Three-Layer Formal Verification of Quantized ML Kernels

From Compiler IR to ISA Specification

Yi-De Wu

FMCAD 2026

National Taiwan University

The Problem: Quantized Inference Correctness

Quantized LLM inference (llama.cpp)

- Q4_0 / Q5_0 / Q8_0: 4–8 bit integer weights on RVV
- Integer datapath: nibble extract, widen-multiply, reduce

The correctness gap

- Integer bugs $\neq$ small FP errors — arbitrary corruption
- Multiple layers: MLIR $\rightarrow$ LLVM IR $\rightarrow$ asm $\rightarrow$ ISA
- Testing cannot cover 2^{384}–2^{640} inputs

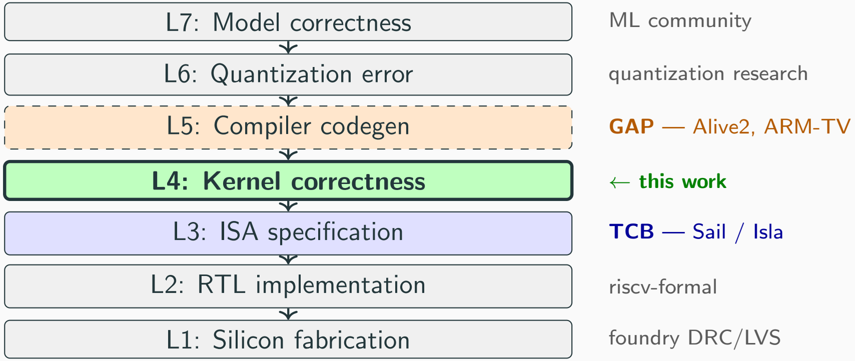
Why formal methods work here:

- Pure bitvector arithmetic
- No FP rounding, no dynamic memory
- No control-flow divergence
- $\Rightarrow$ **Decidable** (QF_BV)

Nobody has done this:

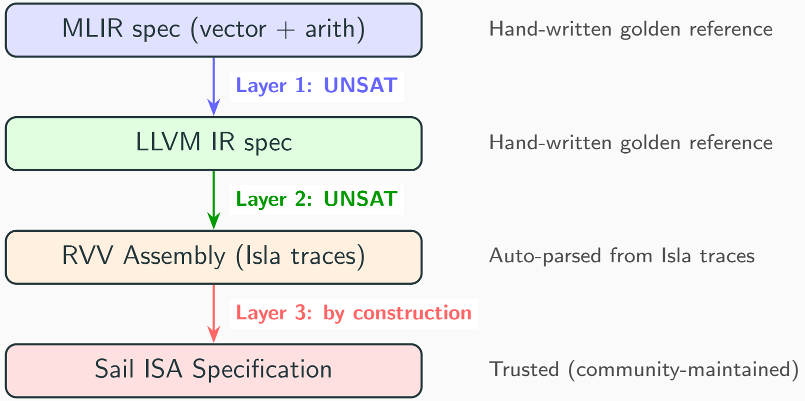
No prior work verifies ML kernels against ISA formal specifications.

Where We Sit: 7-Layer Verification Landscape



We verify **L4**: kernel assembly is faithful to the ISA formal specification.

Pipeline: Three Layers of Equivalence


MLIR spec (vector + arith)
Hand-written golden reference
Layer 1: UNSAT
LLVM IR spec
Hand-written golden reference
Layer 2: UNSAT
RVV Assembly (Isla traces)
Auto-parsed from Isla traces
Layer 3: by construction
Sail ISA Specification
Trusted (community-maintained)

Honest framing: MLIR/LLVM IR are *specifications*, not compiler output.
We verify assembly against these specs, not the compiler itself.

llama.cpp Q4_0 Kernel: 11 RVV Instructions

`ggml_vec_dot_q4_0_q8_0()` — commit 05728db

#	Instruction	SEW	Operation
1–3	`vle8.v v`N`,(a`M`)`	8	load ×3
4	`vand.vx v4,v1,a3`	8	nibble mask
5	`vsrl.vx v5,v1,a4`	8	nibble shift
6	`vsub.vx v6,v4,a5`	8	bias subtract
7	`vsub.vx v7,v5,a5`	8	bias subtract
8	`vwmul.vv v8,v6,v2`	8→16	widen multiply
9	`vwmacc.vv v8,v7,v3`	8→16	widen MAC
10	`vmv.v.x v10,x0`	16	zero init
11	`vwredsum.vs v10,v8,v10`	16→32	widen reduce

$$\texttt{sumi} = \sum_{i=0}^{15} \mathrm{sext}_{32}\Big(\mathrm{sext}_{16}((x_i \,\&\, \texttt{0xF} - 8) \cdot y_i^{\mathrm{lo}}) + \mathrm{sext}_{16}((x_i \gg 4 - 8) \cdot y_i^{\mathrm{hi}})\Big)$$

