中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

# IR-OptSet: An Optimization-Sensitive Dataset for Advancing LLM-Based IR Optimizer

***Zi Yang***[1,2], ***Lei Qiu***[1,3], Fang Lyu[1], Ming Zhong[1], Zhilei Chai[2], Haojie Zhou[2], Huimin Cui[1,3] and Xiaobing Feng[1,3]

[1]SKLP, Institute of Computing Technology, CAS

[2]Jiangnan University, China

[3]University of Chinese Academy of Sciences, Beijing, China

# Compiler Optimization

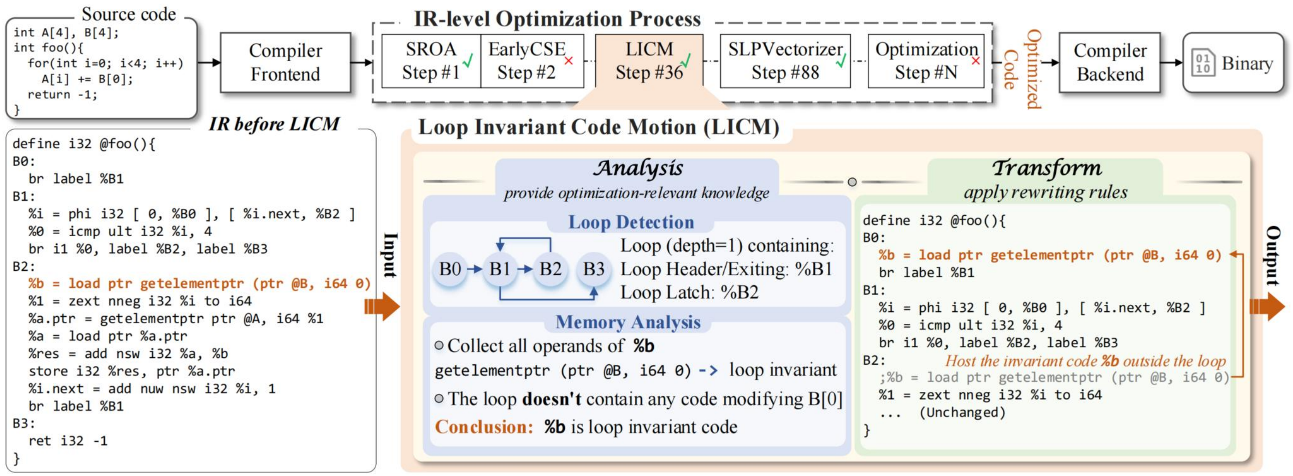
- **Compiler: Source Code -> Machine Code.**
  - ✓ Main tasks: Translation, **Optimization.**
  - ✓ **Optimization:** Optimizing code to improve performance.

- **Challenges in Compiler Optimization**
  - ✓ Reliant on **numerous** manually crafted, rule-based transformation passes over IRs.
  - ✓ Each pass encapsulates **highly specialized** optimization logic.

**->** *Significant manual effort required to design and maintain.*

# Background

## ➢ Traditional IR-level Optimization Process



```
int A[4], B[4];
int foo(){
  for(int i=0; i<4; i++)
    A[i] += B[0];
  return -1;
}
```

```
define i32 @foo(){
B0:
  br label %B1
B1:
  %i = phi i32 [ 0, %B0 ], [ %i.next, %B2 ]
  %0 = icmp ult i32 %i, 4
  br i1 %0, label %B2, label %B3
B2:
  %b = load ptr getelementptr (ptr @B, i64 0)
  %1 = zext nneg i32 %i to i64
  %a.ptr = getelementptr ptr @A, i64 %1
  %a = load ptr %a.ptr
  %res = add nsw i32 %a, %b
  store i32 %res, ptr %a.ptr
  %i.next = add nuw nsw i32 %i, 1
  br label %B1
B3:
  ret i32 -1
}
```

Loop (depth=1) containing:
Loop Header/Exiting: %B1
Loop Latch: %B2

○ Collect all operands of **%b**
`getelementptr (ptr @B, i64 0)` -> loop invariant

○ The loop **doesn't** contain any code modifying B[0]

