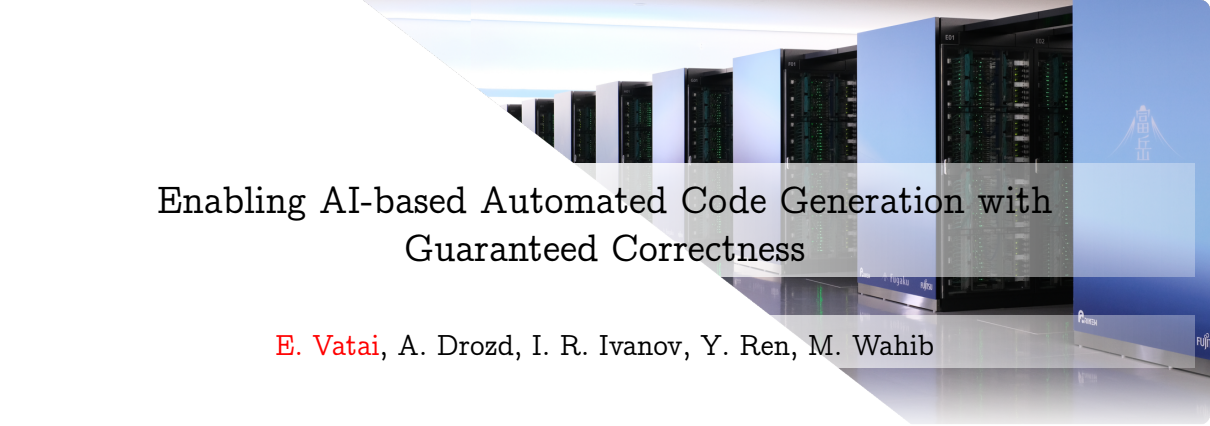


Enabling AI-based Automated Code Generation with Guaranteed Correctness

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

R-CCS, High Performance Artificial Intelligence Systems Research Team


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!"

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!"*

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!"*

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!"*

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!"*

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

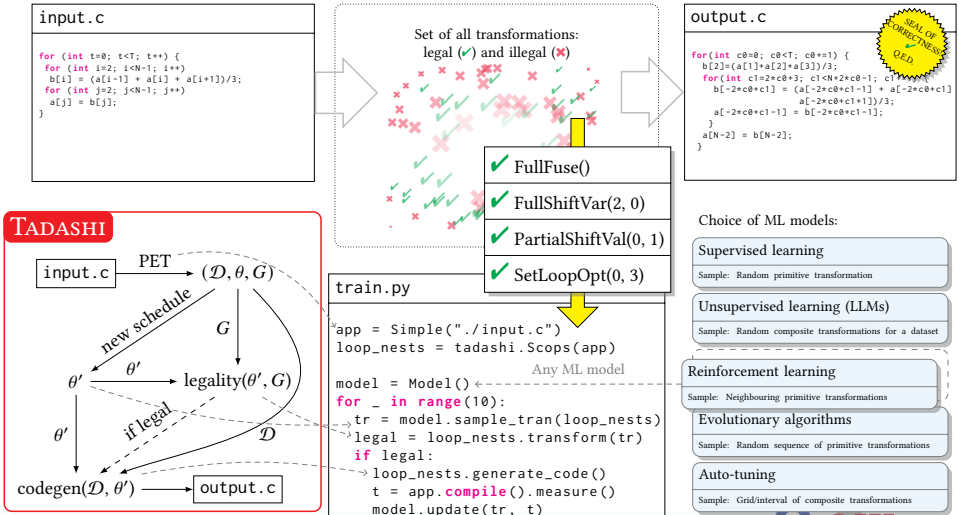
All you need is to sample the set of correct transformations

