Binary Circuit Models

Compiling a Small Binary Language Model into a Logic Circuit

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

August 17, 2026

v0.0.1

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. The model is not software that a circuit interprets; the model *is* 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, verify each netlist bit-exactly against its arithmetic specification, and count its gates. Under binarization the inner product collapses to the celebrated XNOR-and-population-count, which we synthesize at roughly eleven gates per input bit; the entire next-token *head* of a real binary model of embedding width five hundred and twelve over a two-hundred-and- fifty-eight-symbol alphabet compiles to on the order of one and a half million verified gates—a single modest chip. 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, so its feed-forward datapath is a circuit read off its machine-verified sign-valued weights rather than an approximation of a floating-point model; and we discharge the core of the compiler in the Lean 4 proof assistant—six theorems with no `sorry`, including that the signed inner product equals $2\,\mathrm{popcount}(\mathrm{XNOR}) - n$ and that *every* Boolean function is a circuit, the formal statement of the thesis. We give the Boolean-algebraic foundations in full, draw the resulting circuits, emit synthesizable hardware description for the blocks, and lay out the path to silicon. The thesis is simple and, we think, striking: a binary model is a machine one can fabricate.

Contents

I Preliminaries **2**

1 The State of Today **2**

2 Related Work **3**
2.1 Binary and low-bit quantized neural networks 3
2.2 Neural networks as learned logic 3
2.3 FPGA/ASIC accelerators and quantization toolflows 3
2.4 Logic synthesis and Boolean-function representation 4
2.5 Finite-precision networks as Boolean circuits 4

3 Current Problems **4**

4 The Discipline of Circuit Compilation **4**

II Proposal **4**

5 Boolean Foundations **5**

6 The Reduction: a Model is a Boolean Function **6**

7 The Compiled Blocks **6**

8 The Whole Model as a Circuit **8**

9 Attention as a Circuit **9**

10 Native-Binary Training: the circuit is the model **10**

11 The Whole Model, and What It Would Cost as a Chip **11**

12 Formal Verification in Lean **12**

13 Validation **13**

14 Open Problems **13**

15 Conclusion **13**

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 [26] studies neural networks through the lens of circuit complexity, situating threshold units within bounded-depth circuit classes. On the verification side, Narodytska et al. [27] 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. The remainder of this work develops the foundations, exhibits the compiled and verified blocks, and estimates 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.

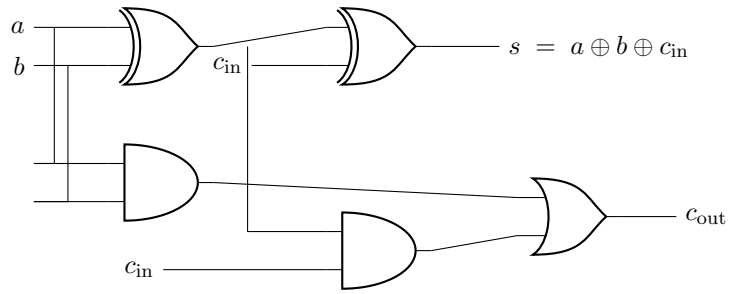


Figure 1: The full adder, the atom of arithmetic: sum $s = a \oplus b \oplus c_{\text{in}}$ and carry $c_{\text{out}} = (a \wedge b) \vee (c_{\text{in}} \wedge (a \oplus b))$. Two XOR, two AND, one OR; verified exhaustively on all eight inputs. A ripple of n full adders is an n-bit adder (40 gates at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

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.

