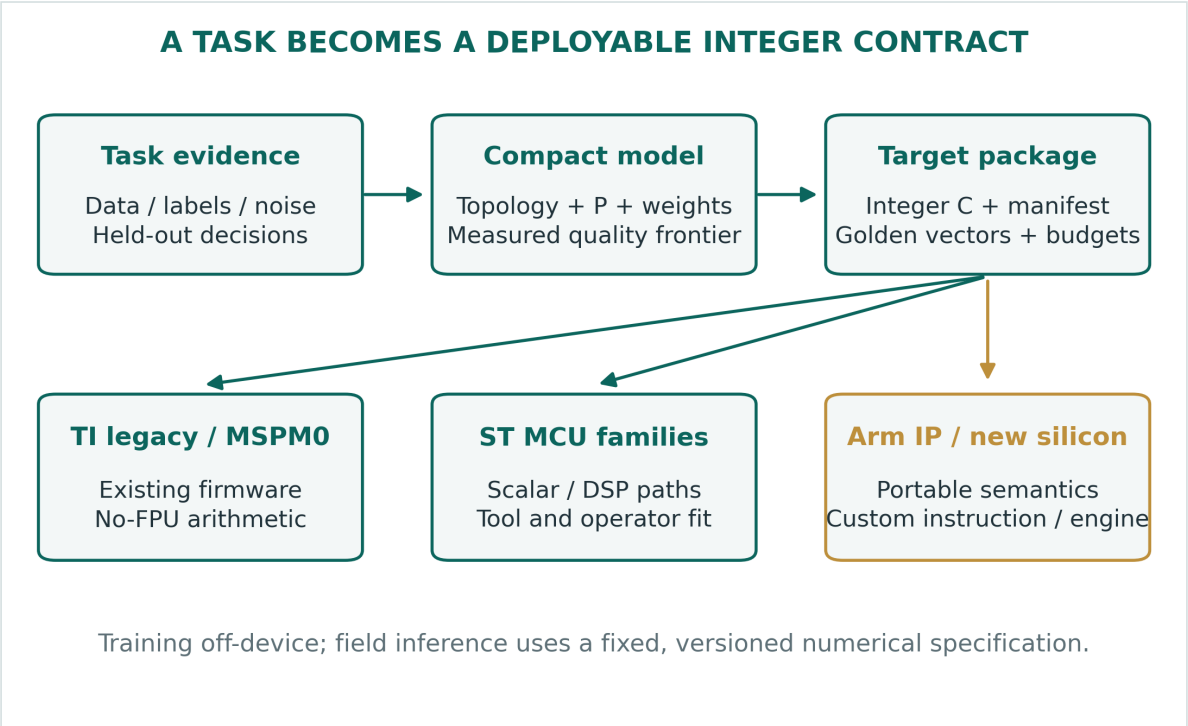
MLIGLON / INTEGER GATE LOGIC

IGL Technical Brief

Knowledge compression, legacy MCU deployment and pathways to specialized silicon
An expanded technical introduction using IGL Laboratory 2.0.1

Laboratory reference 2.0.1 · Brief edition 2.0 | 12 November 2026



For AI scientists, embedded engineers and semiconductor systems teams. Connect the folded IGL node to compact task representations, existing no-FPU devices, modern MCU toolchains and possible future instructions or chips.

Expanded coverage of Texas Instruments, STMicroelectronics and the Arm ecosystem, with worked laboratory exercises, exact arithmetic, field budgets, technical footnotes and a practical licensing evaluation path.

Open the IGL Laboratory

https://mliglon.com/igl-lab/

Contents

Select a chapter or section to jump to it. Page numbers include the cover and contents pages.

1. Backpropagation and the path to embedded AI . . . 5

1.1 From logical neurons to trainable classifiers . . . 5

1.2 Backpropagation has a shared history . . . 6

1.3 Backpropagation, loss functions and optimizers . . . 7

1.4 Architectural progress made gradient learning practical . . . 9

1.5 Success conditions and failure modes . . . 10

1.6 Competitive model families for embedded work . . . 10

1.7 Quantized neural networks are essential comparators . . . 11

2. IGL: the proposition and the mechanism . . . 13

2.1 The engineering proposition and its present evidence . . . 13

2.2 Compare representation, optimization, and implementation separately . . . 13

2.3 What extends from the node to a full network . . . 13

2.4 Why this is a useful semiconductor evaluation opportunity . . . 14

2.5 The IGL node: a folded affine response . . . 14

2.6 Gate replication is a placement problem . . . 15

2.7 Integer mode: preserve the order of operations . . . 16

2.8 P governs resolution, not datatype or mesh quality . . . 17

2.9 What the mechanism does and does not establish . . . 17

3. The knowledge compression effect . . . 18

3.1 Knowledge compression means retaining the distinctions that matter . . . 18

3.2 The XOR fold makes the mechanism visible . . . 18

3.3 Count the description, including the machinery that interprets it . . . 19

3.4 Turn compression into a loss-versus-bytes design frontier . . . 20

3.5 Make the compact representation valuable in a field device . . . 21

4. Laboratory tutorials . . . 22

4.1 Use the Lab as an experimental instrument . . . 22

4.2 Tutorial 1: AND and a graded ramp . . . 23

4.3 Tutorial 2: XOR, the falling segment and a conventional comparator . . . 23

4.4 Tutorial 3: OR corners do not imply a monotone fuzzy OR . . . 24

4.5 Tutorial 4: bias shifts and sign reversal . . . 25

4.6 Tutorial 5: inhibitory weights and asymmetry . . . 26

4.7 Tutorial 6: measure the effect of P . . . 27

5. Training and GPU parallelism . . . 29

5.1 What the single-node teaching search optimizes . . . 29

5.2 Tutorial 7: plateau, restart and reproducibility . . . 30
5.3 From one node to an affected dependency subgraph . . . 31
5.4 A GPU work decomposition that an engineer can evaluate . . . 32
5.5 GPU evaluation: parallel work inside a sequential search . . . 33

