RL4ReAl: Reinforcement Learning for Register Allocation

S. VenkataKeerthy, Siddharth Jain, Anilava Kundu,
Rohit Aggarwal, Albert Cohen, Ramakrishna Upadrasta

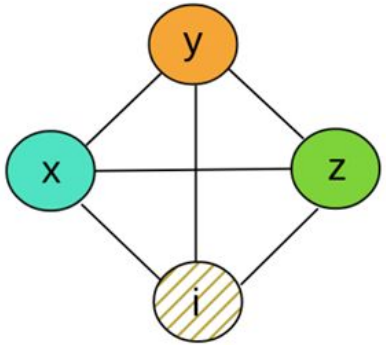
Presented by: Haocheng Ren, Yuning Cong, Xiuru Ruan
Group 18

Outline

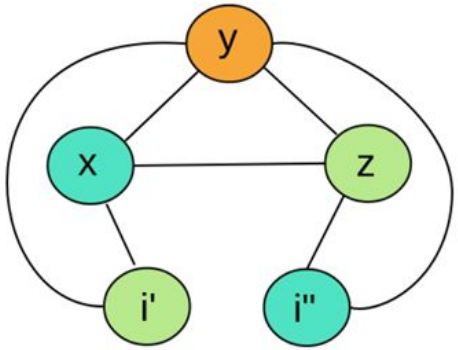
1. Quick Recap
2. Issues of ML in Register Allocation
3. Reinforcement Learning
4. Overall RL4ReAl Workflow
5. Details in the workflow
6. Evaluation
7. Commentary

Quick Recap

1. **Coloring**: assign PR
2. **Spilling**: if no available PR, put into memory
3. **Splitting**: split a variable across multiple registers or between registers and memory



