Binary Circuit Models

Compiling a Small Binary Language Model into a Logic Circuit

Jovonni L. Pharr
Georgia Cyber Warfare Range
info@gacwr.org

August 2026

Abstract

A language model is ordinarily *run*: its weights are loaded into a general-purpose processor, and a fetch–decode–execute loop grinds through billions of floating-point multiply–accumulates to produce each token. This work observes that a model of a particular kind need not be run at all—it can be *built*. A small *binary* model, trained over a finite alphabet with fixed-precision integer arithmetic, is a total function from a finite set of input bits to a finite set of output bits; and a total function on bits is, by the founding theorem of digital design, a Boolean function, hence a network of logic gates, hence a circuit. We make the reduction explicit and, more importantly, concrete: we compile the arithmetic that constitutes such a model—the multiply–accumulate, the threshold nonlinearity, the arg-max token selection—into gate-level netlists of AND, OR, NOT, XOR, and XNOR, count their gates exactly, technology-map them with YOSYS and ABC, co-simulate the Verilog they emit, and prove sixteen of them equivalent to independently written specifications by SAT over their *entire* input spaces—2^{128} vectors for the 64-wide inner product, 2^{208} for an attention head—rather than the few hundred random draws such blocks are usually checked on. Under binarization the inner product collapses to the celebrated XNOR-and-population-count, which we synthesize at nine to eleven gates per input bit (rising with width); the entire next-token *head* of a real binary model of embedding width 512 over a 258-symbol alphabet compiles to 1.47 million verified gates, which YOSYS and ABC technology-map to 0.41 million six-input lookup tables. We then dispatch the objection that attention's soft-max is continuous, by compiling hard, arg-max attention into an exact bit-exact comparator-multiplexer netlist and showing fixed-point soft-max is a finite Boolean function; we train a model *natively* binary and compile every one of the 512 neurons of one of its feed-forward layers, with the trained weights baked in, and prove every one of them equivalent to the layer it came from over its entire 2^{128} input space; and we discharge part of the compiler's specification in the Lean 4 proof assistant—six theorems with no `sorry`. We then carry the netlists into silicon: an open physical-design flow places, routes, times and power-analyses every block in a characterised 130 nm standard-cell library, where the binarized inner product occupies 40.5× less area and draws 32.4× less power than the same arithmetic in int8, and a pipelined compiled neuron of a trained model closes at a measured clock period and emits one result per cycle. We price the one term that is not yet integer—the normalizer—and replace it with a multiplier-free shift-normalizer that removes every general multiplier from an autoregressive step and shrinks it by 40%. Every performance, area, latency and energy statement carries an explicit evidence label—MEASURED, MODELLED or ASSERTED—so the reader always knows which tool stands behind which number. The thesis: the arithmetic of a binary model is a circuit, and we have built it, mapped it, placed it, timed it and in large part *proved* it, block by block.

Contents

A note on evidence

A hardware claim is worth exactly as much as the tool that produced it. Every quantitative statement in this paper therefore carries one of three labels, and the label names the tool that stands behind the number:

measured produced by running a tool and recording what it reported: the gate compiler (an exact count of a netlist it built), YOSYS 0.68 with ABC (technology mapping, cell counts, longest topological path), YOSYS's SAT engine and ABC's combinational equivalence checker (proofs of equivalence over an entire input space), Icarus Verilog 14 (simulation of the emitted Verilog), OpenROAD [26] with the SKY130 open process design kit [27] (floorplanning, placement, routing, static timing and power), or a training log. Every such number is written to a machine-readable receipt by the run that produced it, and a checker asserts that the prose still equals the receipt.

modelled composed analytically from MEASURED primitives under assumptions that are stated at the point of use—for example, a whole-model gate budget summed from per-block counts, or a fixed-point LayerNorm priced at a chosen word width.

asserted taken from an external source and not verified here—for example, a vendor's lookup-table budget for a named device.

Part III carries every block through a physical-design flow, so MEASURED also covers standard-cell area in μm^2, critical-path delay in ns, $F_{\max}$ in MHz and power in μW, always in the `tt_025C_1v80` corner of `sky130_fd_sc_hd`. No number in this paper is a measurement on fabricated silicon: nothing has been fabricated, and no device has been programmed or powered. 130 nm is two decades behind a modern accelerator, so the absolute figures are a consistent yardstick, and the claims we make from them are *ratios* measured on one flow in one library.

Part I
Preliminaries

1 The State of Today

Inference upon a neural network is, today, an act of *interpretation*. The trained weights are a passive data structure; a processor—a central processing unit, a graphics processor, a tensor accelerator—reads them, reads the input, and executes a program of arithmetic that computes the output. The computation is real, but it is mediated: between the model and its result stands a von Neumann machine fetching operands from memory and dispatching them through arithmetic units, one instruction at a time. This is why inference costs what it costs, in energy and in latency: not because the function the model computes is intrinsically expensive, but because a general-purpose interpreter is a heavy way to evaluate a fixed function.

For most models this mediation is unavoidable, because the function is defined over the real numbers and approximated in floating point, a representation whose evaluation is genuinely the province of a floating-point unit. But there is a class of models for which it is not unavoidable at all. A *binary* model—one whose alphabet is finite, whose activations and weights are of fixed and low precision, and whose every operation is an exact integer or logical operation—defines a function from a finite set of input bits to a finite set of output bits. Such a function does not need to be interpreted. It can be *implemented*, directly, as the thing that digital hardware is made of: a circuit of logic gates.

2 Related Work

The bridge this work walks across was built in 1937. In his master's thesis, Shannon [1] established that the behaviour of any network of switches is described exactly by an expression in two-valued Boolean algebra, and conversely that any Boolean expression can be realized by such a network. Digital design has rested on this equivalence ever since: it is why a first course in computer architecture converts circuits to algebra and algebra to circuits, then composes the results into adders, arithmetic-logic units, multiplexers, and registers. Our contribution sits at the confluence of four literatures that carry this equivalence into neural computation.

2.1 Binary and low-bit quantized neural networks

The premise that a multiply–accumulate can be replaced by bitwise logic originates in the binary-network line. BinaryConnect [2] trained networks with binary weights; Binarized Neural Networks [3] extended binarization to activations, so the dominant inner product becomes an XNOR followed by a population count; and XNOR-Net [4] scaled this to ImageNet with per-channel scaling, making the XNOR-popcount kernel that Eq. (3) synthesizes the canonical primitive of the field. Beyond one bit, DoReFa-Net [5] generalized to arbitrary low bitwidths and Ternary Weight Networks [6] added a zero state; training through the non-differentiable quantizer rests on the straight-through estimator [7]. Most relevant to our autoregressive target, BitNet [8] trains 1-bit Transformers from scratch, and BitNet b1.58 [9] shows a ternary-weight LLM matching an FP16 baseline and—in the authors' framing—"opens the door for designing specific hardware" for such models. We take that door literally: where these papers quantize to *run* cheaply on existing processors, we compile the resulting Boolean function into gates.

2.2 Neural networks as learned logic

A second thread treats the network itself as combinational logic. LogicNets [10] co-designs sparse low-bit neurons so each maps to one FPGA lookup table; LUTNet [11] makes the K-input LUT the trainable unit; PolyLUT [12] raises each LUT's expressivity with piecewise polynomials; and NullaNet [13] trains layers to be Boolean functions and then applies *logic minimization* to emit an optimized netlist. The Tsetlin machine [14] learns propositional clauses outright. Most directly related to our framing are the differentiable logic gate networks of Petersen et al. [15], which relax the choice of each two-input gate into a differentiable distribution so an entire AND/XOR/NAND network is learned by gradient descent and discretized to a netlist; the convolutional extension [16] scales this by more than an order of magnitude. These works establish that logic-gate networks are trainable and hardware-efficient. The distinction we draw is one of *direction and object*: they *learn* a circuit that approximates a task (typically an image classifier); we *compile* an already-trained, finite-alphabet language-model head into a circuit that is bit-exactly the same function, and account for its gate count.

2.3 FPGA/ASIC accelerators and quantization toolflows

The mapping from quantized networks to silicon is mature. FINN [17] and FINN-R [18] compile binarized and quantized networks into streaming dataflow architectures on FPGAs, fed by the Brevitas [20] quantization-aware training front end; hls4ml [19] compiles small networks to high-level-synthesis firmware for sub-microsecond inference. These toolflows are our closest engineering neighbors and we borrow their vocabulary; they differ in their standard of correctness: they target a throughput/accuracy operating point under quantization, whereas we hold ourselves to *bit-exact equivalence* between the emitted netlist and an integer specification, block by block.

2.4 Logic synthesis and Boolean-function representation

That we can speak of a "netlist for a function" at all rests on the switching-circuit equivalence [1], whose Shannon expansion (Eq. (1)) is the constructive bridge from expression to circuit. The apparatus of logic synthesis supplies the representations a compiler such as ours would use downstream: reduced ordered binary decision diagrams for canonical representation and equivalence [21]; two-level minimization via ESPRESSO [22]; And-Inverter Graphs as the workhorse intermediate representation [23]; and the ABC system [24], which combines AIG rewriting, SAT, and simulation for industrial-strength synthesis and equivalence checking. We currently verify by exhaustive and random simulation against the integer spec; these tools are the path to *formally* certifying the compiled model and to shrinking it below our current gate counts, as NullaNet's logic-minimization stage already illustrates. The balanced reduction our population count synthesizes is the classical carry-save adder tree of Wallace [25].

2.5 Finite-precision networks as Boolean circuits

Finally, the identification of a fixed-precision network with a Boolean circuit has a theoretical lineage. Parberry [29] studies neural networks through the lens of circuit complexity, situating threshold units within bounded-depth circuit classes. On the verification side, Narodytska et al. [30] give the first *exact* Boolean encoding of a binarized network as a SAT/ILP instance—precisely the "a binary model is a total Boolean function" identity we build on, but turned toward *reasoning about* a fixed-input classifier rather than *emitting synthesizable hardware* for an autoregressive language model. Our reduction is the same identity turned toward fabrication; what remains genuinely open is set out next.

3 Current Problems

If the equivalence is so old and so well used, what remains to do? Two things, and they are what a *binary model* in the sense of the companion work makes newly pointed. First, the models compiled to logic in the prior literature are, almost without exception, *classifiers*: a fixed-input, fixed-output map, one forward pass. A language model is *autoregressive*—it consumes its own output, step after step—so its circuit is not merely combinational but sequential, a datapath with state, and the discipline of compiling it must account for the feedback of the emitted symbol into the next input. Second, and more fundamentally, the objects usually compiled are models of continuous data forced down to low precision, so the compilation is a lossy approximation whose fidelity must be argued. A model whose alphabet is *already* finite and discrete—bytes, opcodes, a vocabulary of a few hundred symbols—has no continuous ground truth to lose; its input and output are digital to begin with, and the compilation to a circuit can be *exact*. The question this work asks is therefore sharper than the general one: not "how well can a network be approximated by logic," but "a binary model already is a Boolean function; what is the circuit, and how large is it?"

4 The Discipline of Circuit Compilation

We name the practice *circuit compilation of neural models*: the exact translation of a fixed-precision, finite-alphabet model into a verified gate-level netlist that computes the identical function, together with the accounting of that netlist's size. Its standard of correctness is not a benchmark score but a *bit-exact equivalence*, block by block, between the arithmetic the model specifies and the gates the compiler emits. We meet that standard block by block: sixteen are *proved* equivalent to an independent specification over their entire input spaces (Table 10), the proofs are pushed wider still in Section 16, and every block whose evidence is a sample

rather than a proof is labelled a sample. The remainder develops the foundations, exhibits the compiled blocks, maps them with a synthesis tool, places them in silicon, and budgets the circuit for a whole model.

Part II
Proposal

5 Boolean Foundations

We recall precisely the algebra on which everything rests, over the two-element set $\mathbb{B} = \{0, 1\}$.

Definition 1 (Logic gates)**.** *The elementary operations are conjunction $a \wedge b$ (AND), disjunction $a \vee b$ (OR), negation $\neg a$ (NOT), exclusive-or $a \oplus b$ (XOR), and its complement $a \overline{\oplus} b = \neg(a \oplus b)$ (XNOR), with the truth tables of Table 1.*

Table 1: The elementary gates.

a	b	$a \wedge b$	$a \vee b$	$\neg a$	$a \oplus b$	$a \overline{\oplus} b$
0	0	0	0	1	0	1
0	1	0	1	1	1	0
1	0	0	1	0	1	0
1	1	1	1	0	0	1

These operations satisfy the axioms of a Boolean algebra—commutativity, associativity, the mutual distributivity of $\wedge$ and $\vee$, the identities $a \vee 0 = a$ and $a \wedge 1 = a$, the complements $a \vee \neg a = 1$ and $a \wedge \neg a = 0$—and De Morgan's laws $\neg(a \wedge b) = \neg a \vee \neg b$ and $\neg(a \vee b) = \neg a \wedge \neg b$. Two consequences are load-bearing for us. The first is *functional completeness*: every function $f : \mathbb{B}^n \to \mathbb{B}$ can be written using only $\{\wedge, \vee, \neg\}$—indeed using only NAND—so any Boolean function whatsoever is realizable in gates. The second is *Shannon expansion*,

$$f(x_1, \ldots, x_n) \;=\; \big(x_1 \wedge f|_{x_1=1}\big) \;\vee\; \big(\neg x_1 \wedge f|_{x_1=0}\big), \tag{1}$$

which recursively decomposes any function into a multiplexer over its sub-functions and is the constructive engine behind the algebra-to-circuit equivalence: an expression is read off as a netlist, and a netlist is read off as an expression.

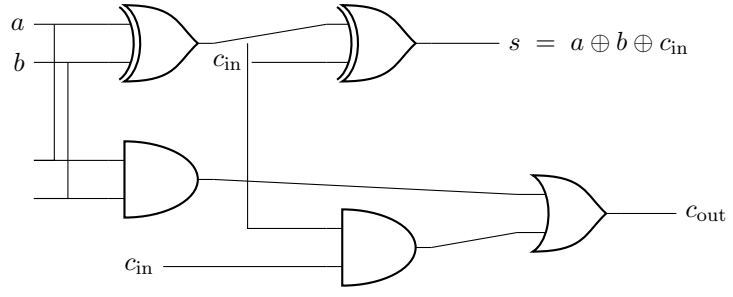


Figure 1: The full adder, the atom of arithmetic: sum $s = a \oplus b \oplus c_{\text{in}}$ and carry $c_{\text{out}} = (a \wedge b) \vee (c_{\text{in}} \wedge (a \oplus b))$. Two XOR, two AND, one OR; verified exhaustively on all eight inputs. A ripple of n full adders is an n-bit adder (40 gates at $n = 8$).

The full adder of Figure 1 is the atom from which arithmetic is built: chaining n of them by their carries yields an n-bit adder, adder trees yield multipliers and population counts, and a subtractor's high bit yields a comparator. Everything below is a composition of this atom.

6 The Reduction: a Model is a Boolean Function

Consider a single layer of a fixed-precision network. Its output is, coordinate by coordinate, an integer multiply–accumulate followed by a nonlinearity,

$$y_j \;=\; \phi\Big(\textstyle\sum_{i=1}^{n} w_{ji}\, x_i\Big), \tag{2}$$

with w_{ji} and x_i integers of bounded width. Each bit of each product $w_{ji}x_i$ is a Boolean function of the bits of w_{ji} and x_i; the accumulation is a tree of adders; the nonlinearity ϕ—a threshold or a sign—is a comparison, whose result is a single bit read from the top of a subtraction. So y_j, bit by bit, is a Boolean function of the input bits, and the whole layer is a Boolean function $\mathbb{B}^N \to \mathbb{B}^M$. A stack of layers composes these functions; the composition is again a Boolean function; it is a circuit.

Binarization makes the circuit small. Restrict weights and activations to $\{-1, +1\}$ and encode them in one bit, $1 \leftrightarrow +1$ and $0 \leftrightarrow -1$. Then $x_i w_i = +1$ exactly when $x_i = w_i$, i.e. exactly when $x_i \overline{\oplus} w_i = 1$, and the inner product becomes a count of agreements minus disagreements:

$$\sum_{i=1}^{n} x_i w_i \;=\; \#\{i : x_i = w_i\} - \#\{i : x_i \neq w_i\} \;=\; 2\,\mathrm{popcount}\big(x \overline{\oplus} w\big) - n. \tag{3}$$

The multiply–accumulate—the operation that dominates the cost of every neural network—collapses to a bitwise XNOR followed by a population count and an affine relabelling that costs no gates. The nonlinearity, a sign, is then simply the high bit of the comparison of this count against a threshold; and the final selection of a token, an arg max over the output scores, is a tree of comparators. These three—XNOR-popcount, compare, arg max—are the whole of what a binarized model computes.