6. Precision, memory and the numerical contract . . . 34

6.1 Begin with the numerical contract and a fair baseline . . . 34
6.2 P is a scale, not a storage type . . . 35
6.3 Separate storage, buses, registers and accumulation . . . 36
6.4 A portable rounding and overflow contract . . . 38
6.5 A defensible deployment and measurement plan . . . 39

7. Texas Instruments: installed MCUs and the DSP path . . . 40

7.1 Add a bounded decision to the device already installed . . . 40
7.2 Inventory the exact TI part and its arithmetic resources . . . 40
7.3 Train off-device and export the whole integer contract . . . 41
7.4 Use native operations while preserving the rounded function . . . 42
7.5 An illustrative budget for four independent two-input tests . . . 43
7.6 Fit the scheduler, battery budget and field-update path . . . 44
7.7 TI MSPM0G5187: an MCU evaluation entry point . . . 45
7.8 TI C7x: AM62D-Q1 is specifically a 256-bit C7504 system . . . 45
7.9 TI streaming engines, address generators and MMA boundaries . . . 46

8. STMicroelectronics: from legacy MCUs to acceleration . . . 47

8.1 Start with the MCU already on the board . . . 47
8.2 Prove arithmetic width before selecting a larger part . . . 47
8.3 Use current ST tools without assuming custom-operator support . . . 48
8.4 Give each integration partner a measurable deliverable . . . 49

9. Arm Holdings and the Cortex-M ecosystem . . . 50

9.1 Separate processor IP from the selected MCU . . . 50
9.2 Make scalar, DSP and MVE implementations interchangeable . . . 50
9.3 Treat Ethos placement as a compiler result . . . 51
9.4 Reserve custom instructions for a silicon partner . . . 51

10. Dedicated IGL instructions and silicon . . . 53

10.1 A progression from software to silicon . . . 53
10.2 Instruction candidates: define the smallest useful operation . . . 54
10.3 An exact integer datapath and its verification obligations . . . 55
10.4 A dedicated engine is also a memory architecture . . . 56
10.5 Performance, energy and commercial decision gates . . . 56

11. From application requirements to MCU deployment . . . 58

11.1 Requirements: define the decision before selecting the model . . . 58
11.2 Data: make preprocessing and calibration part of the model . . . 59
11.3 Architecture: establish the hardware and software boundary . . . 59
11.4 Training: retain an auditable experiment . . . 59
11.5 Refinement: build a scalar C reference before accelerating . . . 60
11.6 Export a small runtime and a versioned model contract . . . 60
11.7 Verification: use a pilot acceptance matrix . . . 61
11.8 MCU deployment, lifecycle, and practical questions . . . 62

12. Evidence, product pilots and licensing . . . 63

12.1 Two verified United States patent grants . . . 63
12.2 Reported results worth reproducing . . . 63
12.3 An evidence ladder for adoption . . . 64
12.4 A bounded technical evaluation before commercial scaling . . . 64
12.5 Training software and the next conversation . . . 65
12.6 A concrete next step for TI, ST and Arm ecosystem sponsors . . . 65

13. Extended technical glossary . . . 66

13.1 IGL and laboratory mechanics . . . 66
13.2 Learning, optimization and model structure . . . 67
13.3 Numerical representation and hardware . . . 68
13.4 Evaluation, delivery and commercial decisions . . . 70
13.5 Compression, field deployment and specialized silicon . . . 71

14. References and verification notes . . . 73

14.1 Historical learning and quantization . . . 73
14.2 Compression theory and representation . . . 74
14.3 Numerical implementation, TI DSP and GPUs . . . 74
14.4 TI legacy microcontrollers . . . 74
14.5 STMicroelectronics devices and tools . . . 75
14.6 Arm IP and tools . . . 75
14.7 Dedicated hardware architecture . . . 75
14.8 MLiglon public evidence and intellectual property . . . 75
14.9 Laboratory and reproduction sources . . . 76
14.10 Illustration and verification notes . . . 76

CHAPTER 01

Backpropagation and the path to embedded AI

1.1 From logical neurons to trainable classifiers

This expanded second edition is for engineers evaluating Integer Gate Logic as a compact learned computation for constrained products. It begins with the methods that made modern machine learning successful, then develops the IGL node, knowledge compression, reproducible laboratory exercises and deployment. Dedicated chapters address TI and ST devices, Arm's ecosystem, and proposed hardware specialization. The aim is a useful engineering evaluation that can support a licensing decision, grounded in explicit numerical contracts and application-level evidence.

A licensing evaluation needs a precise comparison class. A new activation, learning rule, numerical representation and hardware implementation are four different contributions. Success in one does not establish superiority in the others. The history below explains why representation, trainability, generalization and deployment efficiency should be evaluated separately.