(b) Without splitting i
- one spill



(c) After splitting i
- no spills

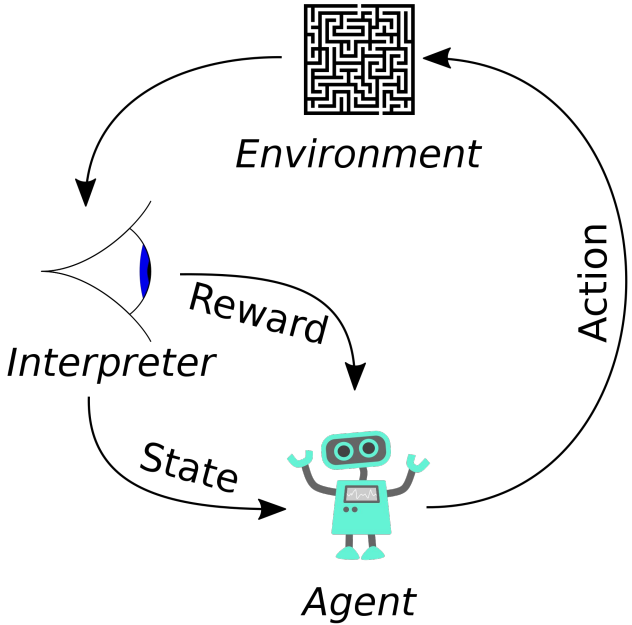
Issues of ML in Register Allocation

Applicability and Effectiveness

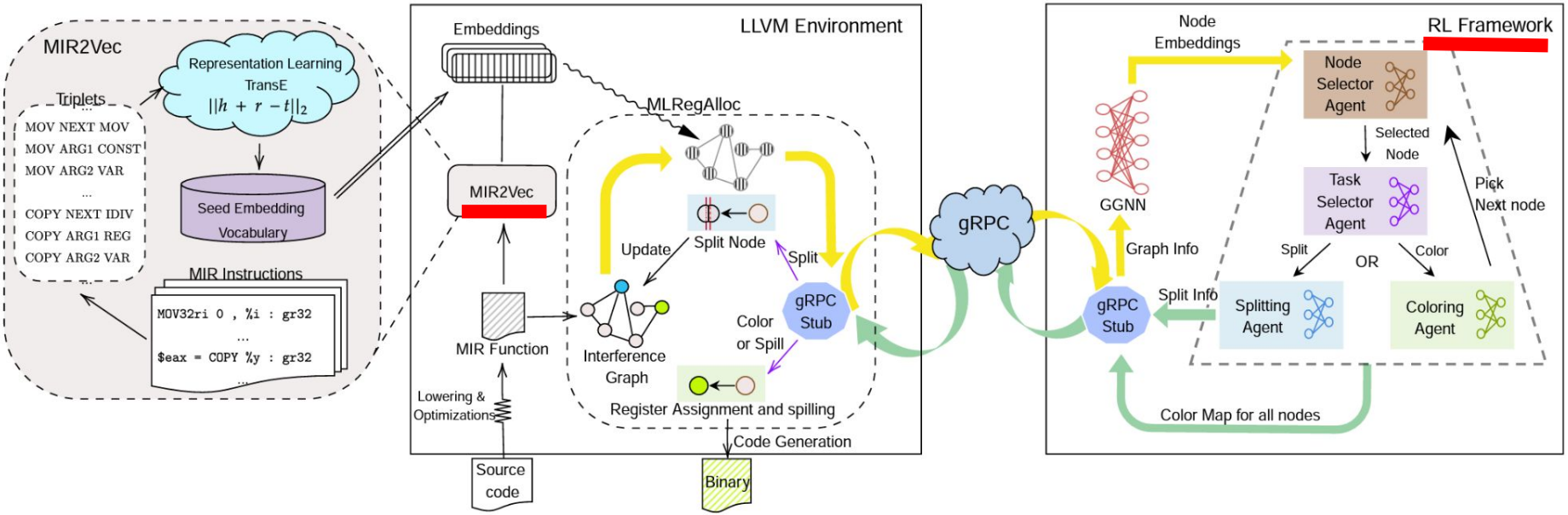
1. Register allocation is complex
    a. sub-tasks (spilling, splitting, etc.)
2. Absolute correctness from ML-based approach
    a. live range
    b. register type
3. Integration
    a. ML models, RL algo in python
    b. compiler frameworks in C++

Reinforcement Learning

1. A branch of ML
2. How?
    a. Agent learns a policy
    b. Positive/Negative reward
3. In register allocation
    a. Hierarchical Multi-Agent RL (state, action, reward)
        i. Node Selection Agent (Top)
            1. State space
            2. Action
            3. Reward
        ii. Task Selection Agent (Mid)
        iii. Coloring Agent (Low)
        iv. Splitting Agent (Low)



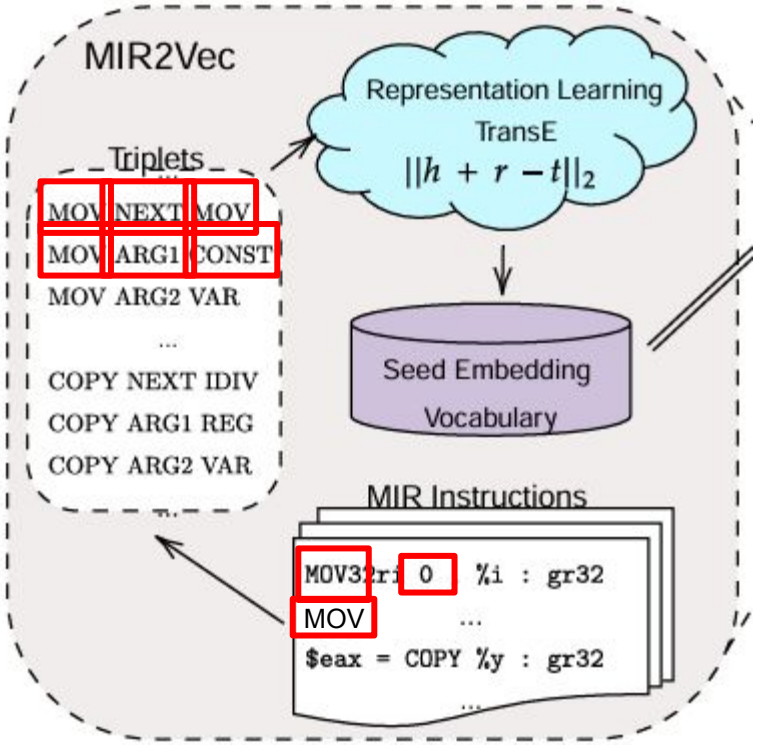
RL4ReAl Workflow



MIR2Vec

1. Create Triplets
   a. (opcode, NEXT, opcode)
   b. (opcode, ARGi, arg)

2. Feed the triplets to TransE to obtain embeddings of each MIR instruction opcode and argument



Register Constraints - I

1. **Type constraint**
   - Virtual registers can only be assigned with physical registers of the same type