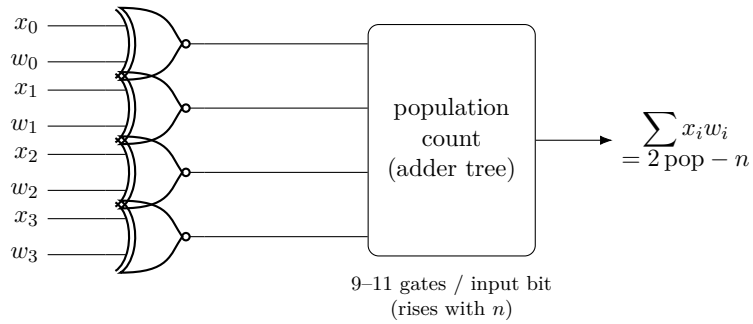


Figure 2: The binarized inner product (Eq. 3): an array of XNOR gates computes the agreement bits, a balanced adder tree of full adders counts them, and an affine relabel gives the signed dot product. The count grows at 9–11 gates per input bit over $n = 16 \ldots 512$, rising slowly with the popcount tree's depth (146 gates at $n = 16$, 5,577 at $n = 512$) [MEASURED].

7 The Compiled Blocks

We implemented a gate-level compiler that builds these blocks from primitive AND/OR/NOT/XOR/XNOR gates, evaluates the resulting netlist on concrete inputs, and checks it against the integer specification. Table 2 reports the outcome. Gate counts are exact—they are counts of a netlist the compiler built, not estimates. The *evidence* for correctness is graded, and the table says which grade each block earns: some are proved equivalent to an independent behavioural model by SAT over their whole input space, some are enumerated exhaustively, and the widest

rest on a sample of seeded random vectors plus deterministic corner cases. A 2^{1024}-input block checked on 500 draws has been sampled, not verified. The compiler also emits synthesizable Verilog for each block, and that Verilog is co-simulated against the Python netlist (Section 17).

Table 2: Compiled gate-level blocks [MEASURED]. "Gates" is an exact count of the netlist the compiler built. "Depth" is the longest combinational path in levels of two-input gates, with *no* pipeline registers. "Evidence" distinguishes a *proof* over the whole input space (YOSYS `miter` + SAT against an independently written behavioural model) from *exhaustive* enumeration, from a *sample* of seeded random vectors plus deterministic corner cases. The distinction matters: for the widest blocks the evidence in this table is 300–500 random draws from input spaces of size 2^{1024} and larger, and Section 16 reports how far a larger solver budget pushes that frontier.

Block	Function	Evidence	Gates	Depth
Full adder	$a + b + c_{\text{in}}$	exhaustive 8/8; SAT-proved	5	3
Ripple adder (n=8)	two's-complement add	SAT-proved over 2^{16}	40	17
XNOR-popcount (n=64)	agreement count	SAT-proved over 2^{128}	664	24
XNOR-popcount (n=512)	agreement count	508 vectors (sample)	5,577	36
Signed dot (n=512)	$2\,\text{popcount} - n$	508 vectors (sample)	5,637	41
Arg-max *value* (K=8)	maximum, *not* its index	SAT-proved over 2^{48}	462	126
Arg-max *index* (K=8, fold)	$\arg\max$, first-max ties	SAT-proved over 2^{48}	506	127
Arg-max *index* (K=8, tree)	$\arg\max$, first-max ties	SAT-proved over 2^{48}	490	36
Arg-max *index* (K=258, tree)	the head's token select	308 vectors (sample)	34,438	125
Hard attention (k=16, d=64)	$V[\arg\max_j Q \cdot K_j]$	308 vectors (sample)	12,906	327
exp ROM (7→12 bit)	fixed-point exp	exhaustive 128/128	684	44
Real trained neuron (n=128)	weights baked in	2,006 vectors, co-simulated	1,300	28

Value and index are different circuits, and the head needs the index. A block that returns the largest score is not the block a language model needs: to close the autoregressive loop of Figure 6 the head must emit the winning *symbol*. The two are separate netlists with separate costs, and Table 2 lists both. Emitting the index costs 2.22× the value block if the reduction is built as a linear fold (62,698 against 28,270 gates at K=258), but only 1.22× if it is built as a balanced tournament tree (34,438 gates), which is also 54× shallower. We use the tree, and every head budget in this paper is costed with it.

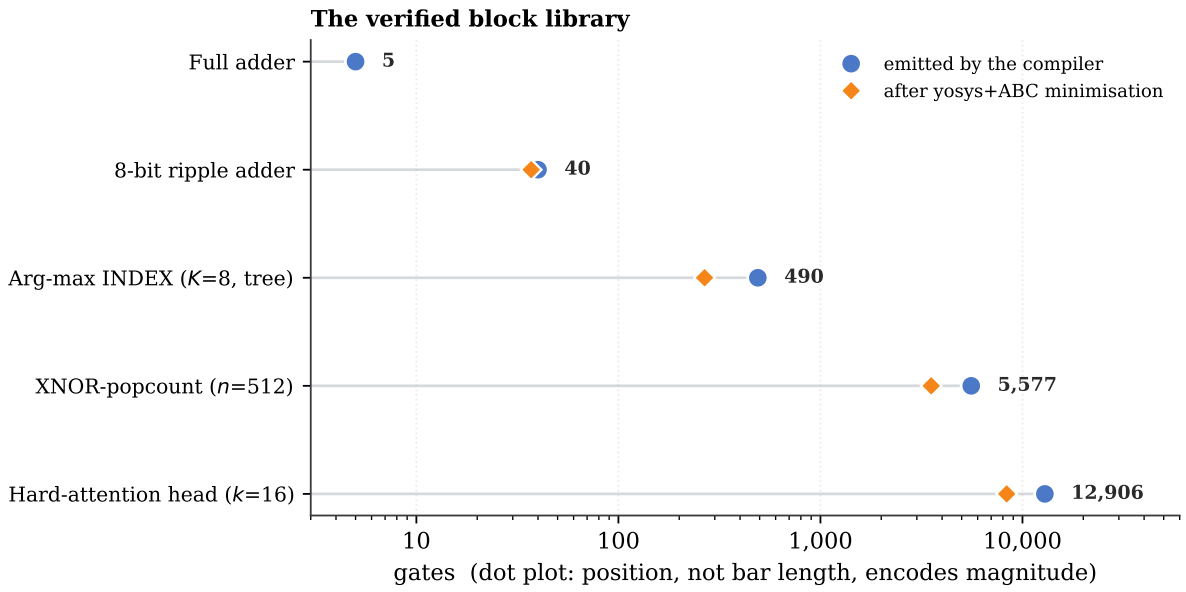


Figure 3: [MEASURED] The block library. Each marker is a self-contained gate netlist; blue is what the compiler emits, orange is what YOSYS+ABC reduce it to (Table 3). This is a *dot plot* on a log axis: position encodes magnitude, which a bar chart on a log axis cannot do, since there bar length is not proportional to value. Gate counts are exact counts of the emitted netlist; the evidence class for each block's correctness is graded in Table 2.

8 Technology Mapping: Gates, Lookup Tables and Depth

A count of two-input gates is not an area, and no claim about a chip follows from one. We therefore ran the emitted netlists through a real synthesis flow—YOSYS 0.68 driving ABC—in two independent configurations: minimisation into a strictly two-input cell library, which says how much of our gate count is slack, and mapping into six-input lookup tables, which is the unit a field-programmable gate array is built from. Both report the longest topological path, so depth is measured after mapping and not only in our own netlist. Table 3 is what the tools reported; the full tool transcripts are kept with the artifact.

Table 3: Technology mapping [MEASURED: YOSYS 0.68 + ABC]. "Emitted" is our compiler's count; "ABC" is after minimisation into a two-input library; "LUT6" is after `abc -lut 6`; "levels" is YOSYS `ltp` on the mapped netlist.

Block	Emitted	ABC	Δ	LUT6	gates/LUT6	LUT levels
8-bit ripple adder	40	37	−7.5%	13	3.08	3
XNOR-popcount n=64	664	413	−37.8%	182	3.65	7
XNOR-popcount n=512	5,577	3,534	−36.6%	1,554	3.59	11
Real neuron, baked n=128	1,300	728	−44.0%	254	5.12	8
Arg-max index K=258 (fold)	62,698	20,162	−67.8%	6,227	10.07	1,027
Arg-max index K=258 (tree)	34,438	17,888	−48.1%	5,555	6.20	31
Hard attention k=16	12,906	8,354	−35.3%	3,241	3.98	53
Hard attention k=64 (tree)	52,682	33,564	−36.3%	13,835	3.81	28
exp ROM 7→12	684	249	−63.6%	30	22.80	4

Two things follow. First, **our gate counts are upper bounds, not sizes**: ABC removes between 35% and 68% of the gates from every non-trivial block, well past the −30% we pre-

registered as the threshold at which a count must be described as a bound. The counts remain exact statements about the netlist the compiler emits, every budget in this paper is built from them, and the tool's own reduction is reported alongside. Second, **the conversion from gates to lookup tables is 3.6 to 6.2 gates per LUT6**, measured rather than assumed, and that exchange rate is what decides the device question in Section 12.

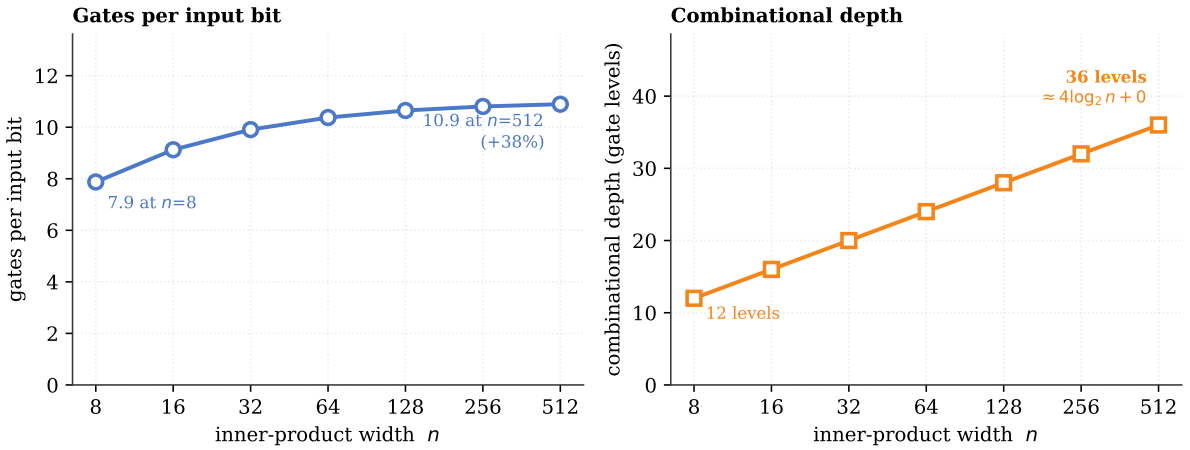


Figure 4: [MEASURED] The cost of the binarized multiply–accumulate (Eq. (3)) is *nearly* linear in the inner-product width, but not exactly: gates per input bit rise from 7.9 at n=8 to 10.9 at n=512, a 38% range over a sweep spanning a factor of 64, because the popcount adder tree deepens. *Left*: gates per input bit on a linear axis, which is the view that makes the rise visible—on log–log totals any power law is a straight line. *Right*: the block's combinational depth, exactly $4\log_2 n$ levels of two-input gates, since the popcount tree contributes one ripple-adder stage per level. The n=512 inner product is therefore 36 levels deep, and with no pipeline register placed that entire depth is one clock period; Section 14 places the registers and measures what they buy.

9 The Whole Model as a Circuit

A binary language model is a stack of such layers ending in a linear *head* that scores each of the V symbols of the alphabet, followed by an arg max that selects the next symbol. Composing the verified blocks at a real model's dimensions gives the size of the head directly. For an embedding width $D = 512$ and an alphabet of $V = 258$ symbols, the head is V binarized inner products of width D followed by a V-way arg max:

$$G_{\text{head}} \;=\; V \cdot G_{\text{dot}}(D) \;+\; G_{\arg\max\text{-index}}(V) \;=\; 258 \cdot 5{,}577 \;+\; 34{,}438 \;=\; 1.47 \times 10^{6} \text{ gates}, \quad (4)$$

[MEASURED] where the second term is the balanced-tree block that emits the winning *index*—the symbol the model must feed back—rather than the winning score. At $D = 768$ the head is 2.20 million gates. Mapped with ABC at the measured 3.59 gates per LUT6 for the dot products and 6.20 for the arg-max tree, the D=512 head is approximately 4.1×10^5 LUT6 [MODELLED from MEASURED per-block ratios], which is within the budget of a mid-to-large device such as a Kintex UltraScale+ KU15P (5.2×10^5 LUT6 [ASSERTED, vendor table]). The *head* of a binary language model is a chip. Section 12 asks the same question of the whole model, and the answer there is different.

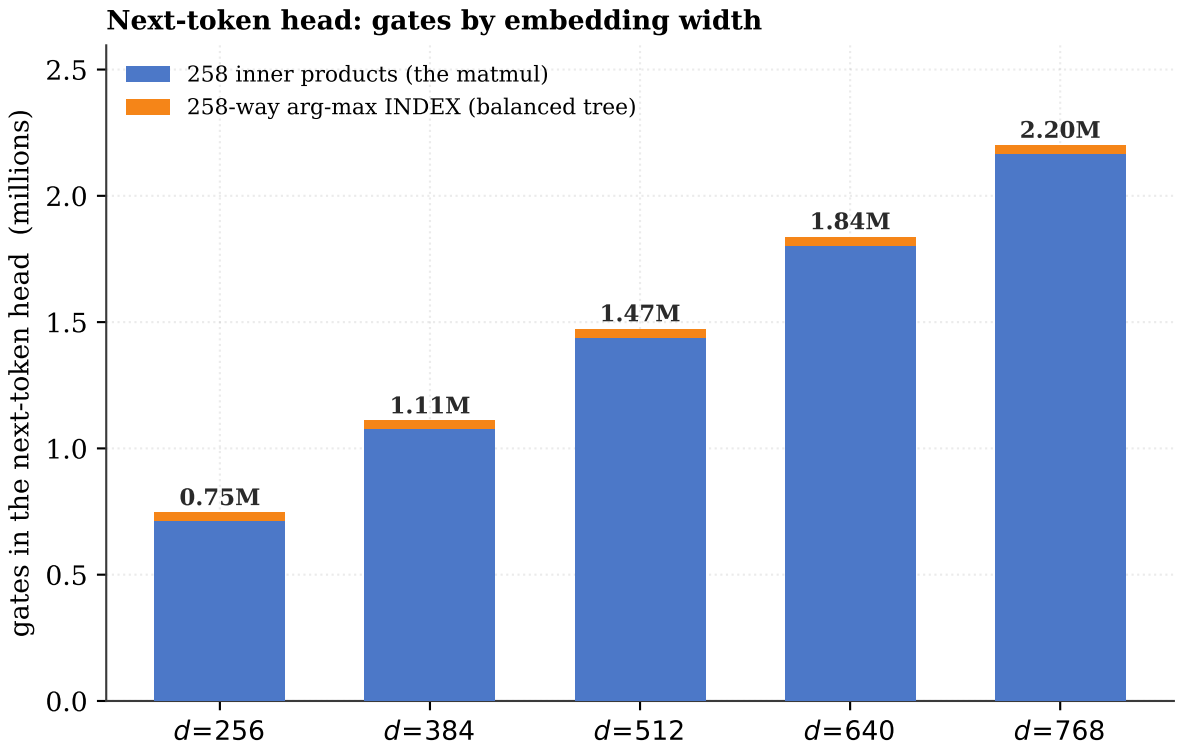


Figure 5: [MEASURED] The gate budget of the next-token head across embedding widths, decomposed into the V=258 binarized inner products (the matmul, which grows with D) and the 258-way arg-max *index* selection, which is nearly constant in D. Device capacity is deliberately not drawn here: it is a question about lookup tables, and Figure 11 answers it against named parts from measured technology mapping.

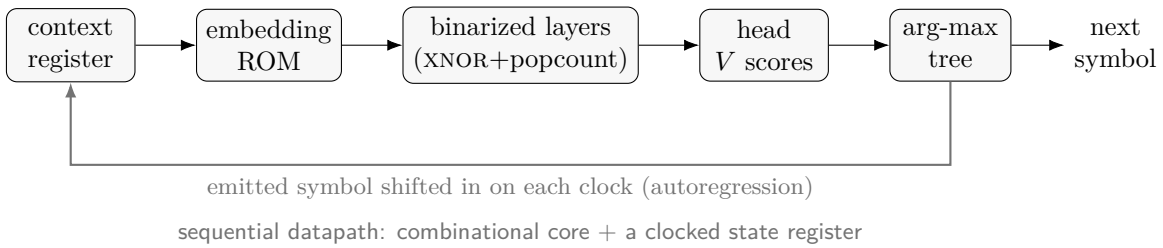


Figure 6: The inference datapath of a binary circuit model. The combinational core (embedding, binarized layers, head, arg-max) computes the next symbol from the context in one pass of logic; a clocked register shifts the emitted symbol back into the context, turning the autoregressive loop into an ordinary finite-state machine. Because the alphabet is finite—258 symbols fit in nine bits—the input and output are digital with no conversion. Two things the diagram does not show: the feedback wire carries a symbol, so the arg-max must emit an *index* and is costed as the index block throughout (Table 2); and "one pass of logic" is 1,255 levels of two-input gates with balanced tournament reductions, against 17,692 if the same reductions are built as left folds (Section 12).