- 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

All you need is to sample the set of correct transformations

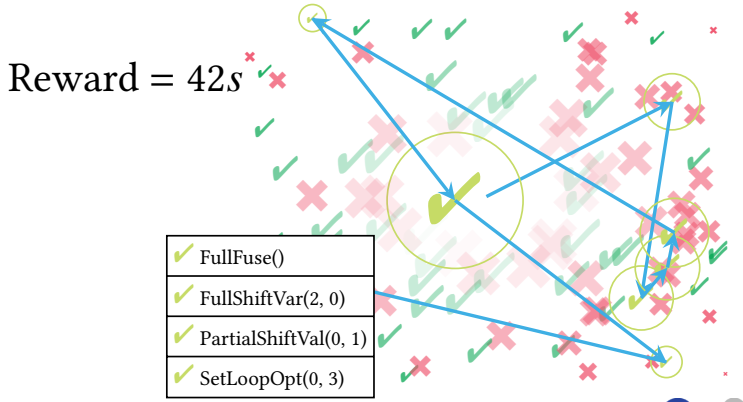
- ML codegen is transformations on a reference implementation
- ML can explore the space of transformations to find a faster version
- But we also need correctness

TADASHI: loop transformations with correctness check



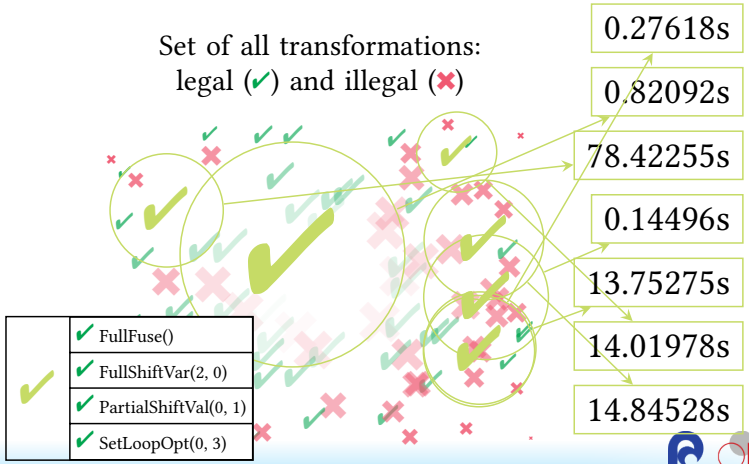
Reinforcement learning

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



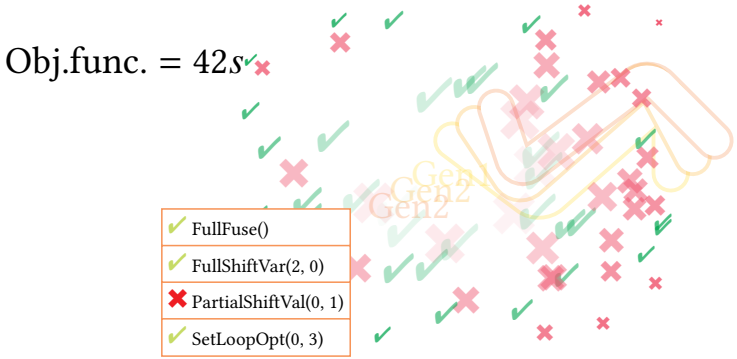
Supervised learning

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



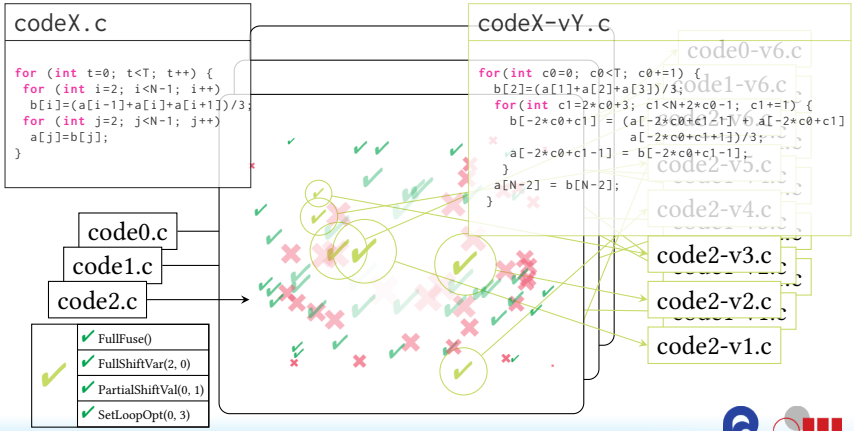
Evolutionary algorithms

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