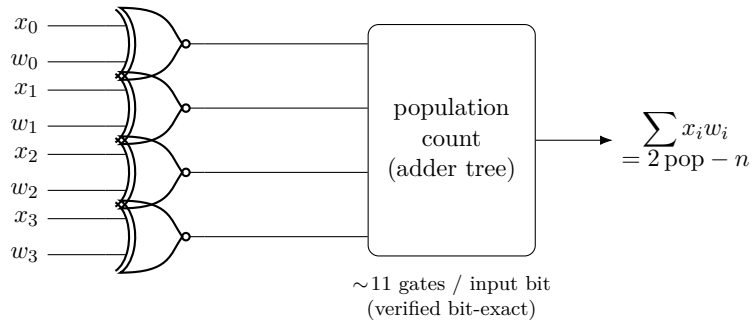


Figure 2: The binarized inner product (Eq. 3): an array of XNOR gates computes the agreement bits, a balanced adder tree of full adders counts them, and an affine relabel gives the signed dot product. Synthesized and verified against random inputs; the count grows at ≈ 11 gates per input bit (146 gates at $n = 16$, 5,577 at $n = 512$).

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: every block is bit-exact, and its gate count is exact. The full adder is verified on all eight of its inputs; the multi-bit blocks on random inputs; the equivalence is not approximate. The compiler additionally emits synthesizable Verilog for each block, so that the netlists are not diagrams but hardware.

Table 2: Compiled and verified gate-level blocks. Every "exact" is a bit-for-bit match to the integer specification; gate counts are exact.

Block	Function	Verification	Gates
Full adder	$a + b + c_{\text{in}}$	exhaustive 8/8	5
Ripple adder (n=8)	two's-complement add	500 random, exact	40
XNOR-popcount (n=64)	binarized inner product	500 random, exact	664
XNOR-popcount (n=512)	binarized inner product	random, exact	5,577
Arg-max (K=8)	index of maximum	300 random, exact	462

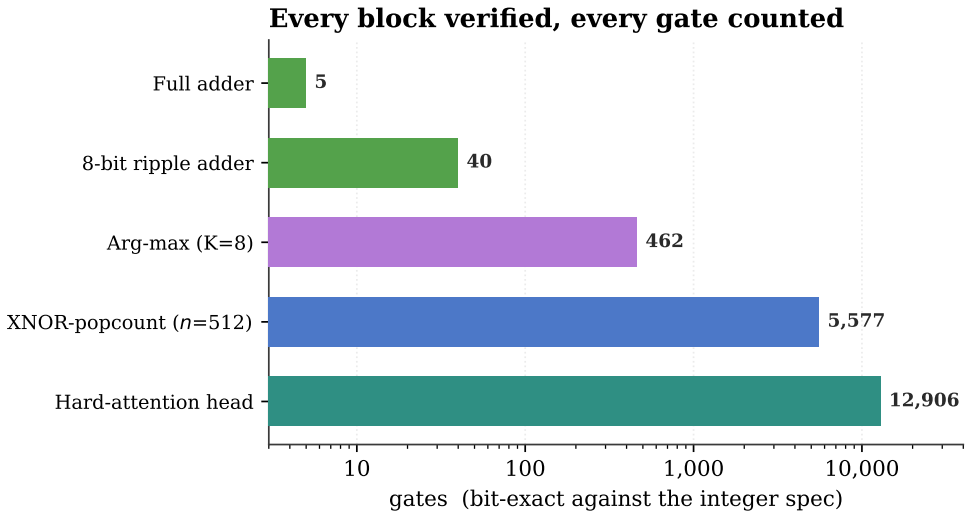


Figure 3: The verified block library at a glance (log scale). Each bar is a self-contained gate netlist whose output was checked bit-for-bit against its integer specification—exhaustively for the full adder, on random inputs for the wider blocks. Gate counts are exact, not estimated.

