

Towards Training Transformers as a Universal Computer

Ruize Xu · Chenxiao Yang · Yanhong Li · David McAllester

University of Chicago · TTIC · Allen Institute for AI

59.5M

parameters

4,096

tokens of context



A learned interpreter for a small, computationally universal language.

The learning question

Can next-token training learn reusable execution rules?

Input: definitions + object facts + bindings + stack

Output: a serialized retrieval or rewrite

Evaluation: held-out, human-written programs

The model predicts one local execution step at a time.

MicroPy execution

RETRIEVAL

- Read definitions
- Look up variable bindings
- Read object attributes

REWRITING

- Choose a conditional branch
- Enter a procedure call
- Combine object updates

One set of operational rules applies across programs

Calls, recursion and mutable objects use a small set of execution rules.

SAT solver execution

Interpreter-generated trace · condensed playback

Try True first. All four clauses must hold.

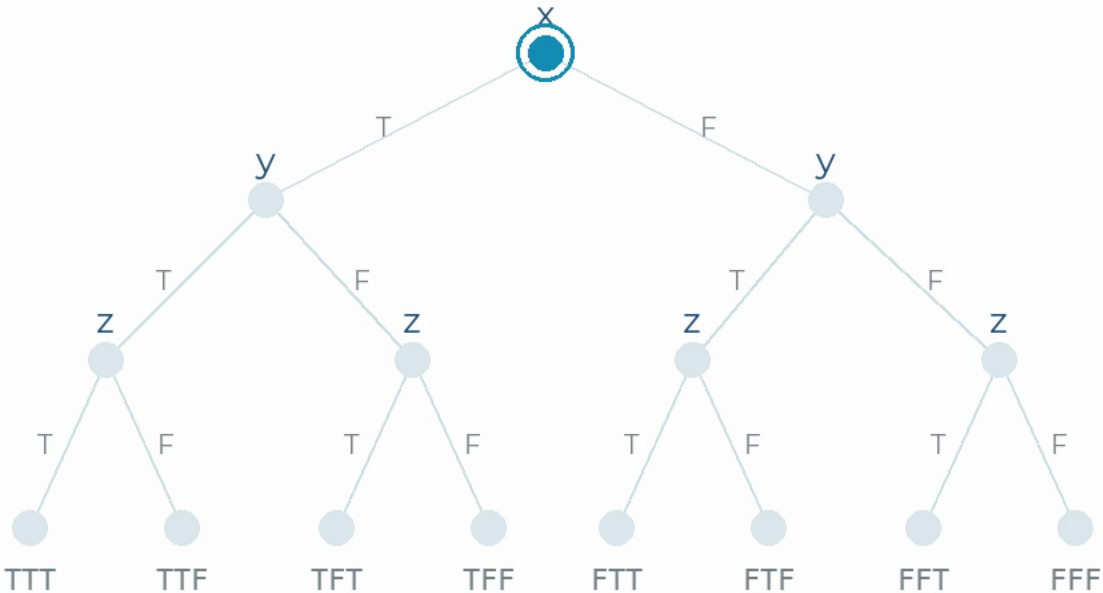
C1	$(\neg x \vee y)$	pending
C2	$(\neg y \vee z)$	pending
C3	$(\neg z \vee \neg x)$	pending
C4	$(x \vee y)$	pending

Current assignment

x = -

y = -

z = -



MicroPy trace 1/1,531 steps

Live context: 1,066 tokens

=> [call] App (sat_solve (0 . var1 0 . cl1 0 . out)) [ret]

Find an assignment that makes every clause true.

SAT solver execution

Interpreter-generated trace · condensed playback

Try True first. All four clauses must hold.