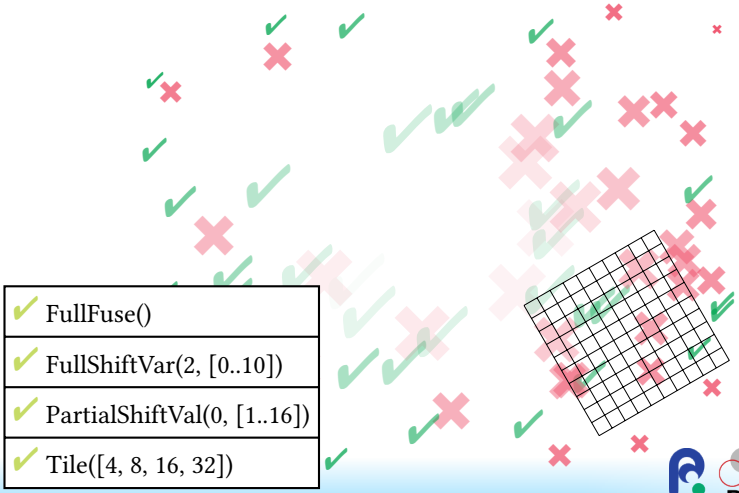
LLM agents

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



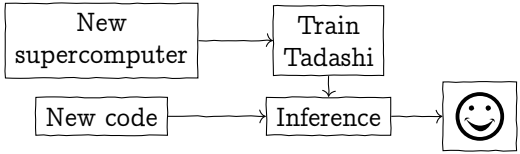
Auto-Tuning

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



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.

Now: Support C, Harness, and Transformations

TrEnum	Args
TILE	tile size
INTERCHANGE	–
FULL_FUSE	–
FUSE	2 loop indices
FULL_SHIFT_VAL	const shift value
PARTIAL_SHIFT_VAL	stmt idx, const shift value
FULL_SHIFT_VAR	coeff, shift iter idx
PARTIAL_SHIFT_VAR	stmt idx, coeff, shift iter idx
FULL_SHIFT_PARAM	coeff, shift param idx
PARTIAL_SHIFT_PARAM	stmt idx, coeff, shift param idx
SCALE	coeff
SET_PARALLEL	–
SET_LOOP_OPT	AstLoopType enum

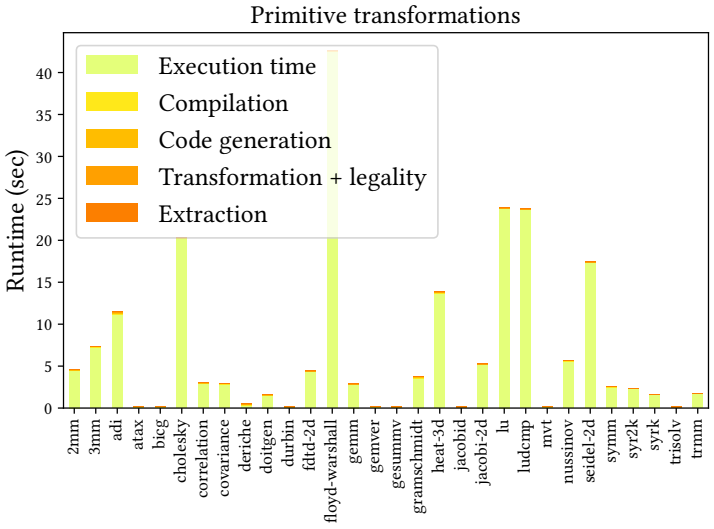
Future:

