Enabling AI-based Automated Code Generation with Guaranteed Correctness

E. Vatai, A. Drozd, I. R. Ivanov, J. E. Batista, Y. Ren, M. Wahib

Riken R-CCS, High Performance Artificial Intelligence Systems Research Team, Japan

Outline

The Team

Emil
VATAI

Aleksandr
DROZD

Ivan R.
IVANOV

João E.
BATISTA

Mohamed
WAHIB


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

ML approaches to correctness in code generation

- Extensive unit testing
  - Coverage
  - Writing tests is not trivial
- Surrogate ML model
- Round-trip
  - Trust issues
- Limit ML to short sequences of instructions
- Limit ML to a set of determined transformations
  - Not general
- Symbolic execution
  - Not well established

Edsger W. Dijkstra

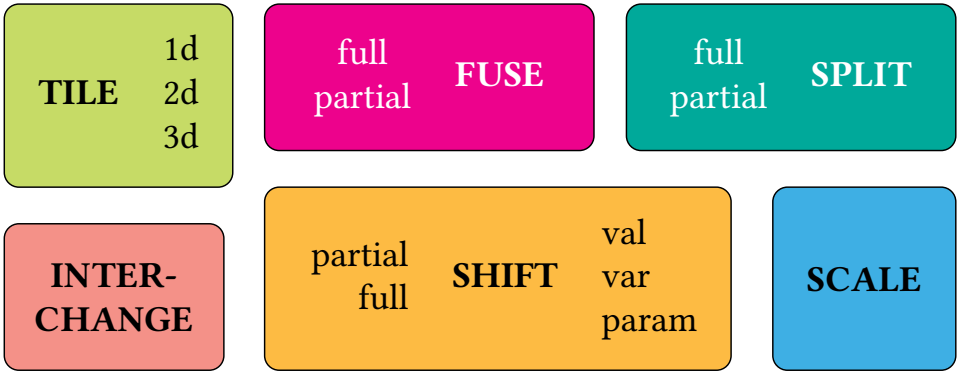
> *"Program testing can be used to show the presence of bugs, but never to show their absence!"*

Some people don't even do that

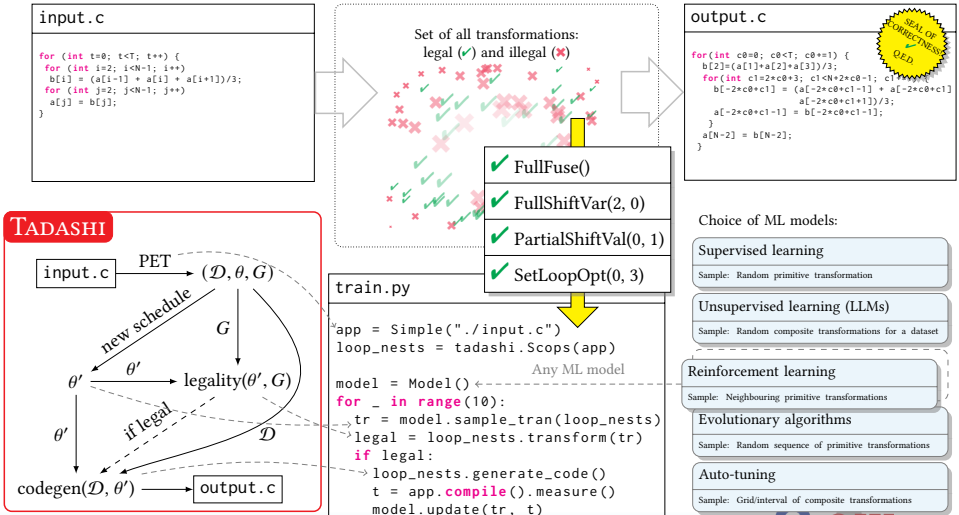
All you need is to sample the set of correct transformations

- ML codegen is transformations on a reference implementation
- ML can explore the space of transformations to find a faster version
- But we also need correctness

Loop Transformations



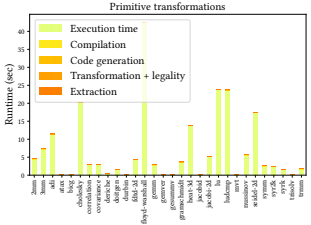
TADASHI: loop transformations with correctness check