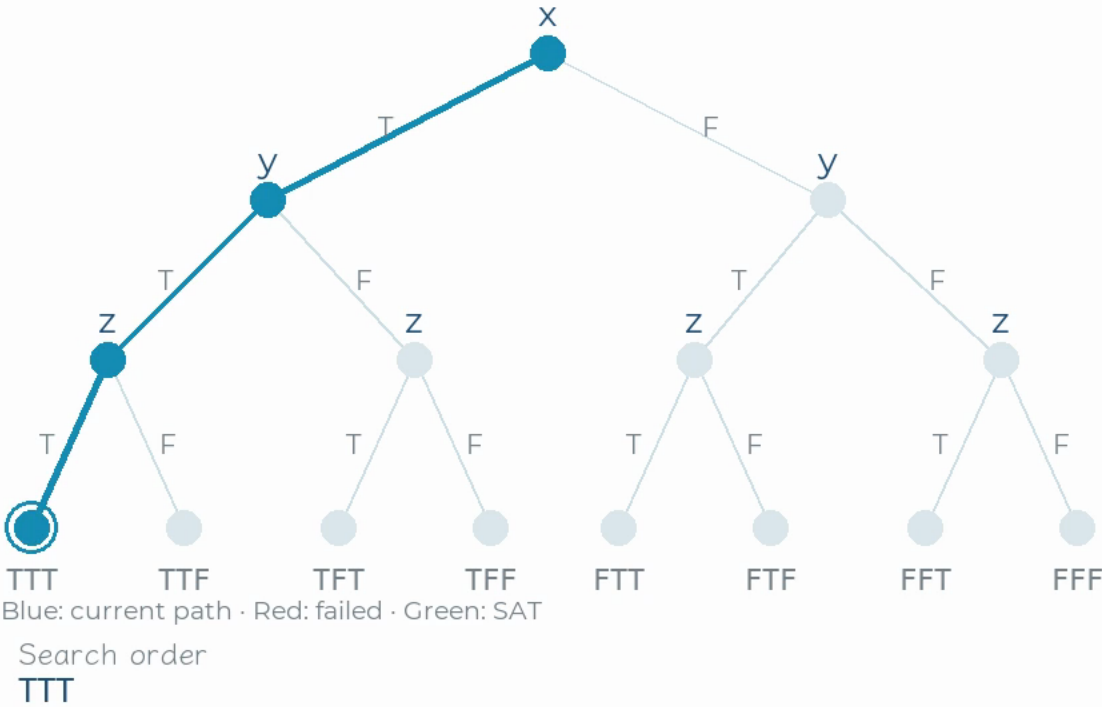
C1	$(\neg x \vee y)$	true
C2	$(\neg y \vee z)$	pending
C3	$(\neg z \vee \neg x)$	pending
C4	$(x \vee y)$	pending

Current assignment

x = T

y = T

z = T



MicroPy trace 270 / 1,531 steps
=> Eff (empty) False [ret]

Live context: 1,312 tokens

Clause 2: not y is false.

SAT solver execution

Interpreter-generated trace · condensed playback

Try True first. All four clauses must hold.

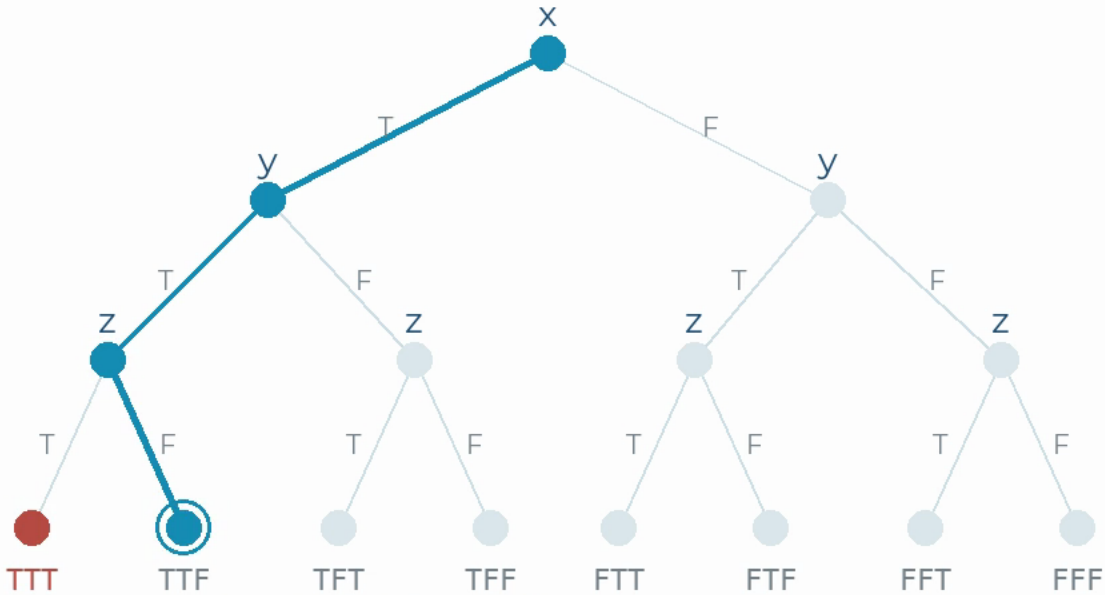
C1	$(\neg x \vee y)$	true
C2	$(\neg y \vee z)$	pending
C3	$(\neg z \vee \neg x)$	pending
C4	$(x \vee y)$	pending