2. **Interference constraint**
   - No adjacent nodes of an interference graph should be allocated the same color

Register Constraints - II

3. **Congruence constraint**

   a. Some registers are **physically part** of some other wider registers

   b. Registers that adhere to this part of relation belong to the same **congruence class**

   c. Virtual registers with overlapping live ranges should not be assigned with physical registers of the same congruence class

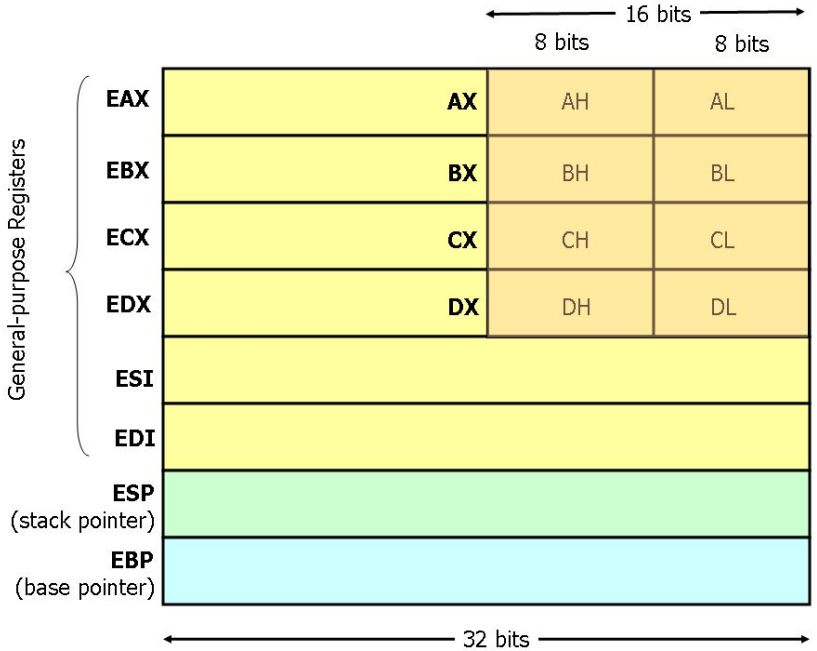
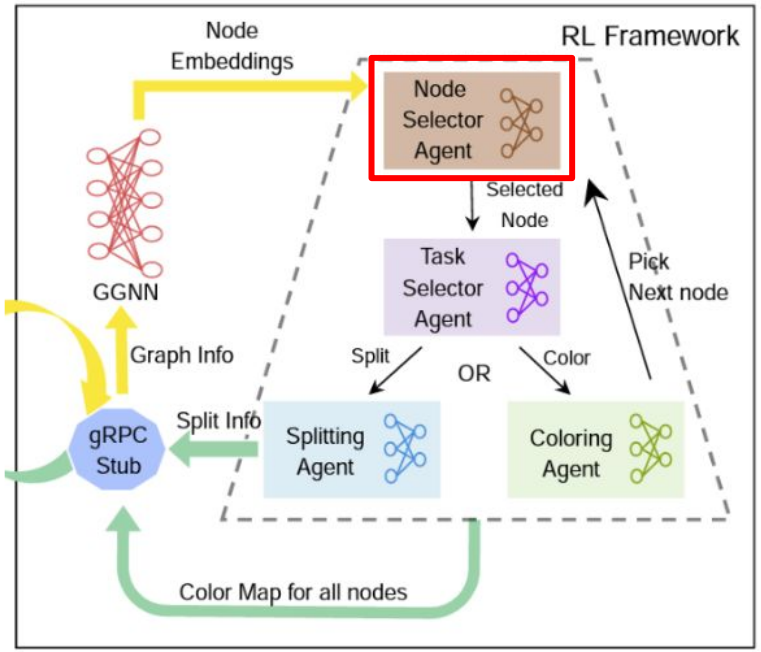


