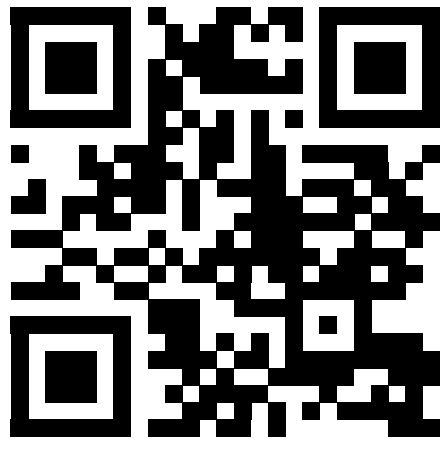


Towards Training Transformers as a Universal Computer

Ruize Xu¹ · Chenxiao Yang² · Yanhong Li³ · David McAllester²

¹ University of Chicago ² TTIC ³ Allen Institute for AI



MicroPy project Personal website

What is MicroPy?

A small, Turing-complete language. Our transformer trains on **randomly generated programs** and generalizes to **human-written programs**.

59.5M parameters · 4,096-token context

What is PENCIL?

A fixed context-management rule: **pop completed computation**; keep its returned effects + value.

After a model return:
prefix [call] work => result [ret]
→ prefix result

Data generation

PCFG **Pretraining-like**
Learn basic MicroPy syntax.

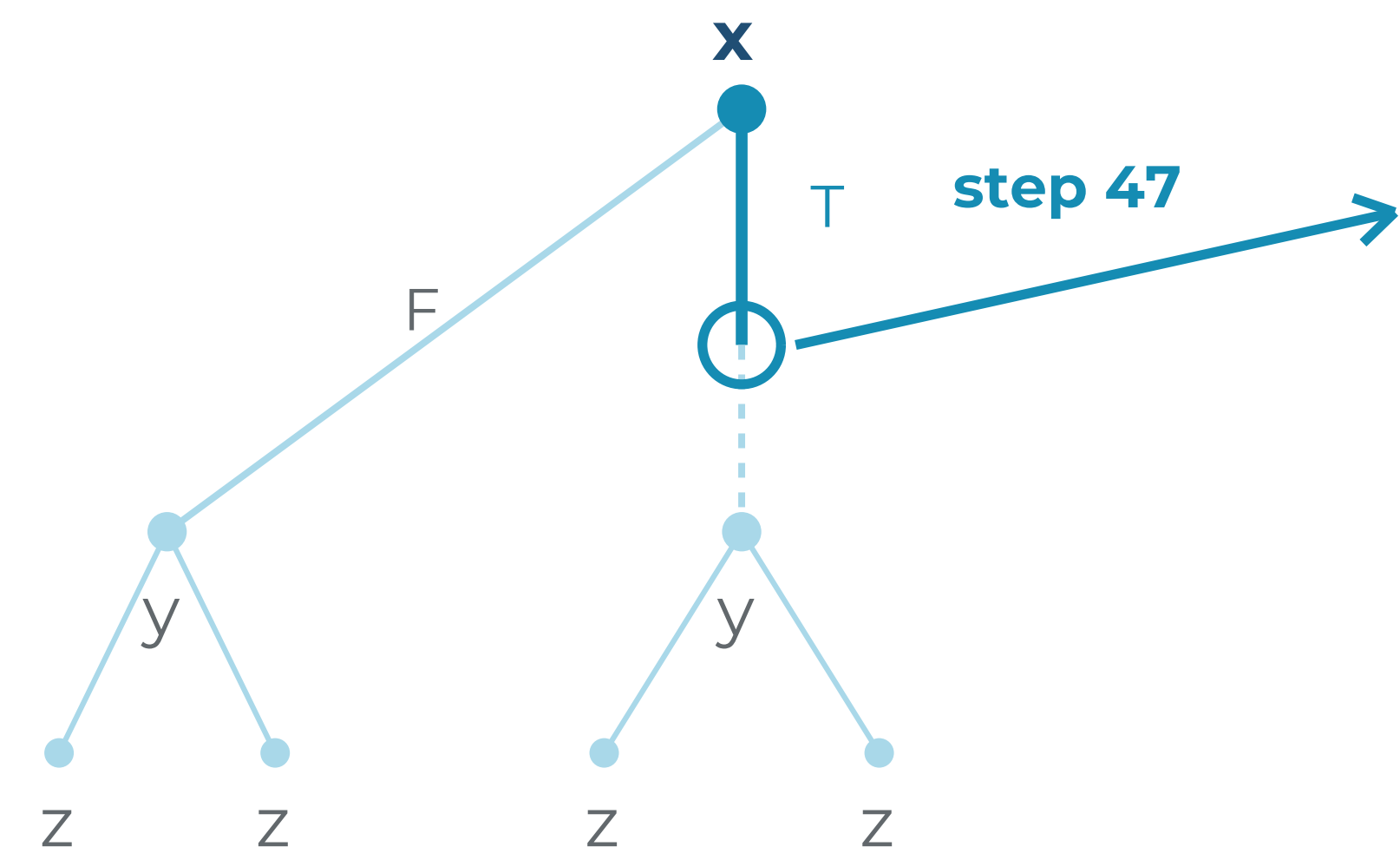
Plan compiler **Post-training-like**
Cover long retrieval and stack reads/writes. Learn simple patterns to generalize to hard programs.
Sample a random intermediate state (stack + bindings + effects) as a plan. Construct matching low-level input/output pairs for the model to learn.