Current assignment

$x = T$

$y = T$

$z = F$



Search order
TTT → TTF

MicroPy trace 572 / 1,531 steps
=> Eff (empty) True [ret]

Live context: 1,304 tokens

Clause 1: y is true. Clause satisfied.

SAT solver execution

Interpreter-generated trace · condensed playback

Try True first. All four clauses must hold.

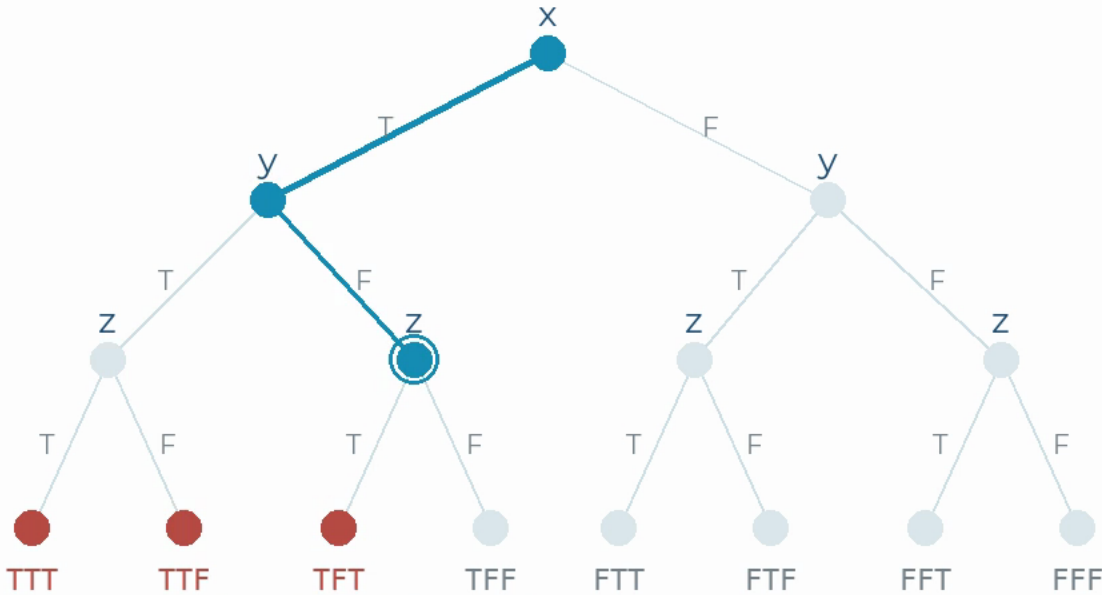
C1	$(\neg x \vee y)$	pending
C2	$(\neg y \vee z)$	pending
C3	$(\neg z \vee \neg x)$	pending
C4	$(x \vee y)$	pending

Current assignment

$x = T$

$y = F$

$z = -$



Blue: current path · Red: failed · Green: SAT

Search order
TTT → TTF → TFT

MicroPy trace 876 / 1,531 steps

Live context: 1,253 tokens

=> second = [ret]

Backtrack: restore the state before choosing z.

SAT solver execution

Interpreter-generated trace · condensed playback

Try True first. All four clauses must hold.

