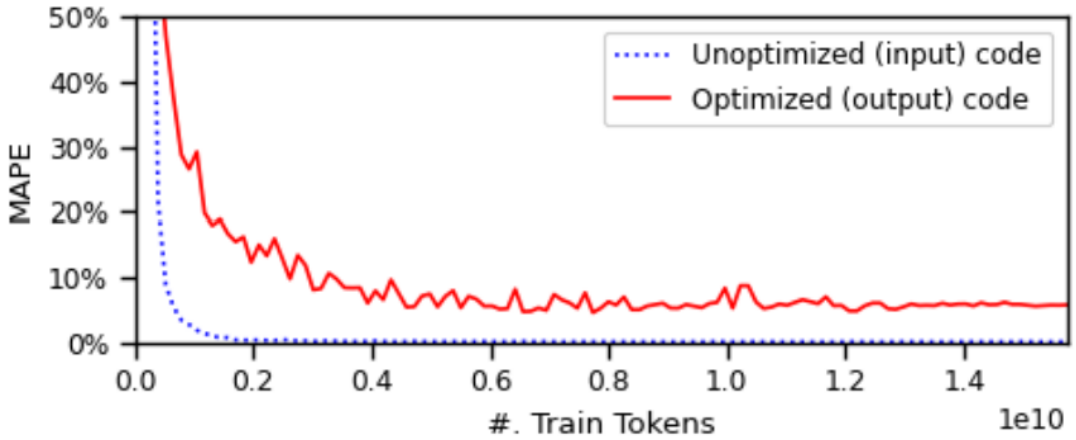

Figure: Accuracy at predicting instruction counts.

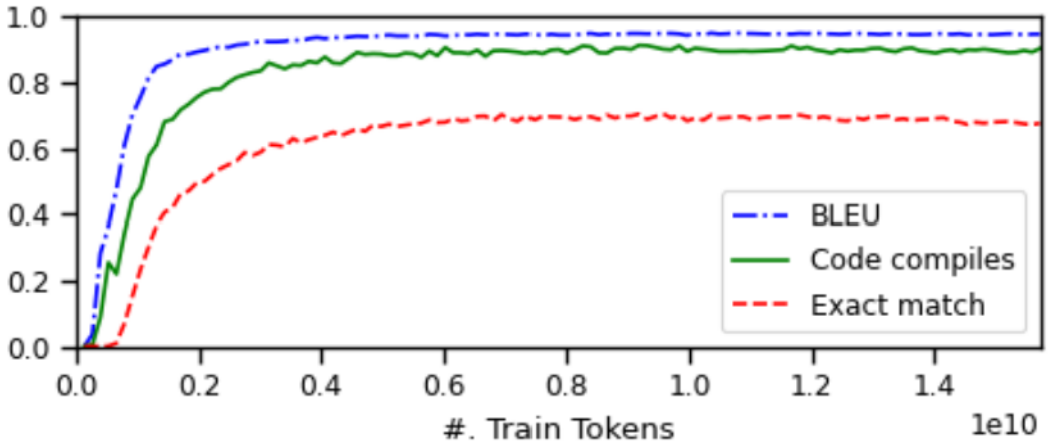

Figure: Model-optimized code metrics.

Comparison to State-of-the-Art

- test datasets: AI-SOCO, POJ-104, CSmith, YARPGen, ExeBench, Transcoder
- baselines
  - AutoPhase: reinforcement learning approach.
  - Coreset-NVP: iterative search with a learned cost model.
  - Autotuner

	additional compilations	functions improved	functions regressed	instructions saved	instructions regressed	overall improvement
Autotuner	2,522,253,069	6,764	0	30,948	0	5.03%
AutoPhase [39]	4,500,000	1,558	8,400	6,522	32,357	-3.85%
Coreset-NVP [20]	442,747	3,985	6,072	16,064	28,405	-1.88%
Our Approach	0	4,136	526	21,935	3,095	3.01%

Figure: Comparison to SOTA.

Comparison to State-of-the-Art

- If a model predicts a pass list other than -Oz, it also evaluates -Oz and selects the best. This prevents regressions w.r.t -Oz at the expense of additional compilations.

	additional compilations	overall improvement
AutoPhase [39]	4,600,000	1.02%
Coreset-NVP [20]	542,747	2.55%
Our Approach	5,721	3.52%

Figure: ”-Oz backup”.