MicroPy code example + model input / output

2-SAT example: choose True/False for x, y, z to satisfy all four clauses.

$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg z \vee \neg x) \wedge (x \vee y)$

Try True first; backtrack on failure.



Current branch: x = True.
Next, try a value for y.

```
sat_assign(  
  LookupAttr(var, Attr("next")),  
  cnf_head_, out_)
```

This step: find the next variable object
var points to x (O.var1).
x.next points to y (O.var2).
Return the y object; y is still unassigned.

Unroll execution; zoom in on step 47
... → 46 → 47 → 48 → ...

MODEL INPUT: DEFINITIONS + OBJECT FACTS + STACK

```
FD sat_assign = lambda(...) Exp47  
D.Exp21 = LookupVar(var)  
Assertion(O.var1 Att.next O.var2)  
Bind(var O.var1)  
... arg0 = [call]  
[call] LookupAttr(Exp21 Att.next)  
obj = Eff(empty) O.var1
```

Function body starts at Exp47.
Exp21 means: read variable var.
Fact: x.next points to y.
Binding: var refers to x.
Caller waits for its first argument.
Look up attribute next
of object O.var1.

[EOP]

MODEL OUTPUT: ONE ATOMIC EXECUTION STEP

=> Eff(empty) O.var2 [ret]

Model learns to **retrieve the value** from facts + stack to finish LookupAttr.

BEFORE PENCIL

```
... arg0 = [call] [call] LookupAttr(Exp21 Att.next)  
=> Eff(empty) O.var2 [ret]  
After completion, PENCIL removes this LookupAttr frame; keeps the result.
```