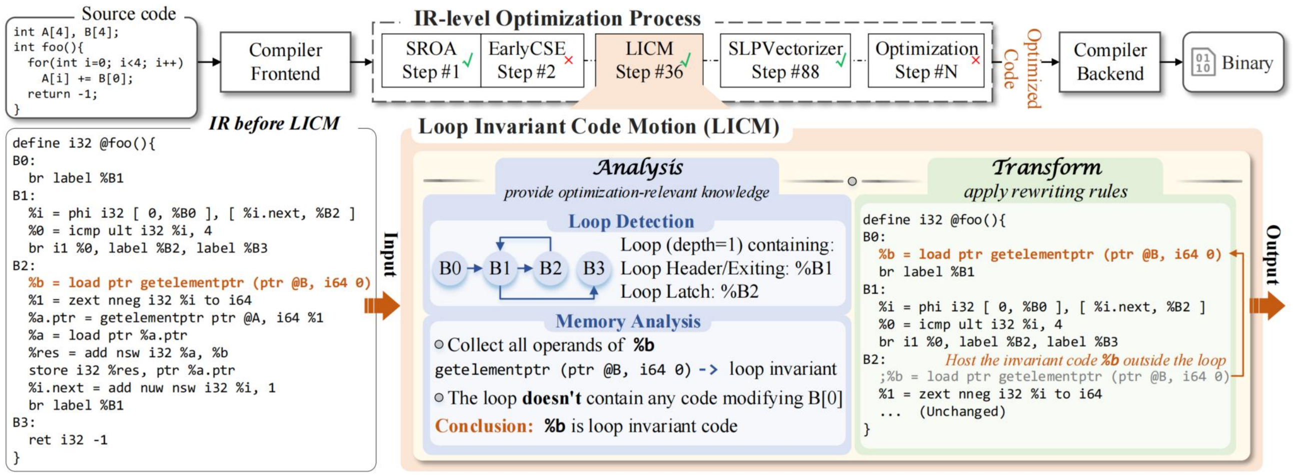
**Conclusion:** **%b** is loop invariant code

```
define i32 @foo(){
B0:
  %b = load ptr getelementptr (ptr @B, i64 0)
  br label %B1
B1:
  %i = phi i32 [ 0, %B0 ], [ %i.next, %B2 ]
  %0 = icmp ult i32 %i, 4
  br i1 %0, label %B2, label %B3
B2:          Host the invariant code %b outside the loop
  ;%b = load ptr getelementptr (ptr @B, i64 0)
  %1 = zext nneg i32 %i to i64
  ...  (Unchanged)
}
```

- ✓ **Analysis Pass:** Provide compiler-specific optimization knowledge.
- ✓ **Transform Pass:** Modify IR based on analysis.

# Background

## ➢ Traditional IR-level Optimization Process



Source code

```
int A[4], B[4];
int foo(){
  for(int i=0; i<4; i++)
    A[i] += B[0];
  return -1;
}
```

IR before LICM

```
define i32 @foo(){
B0:
  br label %B1
B1:
  %i = phi i32 [ 0, %B0 ], [ %i.next, %B2 ]
  %0 = icmp ult i32 %i, 4
  br i1 %0, label %B2, label %B3
B2:
  %b = load ptr getelementptr (ptr @B, i64 0)
  %1 = zext nneg i32 %i to i64
  %a.ptr = getelementptr ptr @A, i64 %1
  %a = load ptr %a.ptr
  %res = add nsw i32 %a, %b
  store i32 %res, ptr %a.ptr
  %i.next = add nuw nsw i32 %i, 1
  br label %B1
B3:
  ret i32 -1
}
```

Loop Detection

Loop (depth=1) containing:
Loop Header/Exiting: %B1
Loop Latch: %B2

Memory Analysis

○ Collect all operands of **%b**
`getelementptr (ptr @B, i64 0) ->` loop invariant

○ The loop **doesn't** contain any code modifying B[0]

**Conclusion:** **%b** is loop invariant code

```
define i32 @foo(){
B0:
  %b = load ptr getelementptr (ptr @B, i64 0)
  br label %B1
B1:
  %i = phi i32 [ 0, %B0 ], [ %i.next, %B2 ]
  %0 = icmp ult i32 %i, 4
  br i1 %0, label %B2, label %B3
B2:          Host the invariant code %b outside the loop
  ;%b = load ptr getelementptr (ptr @B, i64 0)
  %1 = zext nneg i32 %i to i64
  ...  (Unchanged)
}
```