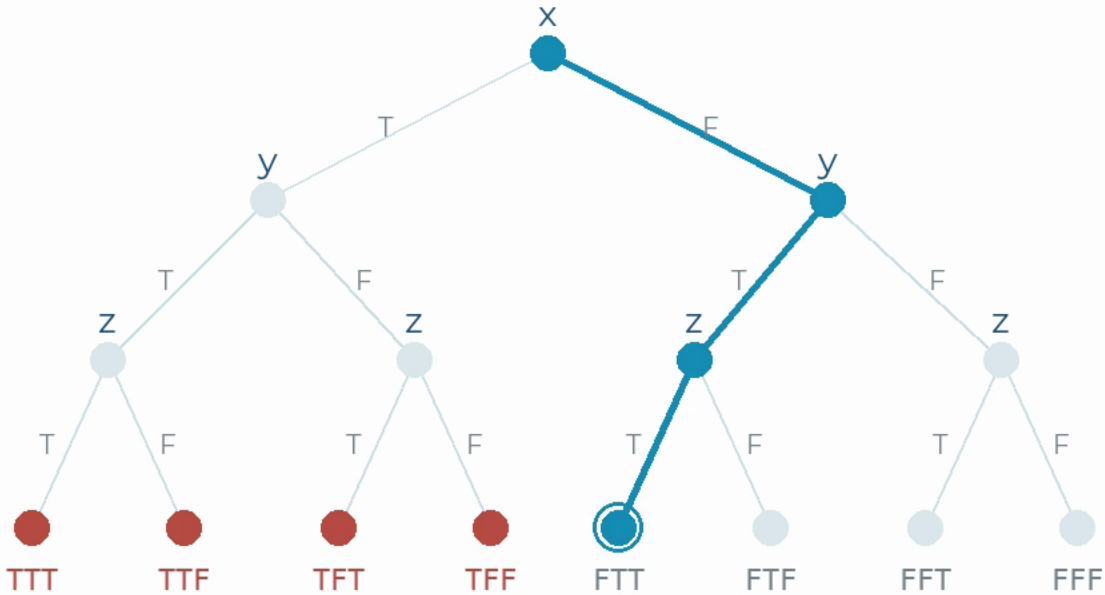
C1	$(\neg x \vee y)$	true
C2	$(\neg y \vee z)$	pending
C3	$(\neg z \vee \neg x)$	pending
C4	$(x \vee y)$	pending

Current assignment

$x = F$

$y = T$

$z = T$



MicroPy trace 1,159 / 1,531 steps

Live context: 1,349 tokens

=> Eff (empty) 0 . cl1 [ret]

Clause 1: not x is true. Clause satisfied.

TTT $\rightarrow$ TTF $\rightarrow$ TFT $\rightarrow$ TFF $\rightarrow$ FTT

MicroPy code + model input / output

RECURSIVE CALL

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

x=True; y and z are unassigned.
O.var1 represents x.
O.var2 represents y.

MODEL INPUT

Functions	FD sat_assign = lambda(...) Exp47
Expressions	D.Exp21 = LookupVar(var)
Object facts	Assertion(0.var1 Att.next 0.var2)
Bindings	Bind(var 0.var1)
Caller	... arg0 = [call]
Top frame	[call] LookupAttr(Exp21 Att.next) obj = Eff(empty) 0.var1

MODEL OUTPUT

=> Eff(empty) 0.var2 [ret]

Trace step 47: one attribute lookup. The full SAT execution continues after this step.

PENCIL context management

A fixed rule pops completed computation and keeps its returned effects and value.

COMPLETED FRAME

```
prefix [call] LookupAttr(...) ... => Eff(empty) 0.var2 [ret]
```

NEXT MODEL INPUT AFTER PENCIL

```
prefix Eff(empty) 0.var2
```

Repeat model prediction and PENCIL reduction until the program ends.

Training data generation

PCFG SAMPLER

Pretraining-like

Sample general MicroPy programs.
Run them and collect trace steps.

Learn basic syntax and
common execution patterns.

PLAN COMPILER

Post-training-like

A plan specifies a target execution state:
stack shape + bindings + effects.

Compile a program that reaches it.
Train on the matched input/output
to cover hard long-tail cases.

Pretraining-like and post-training-like describe sampler roles. Training mixes both sources.

Plan compiler: reaching a target stack

TARGET EXECUTION STATE

```
Bind(var 0.var1)
```

```
Assertion(0.var1 Att.next 0.var2)
```

```
... arg0 = [call]
```

```
[call] LookupAttr(Exp21 Att.next)
```

```
obj = Eff(empty) 0.var1
```

PROGRAM CONSTRUCTION

Bind var to O.var1.