Two properties make the whole model, and not merely its head, a well-posed circuit. First, autoregression is *state*: the emitted symbol is shifted into a context register and the combinational core re-evaluated on the next clock, exactly the structure of a finite-state machine (Figure 6). The model does not need an instruction stream to loop; the loop is a wire from the

output register to the input register. Second, the alphabet is *finite and small*—a few hundred symbols, nine bits—so there is no analog-to-digital boundary to cross: the model's inputs and outputs are already the bits the circuit carries. A binary model asks nothing of the world that a circuit cannot give it.

10 Attention as a Circuit

The one continuous operation in a transformer is the soft-max of its attention, and it is the natural objection to the thesis that a model is a circuit. The objection dissolves at its own limit. As the temperature $\tau \to 0$, the soft-max $\mathrm{softmax}(s/\tau)$ collapses to a one-hot $\arg\max$ over the scores, and $\arg\max$ is a purely combinational operation. *Hard attention*—the $\tau \to 0$ limit—is therefore an exact circuit, and we compile it end to end. The query–key similarities are the same XNOR-popcount inner products of Eq. (3), which are monotone in the signed dot product, so the arg-max over popcounts equals the arg-max over true scores; the winning key is chosen by a comparator tree that emits a one-hot index with first-max tie-breaking; and the value vector is read out by a one-hot multiplexer, giving $\mathrm{out} = V[\arg\max_j Q{\cdot}K_j]$. We check this netlist against a reference $\arg\max$: at $n_{\mathrm{keys}} = 16$ and head width $d = 64$ it agrees on 308 vectors—eight deterministic corner cases and 300 seeded random draws—and costs 12,906 gates [MEASURED]. Those 308 draws from a 2^{1600} input space are a *sample*; the smaller configurations (d=8, k=4) are *proved* equivalent by SAT (Section 8). Its size grows *superlinearly* in the number of keys: from 780 gates per key at k=4 to 1,061 at k=512, a rise of 36%, because the one-hot update that tracks the winner costs $K(K{-}1)/2$ AND gates. Fitting $aK + bK^2$ over $k \in \{4, \ldots, 512\}$ gives a quadratic coefficient whose 95% bootstrap interval, $[0.501, 0.577]$, excludes zero. The curvature only becomes visible past k=64, an octave below the context our whole-model budget assumes, which is why the sweep is carried to k=512. Full soft-max is recovered as the approximate extension: at any fixed numeric precision it too is a finite Boolean function, since its only transcendental piece, exp, is a finite integer look-up table—we realize a seven-bit-to-twelve-bit exp ROM in 684 gates, exhaustively verified over all 128 inputs—followed by an integer normalize-and-compare. Hard attention is exact as a circuit, and fixed-point soft-max is a finite Boolean approximation of bounded, precision-controlled error. Either way attention is not a wall: it is a comparator-mux tree over XNOR-popcount scores, optionally preceded by an integer exp table.

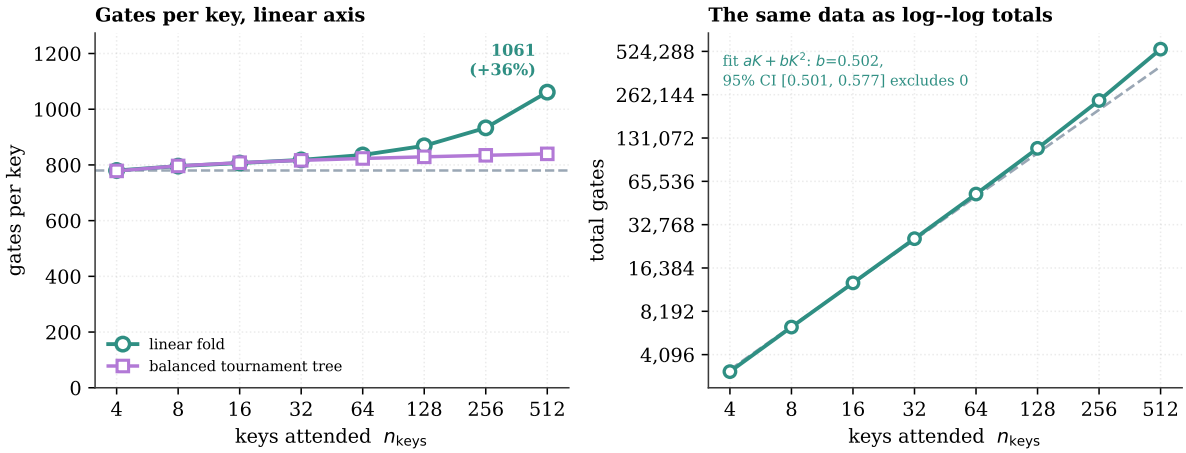


Figure 7: [MEASURED] Hard attention compiled to gates, swept to k=512—the context our whole-model budget assumes. *Left*: gates *per key* on a linear axis, the view in which the growth is plainly not linear: the cost per key rises 780 → 1,061, +36%, against the dashed flat line a linear cost would follow. Building the reduction as a balanced tournament tree (purple) removes almost all of that excess and is far shallower besides (Figure 12). *Right*: the same data as log–log totals, where a superlinear curve is visually indistinguishable from a linear one, which is exactly why the left panel is the one to read. Fitting $aK + bK^2$ gives $b = 0.502$ with a 95% bootstrap interval of $[0.501, 0.577]$, excluding zero. Attention is not a wall: it is a comparator-mux tree whose cost grows faster than the key count, which is why the attention window is the lever that decides the size of the chip.

11 Native-Binary Training: the circuit is the model

The compiler of the preceding sections binarizes a model trained in floating point, so its circuit computes an *approximation* of the model. The remedy is to train the model natively binary, so the circuit computes the model itself. We do this for the feed-forward sub-layers, which carry the majority of a transformer's parameters: each linear map is replaced by a layer whose weights and activations are pushed to $\{-1, +1\}$ by sign in the forward pass, with a per-output magnitude scale and a straight-through estimator for the gradient. The inner product of such a layer is *exactly* the XNOR-popcount of Eq. (3); the layer is not a model to be compiled but a gate netlist that was trained. On the compiled-bytecode corpus a small native model—six layers of width 192, some three million parameters—trains to a held-out 0.57 bits per token against the 8.01 of the uniform baseline [MEASURED, training log].

Three things are worth establishing precisely about a layer of this kind, and each of them is a check that can fail.

The stored weights, and the netlist that computes them. The property that matters is not that a binarizer returns values in $\{-1, +1\}$—it does so by construction, for an untrained or all-zero tensor as readily as for a trained one—but that the *stored* parameter is well behaved and that the emitted netlist computes the layer. Both are checked here. The stored weights of block 0 contain no exact zeros, the one point at which the sign map is ill-defined and a netlist could legitimately disagree with its layer, with a minimum magnitude of 2.6×10^{-8} in `fc1` and 4.3×10^{-8} in `fc2`. And we compile *all* 512 *neurons* of that layer with their weights baked in and diff each netlist against the layer's own agreement count: 512/512 bit-exact over 30,720 checks [MEASURED], with every one of those neurons additionally *proved* over its entire 2^{128} input space in Section 16.

Opcode validity is a saturated metric; entropy is the one to read. A metric that greedily

decodes 400 tokens from a single prompt, assumes even positions are opcodes and scores them against `dis.opmap` is not discriminative: a constant stream of one legal opcode scores 100%, and we verify that it does, as do two alternating opcodes and an opcode interleaved with zeros. All seven native runs report exactly 100.0%, which is the signature of a saturated metric, and the decisive tell is that the *held-out corpus itself*—real compiled Python bytecode—scores 90.0%: a generator that beats its own data on a validity metric is degenerate, not excellent. Opcode entropy shows what the validity score hides. The corpus uses 256 distinct opcodes at 2.651 bits, while greedy decodes from these checkpoints reach 0.39× to 0.78× that entropy, with three-gram repetition of 0.74 to 0.84 [MEASURED]. The pre-registered criterion was entropy within 20% of the corpus and the best checkpoint reaches 78% of it, so entropy and repetition are the numbers reported here in place of a validity percentage. Executability is not measurable on these checkpoints at all, because the corpus they were trained on discards the fields execution requires; that is a retraining experiment (G5), not an evaluation.

What the circuit covers. It is the model for the feed-forward *matmul*, and not yet for the sub-layer as a whole. The trained layer is $y = (\mathrm{sign}(x)\ \mathrm{sign}(W)^\top)\alpha + b$ with a floating-point per-output scale α and bias b, wrapped by LayerNorm before and a GELU after, inside a full-precision residual stream. Some of that does compile, and cheaply: because $\alpha > 0$ and GELU preserves sign, the *next* layer's sign collapses the scale, the bias and the activation into a single constant comparator per neuron,

$$\mathrm{sign}\big(\mathrm{GELU}(\alpha(2p - n) + b)\big) \;=\; \big[\, p \;\geq\; \lceil (n - b/\alpha)/2 \rceil \,\big], \tag{5}$$

which costs 12,070 gates across the 512 neurons, 1.82% on top of the matmul [MEASURED]. That per-neuron threshold is a line item in every whole-model budget in this paper (Figure 10).

The rest does not compile away. `fc2` feeds the residual adder rather than a sign, so its value must actually be produced, scale and bias included. And LayerNorm needs a mean, a variance—d squares, that is d *multipliers*—a reciprocal square root, and a per-channel multiply–add. Priced in 16-bit fixed point from our own verified primitives, one LayerNorm at d=128 is 665,373 gates, so the two per block come to 200% of the 664,520-gate matmul they wrap [MODELLED from MEASURED primitives; the word width and the fully parallel combinational form are our assumptions, and a serial implementation would trade area for latency]. The pre-registered threshold for describing a datapath as containing "no multiplier, no floating-point unit" was 10%; the normalizer is twenty times that, so the multiplier-free statement is scoped to the feed-forward *matmul* and the head. This is the single largest term standing between the model and an all-integer circuit, and Section 15 removes it: a multiplier-free shift-normalizer that costs 3.2× fewer gates and leaves no general multiplier anywhere in an autoregressive step.

Attention and the embedding remain full precision here, and their native binarization is the natural continuation.

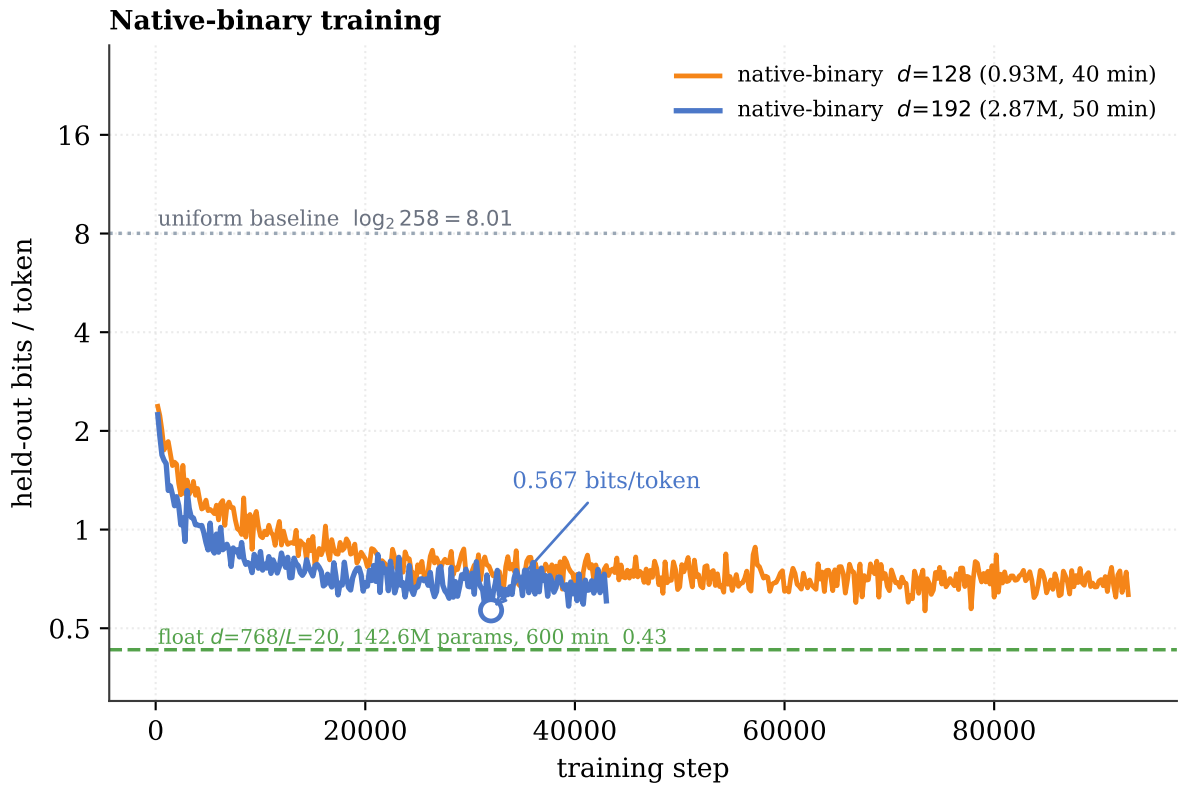


Figure 8: [MEASURED, from training logs] Native-binary training. Two small models whose feed-forward weights and activations are constrained to $\{-1, +1\}$ throughout training drive held-out bits-per-token down from the 8.01 uniform baseline to well under one bit; the larger reaches 0.57. **The green dashed line is a reference, not a control**: the floating-point model behind it has 142.55M parameters against this model's 2.87M and trained for 600 minutes against 50—49.7× the parameters and 12× the budget—so the 0.14-bit gap to it measures size and budget, not the cost of exactness. The iso-parameter, iso-budget control that would measure that cost is pre-registered as experiment G4.

Size, at matched budget. Five native-binary models spanning half a million to eighteen million parameters give bits per token of 0.706/0.686/0.665/0.647/0.686 once every point is re-derived from the training logs at a *matched* wall-clock budget. The series is not monotone. The fitted slope is -0.019 bits per decade with a 95% bootstrap interval of $[-0.056, +0.030]$, which includes zero, and the whole range spanned by the trend is 0.059 bits while two runs at the *same* size differ by 0.095 bits: the apparent effect is smaller than the run-to-run noise. Our pre-registered criterion was that the slope's confidence interval exclude zero, and it does not.

Budget matching is what decides this. Four of the five runs took 2,398–2,987 seconds; an 18.14M run given 7,188 seconds—2.7× the budget—reaches 0.591, while the matched-budget run at the same size reaches 0.686, level with the 0.93M model. Native binarization therefore trains stably across a thirty-fold range of model sizes, with no size effect observable at matched budget. Whether one exists is a question for a controlled iso-token study with multiple seeds, pre-registered as experiment G1.