## ➢ Current Problem:

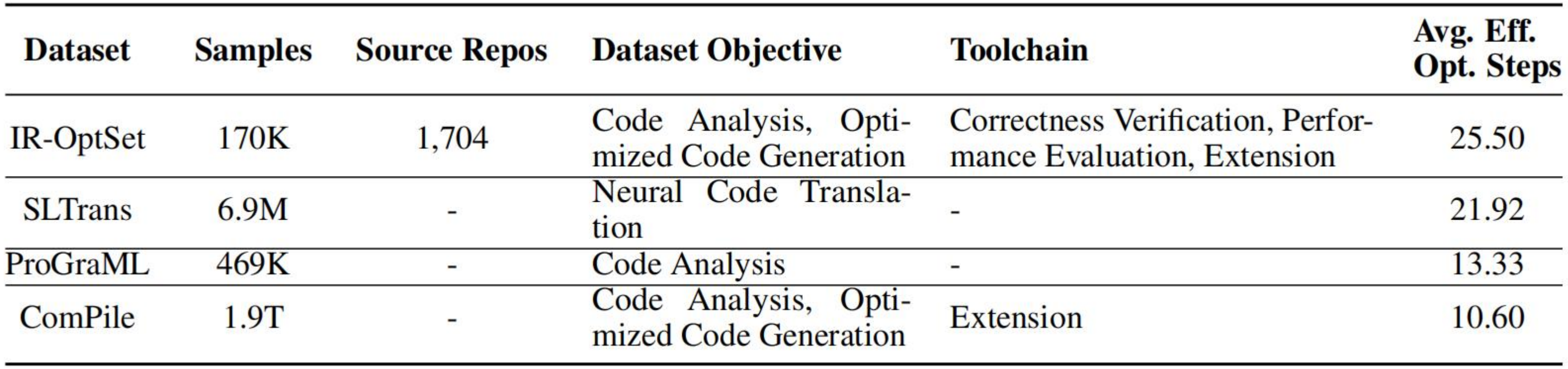
- ✓ Limited accuracy in LLMs for IR optimization.

## ➢ Solution:

- ✓ A dataset to reflect how compilers apply optimizations across diverse program.