McCulloch and Pitts formalized networks of idealized all-or-none elements as logical systems in 1943. Their central achievement connected neural-style circuit structure with computable logical behavior; it was not a modern end-to-end parameter-training system. Rosenblatt's 1958 perceptron work made adaptive connections central to pattern recognition, within a broader probabilistic model than the single threshold unit often used in elementary illustrations. [H01][H02]

For a clean mathematical comparison, consider the elementary classifier h(x)=1 when $w_1x_1+w_2x_2+b>0$, and 0 otherwise. XOR requires positive outputs at (1,0) and (0,1), but negative outputs at (0,0) and (1,1). The following constraints are incompatible:

$$b \le 0, \quad w_1 + b > 0, \quad w_2 + b > 0, \quad w_1 + w_2 + b \le 0$$

Adding the two positive-corner inequalities gives $w_1+w_2+2b>0$. Adding the two negative-corner inequalities gives the same expression ≤ 0. This is a direct separability proof for this model class.

The limitation concerns one affine threshold decision surface. It is not a prohibition on neural networks representing XOR. For example, let r(u)=max(0,u). A network with two hidden ReLU units and a linear output computes

$$y = r(x_1 + x_2) - 2r(x_1 + x_2 - 1)$$

At inputs 00, 01, 10 and 11 the outputs are respectively 0, 1, 1 and 0. This explicit construction establishes representability, without claiming that a particular training run will discover it.

A single custom non-monotone activation can also fold an affine sum into the XOR pattern. It changes the function family, so the single-threshold impossibility proof no longer applies. An engineering comparison must count the activation's implementation, numerical precision and training machinery as well as the number of nominal nodes. Four correct Boolean outputs alone establish neither useful behavior between those corners nor accuracy on a separate application dataset.

1.2 Backpropagation has a shared history

The chain rule, reverse accumulation of derivatives, neural-network learning applications and widespread practical adoption are related milestones, not one indivisible invention. Linnainmaa's 1976 BIT paper presents a reverse traversal of recorded operations for sensitivity calculations and explicitly identifies his 1970 Finnish master's thesis as earlier work. The accessible paper also discusses the storage burden of retaining intermediate information. [H03]

Werbos's original Harvard dissertation is dated August 1974 on its title page. Its ordered-derivative and dynamic-feedback treatment develops backward derivative calculations for estimation and optimization. That is a substantive predecessor to modern computational-graph differentiation; it should not be inflated into a claim that every later neural architecture or training system was already demonstrated there. [H04]

Rumelhart, Hinton and Williams's 1986 paper showed how propagated error derivatives can adjust hidden-unit connections so that internal representations become useful for a task. It is an essential neural-learning milestone. Crediting it does not erase earlier automatic-differentiation and optimization work, or justify assigning the whole history to one author. [H05]

> **Historical scope**
> The 1970 thesis itself was not inspected; its date and relationship are supported by Linnainmaa's own 1976 reference list and introduction. The Werbos thesis scan was inspected for its title page and ordered-derivative treatment. This selected lineage does not adjudicate exhaustive invention priority or patent novelty.