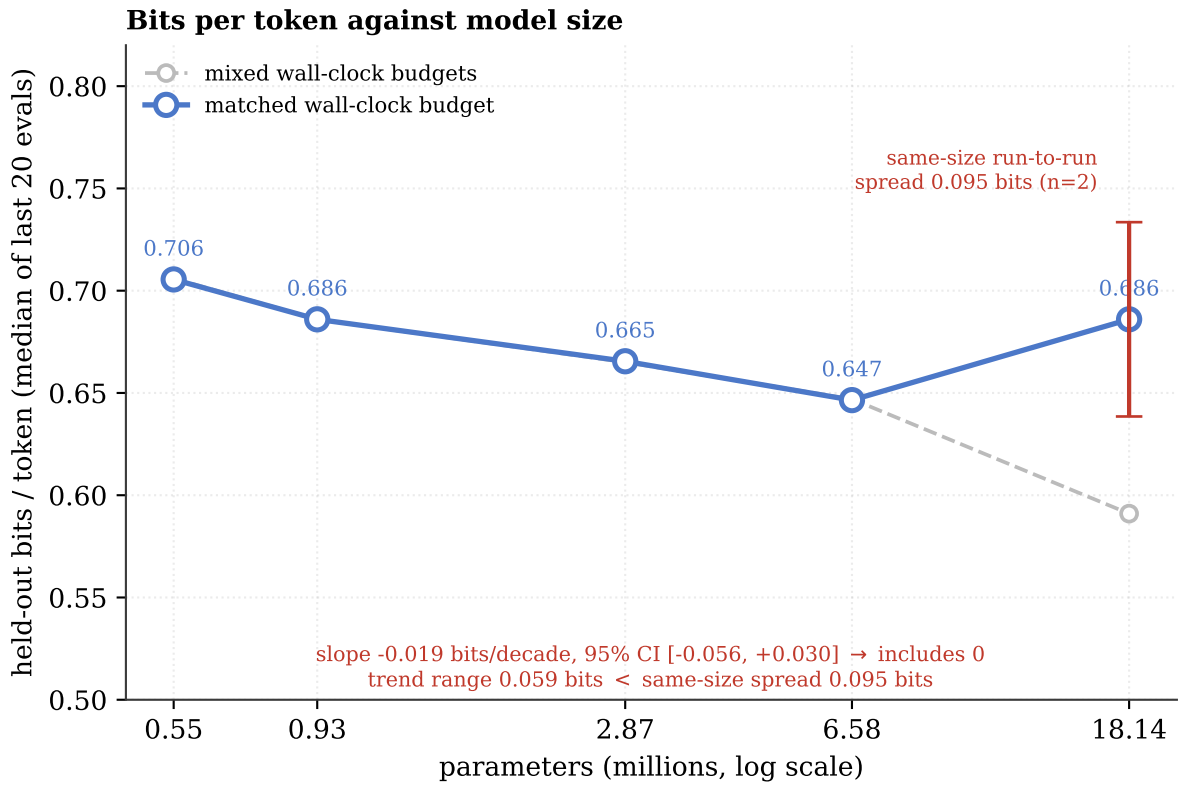


Figure 9: [MEASURED, re-derived from the training logs] **No size effect at matched budget.** Grey: the series with mixed budgets—its top-end point comes from a run given 2.7× the wall-clock of the others. Blue: the matched-budget series, in which the 18.14M model lands back at 0.686, level with the 0.93M model. The red bar is the spread between the two runs at that same size, 0.095 bits, which exceeds the 0.059-bit range of the whole trend, and the fitted slope's 95% interval includes zero. The y-axis spans a full range rather than a window, so the size of the effect is visible against the size of the noise.

To make the compilation fully concrete, we take a single *actual* neuron from this trained model—output 0 of block 0's `fc1` in `binmodel_native_d128.pt`, a fan-in of one hundred and twenty-eight with sixty-five negative weights—and compile it with its weights *baked in*. A constant weight collapses the XNOR: $\mathrm{XNOR}(x, 1) = x$ is a wire and $\mathrm{XNOR}(x, 0) = \neg x$ is a single inverter, so the XNOR array of the general block is replaced by an inverter mask on the sixty-five disagreeing inputs, followed by the popcount adder tree—a specific 1,300-gate circuit [MEASURED], which Icarus Verilog confirms agrees with the Python netlist on 2,006 vectors and which ABC maps to 254 LUT6. Adding the threshold of Eq. (5) makes it a 1,324-gate circuit emitting the single bit the next layer consumes.

Two notes on this example. The saving from baking is real but modest: 1,300 against the general block's 1,363 gates, 4.6%—the inverter mask is nearly as expensive as the XNOR array it replaces, because an XNOR with a constant is an inverter about half the time. And every number here is pinned to one named checkpoint, which each artifact script takes explicitly and defaults to, so the worked example reproduces exactly.

12 The Whole Model, and What It Would Cost as a Chip

We have counted the head; the natural question is the whole model. Composing the compiled blocks at the dimensions of an actual trained native model—the sub-million-parameter model of

the previous section, four layers of width 128—gives an end-to-end budget for one autoregressive step (Fig. 10).

The budget counts *every* term, and the terms that are easy to leave out are the ones that decide it. The attention $Q/K/V/O$ projections are $4d^2$ binarized multiply–accumulates per layer and add 2.79 million gates—29% of the total—on their own. The per-neuron sign thresholds of Eq. (5), the LayerNorms, the residual adders and the positional-embedding table are each a line item. And the attention term is *built* at the key count in question rather than extrapolated from a smaller build by a per-key slope, which, because that cost is superlinear (Section 10), would under-count the 512-key case by 13.7%.

At a local context of 128 keys the model is **24.0 million gates** [MODELLED from MEASURED blocks], of which 13.6 million are the exact-integer datapath—XNOR-popcount, compare and multiplex, read straight off the trained $\{-1, +1\}$ weights—and 10.4 million are fixed-point terms that need a multiplier, dominated by the 6.0 million gates of LayerNorm. At a global context of 512 keys it is 33.5 million, attention having grown from 3.1 to 12.7 million. The design fact this makes sharp is that the attention window, not the parameter count, is the lever that decides whether the chip is dominated by its mixing or by its multiply–accumulates.

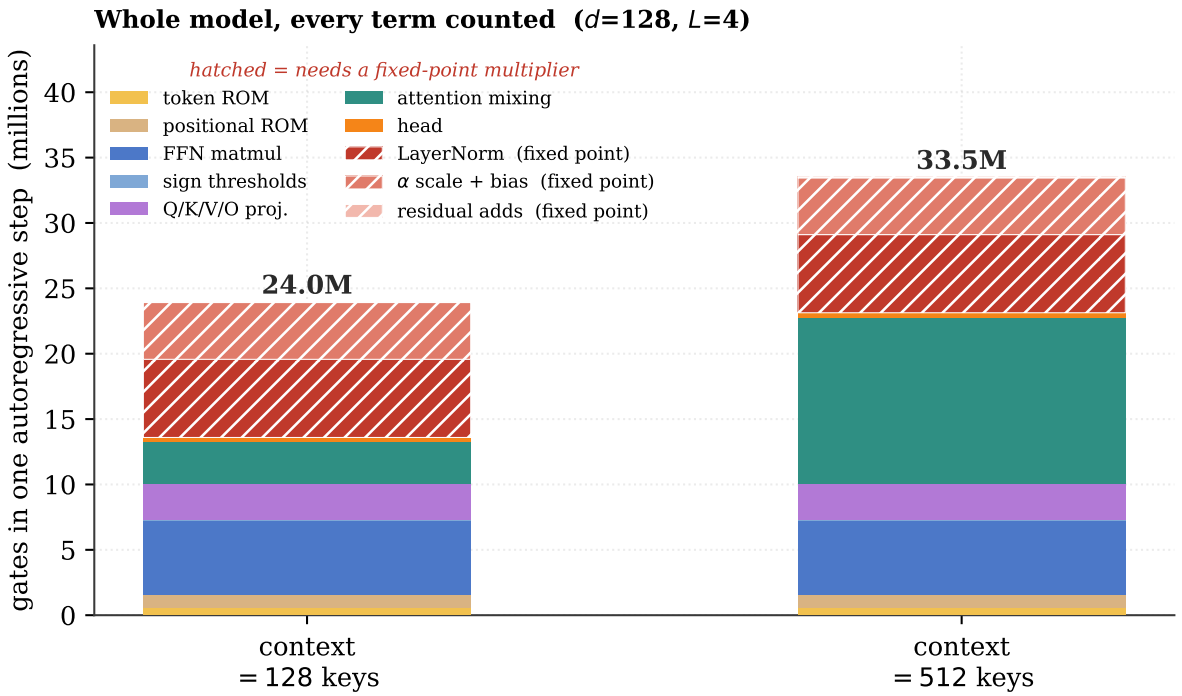


Figure 10: [MODELLED from MEASURED blocks] The whole native-binary model as one autoregressive step of logic, at the dimensions of a real sub-million-parameter model (d=128, four layers), with *every* term counted. Solid components are the exact-integer datapath; hatched components are fixed-point and require a multiplier. The totals are 24.0M gates at a 128-key context and 33.5M at 512. LayerNorm alone (6.0M) is larger than the feed-forward matmul it wraps (5.6M), which is what makes the normalizer the term worth attacking (Section 15).

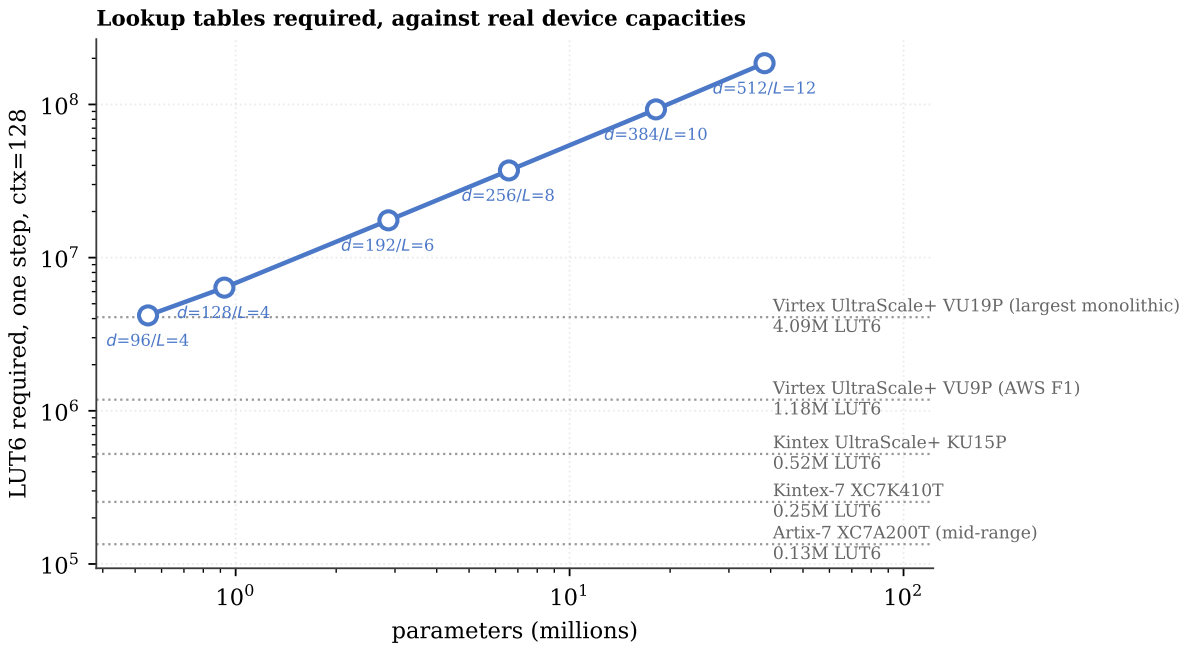


Figure 11: [LUT6 MEASURED via YOSYS+ABC; device capacities ASSERTED from vendor tables] Every native-binary checkpoint's lookup-table requirement for one autoregressive step at a 128-key context, against real device capacities. **Every point lies above every line.** The d=128, L=4, 0.93M model this paper costs in detail needs 6.37×10^6 LUT6, exceeding the largest monolithic FPGA in production by 1.6×; the best model in this work (d=512, L=12, 38.4M parameters) needs 1.86×10^8 LUT6, 45× the largest device; and even the smallest model trained here (d=96, 0.55M) is 1.03× over. Naming the device is what turns "it fits on a chip" into a checkable statement, and once named the whole model does not fit—though its exact-integer datapath does, as the text below shows.

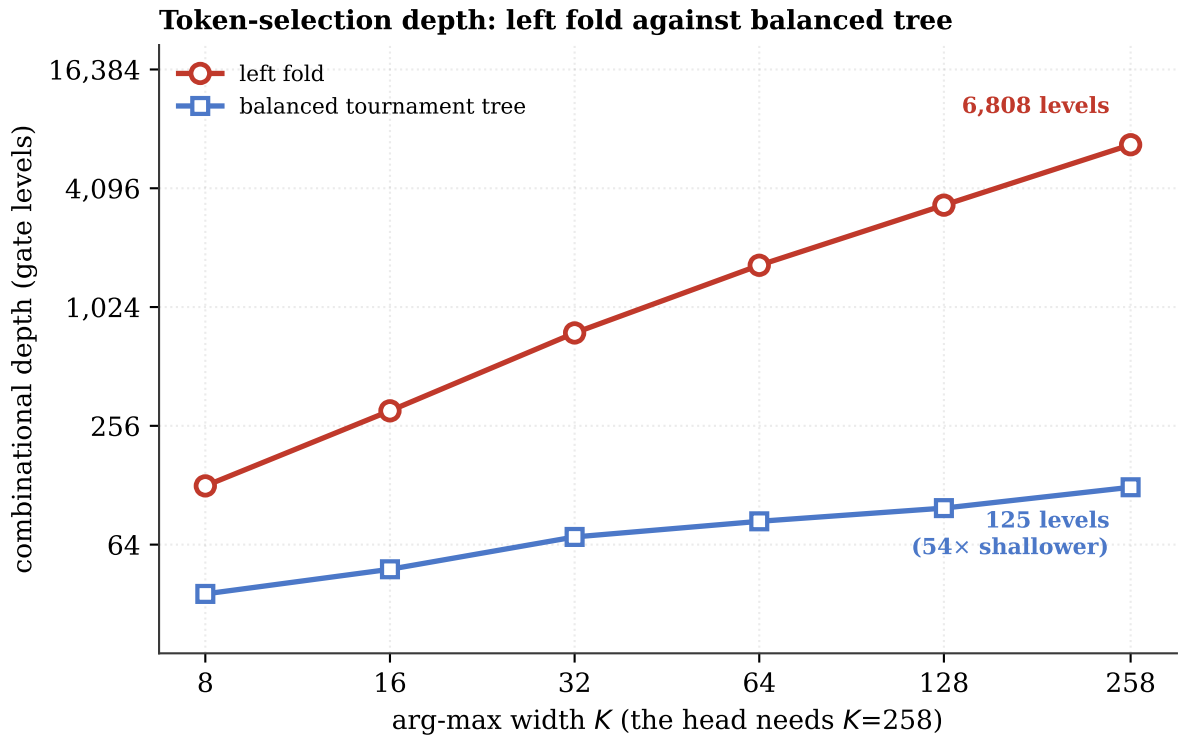


Figure 12: [MEASURED] Combinational depth of the token-selection block against its width. The block the natural way to write a reduction—a left fold—has depth linear in the vocabulary: 6,808 levels of two-input gates at the K=258 the head requires. A balanced tournament tree over a parallel-prefix comparator—implemented, checked against `numpy.argmax`, and SAT-proved equivalent over the entire input space at small widths—is 125 levels, 54× shallower, and uses 45% fewer gates. Depth is what a clock period has to cover, so this is the structural choice that makes a clock rate reachable at all; Section 14 places the registers and measures the rate.

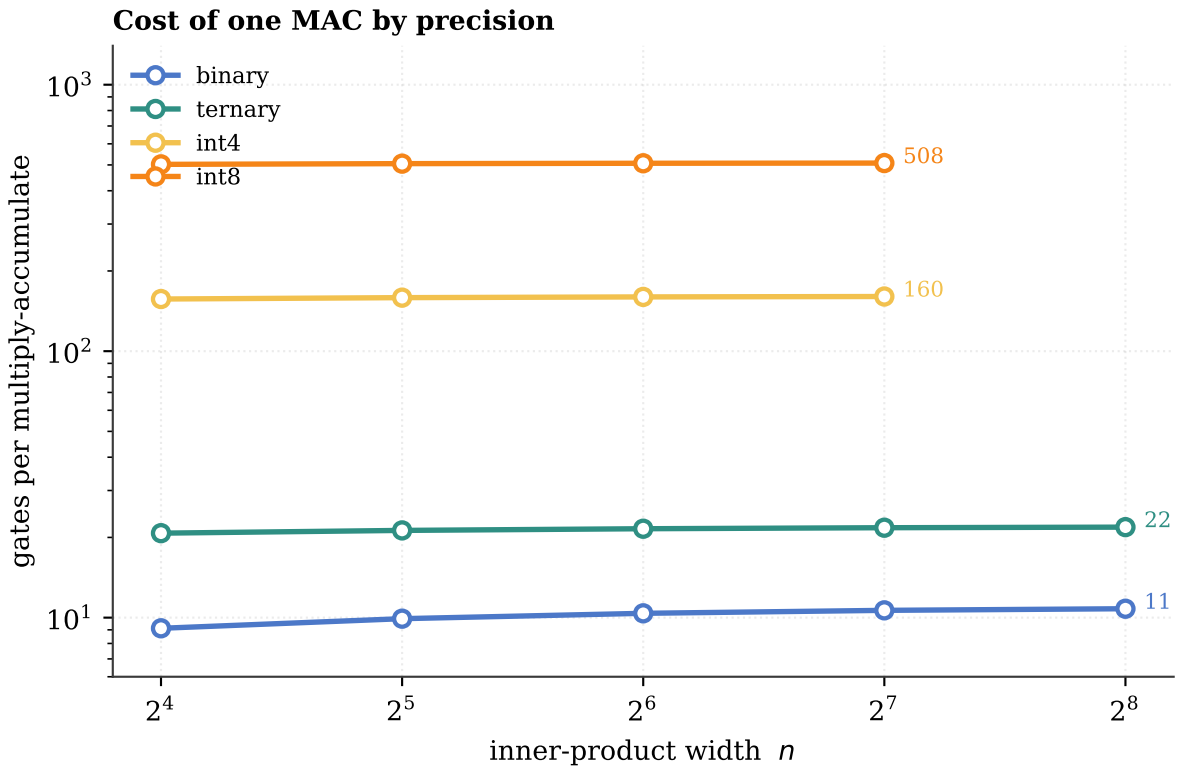


Figure 13: [MEASURED] Gates per multiply–accumulate for an inner product built at four precisions from the same verified primitives, each checked against its integer specification before being counted. Binarization is worth 47.7× the area of an int8 array-multiplier datapath at n=128 and 15.1× that of int4; ternary (BitNet b1.58 style) costs 2.04× binary. This is the *cost* axis; the same four blocks are carried through place-and-route in Section 13.3, where the ratio is restated in placed silicon and power. Whether binary sits on the accuracy–cost Pareto *front* depends on what each scheme achieves at matched training budget, which is experiment G3.