Evaluation of Generated Pass Lists

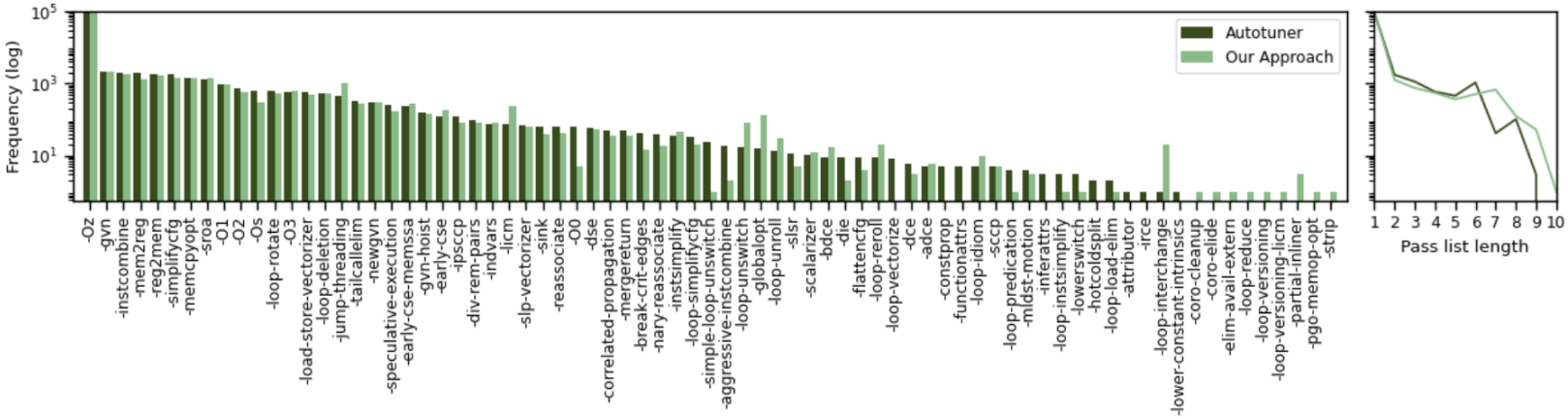

Figure: Pass frequency.

Evaluation of Generated Pass Lists

```
define i32 @f1(i8 %0) {
 %2 = alloca i32, align 4
 %3 = alloca i8, align 1
 store i8 %0, i8* %3, align 1
 %4 = load i8, i8* %3, align 1
 %5 = zext i8 %4 to i32
 %6 = icmp sge i32 %5, 65
 br i1 %6, label %7, label %15

7:
 %8 = load i8, i8* %3, align 1
 %9 = zext i8 %8 to i32
 %10 = icmp sle i32 %9, 90
 br i1 %10, label %11, label %15

11:
 %12 = load i8, i8* %3, align 1
 %13 = zext i8 %12 to i32

<snip 21 lines...>

33:
 %34 = load i32, i32* %2, align 4
 ret i32 %34
}
```

(a) Input code (39 instructions).

```
define i32 @f1(i8 %0) {
 %2 = zext i8 %0 to i32
 %.off = add i8 %0, 191
 %3 = icmp ult i8 %.off, 26
 br i1 %3, label %4, label %6

4:
 %5 = add nsw i32 %2, 191
 br label %10

6:
 %.reload16.off = add nsw i32 %2, 159
 %7 = icmp ult i32 %.reload16.off, 26
 br i1 %7, label %10, label %8

8:
 %9 = icmp eq i8 %0, 32
 %. = select i1 %9, i32 26, i32 1
 br label %10

10:
 %.0.reg2mem.0 = phi i32 [%5, %4],
  [%., %8], [%.reload16.off, %6]
 ret i32 %.0.reg2mem.0
}
```