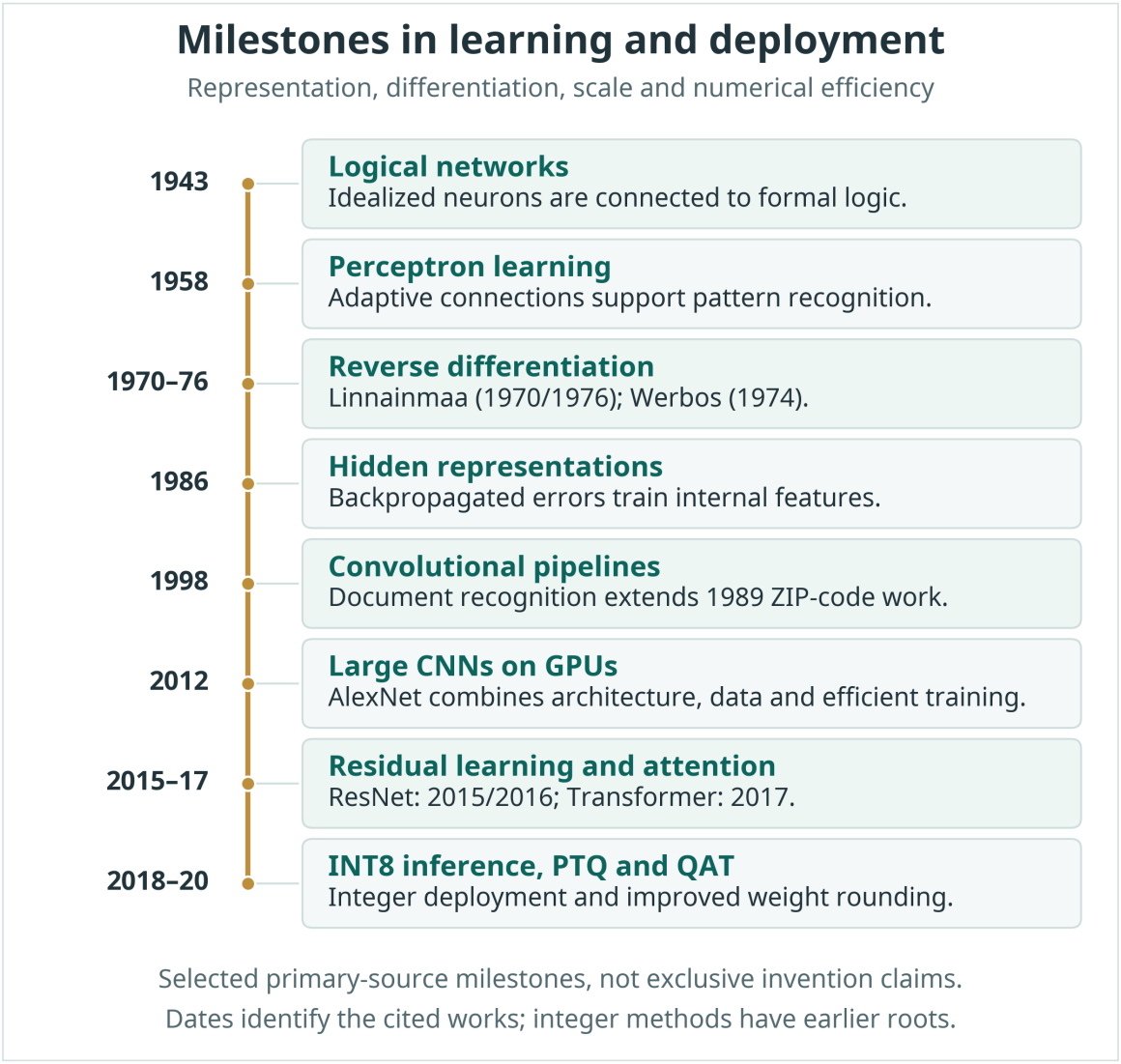


Figure 1. Selected primary-source milestones distinguish logical representation, reverse differentiation, hidden-feature learning, architectural progress and quantized deployment. This timeline preserves shared historical credit. [H01-H10, H17, H19]

1.3 Backpropagation, loss functions and optimizers

For supervised learning, the designer chooses a model fθ and a loss ℓ that compares its prediction with a target. Training commonly minimizes an average loss plus a regularization term. This mathematical objective is distinct from the operational acceptance criterion: low mean training loss need not imply an acceptable false-alarm rate, worst-case error or calibration.

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), t_i) + \lambda R(\theta)$$

θ contains trainable parameters; N is the number of training examples; λ controls the selected regularization penalty R. Validation data are not included in this training average.

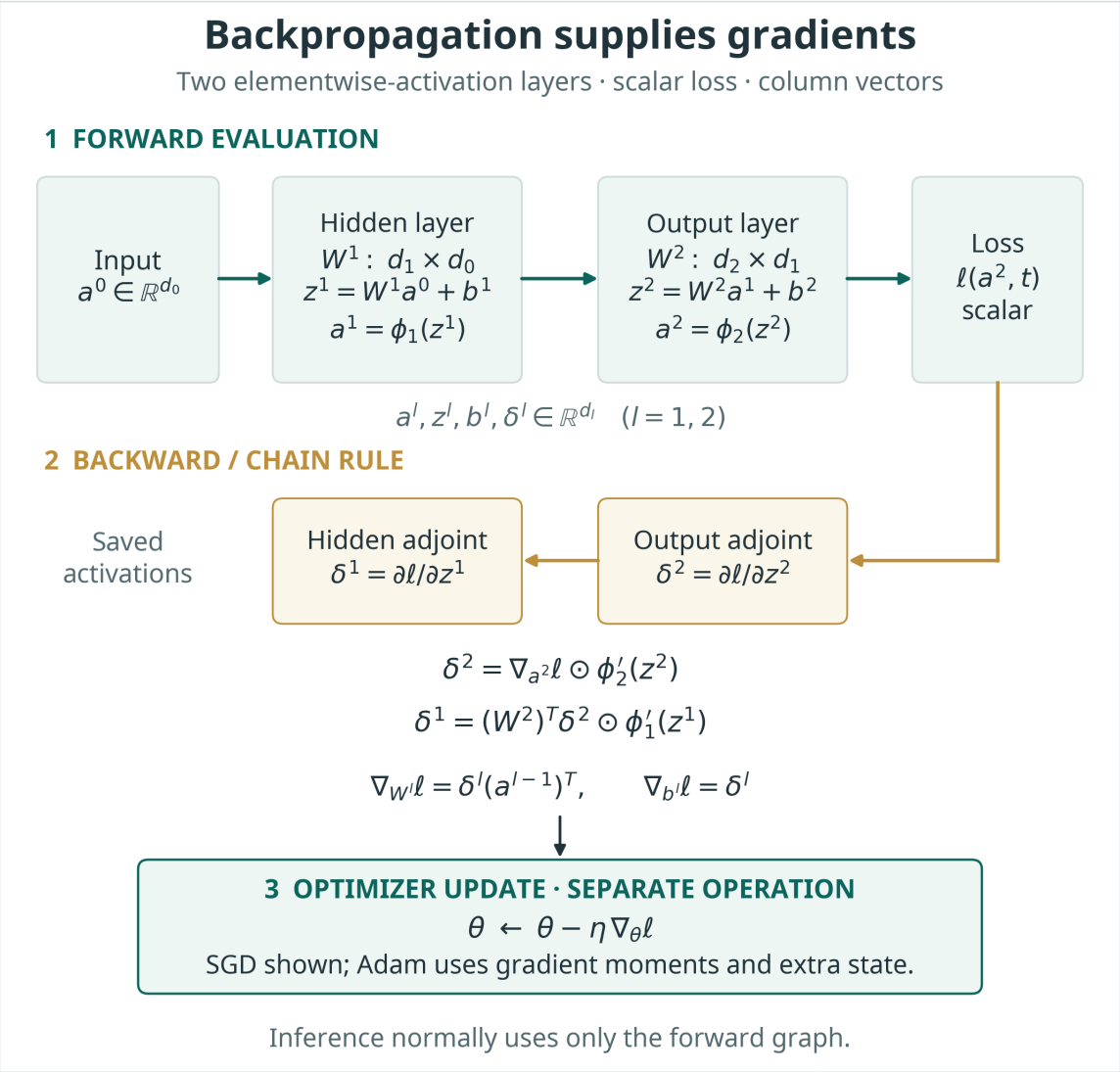


Figure 2. Two-layer forward activations and backward adjoints, with exact dimensions. Parameter-gradient calculation and the optimizer update are separate operations. Equations are direct chain-rule derivations.