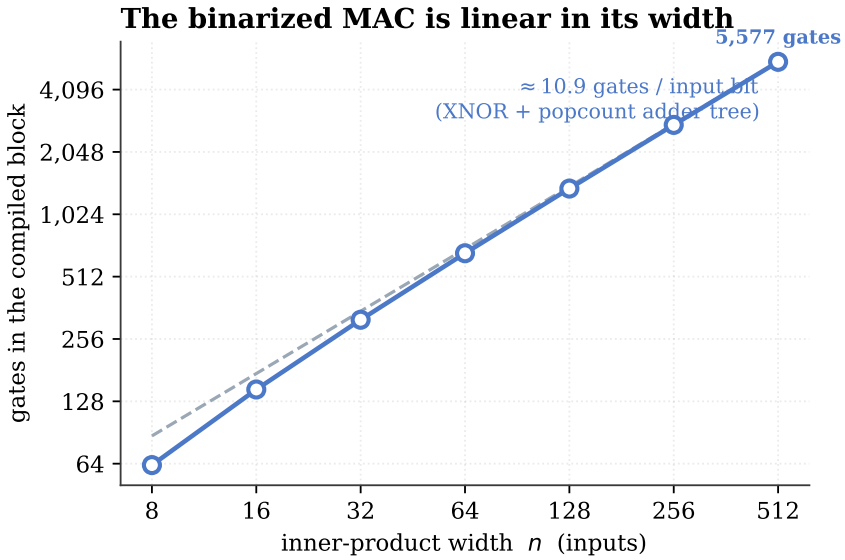


Figure 4: The cost of the binarized multiply–accumulate (Eq. (3)) grows *linearly* in the inner-product width: an XNOR per input bit plus a balanced popcount adder tree, about eleven gates per bit across three orders of magnitude of width. The dashed line is the exact-linear reference.

8 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:

$$G_{\text{head}} \;=\; V \cdot G_{\text{dot}}(D) \;+\; G_{\arg\max}(V) \;\approx\; 258 \cdot 5{,}577 \;+\; 28{,}270 \;\approx\; 1.47 \times 10^{6} \text{ gates}, \qquad (4)$$

about one and a half million gates; at $D = 768$ it is about 2.2 million. These are figures a single field-programmable gate array holds comfortably and a modest application-specific integrated circuit realizes with room to spare. The head of a binary language model is, quite literally, a chip.

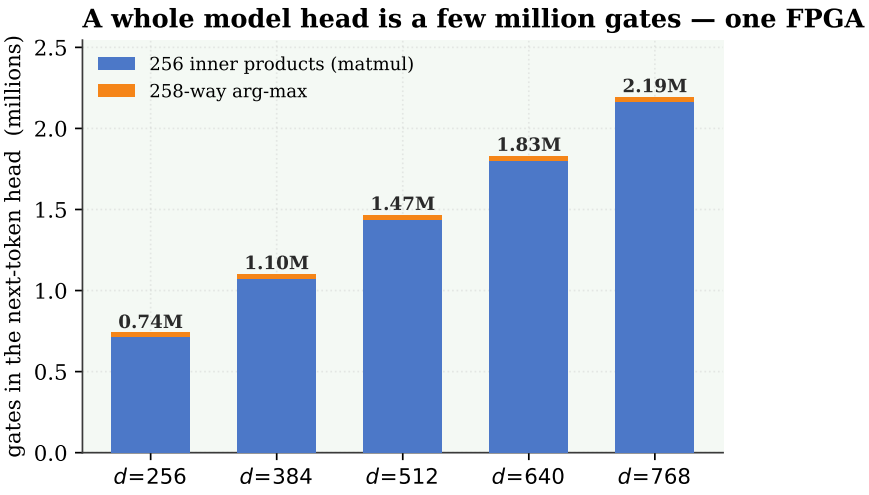


Figure 5: The gate budget of the whole next-token head across embedding widths, decomposed into the V binarized inner products (the matmul, which grows with D) and the V-way arg max token selection (nearly constant). From three-quarters of a million gates at D=256 to 2.19 million at D=768—all within the capacity of a single mid-range FPGA (shaded).