AFTER PENCIL

```
... arg0 = [call]  
Eff(empty) O.var2
```

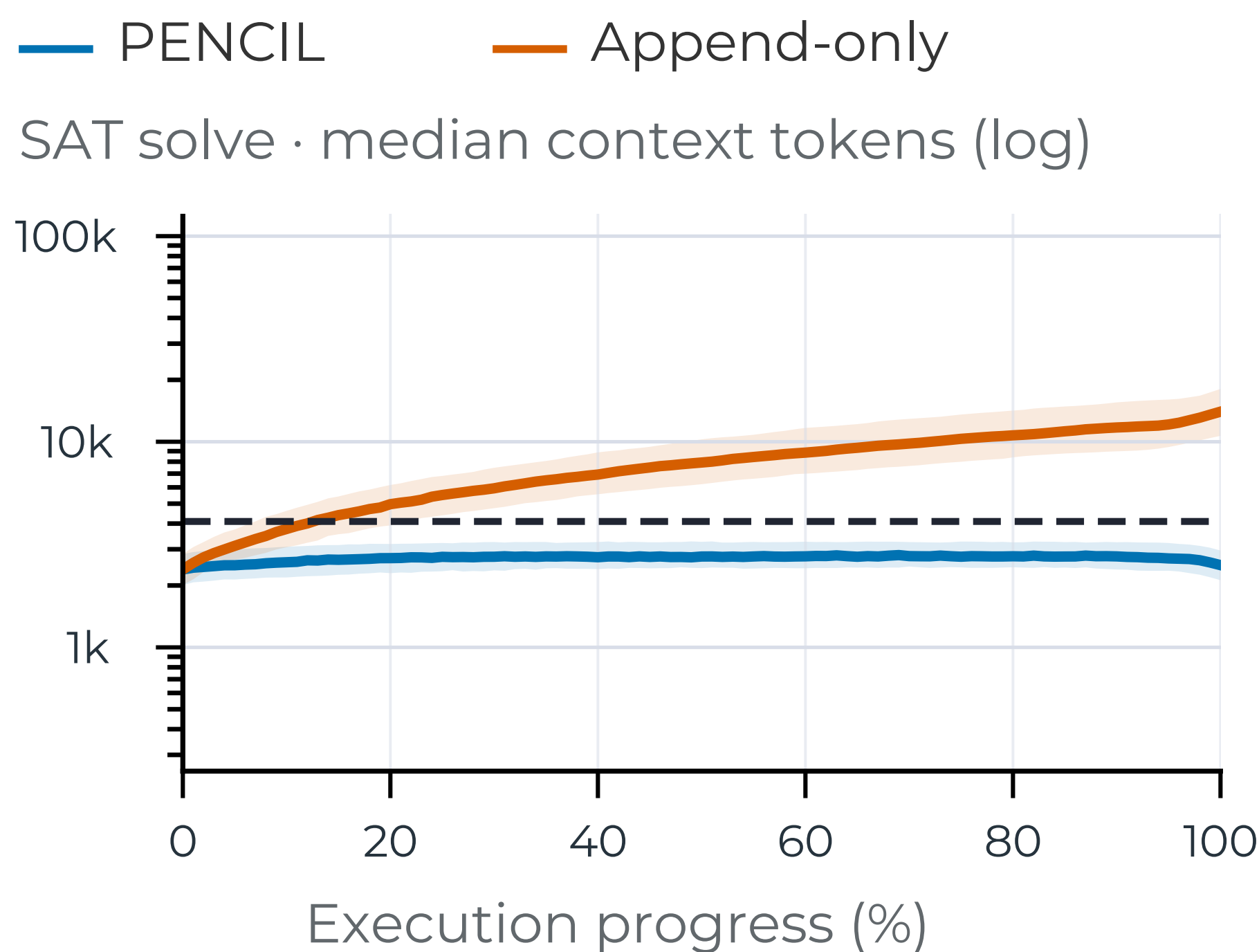
Repeat simple atomic steps → complete the whole program.