For a feedforward layer with column-vector activations, write $z^l = W^l a^{l-1} + b^l$ and $a^l = \varphi^l(z^l)$. Backpropagation applies the chain rule from the loss toward earlier operations. With elementwise activation functions and $\delta^l = \partial \ell / \partial z^l$, the reverse recurrence and parameter derivatives are:

$$\delta^L = \nabla_{a^L} \ell \odot \phi^{L\prime}(z^L)$$

The output-layer sensitivity combines the loss derivative with the final activation derivative. The symbol ⊙ denotes elementwise multiplication.

$$\delta^l = (W^{l+1})^T \delta^{l+1} \odot \phi^{l\prime}(z^l)$$

Each preceding layer receives the downstream sensitivity transformed by the next layer's weights, then applies its own local derivative.

$$\frac{\partial \ell}{\partial W^l} = \delta^l (a^{l-1})^T, \qquad \frac{\partial \ell}{\partial b^l} = \delta^l$$

Shared parameters accumulate contributions from all uses. Batch gradients average the example contributions; the regularizer adds its own derivative.

These equations are a direct chain-rule derivation. The practical advantage is reuse: one reverse traversal can obtain derivatives for every parameter contributing to a scalar loss, instead of separately perturbing each parameter and rerunning the model. Backpropagation computes sensitivities; it does not by itself choose a learning rate, update rule, architecture or stopping criterion. The neural-learning use of this mechanism is exemplified by the 1986 work. [H05]

Stochastic approximation predates deep learning: Robbins and Monro analyzed iterative estimation from noisy observations in 1951. Mini-batch stochastic gradient descent uses a sample-based gradient g_t in the update below. Momentum adds a running direction; learning-rate schedules change how aggressively that direction is followed. [H11]

$$\theta_{t+1} = \theta_t - \eta_t g_t$$

η_t is the learning rate. A mini-batch reduces the cost of one update, while introducing sampling variability relative to the full training gradient.

Adam instead maintains moving estimates of the gradient's first and second moments, applies bias correction, and scales updates coordinate by coordinate. It still depends on gradients, usually supplied by backpropagation. Its moment arrays also consume training memory. Adam is an optimizer, not a replacement for the derivative calculation, and its original empirical results do not guarantee dominance on every workload. [H12]

At inference, a deployed feedforward model normally performs only the forward calculation. Training additionally requires gradients, optimizer state and access to suitable data. A proposed reduction in training cost therefore cannot be inferred from a faster forward kernel; conversely, efficient inference need not require on-device training.

1.4 Architectural progress made gradient learning practical

Useful deep systems emerged from coordinated changes in architectures, optimization, data and implementations. The milestones below matter because they changed the computation being trained or made a favorable inductive bias explicit. They did not replace the chain rule.

Milestone	Technical contribution	Engineering interpretation
LeCun et al., 1989 [H06]	Backpropagation with local connectivity and shared weights for handwritten ZIP-code recognition.	A convolutional prior reuses a detector across image positions; count both computation and reusable parameters.
LeCun et al., 1998 [H07]	Convolutional recognition and globally trained document-processing systems.	End-to-end performance depends on the composed pipeline, not an isolated classifier.
AlexNet, 2012 [H08]	Large CNN training combined non-saturating activations, regularization and an efficient GPU implementation.	Data, software and parallel hardware are part of a reproducible result.
ResNet, 2015/2016 [H09]	Residual blocks learn F(x) with an identity shortcut, producing x+F(x).	A change in parameterization can make much deeper networks easier to optimize.
Transformer, 2017 [H10]	Attention-based sequence processing removed the original model's need for recurrent or convolutional sequence layers.	The operator mix and memory traffic differ from a CNN; a kernel advantage must be tested on the intended graph.

ResNet's preprint appeared in December 2015 and the conference paper in CVPR 2016. Adam similarly appeared as a 2014 preprint before its 2015 conference publication. The 1998 document-recognition paper's "graph transformer networks" are a different construction from the 2017 attention-based Transformer. Dates and overloaded terminology should be preserved accurately. [H07][H09][H10][H12]

For embedded evaluation, the architectural lesson is practical: compare a proposed primitive inside a complete application graph. Surrounding transforms, normalization, activation storage and output processing may dominate even when one inner operation becomes cheaper. Dense attention, for example, explicitly forms pairwise relationships; changing the sequence length changes the workload being measured.

1.5 Success conditions and failure modes

Gradient methods work well when the model can express a useful solution, derivatives provide an informative direction, optimization is sufficiently conditioned and training data represent the intended task. Failure in one of these conditions should not be attributed automatically to the others. The following diagnostic distinctions are engineering guidance, rather than claims that one remedy guarantees success.