Which part of it fits one FPGA. Answering this needs a measured exchange rate rather than an assumption about how many gates a lookup table "absorbs". ABC packs our netlists at 3.6 to 6.2 gates per LUT6 (Table 3); applying the measured per-block ratios, the d=128 model needs approximately 6.4×10^6 LUT6 at a 128-key context and 8.9×10^6 at 512 [MODELLED from MEASURED ratios], against the 4.1×10^6 LUT6 of the largest monolithic field-programmable gate array in production, the Virtex UltraScale+ VU19P [ASSERTED, vendor table]. The trained model does not fit the largest monolithic device made, and it is not close: it is over by a factor of 1.6 at a local context, on the largest and most expensive part of its kind. Figure 11 plots every native checkpoint against real device capacities and none of them fits, from the smallest model trained here (0.55M parameters, 4.20×10^6 LUT6, just over the VU19P's budget) to the largest (38.4M parameters, 1.86×10^8 LUT6, 45× the device).

The interesting statement is the decomposition. The *head* fits comfortably (Eq. (4)). The exact-integer *datapath* of the whole model—the part that is genuinely XNOR-popcount, compare and multiplex, 13.6M gates—maps to 3.48×10^6 LUT6, which is 0.85× the VU19P's budget: **it fits**. What pushes the design over is the fixed-point remainder around it—the norms, the scales, the residual stream—which add 10.4M gates and 2.9×10^6 LUT6 on their own. The device question and the compilation question are therefore the same question, which is why Section 15 attacks the remainder directly.

How deep one token is. A combinational step has a depth, and that depth is what any clock period must cover. Built with left folds, one autoregressive step at d=128, L=4 is **17**,**692** levels of two-input gates [MEASURED]: a left-fold reduction's depth is linear in its width, which costs 5,780 levels for the K=258 token select (6,808 when it emits an index; 1,027 even after LUT mapping) and 2,935 per attention layer at a 128-key context. Built with balanced tournament trees over a parallel-prefix comparator—which we implement, check against `numpy.argmax` and prove equivalent by SAT—the token select is 107 levels and the attention layer 244, and the whole step is **1**,**255** levels: a 14× reduction that also uses 45% fewer gates in the select block. Depth alone is not a token rate: one token per clock is a statement about a pipelined design with a placed clock tree and a timed critical path, and Section 14 builds exactly that, for a block rather than for the whole step.

Energy, from two directions. From the literature: Horowitz's ISSCC-2014 survey [28] puts a 32-bit floating-point multiply–accumulate at roughly 4.6 pJ and an 8-bit integer add at roughly 0.03 pJ at 45 nm [ASSERTED, cited]. From this work: the *area* ratio, measured. A binarized inner product costs 10.6 gates per multiply–accumulate at n=128, against 508 for the same inner product built from int8 array multipliers and 160 for int4—a factor of 47.7 and 15.1 respectively—and it is 2.6× shallower [MEASURED, every block checked against its integer specification before being counted]. A ternary (BitNet b1.58 style) inner product costs 2.04× binary. Turning a gate ratio into a joule ratio takes a physical flow, which is what Part III supplies: Section 13.3 places the same four blocks and reports area and power in silicon.

Part III
Physical Implementation

13 From Netlists to Silicon: place-and-route, timing and power

Every section so far counted things a synthesis tool can report: gates, lookup tables, logic levels. None of those is an area, a delay or a joule. This part supplies all three, by carrying every block in the paper through a complete open physical-design flow: floorplanning, placement, routing, static timing analysis and power analysis, in one named library at one named corner. The flow constrains, places, repairs and analyses, and reports the slack the tool finds; it does not iterate a design to a target frequency and sign it off, so no result here is a timing *closure*, and no physical device has been touched.

The flow. We installed, without administrative privileges, an entirely open physical-design stack: OpenROAD [26] (which embeds OpenSTA for timing and power) together with the `open_pdks` distribution of SkyWater's SKY130 process design kit [27], whose `sky130_fd_sc_hd` standard-cell library is publicly documented and freely redistributable. The path from our compiler to a placed design is:

1. our gate netlist emits Verilog (Section 17);
2. YOSYS 0.68 with ABC maps it onto real SKY130 cells against the library's own timing tables, rather than onto the abstract two-input cells of Table 3;
3. OpenROAD initializes a floorplan, inserts well taps, places the I/O pins, runs global and detailed placement, repairs slew and capacitance violations by buffering and resizing, performs clock-tree synthesis where the design has registers, routes globally, extracts parasitics, and then runs static timing and power analysis.

Everything is reported in the `tt_025C_1v80` corner: typical-typical silicon, 1.80 V, 25°C. Every tool transcript is kept with the artifact, and every placed design is written back out as a DEF, which is what Figure 14 is drawn from.

What these numbers are. They are MEASURED: a physical-design tool ran, and this is what it reported for a design it actually placed. Area, timing and power for an unfabricated design mean exactly what they mean here—what a tool computes from a library's characterized tables—and nothing has been fabricated. SKY130 is a 130 nm node, two decades behind a modern accelerator, so the *absolute* figures are a yardstick rather than a competitive number, and we never compare them to a commercial part. What one consistent flow licenses is *ratios*: block against block and precision against precision.

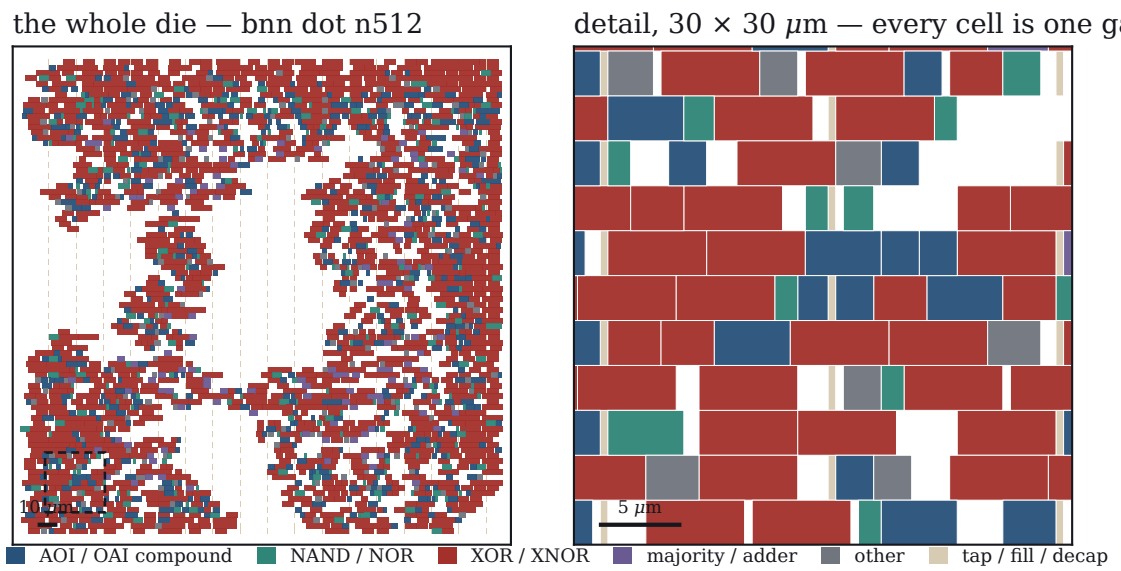
13.1 The block library in silicon

Table 4 is what the tools reported for every block the paper counts gates for.

Table 4: Place-and-route [MEASURED: YOSYS 0.68 + ABC → OPENROAD 2.0, `sky130_fd_sc_hd`, `tt_025C_1v80`]. "Area" is the sum of placed standard-cell areas; "Die" is the side of the square core the floorplanner chose; t_{crit} is the longest path OPENSTA found after parasitic extraction, and $F_{\max}$ its reciprocal for a purely combinational block. α is the switching activity, which we *measured* on the netlist rather than assumed (Section 13.2); "Power" is total power at that activity.

Block	Cells	Area (μm^2)	Die (μm)	t_{crit} (ns)	$F_{\max}$ (MHz)	α	Power (μW)
adder8	25	289	32	2.86	349	0.48	49
argmax_tree_K8_w6	169	1,141	59	3.40	294	0.39	148
exp_rom_7_12	150	1,297	57	1.83	546	0.16	87
maxvalue_K8_w6	173	1,468	64	7.58	132	0.39	197
bnn_dot_n64	304	4,476	90	4.24	236	0.37	448
real_neuron_d128_thresholded	563	5,200	116	4.81	208	0.35	350
real_neuron_d128_baked	558	5,612	119	4.41	227	0.36	380
bnn_dot_n128	623	8,312	129	4.87	206	0.36	537
argmax_tree_K64_w10	2,506	19,426	207	9.07	110	0.39	1233
bnn_dot_signed_n512	2,516	32,014	249	7.04	142	0.36	1185
bnn_dot_n512	2,536	33,148	251	6.35	158	0.36	1230
hard_attention_d64_k16_dv32	6,226	73,338	373	23.35	43	0.34	1946
argmax_tree_K258_w10	10,491	92,714	748	14.55	69	0.37	2964

Three things in this table are worth stating in prose. First, a gate our compiler emits costs between 1.9 and 7.2 μm^2 of placed silicon in this library, a median of 5.0—which is the exchange rate that turns every gate count in this paper into an area, and which sits *below* the area of one standard cell precisely because ABC removes 35–68% of our gates before anything is placed (Table 3); Figure 15 plots the whole ladder from our emitted count through ABC, lookup tables and standard cells to square micrometres. Second, combinational delay tracks logic depth but sub-linearly: the n=512 inner product is 36 levels deep and 6.35 ns, while the 8-bit adder is 17 levels and 2.86 ns, because the mapper flattens shallow chains into complex cells. Third, and most usefully, the whole 258-way token-selection tree of a real head—34,438 gates, the block that emits the next symbol—is 92,714 μm^2 and 14.55 ns, which is 0.093 mm^2: a next-token selector is a tenth of a square millimetre in a 130 nm process.



2,536 standard cells | 33,148 μm^2 of cell area | die 251 × 251 μm
critical path 6.35 ns (F_{max} = 158 MHz) | 1230 μW at α = 0.36

Figure 14: [MEASURED] A real placed design. Every rectangle is one standard cell at the coordinate OpenROAD's placer chose, read back from the DEF the tool wrote; colour is the cell's function class. The red mass is the XOR/XNOR of the population-count adder tree—Eq. (3), as silicon. *Right*: a detail of the densest region, at the scale where individual gates are visible. The flow is OpenROAD 2.0 with `open_pdks` SKY130, library `sky130_fd_sc_hd`, corner `tt_025C_1v80` (1.8 V, 25°C). Nothing has been fabricated; these are a physical-design tool's numbers from a characterized cell library, which is what area, timing and power mean for any unfabricated design.

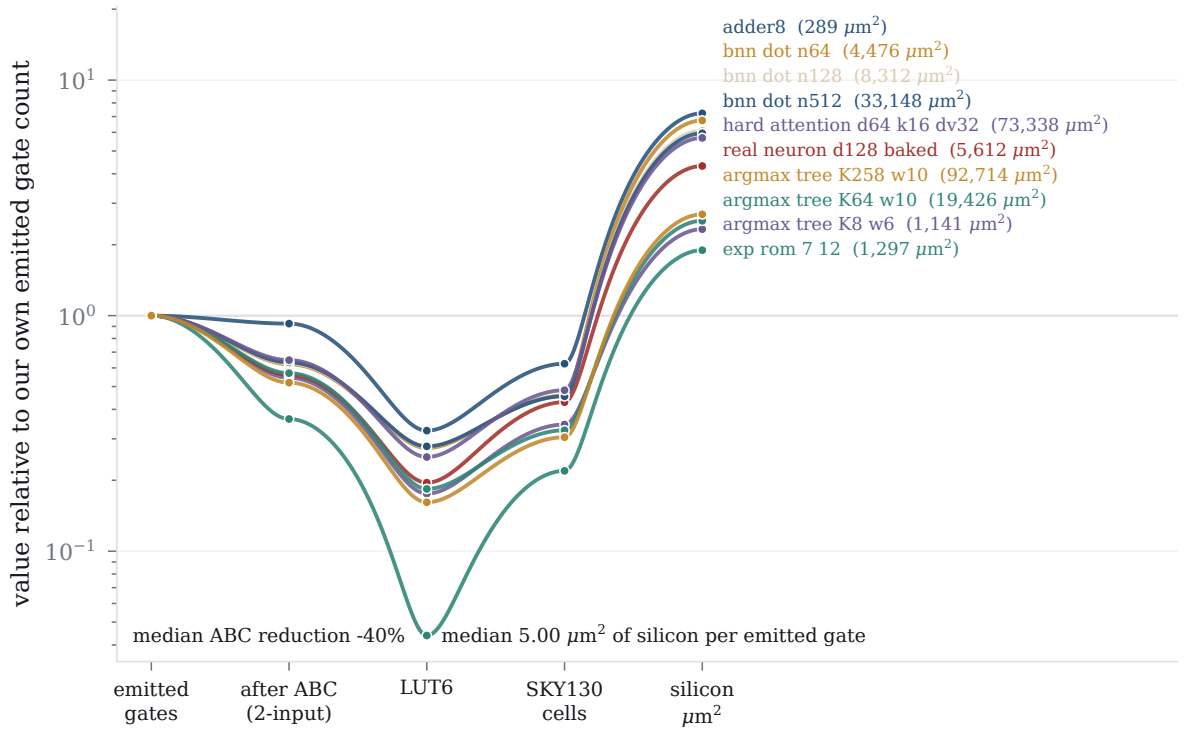


Figure 15: [MEASURED] The ladder from our own gate count to square micrometres. Each ribbon is one block, normalized to the number of gates our compiler emits, so the drop between the first two columns is exactly the slack ABC finds (Table 3) and the rise into the last column is the area a real cell costs. The last column is not a count but an area in square micrometres, plotted on the same relative axis to make the exchange rate visible: the median is 5.0 μm^2 of placed silicon per gate our compiler emits.

13.2 Switching activity, measured rather than assumed

Power is $CV^2f\alpha$, and α—the probability that a net toggles in a cycle—is the one term a physical-design tool cannot know without a stimulus. OpenSTA will *propagate* an assumed input activity through the logic, and its propagation rule for XOR adds the operands' activities without a cap. A population count is a tower of XORs, so under propagation the reported power of our deepest block came out two orders of magnitude above blocks with ten times its cell count. That is an artifact of the estimator, not a property of the circuit.

So we measured it. Each real gate netlist is evaluated on a stream of input vectors and, for every wire, we count how often it changes between consecutive vectors. Under uniformly random inputs the mean toggle rate across the thirteen blocks of Table 4 is 0.34–0.48 for every arithmetic block, and 0.16 for the one pure ROM—remarkably flat, and *not* growing with depth, which is exactly what tells us the propagated figure was an artifact. Under a correlated stream, in which each vector flips five percent of its predecessor's bits (closer to what an autoregressive datapath sees, since the context register shifts in one symbol per step), it falls to 0.06–0.15. Both are handed to OpenROAD as a *global* activity, which does not propagate and so cannot compound. Table 4 reports power at the random-input figure, and the receipt carries the correlated one alongside, between 1.3 and 7.1× lower.

13.3 The precision comparison, in silicon

Section 12 reported that a binarized inner product costs 47.7× fewer *gates* than the same inner product built from int8 array multipliers. A gate count is not an area and not a watt, so we built the same fan-in-64 inner product at four precisions, verified each against its integer specification, and put all four through the identical physical flow. Table 5 is the result: the comparison restated in placed silicon and measured power.