Evidence: context, coverage and generalization

100% token accuracy · 252 held-out programs · 6 task families

PENCIL saves context

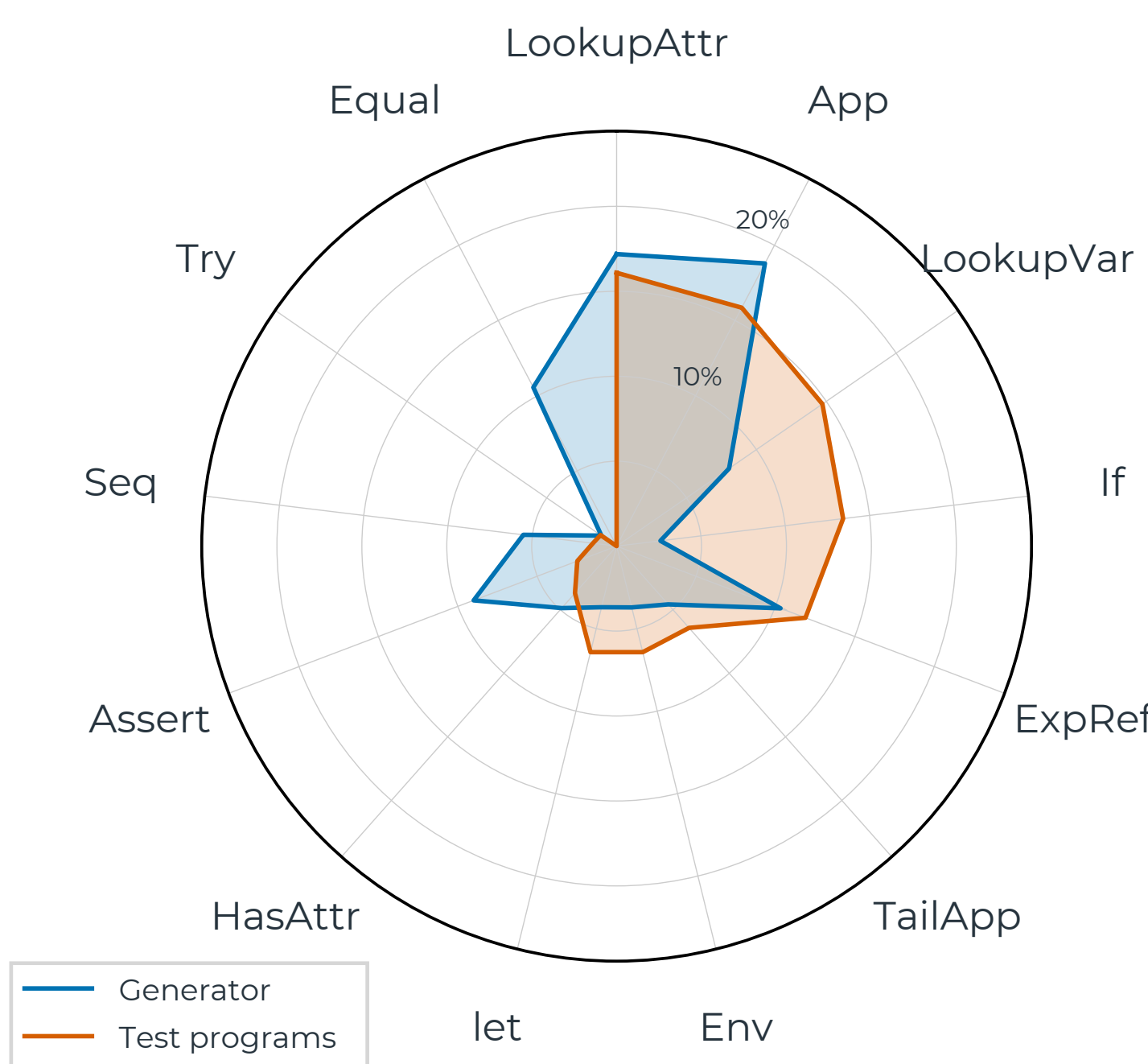
Remove finished calls vs. keep all steps



PENCIL stays within 4,096 tokens; append-only exceeds the budget. Shown: median SAT context. Same traces; only kept text differs.