Figure 1. x86 Registers

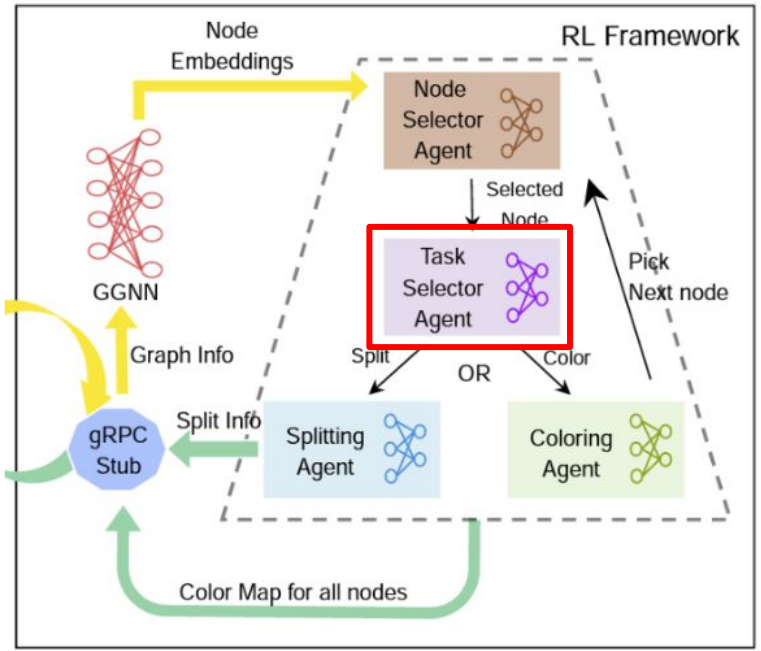
Node Selector

1. **State Space**
   a. Embedding of each vertex in Graph
   b. Spill weights of the vertices
2. **Action**
   a. Select an uncolored vertex for allocation
3. **Reward**
   a. Same reward of the task selector agent



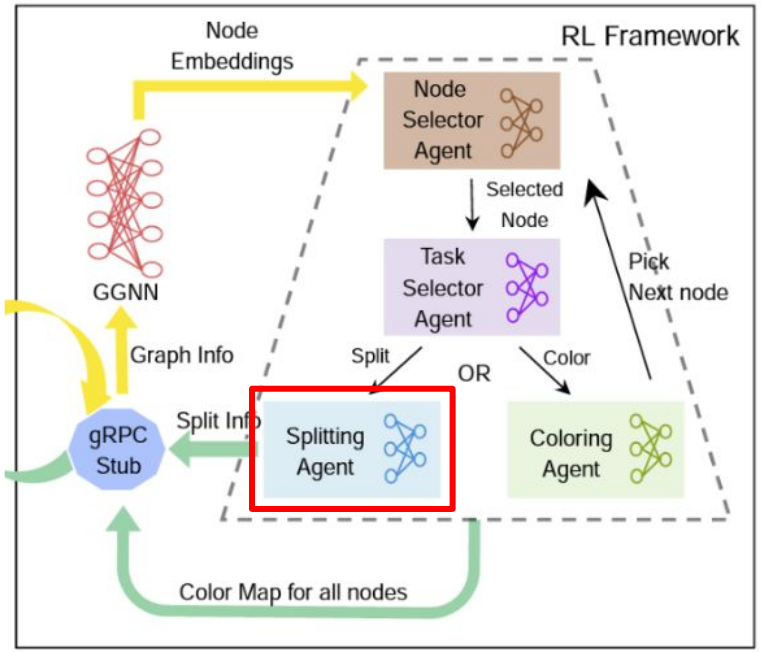
Task Selector

1. **State Space**
   a. Embedding of the vertex v picked by the node selector
   b. Number of available registers
   c. Number of interferences of v
   d. Life-time of v
   e. Spill weight of v
2. **Action**
   a. Select a task among coloring and splitting
3. **Reward**
   a. Same reward of the coloring agent if choosing to color
   b. 0 if choosing to split