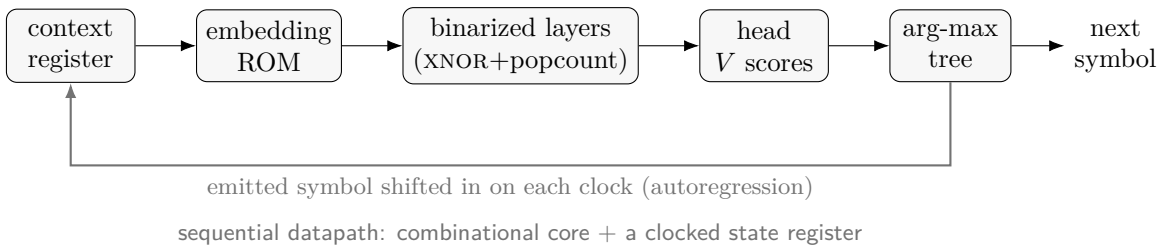


Figure 6: The inference datapath of a binary circuit model. The combinational core (embedding, binarized layers, head, arg max) computes the next symbol from the context in one pass of logic; a clocked register shifts the emitted symbol back into the context, turning the autoregressive loop into an ordinary finite-state machine. Because the alphabet is finite—258 symbols fit in nine bits—the input and output are digital with no conversion.

Two 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.

9 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 out $= V[\arg\max_j Q \cdot K_j]$. We verify this netlist bit-exactly against a reference $\arg\max$: at $n_{\text{keys}} = 16$ and head width $d = 64$ it is 300/300 exact and costs 12,906 gates, and its size grows linearly in both the head width and the number of keys ($\approx$800 gates per key at $d = 64$). 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. We do not claim continuous soft-max is bit-exact as a circuit; we claim that hard attention is exact 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.

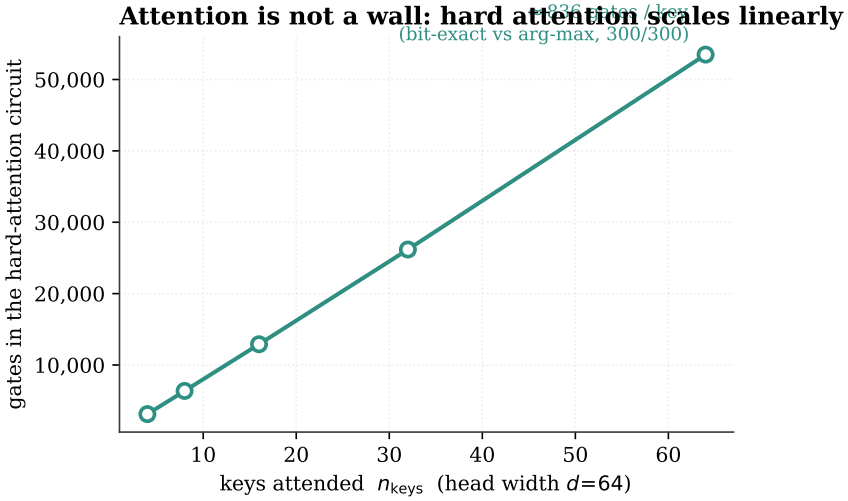


Figure 7: Hard attention compiled to gates: the circuit size grows *linearly* in the number of attended keys (about 800 gates per key at head width d=64), because each key contributes one XNOR-popcount score and one comparator/multiplexer stage. Every point was verified bit-exact against a reference $\arg\max$ (300/300). The continuous soft-max is not needed for the circuit to exist.

10 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), at a hundred-percent opcode validity, and we verify at the end of training that every feed-forward weight is genuinely sign-valued, so the "the circuit is the model" claim is machine-checked rather than asserted. Native binarization costs bits—the floating-point model of the companion work reaches 0.43—but it buys exactness: the feed-forward datapath of this model is a circuit one can read off its weights. Attention and the embedding remain full precision here, and their native binarization is the natural continuation.