# IR-OptSet: Dataset for IR Optimization

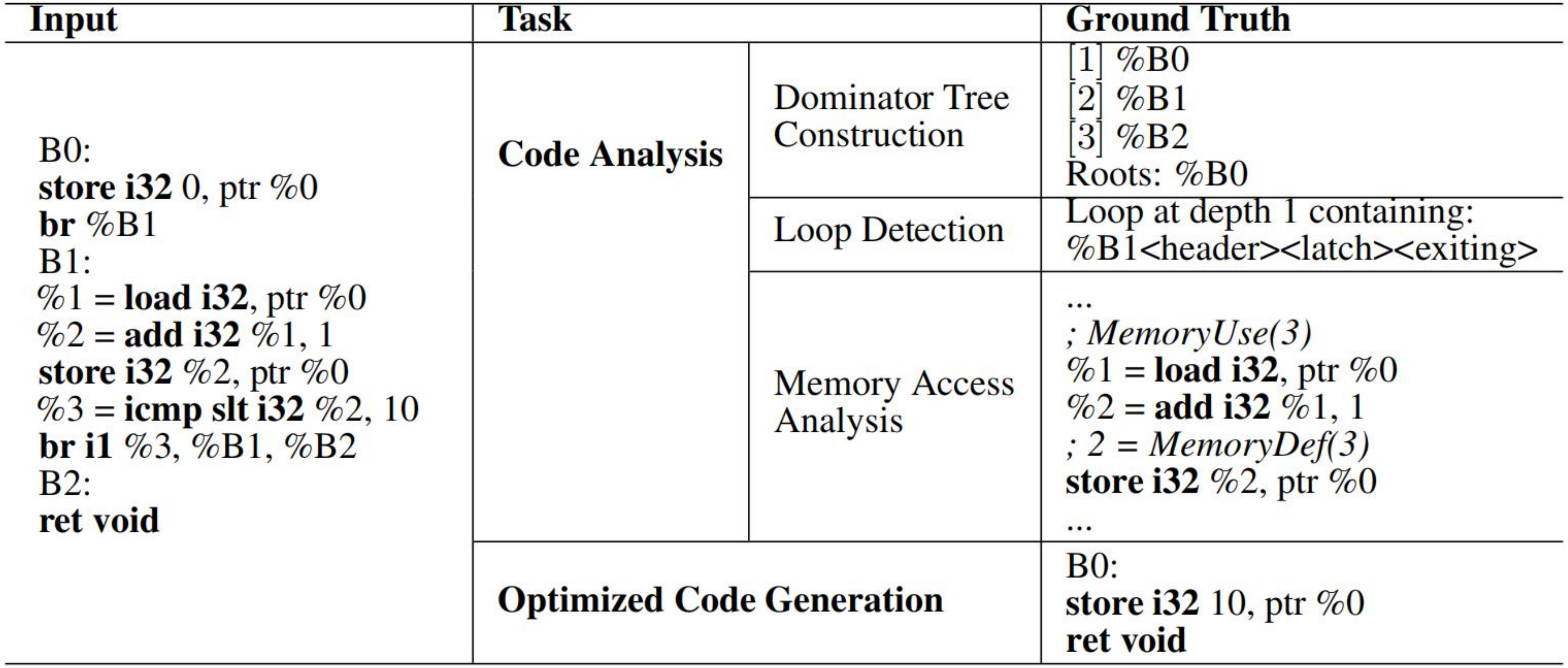
## ➢ Feature



| Dataset | Samples | Source Repos | Dataset Objective | Toolchain | Avg. Eff. Opt. Steps |
|---|---|---|---|---|---|
| IR-OptSet | 170K | 1,704 | Code Analysis, Optimized Code Generation | Correctness Verification, Performance Evaluation, Extension | 25.50 |
| SLTrans | 6.9M | - | Neural Code Translation | - | 21.92 |
| ProGraML | 469K | - | Code Analysis | - | 13.33 |
| ComPile | 1.9T | - | Code Analysis, Optimized Code Generation | Extension | 10.60 |

- ✓ **Optimization-Sensitive:** Targeting real-world programs that trigger various compiler optimizations.
- ✓ **Task-Oriented:** 2 tasks aligned with the traditional IR optimization process.
- ✓ **Comprehensive Toolchain:** 3 tools for evaluation and extension.
- ✓ **Availability:** https://huggingface.co/datasets/YangziResearch/IR-OptSet

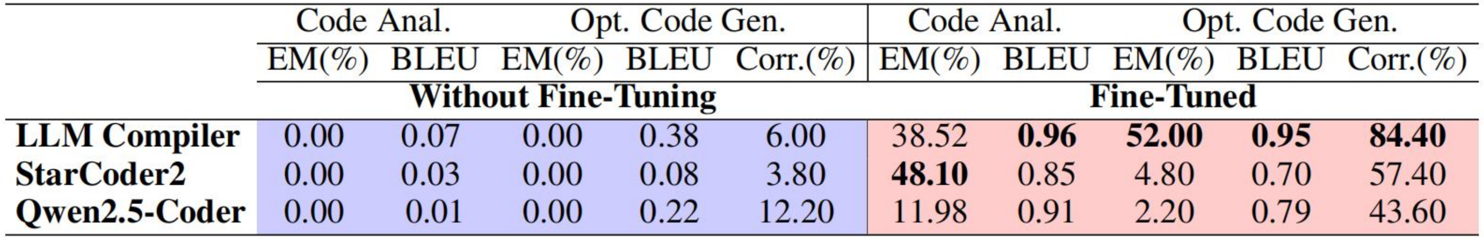
# Two Tasks in IR-OptSet



| Input | Task | | Ground Truth |
|---|---|---|---|
| B0:<br>**store i32** 0, ptr %0<br>**br** %B1<br>B1:<br>%1 = **load i32**, ptr %0<br>%2 = **add i32** %1, 1<br>**store i32** %2, ptr %0<br>%3 = **icmp slt i32** %2, 10<br>**br i1** %3, %B1, %B2<br>B2:<br>**ret void** | **Code Analysis** | Dominator Tree Construction | [1] %B0<br>[2] %B1<br>[3] %B2<br>Roots: %B0 |
| | | Loop Detection | Loop at depth 1 containing:<br>%B1<header><latch><exiting> |
| | | Memory Access Analysis | ...<br>; *MemoryUse(3)*<br>%1 = **load i32**, ptr %0<br>%2 = **add i32** %1, 1<br>; *2 = MemoryDef(3)*<br>**store i32** %2, ptr %0<br>... |
| | **Optimized Code Generation** | | B0:<br>**store i32** 10, ptr %0<br>**ret void** |