Splitting Agent

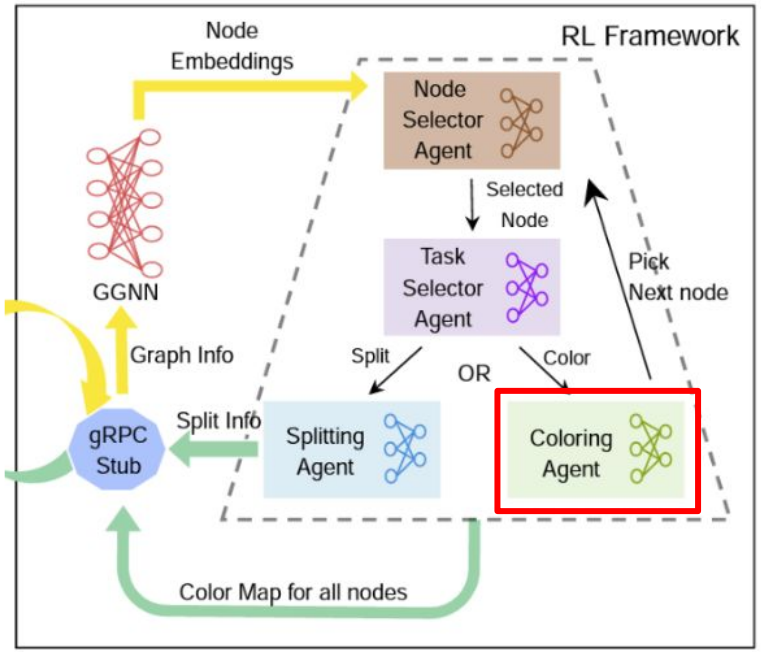
1. **State Space**
   a. Embedding of the vertex v picked by the node selector
   b. Spill weights at each use of v
   c. Distance between each successive use of v
2. **Action**
   a. learn an optimal split point in the live range among all the points of use of v
3. **Reward**
   a. Difference in spill weights of v before and after splitting



Coloring Agent

Decide the register to be assigned

1. **State space**
   a. Embedding of the vertex v picked by the node selector
   b. Number of available registers satisfying the constraints (legal registers)
   c. Number of uncolored vertices in G
   d. Spill weight M(v) of v
2. **Action**
   a. Pick a legal register if available, spill v otherwise
3. **Reward**
   a. +M(v) if a legal register exists, -M(v) otherwise



Experimental Evaluation

1. MIR2Vec representations: train on C++ Boost Library source file and SPEC CPU 2017 benchmarks with –O3; TransE model with triplet (op, $NEXT$, op) or (op, $ARGi$, arg) relation
2. Target Processors: x86 / AArch64
3. Involved registers: general-purpose, vector, floating point
4. RL Training datasets: 5k random functions from SPEC CPU 2017 benchmarks
5. Benchmark: 13 benchmarks in SPEC CPU 2006 and SPEC CPU 2017