- New language
  - Fortran (MLIR, Polygeist)
  - CUDA (PPCG?)
- Better use of the polyhedral model
  - New transformations: Loop fission, unrolling, etc.
  - Expansion and extension nodes
  - More efficient legality check
- Harness/infrastructure
  - Scheduler support (slurm, pjsub)
  - Compiler flags
- ML (target list)
  - Reinforcement Learning
  - LLM Agents
  - Auto-Tuning
  - Supervised Learning
  - Evolutionary Algorithms

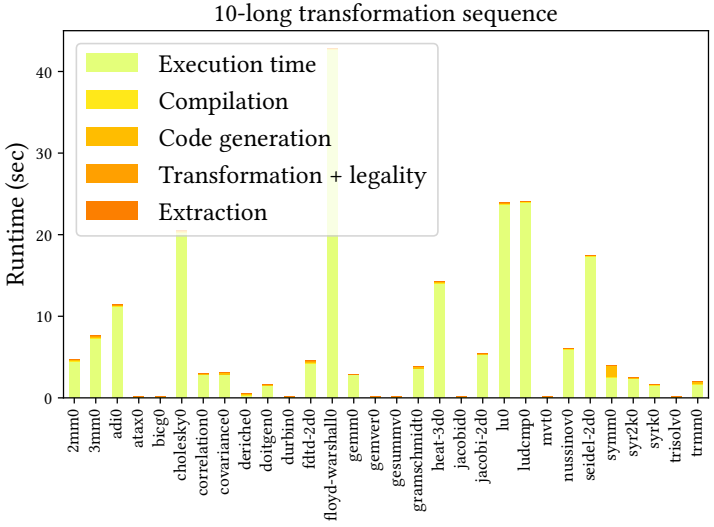
Random RL

```
def train_model(app, num_iter=3, net=Model()):
  loop_nests = LoopNests(app)
  ln = loop_nests[0]
  for i in range(num_iter):
    loop_idx, tr, args = net.transform(ln)
    loop = ln.schedule_tree[loop_idx]
    legal = loop.transform(tr, *args)
    if not legal:
      node.rollback()
      continue
    loop_nests.generate_code("output.c")
    app.compile()
    t = app.measure()
    # net.update(loop_idx, tr, args, legal, t)
```

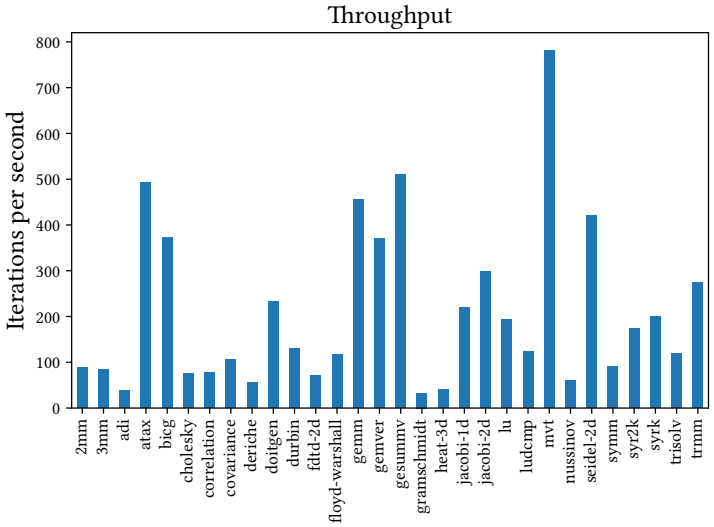
Breakdown (primitive)



Breakdown (10 seq)



Throughput


Throughput
Iterations per second
800
700
600
500
400
300
200
100
0
2mm
3mm
adi
atax
bicg
cholesky
correlation
covariance
deriche
doitgen
durbin
fdtd-2d
floyd-warshall
gemm
gemver
gesummv
gramschmidt
heat-3d
jacobi-1d
jacobi-2d
lu
ludcmp
mvt
nussinov
seidel-2d
symm
syr2k
syrk
trisolv
trmm

Representations

Different levels of abstraction

- 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]