Observed problem	Possible mechanism	Useful check
Training loss remains high	Insufficient representation, a poorly scaled objective, bad optimization settings or an implementation error.	Compare a tiny overfitting test, inspect gradients, and verify the forward/reference calculation.
Early-layer updates are tiny or unstable	Repeated Jacobian products attenuate or amplify sensitivities; saturated or inactive units can remove paths.	Inspect gradient norms, activation distributions, initialization and residual paths.
Training is good; held-out performance is poor	Overfitting, leakage in model selection, insufficient coverage or a train/deployment distribution mismatch.	Use a genuinely separate test set and report subgroup and operating-point metrics.
Reference model works; device output differs	Rounding, saturation, unsupported operators, preprocessing differences or an export/compiler mismatch.	Compare layer outputs and input scaling against a fixed numerical reference.
Accuracy is adequate; deployment misses its budget	Memory movement, workspace, unsupported kernels or unrepresentative timing conditions.	Measure the complete compiled application at the required batch size and device settings.

Bengio, Simard and Frasconi analyzed the difficulty of learning long temporal dependencies with gradient descent. Glorot and Bengio examined activation and initialization effects in deep feedforward networks. These are concrete numerical difficulties, not evidence that all gradient-trained networks fail. Residual connections are one experimentally successful architectural response, with results tied to the tested models and tasks. [H13][H22][H09]

Hard rounding is an especially important boundary: its ordinary derivative is zero between thresholds and undefined at a transition. Directly inserting it into an otherwise differentiable training graph removes the usual informative local signal. One can use surrogate gradients, stochastic estimators or derivative-free optimization, but each changes the training method and its assumptions. Training behavior must therefore be measured independently of the attractive simplicity of an integer forward expression. [H21]

1.6 Competitive model families for embedded work

A responsible baseline set includes methods appropriate to the data and operating constraints. Neural networks are one family among several. For an engineered sensor feature vector, a small linear model or tree may be sufficient; for raw images or sequences, feature learning can change the appropriate comparison. These are candidate-selection principles, not universal rankings.

Family	Learning and inference structure	What to measure
Linear or logistic model	Fits a weighted combination of supplied features; inference is a small dot product and optional link function.	Include the cost and accuracy consequences of feature extraction, not just the final classifier.
Support-vector machine [H14]	Fits a margin-based decision function; kernels allow nonlinear feature-space boundaries.	For kernel inference, support-vector count and kernel evaluation cost matter; a linear SVM has a different footprint.
Random forest [H15]	Combines randomized decision trees through voting or averaging.	Count stored nodes, tree depth, branches and memory accesses. Model size grows with the chosen ensemble.
Gradient-boosted trees [H16]	Adds trees to improve a loss-based objective; XGBoost supplies a scalable implementation with regularization and sparsity-aware processing.	Benchmark the exported ensemble, including missing-value routing, thresholds and accumulation.

The presence of a loss gradient does not make a method neural backpropagation. Boosting can use loss derivatives while constructing decision trees. Conversely, a neural model can be optimized using searches that do not compute derivatives. Keep the model family, optimization algorithm and execution representation separate in technical claims.

For a TI evaluation, use the strongest feasible implementation in each relevant family under the same acceptance test. A large forest should not be selected merely to make a small neural model look compact; an oversized FP32 network should not stand in for every practical neural baseline. Document tuning effort and feature preparation for all contenders, then compare a quality-versus-resource frontier rather than selecting one favorable operating point.

1.7 Quantized neural networks are essential comparators

Integer neural inference is an established engineering approach. Jacob et al. described a quantization scheme and matching training procedure for integer-only inference. Krishnamoorthi's study compared post-training and training-aware techniques across CNN architectures. Their results make a well-tuned quantized network a necessary comparator; they do not imply one universal accuracy penalty or hardware speedup. [H17][H18]

$$q = \mathrm{clip}(\mathrm{round}(r/s) + z,\ q_{\min}, q_{\max}), \qquad \hat{r} = s(q - z)$$

This illustrative affine quantizer represents a real value r by an integer q, scale s>0 and integer zero point z. Rounding, clipping range, granularity and accumulator width must be specified.

Post-training quantization starts with an already trained model. Calibration estimates useful activation ranges from representative data; per-channel weight scales can accommodate channels with different dynamic ranges. Modern PTQ need not mean blind nearest rounding: AdaRound optimizes weight-rounding decisions using a small unlabeled sample and a local reconstruction objective. That is additional optimization, although it is not ordinary full-network supervised retraining. [H18][H19]

Quantization-aware training simulates quantization so that parameters can adapt. Common implementations retain floating-point trainable values and use a surrogate derivative through fake-quantization operations; the exported inference graph can use integer arithmetic. A straight-through estimator is a heuristic gradient substitute, not the exact derivative of discontinuous rounding. QAT therefore does not establish an entirely integer-only training process. [H17][H21]

Low precision does not necessarily cause a significant accuracy loss. The outcome depends on the model, task, bit width, calibration coverage, scale granularity and training procedure. It can be small enough to satisfy the application, or large enough to require QAT, mixed precision or a different architecture. Report measured absolute quality changes and uncertainty rather than assuming degradation from the numerical format alone. [H18][H19]

An 8-bit stored weight occupies one quarter of a 32-bit value's payload, before scales, padding and other state. This arithmetic does not imply a fourfold reduction in complete firmware or RAM. Integer dot products also need suitable accumulators and requantization; unsupported layers may cause conversions or fallback execution. Embedded kernel work such as CMSIS-NN demonstrates why implementation quality matters, but measurements on Arm Cortex-M are not measurements of the selected target. [H20]

Minimum credible neural baseline
Include an appropriate floating-point reference, a calibrated supported INT8 implementation and, when needed to meet the same quality threshold, QAT or mixed precision. Use the intended compiler/runtime and the complete input-to-output path. Record batch-one latency, peak RAM, persistent model storage, energy per inference and quality together. Equalize data splits and preprocessing. These are proposed evaluation controls, not performance results for IGL or any TI device.