Challenge 1: Why Z3 Chokes on SIMD Formulas

Isla outputs one 256-bit formula:

$$\mathit{result} = \underbrace{\mathtt{zext}(e_0) \mid (\mathtt{zext}(e_1) \ll 8) \mid \cdots}_{\text{16 elements}}$$

Why Z3 can't solve this:

- 16 elements are *semantically independent*
- But packed via shift-or, not $\wedge$-conjunction
- Z3 bit-blasts all 256 bits — can't detect the independence structure
- >10 min timeout

Our insight: independence is visible in the AST — each e_i sits inside a `zero_extend` node.

Solution: walk AST *before* Z3, pattern-match `zero_extend` nodes, extract 16 independent 8-bit sub-formulas.

Method	**Time**
Monolithic 256-bit	>600 s
Per-element (ours)	<1 s/elem

Soundness: SIMD lanes have no cross-lane data flow. Reductions verified as single formula.

Challenge 2: Masked Instruction Path Explosion

Problem: `vsub.vx v4,v4,a5, v0.t`

- Sail branches per mask bit: 2^{VL} paths
- Isla times out — even at VL=1
- Needed for Q5_0 conditional 5th-bit

Solution: 3 independent proofs

1. **Isla:** `vmnand.mm` (unmasked) low 16 bits = ∼`v0`
2. **RVV axiom:** mask-undisturbed ITE $\text{result}[i] = \text{ite}(\text{mask}[i], \text{op}, \text{keep})$
3. **Z3 lemma:** $\text{ite}(\neg\text{qh}, n{-}16, n) = n + \text{qh} \ll 4 - 16$

General: works for any masked RVV arithmetic. One axiom from RVV spec.

Per-Instruction Verification: 81 Element Proofs

Instruction	SEW	Spec formula	Elems	Result
`vand.vx`	8	$v1[i]\ \&\ \mathrm{a3}$	16	UNSAT
`vsrl.vx`	8	$\mathtt{LShR}(v1[i], \mathrm{a4})$	16	UNSAT
`vsub.vx`	8	$v4[i] - \mathrm{a5}$	16	UNSAT
`vwmul.vv`	8→16	$\mathtt{sext}(v6[i]) \times \mathtt{sext}(v2[i])$	16	UNSAT
`vwmacc.vv`	8→16	$vd[i] + \mathtt{sext}(v7[i]) \times \mathtt{sext}(v3[i])$	16	UNSAT
`vwredsum.vs`	16→32	$v10[0] + \sum \mathtt{sext}(v8[i])$	1	UNSAT
`vadd.vv`	8	$v2[i] + v1[i]$	16	UNSAT
Total element proofs			**97**	**all unsat**

Auto-parsed from Isla S-expression traces. **Zero manual formula translation.**

Four Verified Kernels

	Q4_0×Q8_0	**Q5_0×Q8_0**	**Q8_0×Q8_0**	**vecadd**
Pattern	dot product	dot product	dot product	**elem-wise**
Instructions	11	7 traces + ITE	4	1
Input space	2^{384}	2^{640}	2^{256}	2^{256}
Output	scalar (i32)	scalar (i32)	scalar (i32)	**vector (16×i8)**
Widening	i8→i16→i32	i8→i16→i32	i8→i16→i32	none
Masking	no	**yes** (5th bit)	no	no
New traces	6	reused + vmnand	reused	**vadd.vv**
All layers	UNSAT	UNSAT	UNSAT	UNSAT

- Q5_0 and Q8_0 **reuse** Q4_0 traces — zero new trace generation
- vecadd: different pattern (no reduction, vector output), proves pipeline generality

Three-Layer Results: All UNSAT

Kernel	Layer 1 MLIR == LLVM	Layer 2 LLVM == Isla	Layer 3 Isla == Sail	Total
Q4_0×Q8_0	UNSAT	UNSAT	by constr.	20.7 s
Q5_0×Q8_0	UNSAT	UNSAT*	by constr.	<1 s
Q8_0×Q8_0	UNSAT	UNSAT	by constr.	<1 s
vecadd	UNSAT	UNSAT	by constr.	<1 s

* Q5_0 Layer 2: 7 Isla traces + 1 RVV-spec axiom (mask-undisturbed ITE)

Metric	Value
Total verification time	<25 s
Lines of Python	2,446
Isla traces parsed	7 unique types
Manual formula translation	**zero**

Portability: Add a Kernel or ISA in 4 Steps

Add a new kernel:

1. Write MLIR spec (∼20 lines)
2. Write LLVM IR spec (∼30 lines)
3. Run `isla-footprint` (automated)
4. Run verification (automated)

Port to a new ISA (e.g., ARM):

1. Replace Sail model (ARM exists)
2. Regenerate Isla traces
3. Adjust LLVM IR spec
4. **MLIR spec unchanged** (ISA-agnostic)

Component reuse:

Component	Kernel	ISA
Parser + decomposer	reuse	reuse
Composer	reuse	reuse
IR parsers	reuse	reuse
MLIR spec	new	reuse
LLVM IR spec	new	new
Isla traces	may reuse	new

Cross-ISA guarantee:

Same MLIR spec ⇒
RISC-V and ARM backends
are transitively equivalent.