Table 5: The precision family in silicon [MEASURED]. Same fan-in (n=64), same flow, same corner, each block checked against its integer specification before it was placed. "Gates/MAC" is an exact netlist count; the remaining columns are what OpenROAD and OpenSTA reported. Power is at each block's own measured switching activity.

Scheme	Gates/MAC	Cells	μm^2/MAC	t_{crit} (ns)	μW/MAC	vs. binary
binary	10.4	304	69.9	4.24	4.65	1.0×
ternary	21.6	615	108.1	5.27	6.87	1.5×
int4	159.8	4,565	768.5	7.76	36.05	11.0×
int8	507.2	16,453	2829.7	12.86	150.52	40.5×

The gate-count ratio and the silicon-area ratio do not agree, and the difference is informative: the 47.7× of Section 12 becomes 40.5× in placed area, because the multiplier's regular structure maps onto complex cells slightly better than our XOR-popcount tree does. The power ratio is the headline. On one flow in one library the int8 datapath draws 32× the power of the binary one at matched activity and matched fan-in, and is 3.0× slower besides. That is a measurement of a placed design, and it is the form in which the energy advantage is stated: a ratio between two designs put through the same flow, rather than a joule figure for a chip that does not exist.

The scope of the comparison. It is a *cost* comparison at equal fan-in. Whether binary also sits on the accuracy–cost front depends on what each scheme achieves at a matched training budget, which is experiment G3 and needs a GPU.

14 Pipelining, and a Measured Token Rate

A throughput claim is a claim about a clocked design, so it needs registers, a clock tree and a timed critical path. This section puts all three in, for a compiled neuron of a real trained model, and measures what they cost.

Construction. The pipeliner cuts a combinational netlist into S stages of equal logic depth and registers every wire that crosses a stage boundary. A wire produced in stage p and last consumed in stage q receives a chain of $q - p$ registers, so that every operand of every gate arrives in the same cycle. That last property—*register balancing*—is what makes a pipeline correct rather than merely registered, and getting it wrong is silent: the circuit computes a mixture of two different inputs and most random vectors will not notice. We therefore check it three ways.

1. **An exhaustive structural invariant.** For *every* gate we assert that no operand is assigned to a later stage than the gate itself. This is checked over all gates, not a sample.
2. **Simulation.** The emitted sequential Verilog is streamed one vector per clock under Icarus Verilog and compared, at the stated latency of $S - 1$ cycles, against the combinational netlist's own evaluation. Every configuration in Table 6 agrees on every vector.

3. **Proof.** For the small blocks we build a miter between the pipelined module and the combinational one delayed by $S-1$ cycles and discharge it with YOSYS's SAT engine over $S+2$ unrolled cycles. For a feed-forward pipeline whose entire state is $S-1$ ranks of registers, the output at any cycle is a function of the previous S inputs alone, so a bound of $S+2$ cycles is *complete* rather than merely bounded: it establishes the equivalence for all inputs and all time. The proof is written independently of the testbench, so the two together also pin the latency: a cross-check neither method gives on its own.

Table 6: Pipelining [MEASURED]. S is the number of stages; S=1 is the purely combinational design and has no registers at all. "Flip-flops" is the register count the balancing requires. Area, T_{clk} (the constraint period minus the slack OpenSTA reports), $F_{\max}$, latency and power are from the same OpenROAD flow as Table 4, with clock-tree synthesis performed for every $S > 1$. Throughput is one result per clock; energy per result is total power divided by $F_{\max}$.

Block	S	Flip-flops	Area (μm^2)	T_{clk} (ns)	$F_{\max}$ (MHz)	Latency (ns)	Throughput (M/s)	Energy (pJ/result)
bnn_dot_n16	1	0	1,008	2.73	366	0.0	366	0.3
	2	12	986	2.26	442	2.3	442	0.6
	4	35	1,554	1.49	671	4.5	671	0.8
	8	83	2,676	1.35	741	9.4	741	1.5
	16	197	4,728	1.41	709	21.1	709	2.7
bnn_dot_n64	1	0	4,476	4.64	216	0.0	216	1.9
	2	28	4,322	3.27	306	3.3	306	2.3
	4	104	5,733	2.19	457	6.6	457	3.4
	8	283	11,024	1.79	559	12.5	559	10.9
	16	577	20,473	1.88	532	28.2	532	24.6
real_neuron_d128_baked	1	0	5,612	4.81	208	0.0	208	2.6
	2	44	6,215	3.94	254	3.9	254	4.2
	4	200	8,409	2.55	392	7.7	392	5.3
	8	486	19,009	1.99	503	13.9	503	20.9
	16	1,005	34,275	2.11	474	31.6	474	44.7
	32	2,036	65,588	1.93	518	59.8	518	83.1
argmax_tree_K64_w10	1	0	19,426	9.47	106	0.0	106	11.9
	4	794	41,206	4.07	246	12.2	246	41.4
	16	5,187	162,339	3.58	279	53.7	279	217.7
	32	11,252	335,434	3.41	293	105.7	293	462.6
hard_attention_d64_k16_dv32	1	0	76,012	12.26	82	0.0	82	23.1
	8	4,549	203,416	3.60	278	25.2	278	92.3
	32	20,824	714,150	4.47	224	138.6	224	526.1

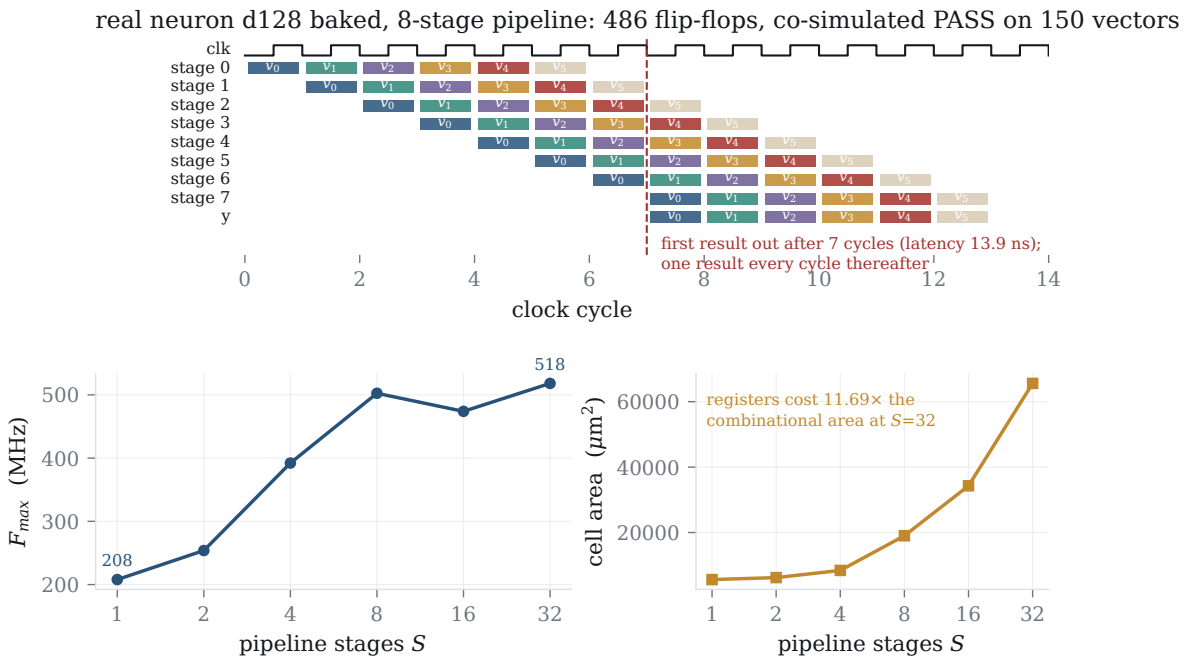


Figure 16: [MEASURED] The pipeline, its waveform, and what it buys. The occupancy diagram is the register-balanced design the compiler emits, simulated under Icarus Verilog against the combinational netlist and proved equivalent by SAT over the unrolled circuit. $F_{\max}$, latency and area are from OpenROAD/OpenSTA after clock-tree synthesis in `sky130_fd_sc_hd`, `tt_025C_1v80`. The registers are not free: they are the price of the throughput, and the right-hand panel is that price.

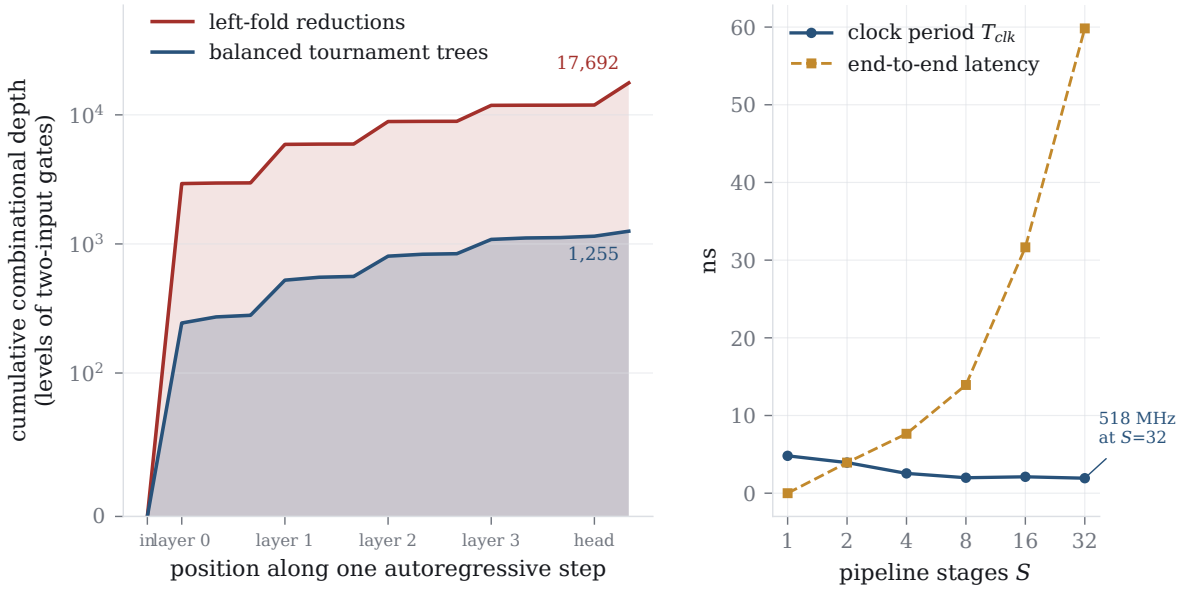


Figure 17: [MEASURED] *Left*: cumulative combinational depth along one autoregressive step, for the same model with its reductions built as left folds and as balanced tournament trees. The vertical axis is symmetric-log, so the shape is the profile of the critical path rather than a bar chart of totals; depth from the gate compiler, in exactly the decomposition the whole-model budget sums. *Right*: what a clock edge costs, measured on one compiled neuron of the trained model with OpenROAD and OpenSTA after clock-tree synthesis. Deepening the pipeline buys clock period until the registers themselves become the path.

The rate this establishes. One compiled neuron of a real trained model, pipelined eight deep, closes at a measured T_{clk} in this library and emits one result per clock thereafter, at a measured area and a measured energy per result—all in Table 6. That is a token rate for a block, not for the model: pipelining the whole step means registering a datapath of the depth Figure 17 plots, and the register count grows with the number of wires crossing each cut. Table 6 shows the trade turning against itself within a single block, where sixteen stages cost several times the combinational area for a clock period that has stopped improving, so a whole-model pipeline is a design problem—where to cut, and how wide the cuts are—rather than a parameter sweep. It is named as an open problem in Section 19. What this section measures is what a pipeline *costs* on real blocks.

15 Removing the Float Remainder

One block stands between this model and an all-integer circuit: LayerNorm. Priced in 16-bit fixed point it needs d squares, hence d multipliers, plus a reciprocal square root and a per-channel multiply–add, and the two per block come to 200% of the feed-forward matmul they wrap. That single block is why 43% of the gate budget is not the exact-integer datapath, and why the device question came out the way it did. This section removes it.

The price of the block being replaced. The 665,373-gate figure for one d=128 LayerNorm is composed *arithmetically* from primitive costs. Building it as an actual netlist from the same primitives gives 844,719 gates—27% more, because arithmetic composition under-counts the width growth in the accumulation trees. The built figure is the one we compare against from here on, so the replacement is measured against the larger and less favourable baseline.

Shift-normalization. A normalizer does two things a transformer depends on: it removes the mean, and it divides by a measure of scale. Neither needs a multiplier. Our integer normalizer builds

$$
\begin{aligned}
\mu &= \tfrac{1}{d}\textstyle\sum_j x_j && d \text{ a power of two: a } \textit{wired} \text{ shift, zero gates}\\
c_j &= x_j - \mu && d \text{ adders}\\
\sigma_* &= \tfrac{1}{d}\textstyle\sum_j |c_j| && \text{L1: abs + one adder tree, } \textit{no multiplier}\\
\text{or}\quad \sigma_* &= \tfrac{1}{d}\textstyle\sum_j c_j^2 && \text{L2: } d \text{ squarers, the exact variance}\\
e &= \lfloor \log_2 \sigma_* \rfloor && \text{a leading-one detector (priority encoder)}\\
y_j &= \big(c_j \ll K\big) \gg e && \text{one barrel shifter: } \textit{no divider}\\
y_j &\leftarrow y_j \cdot r/2^M, \quad r = \mathrm{ROM}_M(\sigma_*\text{'s mantissa}) && M \textit{ conditional adds}\text{: no general multiplier}\\
\mathrm{out}_j &= \gamma_j\, y_j + \beta_j && \gamma_j, \beta_j \text{ constants: signed-digit shift–add}
\end{aligned}
\tag{6}
$$

The one place a scale factor must be recovered rather than shifted away is the mantissa of σ_*, and we recover it without a multiplier: writing $\sigma_* = 2^e(1+f)$, the top M bits of f (read off a left-normalized word) address a 2^M-entry ROM holding $r \approx 2^M/(1+f)$, and the product $y \cdot r/2^M$ is evaluated as M conditional adds of wired shifts of y, gated by r's bits. The L2 variant reads a reciprocal-*square-root* table instead and halves the exponent by a wired shift, which is exact because $\lfloor \log_2 \sqrt{v} \rfloor = \lfloor e_v/2 \rfloor$ with the discarded parity bit folded into the table's address.

The sign specialization. In the all-binary circuit this paper budgets, *every* normalizer's consumer is a sign: the feed-forward layer's first operation is $\mathrm{sign}(x)$, the $Q/K/V/O$ projections binarize, and the head binarizes. For such a consumer the normalization disappears entirely, because for $\gamma_j > 0$

$$
\big[\, \gamma_j\, c_j\, 2^{K-e} + \beta_j \;\geq\; 0 \,\big] \;=\; \big[\, c_j \;\geq\; t_j\, 2^{\,e-K} \,\big], \qquad t_j = -\beta_j/\gamma_j, \tag{7}
$$

and $t_j 2^{e-K}$ is a *constant* selected by the shared exponent e: a sixteen-entry ROM per channel followed by one signed comparator. No barrel shifter, no divider, no multiplier anywhere. At d=128 in 16-bit fixed point this block is 62,669 gates and 70 levels deep—13.5× smaller and 9× shallower than the fixed-point LayerNorm it replaces, emitting the one bit per channel that the next binarized layer consumes.

Correctness. Each of these is a netlist, and each is checked against an independently written integer reference on deterministic corner cases plus seeded random vectors at every width we build, and re-checked against the reference at the widths used in the accuracy experiment. The *accuracy* experiment additionally feeds real activation vectors, captured from the trained model, through the actual gate netlist and requires bit-exact agreement with the arithmetic used to compute bits per token—so the accuracy numbers below describe the circuit whose gates we counted, and the script exits nonzero if they do not.

15.1 Gates *and* bits per token

Table 7 is the trade. Bits per token are measured on the real trained checkpoint over fixed held-out batches, identical for every variant, with a paired bootstrap interval on the difference.

Table 7: The normalizer, both axes [MEASURED]. Gates and depth are exact counts of netlists built at d=128 in 16-bit fixed point. "Mult." counts *general* $W \times W$ multipliers. Bits per token are held-out, on the d=128/L=4 checkpoint, over batches identical across variants; the interval is a paired bootstrap on the per-batch difference from the float baseline. "Sign agree." is the fraction of sign decisions—the bit a binarized consumer actually reads—that are unchanged, measured at matched inputs. **This is a drop-in swap into a model trained with float LayerNorm: it is an upper bound on the harm, not the number a model trained with shift-norm would reach.**

Normalizer	Gates	Depth	Mult.	bits/token	Δ vs. float	sign agree.
float LayerNorm (as trained)	—	—	—	0.7404	—	—
fixed-point LayerNorm W=16	844,719	634	384	0.7401	−0.0003 [-0.0006,+0.0001]	99.99%
L1 shift-norm M=0 (power of two)	138,163	240	0	2.1573	+1.4169 [+1.2439,+1.5897]	98.77%
L1 shift-norm M=4 (recip ROM)	225,011	260	0	1.1637	+0.4233 [+0.3570,+0.4863]	99.22%
L1 shift-norm M=6 (recip ROM)	266,665	310	0	1.1387	+0.3983 [+0.3107,+0.4887]	99.19%
L2 shift-norm M=4 (rsqrt ROM)	527,366	385	128	0.7948	+0.0544 [+0.0419,+0.0662]	99.53%
L2 shift-norm M=6 (rsqrt ROM)	569,566	483	128	0.7417	+0.0013 [-0.0005,+0.0031]	99.48%
shift-norm, sign consumer only	62,669	70	0	*emits one bit per channel; see text*		

The table says four things.