Separate learning from execution

A comparison of mechanisms, not a performance ranking

Approach	How parameters are fitted	What runs on the device
BP + FP32 inference	Backprop computes gradients. An optimizer updates weights in a chosen numeric format.	Floating-point forward graph. Backprop and optimizer state are normally omitted.
BP-trained + PTQ / QAT	PTQ: quantize a trained model. QAT: adapt with simulated quantization; training state is commonly floating point.	Integer or mixed-precision forward graph, with supported kernels and any required format conversions.
Discrete IGL search (reviewed demo)	Bounded derivative-free search of three node parameters. Four Boolean training pairs; P and mode remain fixed.	One JavaScript node evaluator: affine sum + five-region activation, with INT or Real mode; no network scaling test.
Tree / linear models	Fit coefficients, splits or ensembles. Loss gradients may be used without neural backpropagation.	Branches and votes, or small dot products, plus any feature extraction.

PTQ: post-training quantization · QAT: quantization-aware training

Match task quality and measure the complete deployed pipeline.

Figure 3. Fitting parameters and deployed computation are distinct choices. PTQ and QAT are essential baselines. The IGL row explicitly describes the reviewed teaching demo; broader training is discussed later. [H14-H20, L01, L03]

CHAPTER 02

IGL: the proposition and the mechanism

2.1 The engineering proposition and its present evidence

IGL offers a specific engineering hypothesis: a network built from compact, logic-capable nonlinear nodes, combined with discrete parameter search and reuse of unaffected computations, may meet a useful classification target with less implementation cost. MLiglon presents integer parameters, non-monotonic activation, and chain isolation as complementary elements of that approach. Its current homepage also explicitly qualifies the relationship between parameter storage, runtime memory, speed, and energy. These are separate measurements; improvement in one does not establish improvement in all. [I01]

For a prospective semiconductor partner, the immediate opportunity is to test that hypothesis on a constrained application with a clear acceptance threshold. The public laboratory makes the basic node inspectable: users can change parameters and observe Boolean or graded responses. That is useful mathematical evidence, but it is not a measurement of a production network, a CUDA trainer, or a TI deployment. The broader architecture and its reported benchmark implementations must be evaluated separately from the educational two-input demonstration. [I02, I07]

2.2 Compare representation, optimization, and implementation separately

Backpropagation computes derivatives of a loss through a computational graph; an optimizer uses those derivatives to change parameters. MLiglon's chain isolation approach instead measures the loss produced by proposed parameter changes, without a backward derivative pass. This comparison concerns the optimization method. It does not imply that every conventional network is dense, uses FP32 inference, or requires expensive training hardware for every task. [I08]

Integer inference is established outside IGL. Jacob and colleagues described quantized weights and activations, with a compatible training procedure, for integer-arithmetic-only inference in their CVPR 2018 paper. Consequently, an FP32 reference alone cannot establish superiority over the relevant embedded alternatives. Include an optimized quantized baseline and distinguish savings from narrower values from savings caused by fewer parameters, different connectivity, or a different input representation. MLiglon's own INT8 comparison acknowledges these distinctions. [I09, H17]

The single-node XOR example is a precise representational result: a non-monotonic activation can produce the required fall in output when both inputs are active. It does not show that conventional multilayer networks cannot learn XOR, nor that IGL will be more accurate on every dataset. MLiglon's logic page expressly recognizes conventional multilayer XOR implementations. [I16]

2.3 What extends from the node to a full network

The foundational patent describes layered networks, including embodiments in which a non-output node feeds one downstream node, as well as convolutional, locally interconnected, and hybrid arrangements. These topology choices matter: the efficient isolated path of a restricted network should not be assumed for every possible graph. The current IGL-CNN description includes shared kernels, receptive fields, channels, and pooling. A convolutional kernel therefore represents more computation than the laboratory's two-input illustration. [I03, I07]

The separate training chapter develops the dependency reuse, candidate scoring and consistent-update requirements. The teaching application remains deliberately limited to one node; it does not expose a network architecture editor, production GPU trainer or MCU deployment compiler. [L01, L03]

2.4 Why this is a useful semiconductor evaluation opportunity

A semiconductor team can evaluate IGL at several boundaries: a compact sensor feature model that fits existing SRAM, a DSP kernel that packs integer operands effectively, or a training system that explores discrete hardware-aware candidates. The useful result is a task-level design improvement at matched quality. If a smaller representation avoids an external-memory dependency, frees CPU/DSP headroom, or lowers measured energy per decision, that can become a product-level argument for licensing. These are concrete pilot objectives rather than assumed outcomes.

The laboratory reduces the cost of the first conversation: an engineer can inspect the exact node, understand why XOR works, reproduce precision effects and challenge the arithmetic assumptions before evaluating a full model. A bounded joint pilot with MLiglon is the practical next step toward a defensible license recommendation.