Provide O.var1.next = O.var2.

```
sat_assign(
```

```
    LookupAttr(var, Attr("next")),
```

```
    cnf_head_, out_)
```

Evaluating var leaves the caller waiting and reaches the target LookupAttr frame.

The paper's sampled plan

PLAN (DSL)

```
01  Seq[noop]
02  EnvPush[y0=VAR(x0), y1=0.env2]
03  EffPush[2]
04  IfThen[else=noop]
05  TrySecond[prefix=has_next]
06  Seq[tailapp(y0)]
```

BINDING REQUIREMENT

x0
↓
0.env0
↑
y0

y0 must refer to the same object as x0; the final operation calls through y0.

Compile into positions that retain frames

COMPILED EXPRESSION FRAGMENTS

Seq(Exp36, Exp32)

first child

If(Exp38, False, False)

condition

Try(Exp34, Exp37)

second attempt

FRAME VISIBLE DURING EVALUATION

→ Seq ... x =

→ If ... test =

→ Try ... second =

Blue identifies the child being evaluated and the corresponding pending frame.

The resulting stack matches the plan

TARGET FRAME SEQUENCE

01	Eff	07	Eff
02	Env	08	If
03	Seq	09	Try
04	Eff	10	Seq
05	Env	11	Eff
06	Eff	12	TailApp

MATCHED TRACE (EXCERPT)

```
... outer frames and bindings ...
Env(... Bind(y0 0.env0)
      Bind(y1 0.env2))
Eff(Assertion(0.env0 ... True))
Eff(Assertion(0.env2 ... False))
If(Exp38 False False) test =
Try(Exp34 Exp37) second =
Seq(Exp36, Exp32) x =
Eff(empty)
TailApp(_tailapp_id Args(0.env0))
```

Env contains active bindings; Eff contains pending state updates. Unrelated context is collapsed.

One tail call, one matching completion

ARGUMENT FROM THE ACTIVE ENVIRONMENT

$y_0 \rightarrow 0.\text{env}_0$

DEEPEST CALL

`TailApp(_tailapp_id Args(0.env0))`

MATCHING COMPLETION

`[call] let(z) be(0.env0) in Exp10 [ret]`

The same object is passed as an argument, then bound to the callee's parameter.

Keep the matched training example

ONE EXECUTION TRACE

... earlier execution steps ...

prompt: ... TailApp(_tailapp_id Args(0.env0))

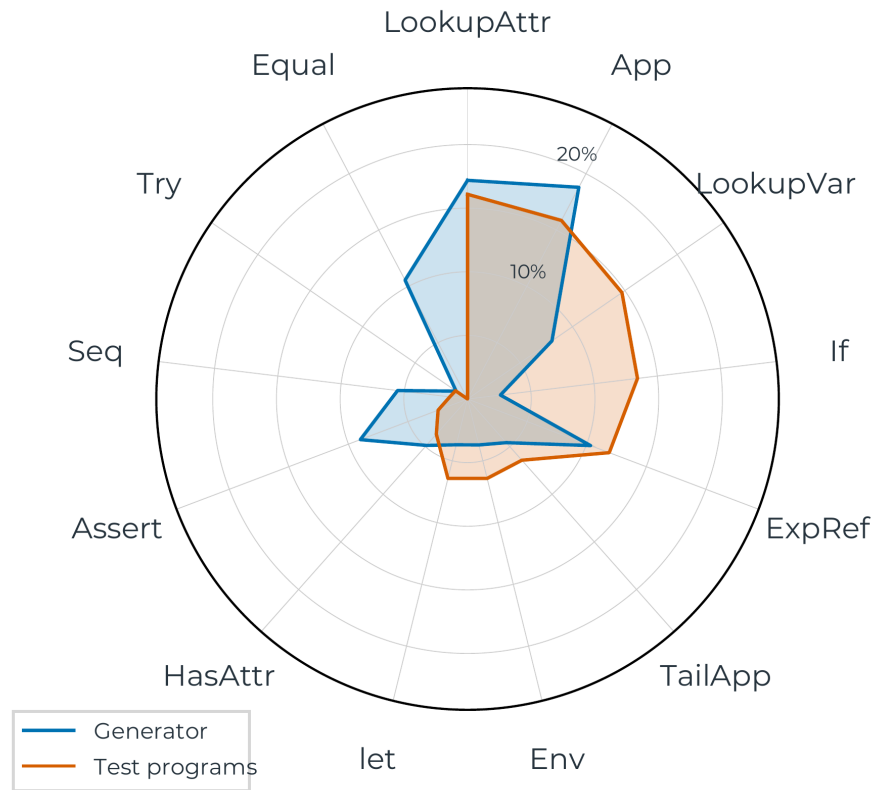
completion: [call] let(z) be(0.env0) in Exp10 [ret]