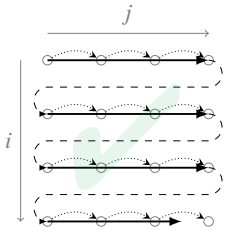
Polyhedral model

Components

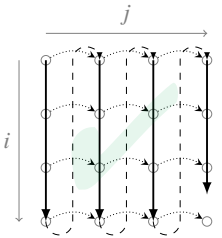
1. $\mathcal{D}_S = \{S[\vec{i}] \in \mathbb{Z}^n : \mathbf{A}\vec{i} + \mathbf{b} \leq \mathbf{0}\}$
2. $\theta(S[i,j]) = t = (i,j)$
3. $G = (V, E)$:
   - $V = \{S_0, S_1, \ldots\}$,
   - $E = \{S_i[\vec{d}] \mapsto S_j[\vec{r}], \ldots\}$

Mini example

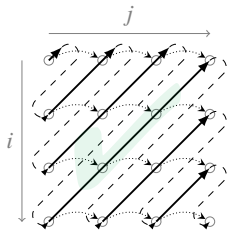
```
for(int i = 0; i < N; i++)
  for(int j = 1; j < M; j++)
    A[i, j] += A[i, j-1]; // S[i
```



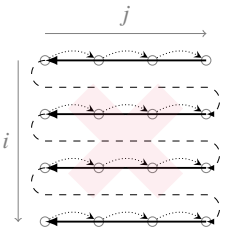
(a) $\theta(S[i,j]) = (i,j)$



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

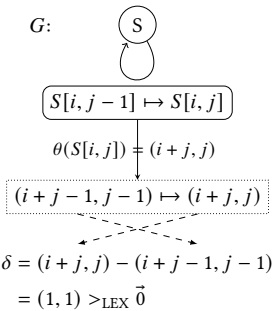


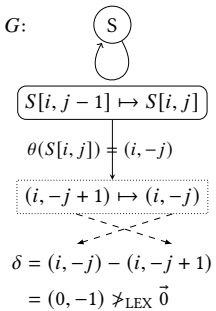
(c)
$\theta(S[i,j]) = (i+j,j)$



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

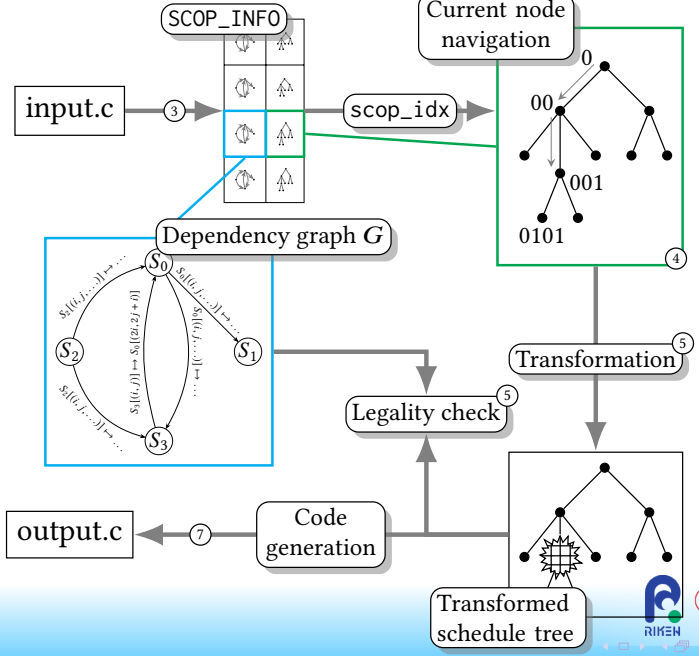
Legality check





End-to-end example

```
import tadashi
app = tadahis.Simple("input.c")
scops = tadahi.Scops(app)
node = scops[0].schedule_tree[1]
tr, args = tadashi.TrEnum.TILE, 16
legal = node.transform(tr, args)
if legal:
    scops.generate_code(output_path="output.c")
    app.compile()
    t = app.measure()
    printf(f"Walltime: {t}")
```


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.