Benchmarks – Hot Functions

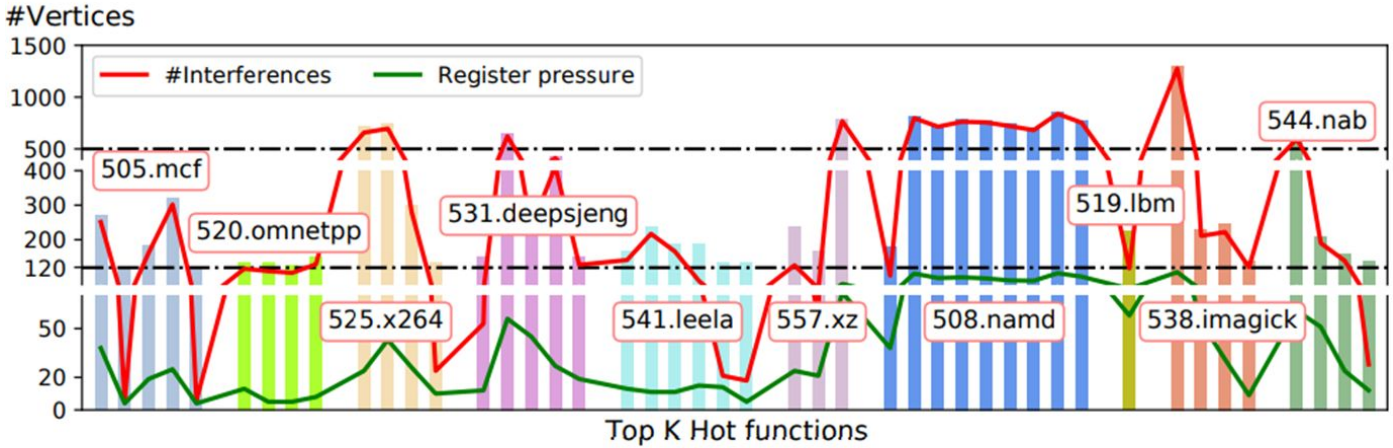


Figure 4. Correlation between #Vertices, #Interferences and Register Pressure

- Register Pressure: Maximum number of overlapping live ranges
- Most hot functions have over 120 vertices: near linear correlation with register pressure
- < 120 vertices: Greedy

Evaluation-x86

- L: Trained with Local reward
- G: Trained with Global reward
- Evaluated speedup(+)/slow-down(-) by seconds
- Close to Greedy
- Policy Improvement Cycle

Table 3. Runtime improvement (s) on x86 over BASIC, highlighting the Highest and Second highest improvements.

Benchmarks	Runtime BASIC	Diff. from BASIC (BASIC- x)			
		PBQP	GREEDY	RL4REAL	
				L	G
401.bzip2	360.6	-7.3	7.5	-1.1	10.8
429.mcf	233.8	1.4	-2.9	2.7	-3.6
445.gobmk	322.3	-3.3	6.4	2.4	1.7
456.hmmer	284.3	1.8	6.1	5.0	-37.6
462.libquantum	256.4	-10.1	-1.1	-2.2	-6.7
471.omnetpp	305.7	0.7	0.4	1.2	1.2
433.milc	349.1	-16.6	0.1	-13.8	-7.0
470.lbm	184.0	-7.9	3.0	2.3	1.4
482.sphinx3	366.0	-37.5	1.6	-3.1	-2.7
505.mcf_r	344.9	4.5	-1.6	8.6	-4.7
520.omnetpp_r	475.7	6.4	6.4	2.4	2.8
531.deepsjeng_r	299.9	4.6	16.0	9.9	12.8
541.leela_r	439.5	1.6	7.1	0.4	1.9
557.xz_r	371.5	-0.6	11.9	12.1	-8.5
508.namd_r	236.5	2.5	23.5	9.1	23.8
519.lbm_r	261.8	1.4	57.7	50.9	58.1
538.imagick_r	479.3	-16.9	115.5	118.8	118.4
544.nab_r	417.5	5.8	132.1	131.3	134.4
	Average	-3.9	21.7	18.7	16.5

Evaluation-Runtime

- Take a deeper dive into comparison with Greedy
- Greedy show no improvement over Basic on 2006 due to BZ2_compressBlock
- RL4REAL fix the failure

Table 2. % runtime diff. over BASIC on hot functions

	SPEC CPU 2017			SPEC CPU 2006		
	GREEDY	RL4REAL		GREEDY	RL4REAL	
		L	G		L	G
Average	6.2	**7.3**	4.8	**-1.5**	-2.1	-1.6
# (val>0)	**23**	**23**	17	16	**17**	13
# (val<0)	**8**	**8**	14	19	**18**	22
Max	44.0	**44.4**	41.3	**12.7**	10.4	6.2
Min	-7.7	**-4.4**	-10.8	-51.4	-52.5	**-13.1**

Table 4. % speedup of GREEDY and RL4REAL over BASIC on SPEC 2006 and 2017 (x86)

B/M	Functions	GREEDY	RL4REAL	Diff.
Top 5 functions with highest % speedup (over GREEDY)				
401	BZ2_compressBlock	-51.3	-5.2	46.1
445	do_get_read_result	-12.0	-0.5	11.5
482	mgau_eval	-6.0	0.3	6.3
429	price_out_impl	-0.8	2.3	3.2
445	subvq_mgau_shortlist	-9.8	-6.9	2.9

Comments

Pros:

1. Developed an overall RL framework for register allocation on real-world dataset
2. Architecture independent: MIR specific

Cons:

1. No single allocator that performs best across all benchmarks
2. Network structure for sub-task agents are relatively trivial (only FC)
3. Relies on LLVM-gRPC as integration method
4. End-to-end structure lacks detailed intermediate analysis

Thank you!

Q&A