... later execution steps ...

Randomize outer frames, bindings, and distractor effects around the target.

The matching input/output pair is the training example. Other steps from that run are discarded.

Data coverage and hard cases



Operation proportions within each corpus

LONG EFFECT WRITES

Under-covered: wrong object reference
Token 108: obj182 instead of obj47

Covered: correct obj47

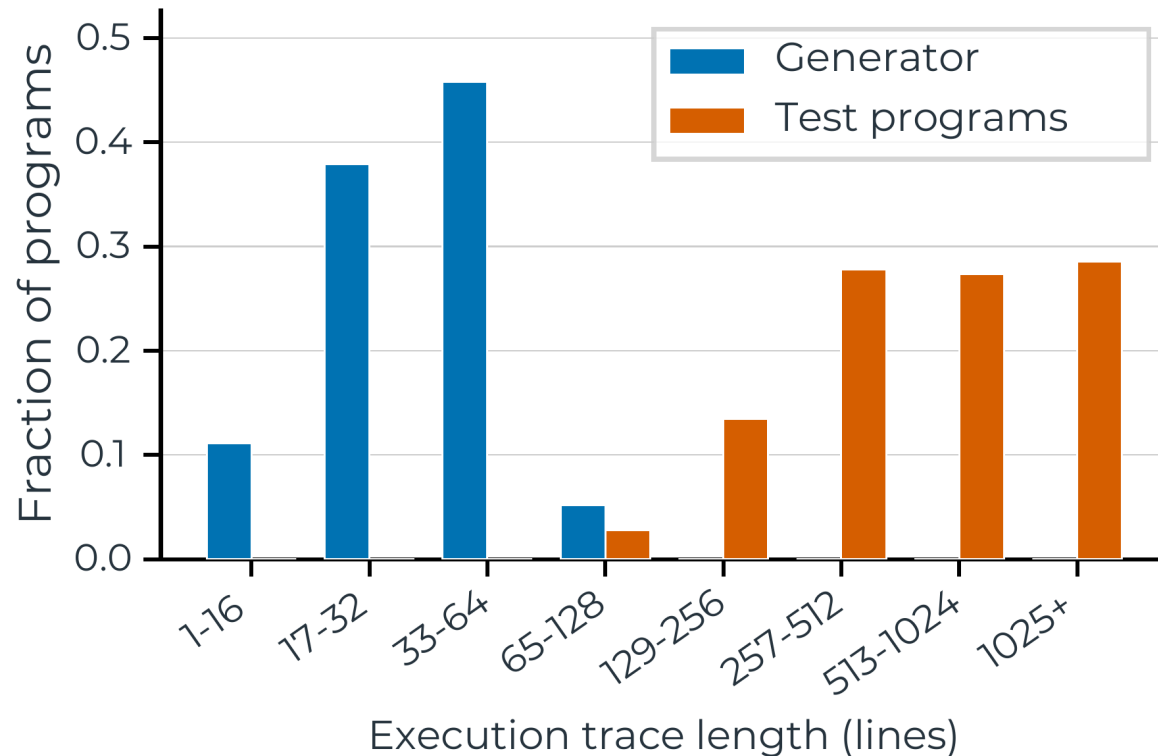
ZERO-ARGUMENT CALLS

Under-covered: 7 token errors
in 3,544 trace rows

Covered: no call-boundary errors

Coverage fixes the shown errors. The cases do not isolate the compiler in a controlled ablation.

Short training traces, longer test traces



128

training trace cap

7,552

longest test trace (59x)

Execution length is measured in trace lines, not source-code lines or context tokens.