The fixed-point LayerNorm *the paper prices is numerically free and physically enormous.* It costs −0.0003 bits per token, an interval straddling zero, and 844,719 gates with 384 multipliers.

The L2 shift-norm is free too, for a third of the multipliers. At M=6 it costs +0.0013 bits per token with a 95% interval of $[-0.0005, +0.0031]$—an interval that *includes zero*, so at this sample size we cannot distinguish it from the model as trained—for 569,566 gates and 128 multipliers rather than 384. It keeps LayerNorm's own variance and removes only the divider, the reciprocal square root and the $2d$ per-channel affine multiplies. This is the strongest result of the section: the reciprocal square root, the divider and the affine multiplies of a LayerNorm are *not needed*, and dropping them costs nothing we can measure. It is *not multiplier-free*: it keeps the d squarers of the variance, and is described here as the L2 variant throughout.

The fully multiplier-free L1 shift-norm is cheap, and as a drop-in it is accurate where the circuit needs it and inaccurate where the trained checkpoint needs it. At M=6 it is 266,665 gates, 3.2× smaller than fixed-point LayerNorm, with *zero* general multipliers. It costs +0.398 bits per token on a 0.740-bit baseline. The sign-agreement column says something different: 99.19% of the sign decisions a binarized consumer would read are unchanged. Those two facts are consistent, and the reason matters. In *this checkpoint* only one of the three normalizers per block feeds a binarized layer; the other two feed a full-precision attention and a full-precision head, where an absolute scale error propagates directly into the logits. In the all-binary circuit this paper budgets, every normalizer feeds a sign, and the quantity that matters is the one in the last column. Both are reported, and each result rests on the column that matches its circuit.

The mechanism of the L1 penalty is identifiable, and it is a modelling assumption rather than a defect of the circuit. Replacing the L2 standard deviation with an L1 dispersion is exact only for a fixed distribution shape, and this model's activations are heavy-tailed enough that the ratio drifts from token to token. A model *trained* with an L1 normalizer would not face that mismatch; that is the experiment pre-registered, with both a success and a failure criterion, in Section 19.

Calibration, and why the sign column is sensitive to it. The single calibration scalar folded into γ must be fitted against a target that respects the sign of γ: 32.8% of this checkpoint's LayerNorm gains are *negative*, and a fit that clamps the denominator to a positive floor calibrates a third of the channels against a target with the wrong sign, which moves the measured sign agreement from 99.2% down to 86% without changing a single gate. The numbers reported here use the sign-respecting fit, and the accuracy harness feeds real activation vectors through

the actual gate netlist so that the reported figures describe the circuit whose gates we counted.

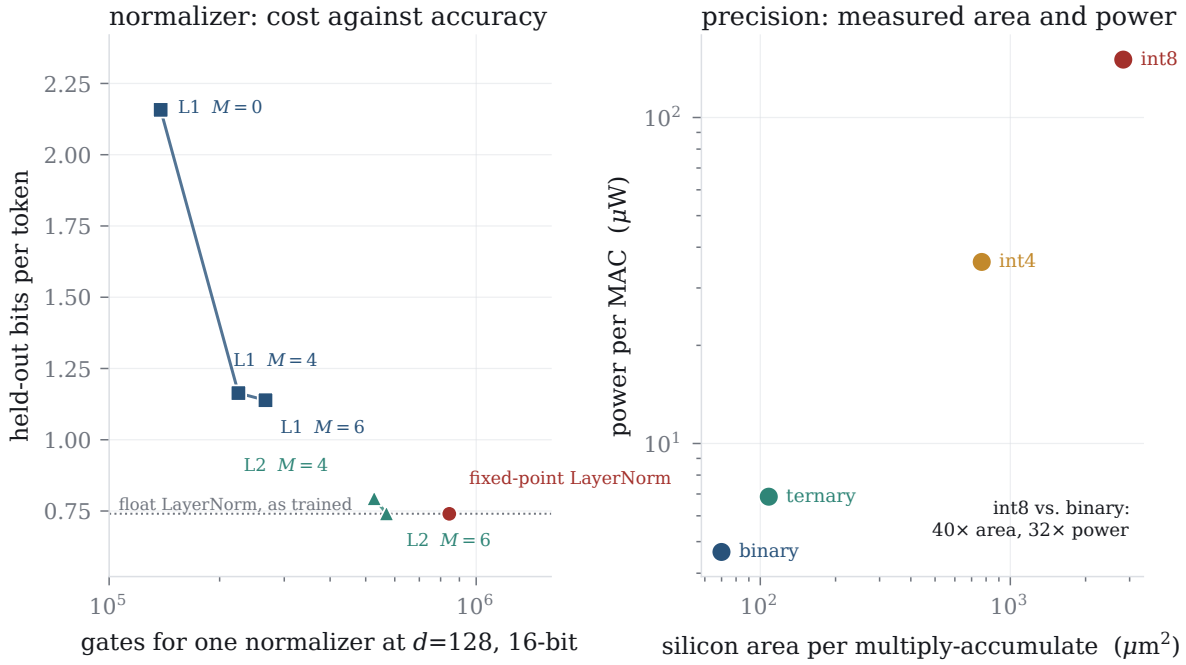


Figure 18: [MEASURED] Cost against accuracy, on both axes at once. *Left*: gates for one normalizer at d=128 in 16-bit fixed point against held-out bits per token on the trained d=128/L=4 checkpoint over paired batches; the dotted line is the model as trained, and each point's general-multiplier count and sign-agreement fraction are the corresponding columns of Table 7. This is a drop-in swap—no model was retrained—so it is an upper bound on the harm. *Right*: placed silicon area and power per multiply–accumulate for the precision family, each block verified against its integer specification and then placed and routed in `sky130_fd_sc_hd` at `tt_025C_1v80` with the switching activity measured on the netlist. Neither panel is a Pareto *front* in the sense a design decision needs, because the accuracy of a model *trained* at each operating point is experiment G3.

15.2 The whole model, re-budgeted

Given a normalizer that does not need a multiplier, and given that the per-output scale α of Section 11 is a positive constant that can be rounded to a power of two—making it a wired shift and leaving only the bias—the whole-model budget can be recomputed with the float remainder removed. Table 8 does that, at the dimensions of the same real checkpoint.

Table 8: The whole model with the float remainder removed [MODELLED from MEASURED blocks]. Each row is one autoregressive step of the d=128, L=4 model. "Multipliers" counts general $W \times W$ multipliers in the whole step. LUT6 uses the measured per-block ABC packing ratios of Table 3; "Fits" asks whether that lookup-table count is inside a device from the vendor table (ASSERTED). **The last row assumes every normalizer feeds a** sign**, which is true of the all-binary circuit this budget describes and is not true of the trained checkpoint, whose attention and head are full precision.**

Configuration	Gates	Multipliers	Norm. (each)	LUT6	Fits
attention window = 128 keys					
fixed-point LayerNorm, float alpha (as published)	25,621,263	6,016	844,719	6,821,082	no device in the table
fixed-point LayerNorm, power-of-two alpha	21,484,303	3,456	844,719	5,671,926	no device in the table
L2 shift-norm M=6 (exact variance), POT alpha	19,007,926	1,152	569,566	4,984,044	no device in the table
L1 shift-norm M=6 (multiplier-free), POT alpha	16,281,817	0	266,665	4,226,791	no device in the table
sign-consumer shift-norm (all norms feed a sign), POT alpha	14,445,853	0	62,669	3,716,801	Virtex UltraScale+ VU19P (largest monolithic)
attention window = 512 keys					
fixed-point LayerNorm, float alpha (as published)	35,158,703	6,016	844,719	9,324,347	no device in the table
fixed-point LayerNorm, power-of-two alpha	31,021,743	3,456	844,719	8,175,192	no device in the table
L2 shift-norm M=6 (exact variance), POT alpha	28,545,366	1,152	569,566	7,487,309	no device in the table
L1 shift-norm M=6 (multiplier-free), POT alpha	25,819,257	0	266,665	6,730,057	no device in the table
sign-consumer shift-norm (all norms feed a sign), POT alpha	23,983,293	0	62,669	6,220,067	no device in the table

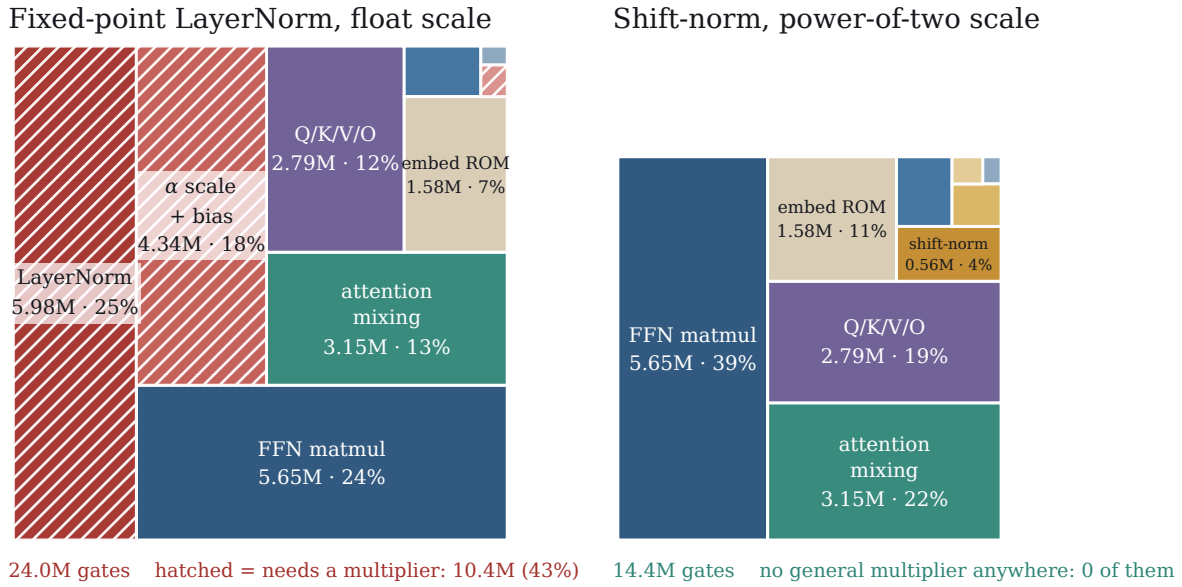


Figure 19: [MODELLED from MEASURED blocks] Where the gates are, and what removing the float remainder removes. Tile area is proportional to gates and *the two panels share one area scale*, so the right-hand model is smaller on the page by exactly the fraction of gates the replacements remove. *Left*: the budget with a fixed-point LAYERNORM and a floating-point output scale; the hatched tiles are the terms that are not integer logic and need a fixed-point multiplier, and LAYERNORM alone is larger than the feed-forward matmul it wraps. *Right*: the same model with the normalizer replaced by the multiplier-free shift-norm of Eq. (6) and the per-output scale rounded to a power of two. The accuracy cost of those replacements is measured separately, in Table 7 and Figure 18.

Which model this describes. A model of this shape *can* be all-integer. With shift-normalization and a power-of-two output scale there is no general multiplier anywhere in one autoregressive step, and the step is 40% smaller than the budget Section 12 reports. That is the substantial design result of this part, and it converts the open item Section 11 identified into a costed circuit.

Two different models are being talked about, and the distinction decides the device question. Table 8's last row describes a *redesigned* model, costed and not yet trained: if every normalizer were the sign-consumer form of Eq. (7) and every output scale a power of two, the step would map to 3.7×10^6 LUT6 by the measured packing ratios, below the VU19P's 4.1×10^6. The trained d=128 checkpoint this paper reports—float norms, float scales, a float residual stream—needs 6.4×10^6 LUT6 and does not fit the largest monolithic device made. The fidelity of the redesign is the sign-agreement figure of Table 7, 99.2% for the multiplier-free variant: high, and a per-decision agreement rate rather than a language-modelling result, which is why training a model with these primitives is the pre-registered next experiment. The precise reading of that row is that *the arithmetic of an all-integer binary model of this size fits one very large device*—the same shape of claim Section 12 makes about the exact-integer datapath, now extended to the terms that datapath excluded.

16 Extending the Proofs

Table 10 proves sixteen blocks equivalent to independent behavioural models over their entire input spaces at a 200-second solver budget. Two things extend that frontier here: a larger budget on the widest blocks, and a proof of *every* neuron of a real trained layer rather than a sample of them. Table 9 reports what is now proved.

Table 9: Proofs added in this pass [MEASURED: YOSYS 0.68, `miter -equiv` + `sat -verify -prove`, and YOSYS `aigmap` → ABC `cec` on the wide blocks]. The pipeline rows are *sequential* equivalences, discharged on the unrolled circuit; because the pipeline is feed-forward with $S-1$ ranks of state, a bound of $S+2$ cycles is complete rather than partial. The negative-control rows are required to FAIL: a proof harness that cannot detect a deliberately wrong design is plumbing, not evidence.

Object proved	Input space	Gates	Solver (s)	Verdict
signed_dot_n64	2^{128}	709	24	PROVED
popcount_n128	2^{256}	1,363	85	PROVED
popcount_n256	2^{512}	2,766	1933	PROVED
argmax_idx_K64_w10	2^{640}	9,195	6	PROVED
argmax_tree_K64_w10	2^{640}	7,686	3752	PROVED
popcount_n512	2^{1024}	5,577	439	TIMEOUT
hard_attn_d64_k16_dv32	2^{1600}	12,906	463	TIMEOUT
hard_attn_tree_d64_k16_dv32	2^{1600}	12,928	467	TIMEOUT
argmax_tree_K258_w10	2^{2580}	34,438	9013	TIMEOUT
every neuron of a trained layer (512 of 512), sign bit	2^{128} each	≈1,300	61 med.	PROVED 512/512
32 of them, full popcount value	2^{128} each	≈1,300	—	PROVED 32/32
per-neuron negative control (threshold +1)	2^{128} each	—	—	FAILED 8/8 (as required)
negative control (one output bit inverted)	—	—	—	FAILED 3/3 (as required)
bnn_dot_n16 pipeline, $S=1$ vs. combinational	all inputs, all time	0 FF	—	PROVED
bnn_dot_n16 pipeline, $S=2$ vs. combinational	all inputs, all time	12 FF	—	PROVED
bnn_dot_n16 pipeline, $S=4$ vs. combinational	all inputs, all time	35 FF	—	PROVED
bnn_dot_n16 pipeline, $S=8$ vs. combinational	all inputs, all time	83 FF	—	PROVED

Every neuron of a trained layer, proved. The middle block of that table is the result we care most about, and it rests on a control. Eight neurons were deliberately corrupted—threshold off by one, a single-bit difference in one output—and the harness reports FAILED on **8/8** of them. A prover that cannot fail proves nothing; this one fails exactly when it should, so a PROVED verdict from it is evidence.

On that footing: compiling all 512 neurons of a real trained layer with their weights baked in and diffing each against the layer on 60 random vectors is 30,720 checks—and a *sample* from a space of size 2^{128} per neuron. We replace that sample with a proof, for every neuron. For each one we build a miter against a behavioural model whose weight mask and threshold are read

from the checkpoint, and ask whether *any* of the 2^{128} inputs can distinguish them. **512/512 neurons are proved** on the sign bit—the bit the next layer consumes—each over the whole 2^{128} input space of that neuron, at a median of 61 seconds each and 1.9 hours of wall time on sixteen shared cores. This is exhaustive per neuron: not a sample of neurons, and not a sample of inputs. The claim "the netlist *is* this trained layer" rests on a proof for every neuron of the layer.

Two further statements are separate results. First, the full popcount *value*—the entire integer the neuron computes, not only its sign—is proved for **32/32** of the neurons attempted, over 2^{128} each. That is a stronger property on a smaller set, and the 512/512 above should not be read as covering it. Second, one *monolithic* query—all 512 outputs proved simultaneously as a single Boolean function $\mathbb{B}^{128} \to \mathbb{B}^{512}$, which shares structure across the cones and would be a stronger statement still—does not close in the budget we gave it, nor does the same construction in groups of 32 outputs. That is a solver-capacity limit, and the per-neuron result does not depend on it: 512 proofs, each over the same 2^{128} space that the single query would have covered, already establish the equivalence for every output of the layer. The monolithic query remains open.