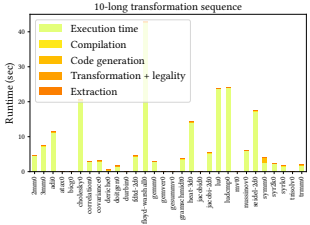


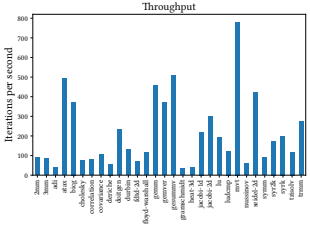
End-to-end example

```
1  import tadashi
2  from tadashi.apps import Simple
3  app = Simple("input.c")
4  ### Specify transformation ###
5  node = app.scops[0].schedule_tree[1]
6  tr = tadashi.TrEnum.FULL_SHIFT_VAR
7  args = [1, 13]
8  ### Transform ###
9  legal = node.transform(tr, *args)
10 tapp = app.generate_code()
11 ### Measure ###
12 app.compile()
13 tapp.compile()
14 speedup = app.measure() / tapp.measure()
15 print(f"{speedup=}")
```

Overhead breakdown (primitive, 10 seq), Throughput



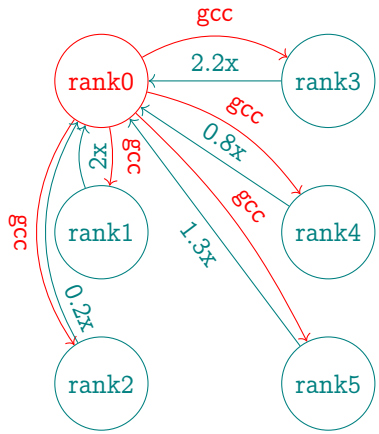




Benchmarking harness

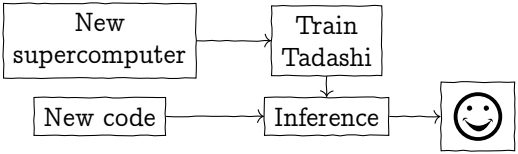
The compiler spends a few minutes optimising the code on a single node. Why not use the entire supercomputer?

- Benchmarking harness
  - Distribute compiling and running of benchmarks
  - Collect the speedups to remote nodes
  - Uses `MPI4py`, and exposed to the user as `concurrent.futures` (from the Python standard)



Grand vision

Large scale optimisation framework

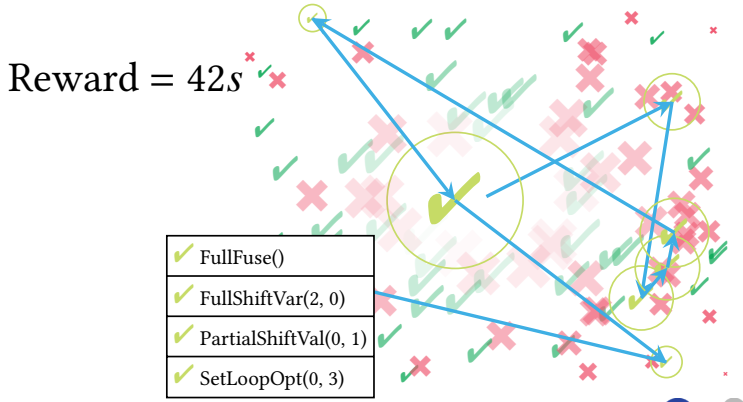


Wise words

Spending 6h on the whole machine to make the code 5% faster will pay off. – N. D.