Results on the held-out suite

100% token accuracy

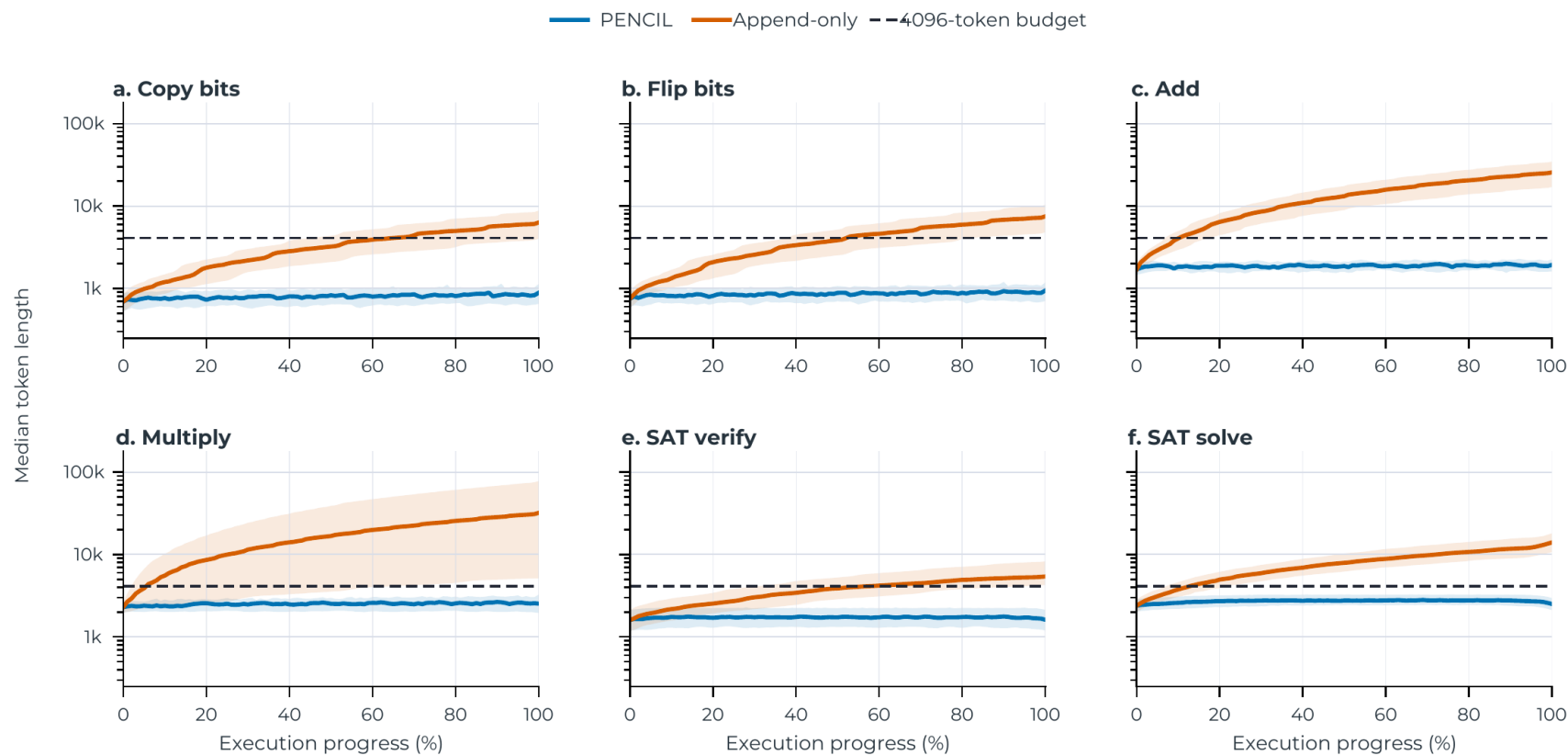
252 held-out programs across six task families

Bit copying and flipping

Binary addition and multiplication

SAT verification and solving

PENCIL saves context



Same programs and completions. Only retention of previous text changes.

Learned execution rules

Local execution rules compose
across unseen MicroPy programs

Targeted training covers difficult runtime states.
PENCIL reclaims completed computation.