VLEN Portability: Proved

Claim: per-element formulas are identical regardless of VLEN/VL.

Proof: generated traces at VL=16 and VL=8 (simulating VLEN=128), proved element-by-element equivalence in Z3.

Instruction	**VL=16 elems**	**VL=8 elems**	**Result**
`vand.vx v4,v1,a3`	16 proven	8/8 match VL=16	UNSAT
`vsrl.vx v5,v1,a4`	16 proven	8/8 match VL=16	UNSAT
`vsub.vx v6,v4,a5`	16 proven	8/8 match VL=16	UNSAT
Total cross-VL proofs			**24/24 UNSAT**

Why: Sail computes each element independently. Changing VL changes *how many* elements, not *what* each computes. ⇒ Proofs at VLEN=256 transfer to 128, 512, 1024, . . .

Honest Scope: What We Do and Don't Verify

L4 Concern	Status	Remaining
Integer datapath	• Verified (Q4_0, Q5_0, Q8_0, vecadd)	GEMM, conv, softmax
Floating point	—	Hard (FP theory in Z3)
Memory	◦ Axiomatized	Address, alignment
Control flow	1 iteration	Loops, vsetvli
Concurrency	—	Very hard (memory model)

TCB (what we trust):

- Sail RISC-V model (30K lines)
- Isla symbolic executor (18K lines Rust)
- Z3 solver (standard assumption)
- Our parsers (1,268 lines Python)

Not verified:

- C $\rightarrow$ asm compiler (L5 gap)
- FP scale multiplication
- Memory addresses / stores
- `vsetvli` configuration
- Q5_0 mask ITE (1 axiom)

Related Work: No Prior Work Spans All Five Dimensions

Work	ISA spec	Compiler IR	Vector	Auto	ML
Islaris (PLDI'22)	• Sail	—	—	○	—
Alive2 (PLDI'21)	—	• LLVM	—	•	—
ARM-TV (OOPSLA'25)	○ hand	• LLVM	—	•	—
VeriISLE (ASPLOS'24)	• Sail	—	—	•	—
MLIR-TV (CAV'22)	—	• MLIR	○	•	—
Verif. Dialects (PLDI'25)	—	• MLIR	—	•	—
Dubey et al. (arXiv'25)	—	—	• GPU	•	• FP
This work	• Sail	• MLIR+LLVM	• RVV	•	• int

Key differentiators:

- First to use Isla traces for *kernel* verification (vs. systems code)
- First to solve SIMD scalability (per-element + mask decomposition)
- ARM-TV uses hand-written lifter; we use authoritative Sail model

Who Benefits: Follow-Up Opportunities

Community	What they reuse	Follow-up enabled
ISA designers	Traces + decomposer	Validate new vector instructions against Sail spec
Compiler devs	MLIR/LLVM specs as validation oracles	Translation validation for LLVM RVV codegen (closes L5 gap)
ML framework devs	Pipeline (automated)	Write MLIR spec ($\sim$20 lines), verify kernel — no FM expertise
HW verification	Per-element decomposition + mask decomposition	Applies to *any* packed-SIMD (ARM SVE, x86 AVX, . . .)

Key: each community reuses a different layer of our pipeline.

The three-layer architecture is not just a proof strategy — it is a *separation of concerns* that enables independent follow-up.

Conclusion

What we built:

- Three-layer pipeline: MLIR $\leftrightarrow$ LLVM IR $\leftrightarrow$ Isla (Sail)
- 4 kernels verified: Q4_0, Q5_0, Q8_0, vecadd — all UNSAT
- 2,446 LOC Python, $<25\,\mathrm{s}$, fully automated

Three scalability techniques (the engineering novelty):

1. Per-element AST decomposition: $>10\,\mathrm{min} \to <1\,\mathrm{s}$
2. Mask decomposition: 2^{VL} paths $\to$ 3 lemmas
3. Linear composition: $O(n)$ via Z3 `substitute()`

Future: L5 translation validation (LLVM RVV codegen) | Cross-ISA (ARM via same MLIR spec) | VLEN-parametric proofs

Artifact: `https://github.com/yiidtw/rvs1`