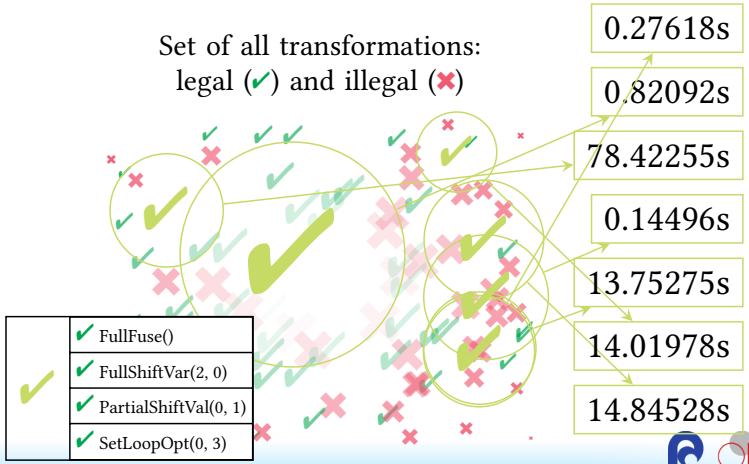
Reinforcement learning

- Actions = primitive transformations; Reward = walltime; Environment = correctness



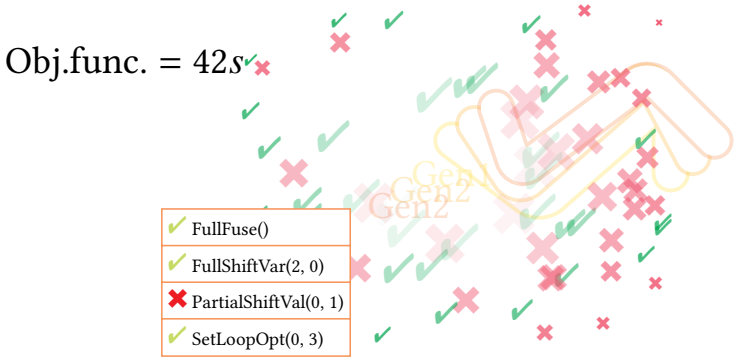
Supervised learning

- Dataset generation (legal transformations); Sample label: runtime



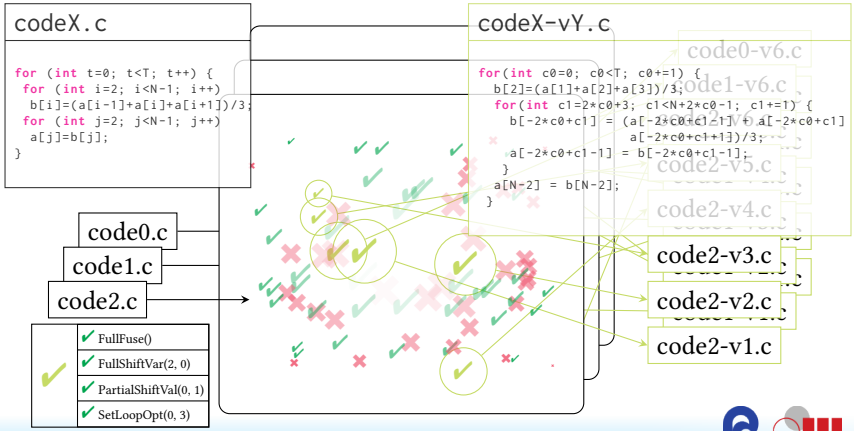
Evolutionary algorithms

- Exploration and explorations; Candidates = transformations; Objective function = runtime



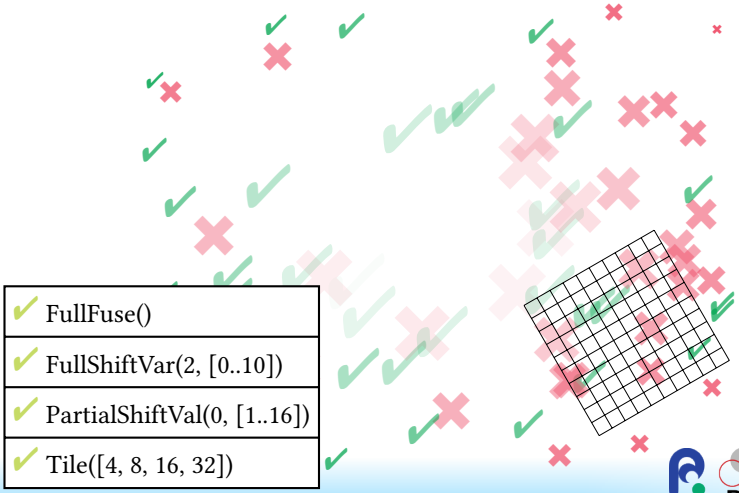
LLM agents

- Dataset generation; High quality correct transformations; Caveat! Hallucinations are possible!