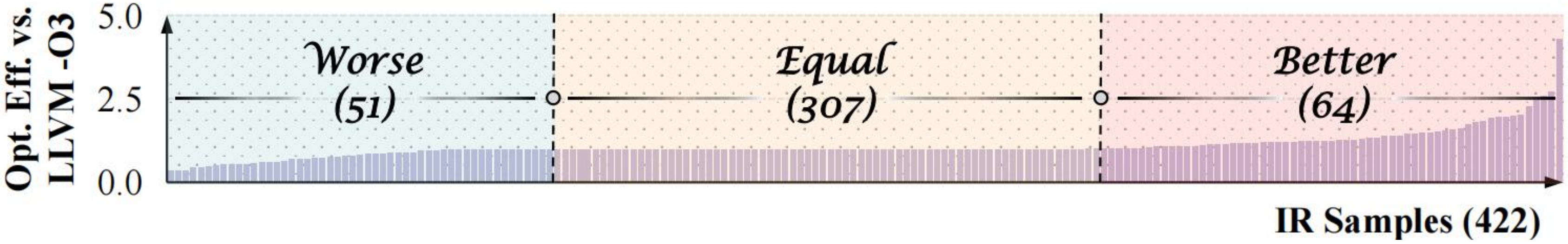
# Evaluation

- **Setup:** 3 LLMs for fine-tuning.
  - ✓ LLM Compiler FTD 7B: LLM-based IR optimizer.
  - ✓ StarCoder2-3B & Qwen2.5-Coder-1.5B: General-purpose code LLMs.
- **RQ.1 Improvements in IR analysis and optimization through fine-tuning.**



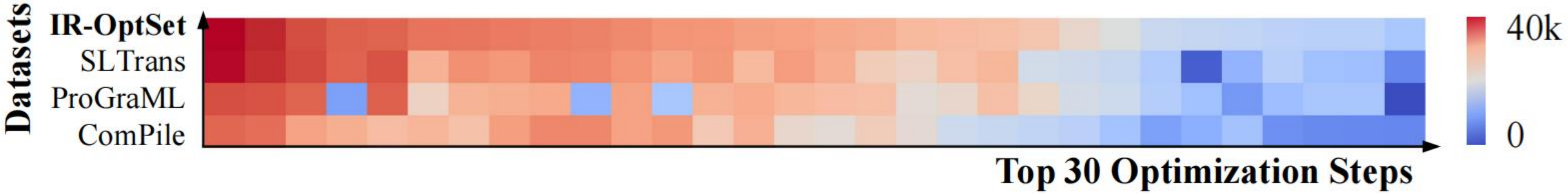
| | Code Anal. | | Opt. Code Gen. | | | Code Anal. | | Opt. Code Gen. | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | EM(%) | BLEU | EM(%) | BLEU | Corr.(%) | EM(%) | BLEU | EM(%) | BLEU | Corr.(%) |
| | **Without Fine-Tuning** | | | | | **Fine-Tuned** | | | | |
| **LLM Compiler** | 0.00 | 0.07 | 0.00 | 0.38 | 6.00 | 38.52 | **0.96** | **52.00** | **0.95** | **84.40** |
| **StarCoder2** | 0.00 | 0.03 | 0.00 | 0.08 | 3.80 | **48.10** | 0.85 | 4.80 | 0.70 | 57.40 |
| **Qwen2.5-Coder** | 0.00 | 0.01 | 0.00 | 0.22 | 12.20 | 11.98 | 0.91 | 2.20 | 0.79 | 43.60 |

- **RQ.2 Potential to surpass traditional compiler performance.**
  - ✓ LLM Compiler FTD IR-OptSet vs. LLVM 19.0.1 -O3



# Evaluation

- **RQ.3 Diversity in transformations compared to existing datasets.**
  - ✓ IR-OptSet vs. 3 IR-oriented datasets (SLTrans, ProGraML, and ComPile).
  - ✓ Effectiveness frequency of the top 30 most commonly used optimization steps.