(b) Autotuned code (14 instructions) using passes: `-reg2mem -instcombine -Os -O1`.

```
define i32 @f1(i8 %0) {
 %2 = zext i8 %0 to i32
 %.off = add i8 %0, 191
 %3 = icmp ult i8 %.off, 26
 br i1 %3, label %6, label %._crit_edge

._crit_edge:
 %.off24 = add i8 %0, 159
 %4 = icmp ult i8 %.off24, 26
 br i1 %4, label %6, label %._crit_edge9

._crit_edge9:
 %5 = icmp eq i8 %0, 32
 %spec.select = select i1 %5,
  i32 26, i32 1
 ret i32 %spec.select

6:
 %.sink = phi i32 [191, %1],
  [159, %._crit_edge]
 %7 = add nsw i32 %.sink, %2
 ret i32 %7
}
```

(c) Model-optimized code (13 instructions) and pass list: `-reg2mem -simplifycfg -mem2reg -jump-threading -Os`.

Evaluation of Generated Code

error category	n
type error	5,777
instruction forward referenced	1,521
undefined value	1,113
invalid redefinition	616
syntax error	280
invalid value for constant	144
undefined function	112
index error	98
other	83
Total	9,744

Figure: ”compile-error”.

Additional Experiments

Figure: Training data size.

Evaluation of Single Pass Translation

- dataset: 10,000 unique (prompt, answer) examples for each of the 60 passes for a total of 600k examples
- model: 74 GPU days

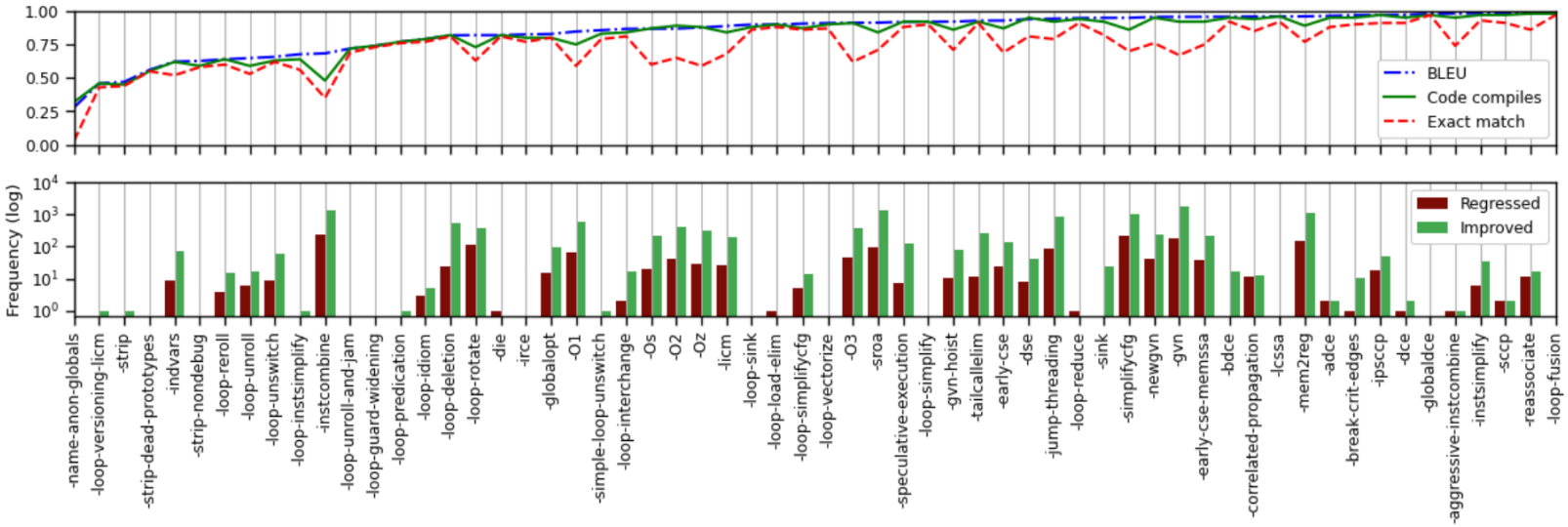

Figure: Single pass translation.

主要内容

Disscussion

- Context Window: A common concern with LLMs.
- Math Reasoning and Logic: A chain-of-thought approach in which models are taught to decompose complex reasoning problems into incremental steps will prove fruitful.
- Inference Speed: Two orders of magnitude more time than Compilers, much faster than the Autotuner.

谢谢大家！