Same operations, new mix

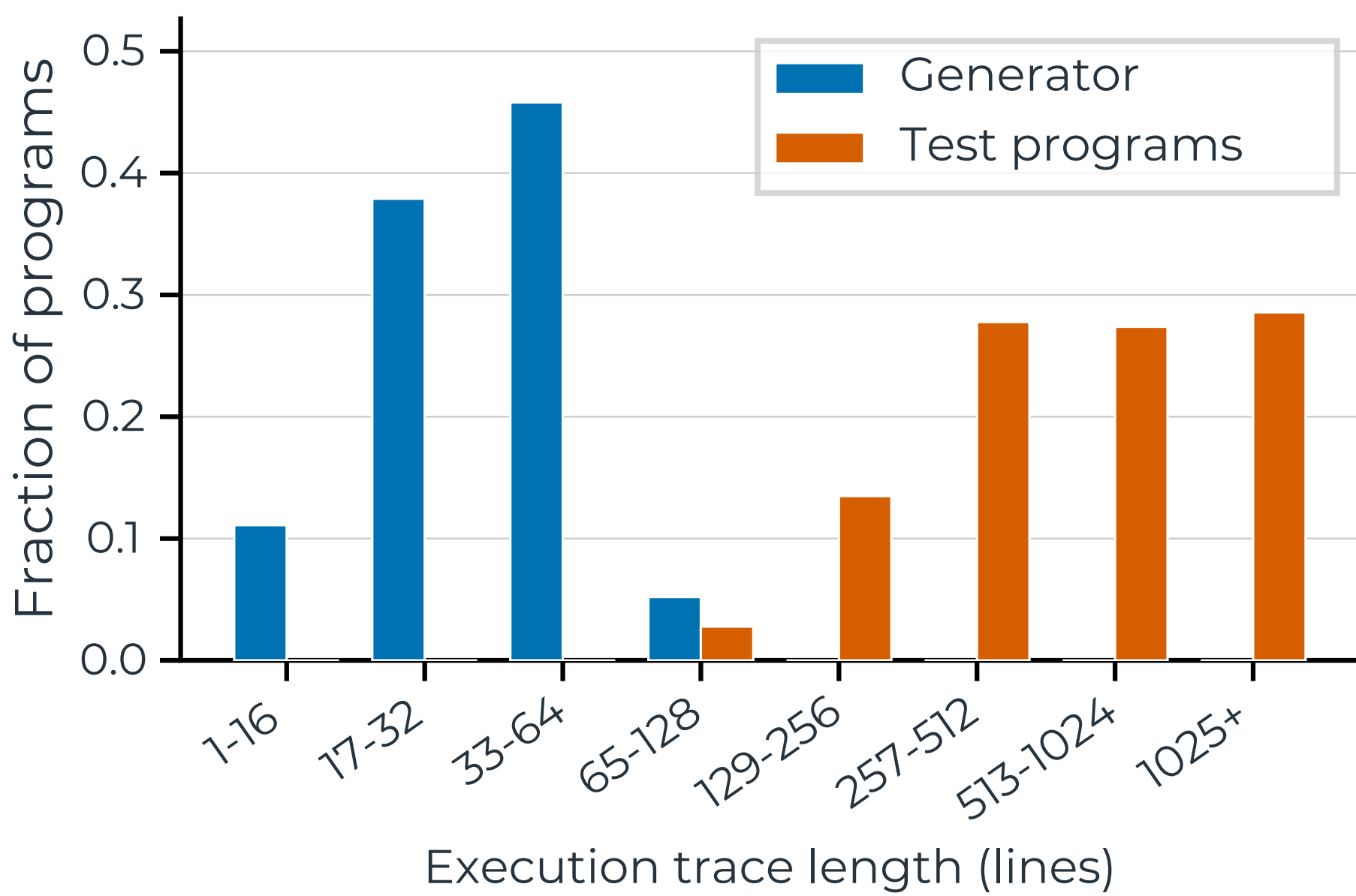
Operation frequency, train vs. test



Training covers every test operation. Tests use more If, LookupVar, ExpRef and TailApp steps. Coverage with a distribution shift.

Execute 59× longer traces

Random training; human-written tests



Random train: ≤ **128 trace lines**. Human tests: **7,552 lines (59×)**. Length = execution-trace lines, not source code or context tokens.