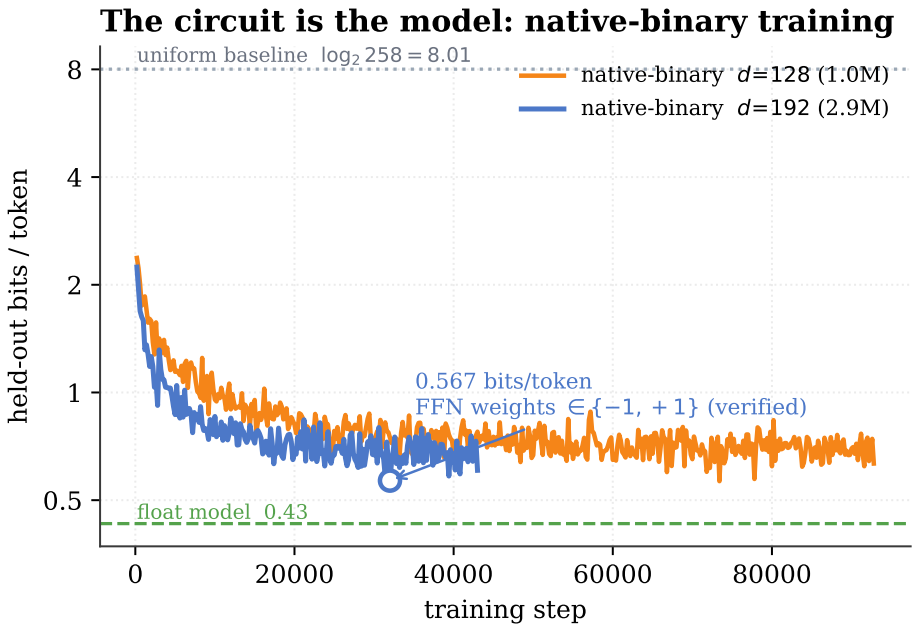


Figure 8: Native-binary training (log scale). Two small models whose feed-forward weights and activations are constrained to $\{-1, +1\}$ throughout training drive held-out bits-per-token down from the 8.01 uniform baseline to well under one bit; the larger reaches 0.57, verified at convergence to have every feed-forward weight sign-valued. The gap to the floating-point model (0.43, dashed) is the measured price of exactness: these weights *are* the circuit, with no post-hoc rounding.

The result is not a single point but a trend (Fig. 9): across five native-binary models from half a million to eighteen million parameters—a range of some thirty-fold—held-out bits per token falls from 0.71 to 0.59, and at every size we verify that the entire feed-forward weight set is sign-valued. Two honesty caveats bound this claim, and we state them plainly. First, we report the *median of the late-training evaluations* rather than the single best draw, because the running minimum of a noisy per-batch validation favours the smaller models that were evaluated more often; the robust statistic gives the cleaner trend shown. Second, the points share a fixed wall-clock budget, not a fixed token budget—the smaller models in fact saw more

data—so this is a demonstration that native binarization succeeds and improves with size under a fixed budget, not a controlled scaling law; the controlled study, holding tokens-per-parameter fixed, is future work. The honest reading is that larger native-binary models are more capable while remaining a circuit at every size.