Auto-Tuning

- Brute forcing on a bounded region of the space (grid search)



ML opportunities

Different levels of abstraction/representations:

- source code [Stein: Paraphrasing etc.]
- abstract syntax tree (AST) [Shido: Tree-LSTM etc.]
- polyhedral [Baghdadi: Tiramisu]
- dependency graphs [Cummins: ProGraML]
- intermediate Representations (IR) [Ben-nun: inst2vec]
- assembly instructions [Deepmind: Faster sorting]

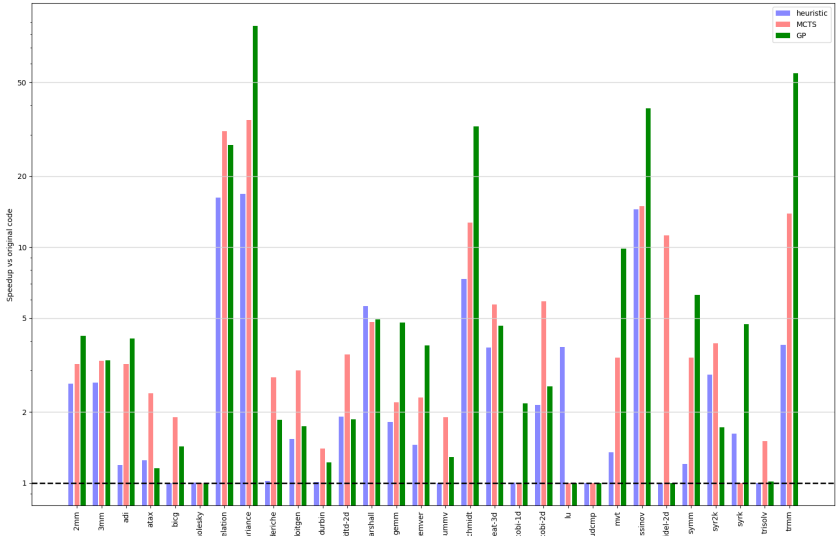
Three levels of transfer learning

1. No Transfer: Train from scratch for each (SW, HW) pair
2. Transfer to new software
3. Transfer to new hardware

Less retraining, Better exploration $\Rightarrow$ better results for a given kernel

Speedups: PoC, MCTS, GP

for heuristic, Monte Carlo Tree Search, Genetic Programming



Real world applications

- miniAMR 5%
- Finite-difference time-domain FDTD 13x

Known limitations

- C only
- SCoP (polyhedral transformations)
- PET/ISL
- CPU only

That's all folks!

- i. https://vatai.github.io/
- ii. https://github.com/vatai/tadashi/
- iii. https://arxiv.org/abs/2410.03210

i. 

ii. 

iii. 

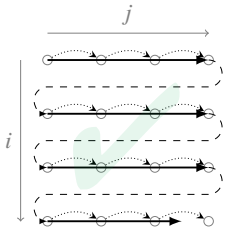
Polyhedral model

Components

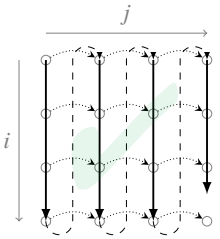
1. $\mathcal{D}_S = \{S[\vec{i}] \in \mathbb{Z}^n : \mathbf{A}\vec{i} + \mathbf{b} \leq \mathbf{0}\}$
2. $\theta(S[i,j]) = t = (i,j)$
3. $G = (V, E)$:
   - $V = \{S_0, S_1, \ldots\}$,
   - $E = \{S_i[\vec{d}] \mapsto S_j[\vec{r}], \ldots\}$

Mini example

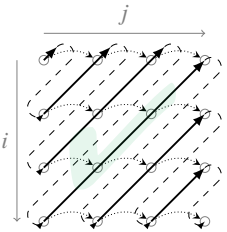
```
for(int i = 0; i < N; i++)
  for(int j = 1; j < M; j++)
    A[i, j] += A[i, j-1]; // S[i
```