The wide blocks, at a larger budget. Raising the per-block budget from about seven minutes to two and a half hours closes the 64-way arg-max index (tree) over 2^{640} inputs in 63 minutes and the n=256 population count over 2^{512} inputs in 32 minutes. 4 of the 9 wide blocks—the n=512 population count (2^{1024}), the k=16 hard-attention head (fold) (2^{1600}), the k=16 hard-attention head (tree) (2^{1600}), the 258-way token selector (2^{2580})—did not close within 2.5 hours of solver time each, so their evidence in Table 2 is a labelled sample. This is a capacity limit of the solver on this machine, not a property of the designs: the same engine, on the same machine, proves a 2^{640} block in seconds and a 2^{512} one in half an hour. The boundary is where the search runs out of time, and it moves with the budget—two of the blocks named here as open were on the other side of it before this pass.

The engine that makes this reachable. The binding constraint is the solver, not the statement. YOSYS's own `sat -prove` on a monolithic miter does not finish an n=128 population count in forty minutes, while ABC's combinational equivalence checker—which reduces both designs to And-Inverter Graphs and *fraigs* them before calling SAT—proves the same statement in 75 seconds. Both are complete decision procedures over the entire input space, so this is a choice of speed and not of evidential standard, and the negative control is re-run against the engine actually used, which correctly reports FAILED on every deliberately broken design (Table 9). The widest thing it reaches is worth naming: the 64-way arg-max *index*—the one-hot fold that tracks which key wins—is proved equivalent to `numpy.argmax`'s tie-breaking over all 2^{640} inputs in 6.5 seconds, where Table 10's widest arg-max was K=16 over 2^{112}.

Part IV
Verification and Limits

17 Proofs: Netlists by SAT, Specifications in Lean

Simulation checks a circuit on the inputs one thinks to try; a proof checks it on all of them. We pursue this at two levels, and it matters which is which.

At the level of the *netlist*, we now have machine proofs: YOSYS builds a miter between an emitted netlist and an independently written behavioural model and its SAT engine shows no input can distinguish them. This covers the adder, the popcount up to the widths the solver

reaches, the arg-max blocks and the attention heads, over their entire input spaces rather than a sample. The harness is validated by a negative control—inverting one output bit makes every case fail—so a pass is evidence rather than plumbing. Separately, all eighteen emitted Verilog modules are co-simulated against the Python netlist under Icarus Verilog and agree on every vector applied, which closes the one gap a gate-level check cannot: a Verilog printer can be wrong without changing a single number in a netlist count.

Table 10: Combinational equivalence by SAT [MEASURED: YOSYS 0.68, `miter -equiv` + `sat -verify -prove`]. Each emitted netlist is checked against a behavioural Verilog model written independently from the specification. A PROVED row means no input vector in the stated space can distinguish them—not a sample. The harness is validated by a negative control: inverting a single output bit of the implementation turns every row to FAILED.

Block	Input space	Gates	Verdict
Ripple adder n=4, 8, 16	$2^8, 2^{16}, 2^{32}$	20, 40, 80	PROVED
XNOR-popcount n=8	2^{16}	63	PROVED
XNOR-popcount n=16	2^{32}	146	PROVED
XNOR-popcount n=32	2^{64}	317	PROVED
XNOR-popcount n=64	2^{128}	664	PROVED (34 s)
Signed dot n=64	2^{128}	709	PROVED (73 s)
Arg-max value K=8	2^{48}	462	PROVED
Arg-max index K=8 (fold)	2^{48}	506	PROVED
Arg-max index K=8 (tree)	2^{48}	490	PROVED
Arg-max index K=16 (fold)	2^{112}	1,318	PROVED
Arg-max index K=16 (tree)	2^{112}	1,260	PROVED
exp ROM $7 \to 12$	2^7	684	PROVED
Hard attention d=8, k=4	2^{56}	421	PROVED
Hard attention d=16, k=8	2^{208}	1,708	PROVED

All sixteen blocks attempted are proved, at a 200-second per-block solver budget; the two 2^{128} cases sit at 19 and 73 seconds, so the budget is the binding constraint rather than the method, which is why Section 16 raises it and reports how much further the frontier then goes.

At the level of the *specification*, we formalize in the Lean 4 proof assistant the arithmetic identities the compiler targets. Six theorems compile with no `sorry` and no added axioms (`#print axioms` reports only Lean's standard `propext`, `Classical.choice` and `Quot.sound`; the full adder depends on none). We prove `full_adder_correct`—that the gate definitions satisfy $s + 2c_{\text{out}} = a + b + c_{\text{in}}$ for all inputs; `ripple_adder_correct`, its n-bit extension by induction; `xnor_popcount_fin`, the identity at the heart of the paper, that the signed inner product of two $\{-1, +1\}$ vectors equals $2\,\text{popcount}(\text{XNOR}) - n$ (Eq. (3)); `argmax_correct`, that the comparator-tree maximum returns a list member dominating every element; and—the formal statement of the paper's thesis—`circuit_complete`, that *every* total Boolean function $f : (\text{Fin}\, n \to \text{Bool}) \to \text{Bool}$ is realized by a circuit over AND/OR/NOT, proved by Shannon expansion (Eq. (1)), with a vector-valued corollary `circuit_complete_vec`. The development is 274 lines, builds against `mathlib`, and its compiled artifact ships with the paper.

What the Lean development covers. Lean carries one half of the correctness argument. Lean's `Circuit` inductive type and its evaluator are a *separate datatype* from the Python compiler's: there is no extraction, no code generation and no refinement proof connecting them, so nothing in Lean constrains what the compiler emits, and nothing in Lean mentions the popcount adder tree or the Verilog emitter. The theorem `argmax_correct` shows that `treeMax` returns a list member dominating every element, for a reduction written as a left fold, and says nothing about the *index* the head must emit. `circuit_complete` is Shannon expansion over an abstract inductive type: a textbook result, and the formal statement of this paper's thesis. The

division of labour is clean and deliberate: Lean certifies the arithmetic the compiler *targets*, and the SAT and combinational-equivalence engines certify that particular emitted netlists *hit* that target, over entire input spaces. Joining the two—proving the emitter correct once rather than checking its outputs one netlist at a time—is the next step, and is named in Section 19.

18 Validation

Every quantitative claim in this work is backed by a machine-readable receipt, written by the run that produced it and shipped with the paper: technology mapping with the full tool transcripts, the SAT and combinational-equivalence proofs with their transcripts, the Icarus Verilog co-simulation, the 512-neuron layer check and its fixed-point pricing, the scaling re-derivation, the precision cost curve, place-and-route with timing and power for every block and every placed design as a DEF, the measured switching activity, the register-balanced pipelines with their flip-flop counts and sequential-equivalence proofs, the normalizer's gate cost and its held-out bits per token with paired bootstrap intervals, the all-integer re-budget, and the per-neuron proofs with their negative control. Every experiment is a script that runs on CPU alone, and one driver reruns all of them end to end.

Three properties make that machinery a check rather than a claim. The checkpoint is pinned: every artifact script takes it explicitly and defaults to the one this paper reports, so a worked example cannot silently follow whichever model was written most recently. The figures are computed at draw time from the compiler, a training log or a receipt, and the figure module carries a provenance check that fails if a hardcoded data series is introduced. And every table added in the physical pass is *generated* from the receipts and included into this source, so a table cannot drift from the experiment that produced it, while the hand-written prose numbers are checked against their receipts by a script that fails loudly when the two disagree.

19 Open Problems

Attention (Section 10) is not a wall: hard, arg-max attention is an exact circuit and fixed-point soft-max is a finite Boolean function, at a cost that is superlinear in the key count. The residue is the quantitative question of how coarse a fixed-point soft-max a trained model tolerates before its outputs drift.

Binarization without loss is demonstrated for the feed-forward *matmul* and the head, and not for the sub-layer as a whole: the per-output scale, the bias, the residual stream and LayerNorm remain full precision, and priced in fixed point the norms are larger than the matmul they wrap (Section 11). Section 15 costs the replacement and shows the redesign is all-integer; extending *native training* to the norms, the attention projections and the embedding is the substantial open item. Whether size helps at all under a controlled budget is open too: no effect is observable at matched budget, and resolving it needs an iso-token study with multiple seeds.

The four open problems. First, and most important, *train* a model with the shift-normalizer of Eq. (6) rather than swapping it in. Section 15 measures the drop-in cost of that swap on a model trained with float LayerNorm—an upper bound on the harm—and for the fully multiplier-free L1 variant that bound is +0.398 bits per token even though 99.2% of the sign decisions a binarized consumer reads are unchanged, the penalty coming from the normalizers that feed this checkpoint's full-precision attention and head, which the all-binary circuit does not have. Whether a model trained *with* an L1 dispersion recovers is the question the whole all-integer result turns on; it is pre-registered as experiment G7, with the criterion that a shift-norm model match a LayerNorm model of the same size at the same token budget to within 0.02 bits per token over three seeds. Second, pipeline the *whole* step rather

than individual blocks: Table 6 shows the register cost turning against itself within one block at sixteen stages, so the question is where to cut a datapath of the depth Figure 17 plots, and that is a design problem rather than a parameter sweep. Third, close the physical flow to a DRC-clean, detail-routed layout with a power grid on at least one block; we run global routing and post-route parasitic extraction, and detailed routing on the smaller blocks, but no design here has been signed off. Fourth, build the accuracy axis of the precision trade of Figure 13, so the area and power advantages measured in Part III can be stated as a Pareto result rather than a cost ratio, and join the two halves of the correctness argument by proving the Verilog emitter correct rather than checking its outputs one netlist at a time. Each of these is a runnable script with a pre-registered success criterion.

Why 130 nm. The node is not the proposal. SKY130 is the only open, fully documented flow in which every number here can be reproduced by a reader without a licence, and it is used for ratios rather than for absolutes. A commercial flow on a modern node would change every absolute figure in Part III and, we expect, few of the ratios; establishing that needs a vendor toolchain, and is queued as such.

There is, finally, a connection to the companion general-purpose neural computer. That work learns an interpreter of a real instruction set and runs it deterministically; this work turns a learned model into a circuit. Their union is a neural system whose learned behaviour is not merely deterministic and auditable but *physical*—compiled past the processor entirely, into gates.

20 Conclusion

A model that is binary is not condemned to be interpreted. Its arithmetic is Boolean, its Boolean functions are circuits, and we have compiled those circuits, counted their gates, mapped them with a real synthesis tool, simulated the hardware description they emit, and proved several of them equivalent to their specifications over their entire input spaces: a full adder of five gates, an inner product of nine to eleven gates per bit, a whole next-token head of 1.47 million gates and about 0.41 million lookup tables.

Then we built it in silicon. Every block goes through an open physical-design flow in a characterised 130 nm standard-cell library: the 258-way token selector of a real head is $0.093\,\mathrm{mm}^2$ and 14.55 ns; a gate our compiler emits costs a median $5.0\,\mu\mathrm{m}^2$ of placed silicon; the same fan-in-64 inner product built from int8 array multipliers occupies $40.5\times$ the placed area of the binarized one, draws $32.4\times$ the power at matched switching activity, and is $3.0\times$ slower; and a compiled neuron of the trained model, pipelined and clock-tree synthesised, closes at a measured period and emits one result per clock at a measured energy per result. The one term that was not integer logic—the normalizer—is replaced by a multiplier-free shift-normalizer that removes every general multiplier from an autoregressive step and shrinks it by 40%, at a drop-in accuracy cost we measure rather than assume: indistinguishable from zero for the variant that keeps the exact variance, and +0.40 bits per token for the entirely multiplier-free variant, which preserves 99.2% of the sign decisions a binarized consumer actually reads.

Two measurements bound the result precisely. The trained d=128 checkpoint needs 6.4 million lookup tables and does not fit the largest monolithic field-programmable gate array made, while its exact-integer datapath, at 3.48×10^6 LUT6, fits one comfortably—so the device question and the compilation question are the same question, and Section 15 answers it for a redesigned model that has been costed and not yet trained. And no number here is a measurement on fabricated silicon: these are what a physical-design tool computed for designs it placed, in one library at one corner, used as ratios.

The result is broad and it is measured: the arithmetic that dominates a binary language model compiles to a netlist that is proved equivalent to its specification over entire input spaces,

maps to standard cells a tool will place and route, and costs roughly 40× less silicon and 32× less power than the same arithmetic in int8. A small binary language model is, to a degree ordinary models are not, a machine one could build.

References

[1] C. E. Shannon. *A Symbolic Analysis of Relay and Switching Circuits.* MIT M.Sc. thesis, 1937.

[2] M. Courbariaux, Y. Bengio, J.-P. David. *BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations.* NeurIPS, 2015. arXiv:1511.00363.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio. *Binarized Neural Networks.* NeurIPS, 2016. arXiv:1602.02830.

[4] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi. *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks.* ECCV, 2016. arXiv:1603.05279.

[5] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou. *DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients.* 2016. arXiv:1606.06160.

[6] F. Li, B. Zhang, B. Liu. *Ternary Weight Networks.* 2016. arXiv:1605.04711.

[7] Y. Bengio, N. Léonard, A. Courville. *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.* 2013. arXiv:1308.3432.

[8] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, F. Wei. *BitNet: Scaling 1-bit Transformers for Large Language Models.* 2023. arXiv:2310.11453.

[9] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, F. Wei. *The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits.* 2024. arXiv:2402.17764.

[10] Y. Umuroglu, Y. Akhauri, N. J. Fraser, M. Blott. *LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications.* FPL, 2020. arXiv:2004.03021.

[11] E. Wang, J. J. Davis, P. Y. K. Cheung, G. A. Constantinides. *LUTNet: Rethinking Inference in FPGA Soft Logic.* FCCM, 2019. arXiv:1904.00938.

[12] M. Andronic, G. A. Constantinides. *PolyLUT: Learning Piecewise Polynomials for Ultra-Low Latency FPGA LUT-based Inference.* ICFPT, 2023. arXiv:2309.02334.

[13] M. Nazemi, G. Pasandi, M. Pedram. *NullaNet: Training Deep Neural Networks for Reduced-Memory-Access Inference.* 2018. arXiv:1807.08716.

[14] O.-C. Granmo. *The Tsetlin Machine.* 2018. arXiv:1804.01508.

[15] F. Petersen, C. Borgelt, H. Kuehne, O. Deussen. *Deep Differentiable Logic Gate Networks.* NeurIPS, 2022. arXiv:2210.08277.

[16] F. Petersen, H. Kuehne, C. Borgelt, J. Welzel, S. Ermon. *Convolutional Differentiable Logic Gate Networks.* NeurIPS, 2024. arXiv:2411.04732.

[17] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, K. Vissers. *FINN: A Framework for Fast, Scalable Binarized Neural Network Inference.* ACM/SIGDA FPGA, 2017. arXiv:1612.07119.

[18] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'Brien, Y. Umuroglu, M. Leeser, K. Vissers. *FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks.* ACM TRETS 11(3), 2018. arXiv:1809.04570.

[19] J. Duarte et al. *Fast Inference of Deep Neural Networks in FPGAs for Particle Physics.* JINST 13(07):P07027, 2018. arXiv:1804.06913.

[20] A. Pappalardo. *Brevitas: Neural Network Quantization in PyTorch.* Software, Xilinx/AMD, 2023. https://github.com/Xilinx/brevitas.

[21] R. E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Trans. Computers C-35(8):677–691, 1986.

[22] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer, 1984.

[23] A. Kuehlmann, V. Paruthi, F. Krohm, M. K. Ganai. *Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification.* IEEE TCAD 21(12):1377–1394, 2002.

[24] R. Brayton, A. Mishchenko. *ABC: An Academic Industrial-Strength Verification Tool.* CAV, 2010.

[25] C. S. Wallace. *A Suggestion for a Fast Multiplier.* IEEE Trans. Electronic Computers EC-13(1):14–17, 1964.

[26] T. Ajayi et al. *Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project.* DAC, 2019.

[27] R. T. Edwards and SkyWater Technology Foundry. *SkyWater SKY130 Open Source Process Design Kit.* 2020–. https://github.com/google/skywater-pdk (cell library `sky130_fd_sc_hd`, distribution `open_pdks`).

[28] M. Horowitz. *Computing's Energy Problem (and what we can do about it).* ISSCC, 2014.

[29] I. Parberry. *Circuit Complexity and Neural Networks.* MIT Press, 1994.

[30] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, T. Walsh. *Verifying Properties of Binarized Deep Neural Networks.* AAAI, 2018. arXiv:1709.06662.

[31] J. L. Pharr. *Large Binary Models.* Georgia Cyber Warfare Range, 2026.

[32] J. L. Pharr. *The General Purpose Neural Computer.* Georgia Cyber Warfare Range, 2026.