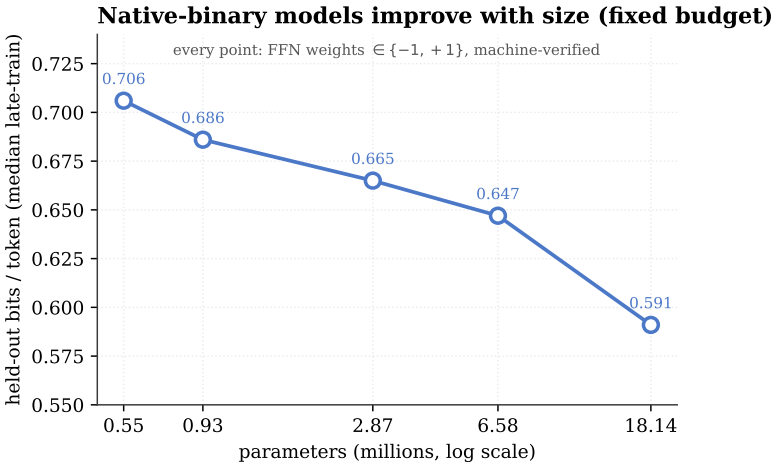


Figure 9: Native-binary models improve with size (fixed budget). Five models spanning some thirty-fold in size each train with feed-forward weights machine-verified to lie in $\{-1, +1\}$; held-out bits per token (median of the late-training evaluations, not the noise-mined running minimum) falls from 0.71 to 0.59. This is an empirical trend under a fixed *wall-clock* budget, not a controlled scaling law—the points are not matched on tokens seen, and in fact the smaller models saw *more* data—so larger size helps *despite*, not because of, more training. A controlled iso-token study is future work. Every point is a model whose feed-forward datapath is a gate netlist read off its weights.

To make "the circuit is the model" fully concrete, we take a single *actual* neuron from this trained model—the first output of the first block's feed-forward layer, 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 whole XNOR array of the general block disappears, leaving an inverter mask on the sixty-five disagreeing inputs followed by the popcount adder tree—a specific 1,300-gate circuit, verified bit-exact against that neuron's own $2\,\mathrm{popcount} - n$ on random inputs. The neuron is not a set of weights a processor multiplies; it is this netlist, and one could etch it.

11 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 verified 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 gate budget for one autoregressive step (Fig. 10). The feed-forward datapath of all layers is 5.6 million gates and is *exact*, read straight off the trained $\{-1, +1\}$ weights; the next-token head is a further 0.38 million; the embedding is a read-only-memory of half a million. The remaining term is attention, and here the compilation reveals a real design fact rather than a nuisance: with a full context of five hundred and twelve keys the hard-attention circuits of all layers reach 12.5 million gates and *dominate* the model, whereas a local window of one hundred and twenty-eight keys holds them to 3.1 million. The whole model is therefore roughly **ten million gates** at a local context and twenty at a global one—and the lever that matters for a fabricated model is not the arithmetic, which binarization has already made cheap, but the *attention window*, which sets whether the

chip is dominated by its mixing or by its multiply–accumulates.

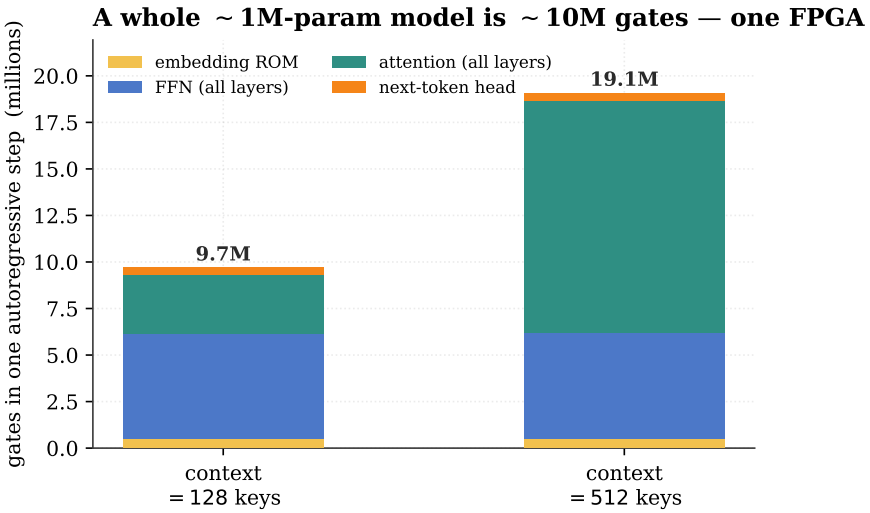


Figure 10: The whole native-binary model as one autoregressive step of logic, composed from the verified blocks at the dimensions of a real sub-million-parameter model (d=128, four layers). The feed-forward datapath and head are exact; attention is the hard-attention circuit. At a local context (128 keys) the model is ∼10M gates; at a global one (512) attention dominates and it is ∼19M. Both are within the capacity of a single contemporary field-programmable gate array.