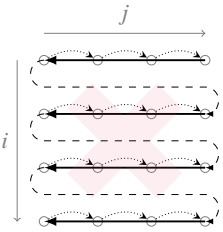
a $\theta(S[i,j]) = (i,j)$



b $\theta(S[i,j]) = (j,i)$



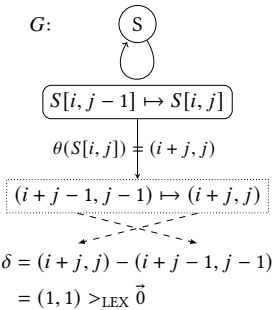
c
$\theta(S[i,j]) = (i+j,j)$



d $\theta(S[i,j]) = (i,-j)$

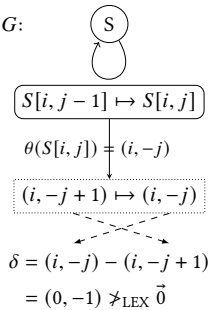
Legality check

G:



$\delta = (i+j,j) - (i+j-1,j-1)$
$= (1,1) >_{\text{LEX}} \vec{0}$

G:



$\delta = (i,-j) - (i,-j+1)$
$= (0,-1) \not>_{\text{LEX}} \vec{0}$

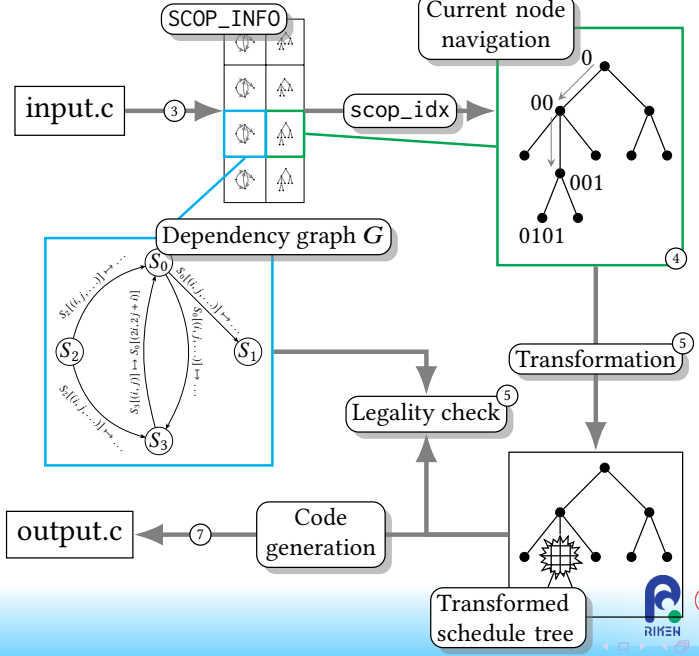
Symbolic representation

$$\{(i,j) : 1 \le i \le N \wedge 1 \le j \le i\}$$

$$\begin{aligned} 1 &\le i \\ i &\le N \\ 1 &\le j \\ j &\le i \end{aligned}$$

$$\begin{aligned} 0 &\le i - 1 \\ 0 &\le N - i \\ 0 &\le j - 1 \\ 0 &\le i - j \end{aligned}$$

N	i	j	const
0	1	0	-1
1	-1	0	0
0	0	1	-1
0	1	-1	0


SCOP_INFO
input.c
3
scop_idx
Current node navigation
0
00
001
0101
4
Dependency graph G
S_0
S_1
S_2
S_3
$S_0[\{i,j,\ldots\}] \mapsto \ldots$
$S_0[\{i,j,\ldots\}] \mapsto \ldots$
$S_1[\{i,j,\ldots\}] \mapsto \ldots$
$S_2[\{i,j,\ldots\}] \mapsto \ldots$
$S_1[\{i,j\}] \mapsto S_0[\{2i,2j+0\}]$
Transformation
5
Legality check
5
Transformed schedule tree
Code generation
output.c
7

That's all folks!

- i. https://vatai.github.io/
- ii. https://github.com/vatai/tadashi/
- iii. https://arxiv.org/abs/2410.03210

i. 

ii. 

iii.