Cross-Dialect Generalization Without Retraining

Benchmarks and evaluation of schema-derived constrained decoding for MLIR

Plawan Kumar Rath · arXiv 2607.18254

MLIR runs modern ML, but LMs rarely see it

EVERYWHERE

The compiler layer of ML

TensorFlow, JAX, PyTorch Inductor and IREE all lower through MLIR.

SCARCE

Trace amounts in pretraining

Models invent ops, misuse types and fail the verifier.

GROWING

New dialects every year

One fine-tuned model per dialect does not scale.

“Can each dialect’s own schema replace fine-tuning?”

Every dialect ships an Operation Definition Specification (ODS): its ops, operands, types and rules. We turn it into decoding constraints at inference time.

No public benchmark paired English with MLIR

NL-TO-CODE BENCHMARK	TARGET
HumanEval · MBPP	Python
APPS · CodeContests	Python, C++
MultiPL-E	General-purpose languages
This work	**MLIR, three dialects**

No source text

MLIR in the wild is compiler test code, with no English description.

A strict verifier

Every reference must pass mlir-opt or iree-compile.

Text metrics mislead

Equal programs can differ in names; one wrong dtype breaks a program.

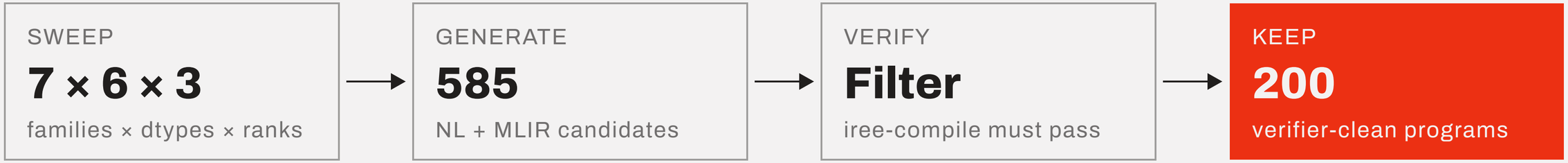
We release the first NL→MLIR benchmarks

BENCHMARK	DIALECT	PAIRS	SOURCE
MLIR-Spec-150	arith + func + memref	150	Hand-written
Linalg-Spec-30	linalg	30	Hand-written
StableHLO-Spec-30	stablehlo	30	Hand-written
StableHLO-Held-Out-200	stablehlo	200	Parametric sweep

Plus a 25-program stress set and a 30-program functional set · Apache-2.0

Every pair is written or generated, then verified

PARAMETRIC SWEEP · STABLEHLO-HELD-OUT-200



HAND-WRITTEN · MLIR-SPEC-150, LINALG-SPEC-30, STABLEHLO-SPEC-30



Hand-written sets reflect real use; the sweep checks for author-curation bias. All references are verifier-clean at release.

Three constraint layers, all derived from ODS

SOURCE

ODS schema

276 op records, read with llvm-tblgen.