Ten million two-input gates is a real and unremarkable size. A large contemporary field-programmable gate array carries several million lookup tables, each of which absorbs a handful of such gates, so a model of this class fits on one device with room to spare, and a modest application-specific integrated circuit realizes it comfortably. The character of the resulting machine is worth stating plainly, because it is the point of the exercise. The datapath is feed-forward and contains no multiplier, no floating-point unit, and no instruction fetch; it is a sheet of combinational logic that turns a context register into the next symbol in a single pass, and it *pipelines*, so that once filled it emits one token per clock. Its dominant operation is the XNOR-popcount, which the binary-network hardware literature has repeatedly found to be an order of magnitude more energy-efficient than the equivalent floating-point multiply–accumulate, precisely because it replaces a multiplier with a single gate. We do not quote a joule figure we have not measured; we observe that the arithmetic that dominates a conventional model's energy has here been removed, and that what remains to be engineered is the memory traffic of attention, not the multiply.

12 Formal Verification in Lean

Simulation checks a circuit on the inputs one thinks to try; a proof checks it on all of them. We therefore formalize the compiler's core in the Lean 4 proof assistant and discharge its correctness with the kernel, not with test vectors. 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 $f : (\mathrm{Fin}\, n \to \mathrm{Bool}) \to \mathrm{Bool}$ is realized by a circuit over AND/OR/NOT, proved by Shannon expansion (Eq. (1)), with a vector-valued corollary `circuit_complete_vec`. The informal claim that a finite-alphabet model is a Boolean function and hence a circuit is thus not merely argued but machine-checked; the Lean source builds against `mathlib` and is part of the artifact.

13 Validation

Every claim of this work is a verified artifact rather than an argument. The gate-level compiler, the bit-exact equivalence of each block against its integer specification, the exact gate counts of Table 2, the scaling of the inner product at ≈ 11 gates per input bit, the bit-exact hard-attention netlist of Section 9, the machine-verified native-binary model of Section 10, the six Lean theorems of Section 12, and the emitted synthesizable Verilog for the adder, the multiply–accumulate, and the attention head are all reproducible from the accompanying code, which reads the latest trained binary model from the companion large-binary-models project. What we have *not* yet done is synthesize a complete trained model onto silicon, which we state plainly among the open problems rather than imply.

14 Open Problems

Two of the three matters we first posed as open are now discharged, and we state what remains with the same plainness. *Attention* (Section 9) is no longer a wall: hard, arg-max attention is an exact circuit, and fixed-point soft-max is a finite Boolean function; the residue is the quantitative one of how coarse a fixed-point soft-max a trained model tolerates before its outputs drift, a quantization-aware training question. *Binarization without loss* (Section 10) is demonstrated for the feed-forward datapath, whose weights are trained natively into $\{-1, +1\}$ so the circuit is the model; extending native binarization to the attention projections and the embedding, and closing the bit gap to the floating-point model, is the continuation.

What remains genuinely open is *realization*: mapping the netlists onto a field-programmable gate array and measuring the true latency and energy of a binary model that runs as physics rather than as a program. Two nearer steps precede it. The first is *formal equivalence at scale*: our simulation checks and the Lean proofs of Section 12 certify the block library, but certifying a whole compiled model calls for the standard synthesis toolchain—And-Inverter-Graph rewriting and SAT-based equivalence checking (ABC)—which would also *minimize* the netlist below our un-optimized gate counts. The second is the synthesis pass itself, which reports post-mapping lookup-table counts and timing; we emit synthesizable Verilog toward exactly this, and a technology-mapped realization is the measurement that turns a gate count into a chip.

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.

15 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 and verified those circuits and counted their gates: a full adder of five gates, an inner product of eleven gates per bit, a whole next-token head of a million and a half. The claim is not that this is cheap in the abstract but that it is *concrete*:

a small binary language model is, to a degree that ordinary models are not, a machine one can build. The processor was always a means; for this class of model, it is an unnecessary one.

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

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

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

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