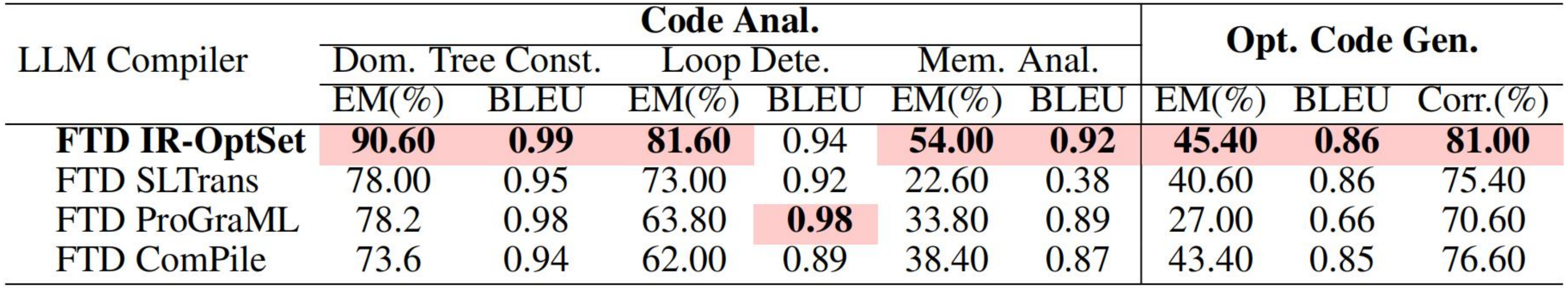


-> *IR-OptSet consistently triggers the highest effectiveness frequency across nearly all top-30 optimization steps.*

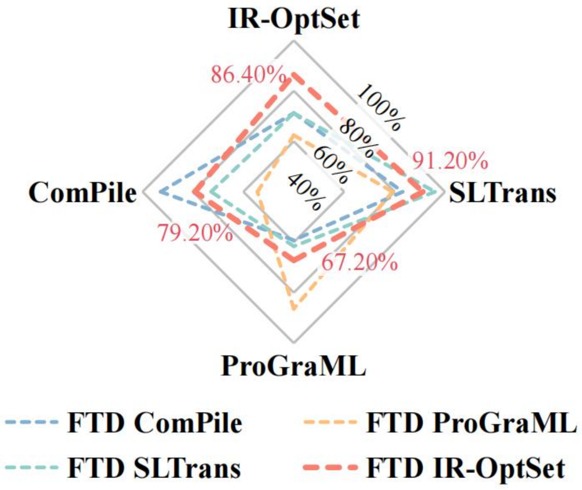
# Evaluation

- **RQ.4 Generalization in transformations compared to existing datasets.**
  - ✓ IR-OptSet vs. 3 IR-oriented datasets (SLTrans, ProGraML, and ComPile).
  - ✓ **Training set:** IR-OptSet/SLTrans/ProGraML/ComPile.

    -> 4 variants: LLM Compiler FTD IR-OptSet, SLTrans, ProGraML, and ComPile.
  - ✓ **Test set:** 500 samples in total, with 125 randomly selected from each dataset.



| LLM Compiler | Code Anal. | | | | | | Opt. Code Gen. | | |
|---|---|---|---|---|---|---|---|---|---|
| | Dom. Tree Const. | | Loop Dete. | | Mem. Anal. | | | | |
| | EM(%) | BLEU | EM(%) | BLEU | EM(%) | BLEU | EM(%) | BLEU | Corr.(%) |
| **FTD IR-OptSet** | **90.60** | **0.99** | **81.60** | 0.94 | **54.00** | **0.92** | **45.40** | **0.86** | **81.00** |
| FTD SLTrans | 78.00 | 0.95 | 73.00 | 0.92 | 22.60 | 0.38 | 40.60 | 0.86 | 75.40 |
| FTD ProGraML | 78.2 | 0.98 | 63.80 | **0.98** | 33.80 | 0.89 | 27.00 | 0.66 | 70.60 |
| FTD ComPile | 73.6 | 0.94 | 62.00 | 0.89 | 38.40 | 0.87 | 43.40 | 0.85 | 76.60 |



# Thanks!

yangzi.research@outlook.com