C1

Op grammar

Only real ops, in valid syntax.

```
linalg.matmul
```

C2

Type splits

Integer ops take integers; float ops take floats.

```
arith.addi : i32
```

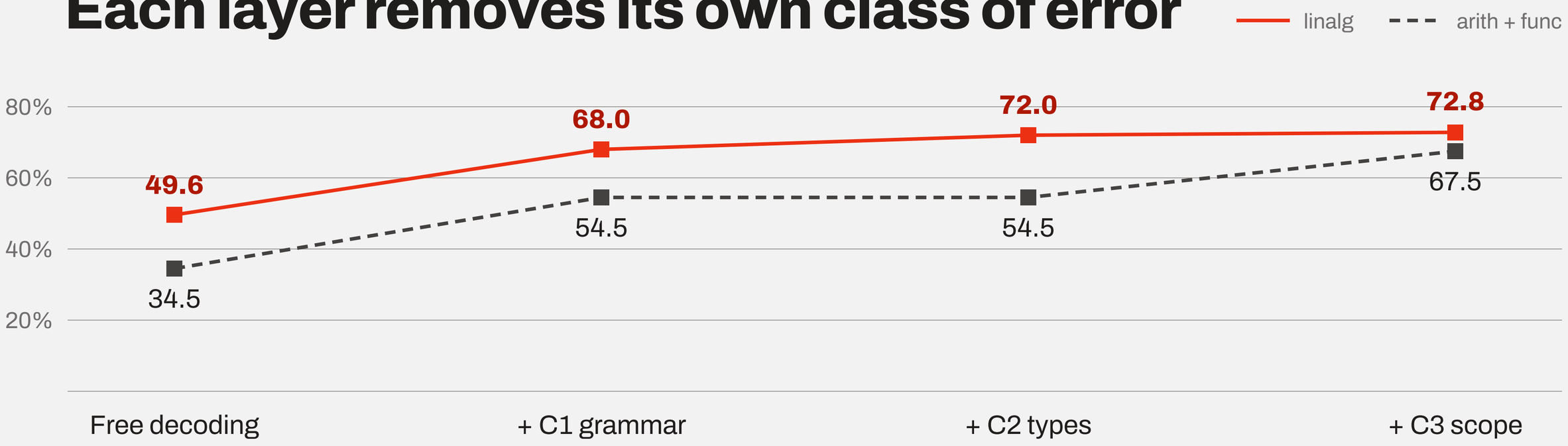
C3

SSA scope check

Every value defined before use. Reject and retry, up to 5 times.

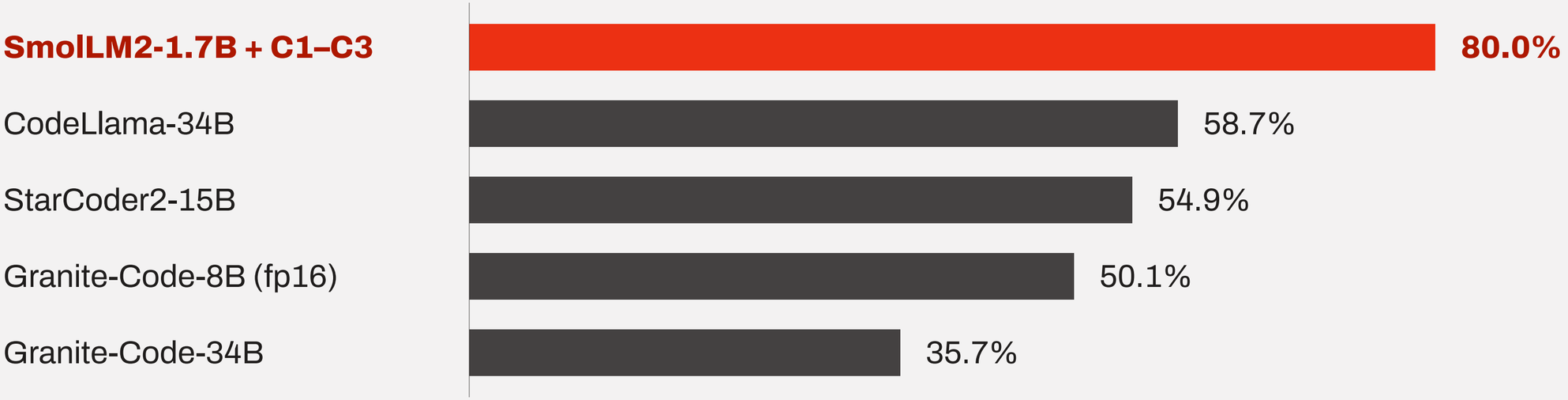
C1 and C2 mask tokens while decoding · C3 filters whole programs · No training of any kind

Each layer removes its own class of error



SmolLM2-1.7B verify-valid rate, seed-0 ladder (n=200 arith + func, n=125 linalg). C2 drives linalg (+4.0 pp); C3 drives arith + func (+13.0 pp).

On linalg, a 1.7B model beats 15–34B code LMs



Verify-valid on linalg, three-seed mean (n=125 per seed), same five-retry budget for all. Every SmolLM2 seed scores 80.0%; CIs do not overlap.

8–25×

faster per generation than 15–34B baselines, on one Apple M4 Max laptop

1.7–1.9 s vs 16–40 s

Seconds per generation, same five-retry budget.

0 lines of new constraint code

To port from core dialects to StableHLO.

The gains depend on the dialect

	BENCHMARK	SMOLLM2-1.7B	15–34B BASELINES	OUTCOME
■	Linalg-Spec	80.0%	35.7–58.7%	Robust win
■	StableHLO-Spec-30	63.3%	33.3–60.0%	Win on hand-written set
■	StableHLO-Held-Out-200	61.5%	98.0–100%	Baselines win
■	MLIR-Spec (arith + func)	53.2%	51.5–66.8%	No win

Wins where the verifier checks structure; not where it checks attribute values. Verify-valid is not functional correctness.

Summary and resources

- Four NL→MLIR benchmarks: 410 pairs across three dialects
- A three-layer constraint stack generated from ODS, with no training
- SmolLM2-1.7B reaches 80.0% on linalg, 8–25× faster than 15–34B models
- Scoped honestly: no win where attribute values decide validity

Thank you

Benchmarks, decoder, every generation and a Docker image are public.

github.com/plawanrath/slm-to-mlir-constrained-emitter

huggingface.co/datasets/plawanrath

arXiv 2607.18254 · plawan@meta.com