2.5 The IGL node: a folded affine response

In the laboratory, a node receives two normalized inputs x_1 and x_2, forms a weighted sum with bias B, and passes that sum through a continuous, non-monotonic, piecewise-linear activation. P is a positive scale. The essential change from an ordinary monotone threshold or rectifier is the falling middle segment: increasing the weighted sum can decrease the output. This fold makes a single node capable of all 16 two-input Boolean truth tables when suitable parameters are chosen. [L01]

$$D = W_1x_1 + W_2x_2, \qquad S = D + B, \qquad Y = f_P(S), \qquad q = Y/P$$

Real mode. x_1,x_2 lie in [0,1]; q lies in [0,1]. D is the dot product, S the activation input, Y the raw output, and q the normalized output.

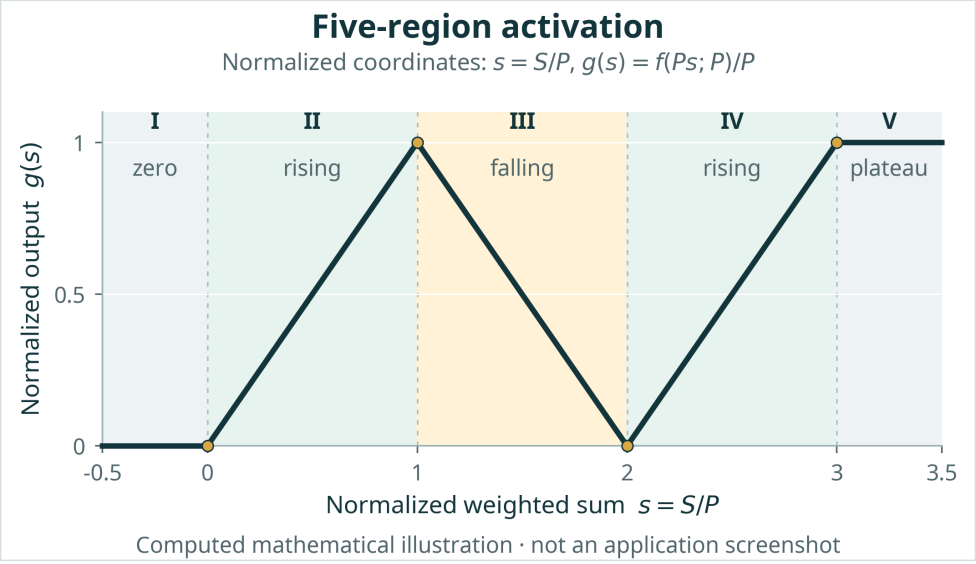


Figure 4. The five-region activation, calculated from the laboratory equations. The output rises, falls, rises again, then saturates. This is a mathematical illustration, not a screenshot.

Region	Activation input S	Raw output Y	Normalized form g(s), s=S/P
I · zero	S ≤ 0	0	0
II · rising	0 < S ≤ P	S	s

Region	Activation input S	Raw output Y	Normalized form g(s), s=S/P
III · falling	$P < S \leq 2P$	2P - S	2 - s
IV · rising	$2P < S \leq 3P$	S - 2P	s - 2
V · plateau	$S > 3P$	P	1

At the boundaries the pieces meet continuously: q=0 at s=0, q=1 at s=1, q=0 at s=2, and q=1 at s=3. The displayed region label follows the left-hand interval at these joins: P is region II, 2P region III, and 3P region IV. A value in [0,1] is not automatically a calibrated probability; that interpretation requires an appropriate task, training objective, and calibration evidence. [L01]

2.6 Gate replication is a placement problem

For the binary pairs 00, 01, 10 and 11, the four sums are B, W_2+B, W_1+B and W_1+W_2+B. Parameters place those sums at selected peaks, troughs or flat regions. For XOR, W_1=W_2=P and B=0 give sums 0,P,P,2P and outputs 0,1,1,0. Adding P to the bias instead gives XNOR: P,2P,2P,3P map to 1,0,0,1. These are exact constructions; a training algorithm is not required to verify them.

$$(S_{00}, S_{01}, S_{10}, S_{11}) = (B, W_2 + B, W_1 + B, W_1 + W_2 + B)$$

The table below lists one construction per function. Multiplying each coefficient by the chosen P produces the actual parameters. A and the input symbol B in gate names refer to x_1 and x_2; the parameter B in the formula is the bias.

Gate	(W_1/P, W_2/P, B/P)	Outputs 00,01,10,11
FALSE	(0, 0, 0)	0000
AND	(1, 1, -1)	0001
A AND NOT B	(1, -1, 0)	0010
A	(1, 0, 0)	0011
NOT A AND B	(-1, 1, 0)	0100
B	(0, 1, 0)	0101
XOR	(1, 1, 0)	0110
OR	(2, 2, -1)	0111
NOR	(-1, -1, 1)	1000
XNOR	(1, 1, 1)	1001
NOT B	(0, -1, 1)	1010
A OR NOT B	(2, 1, 1)	1011
NOT A	(-1, 0, 1)	1100
NOT A OR B	(1, 2, 1)	1101
NAND	(-2, -2, 3)	1110
TRUE	(0, 0, 1)	1111

All listed constructions fit the laboratory's ±3P parameter bounds and work at every supported P, including INT mode, because the binary inputs and these parameters are integral on the scale. Multiple parameter settings can realize the same corners. Their interior surfaces, sensitivity to noise, arithmetic cost and

margins can differ. Gate replication therefore supplies building blocks and insight; application training must select useful behavior over the actual input distribution. [L01]

2.7 Integer mode: preserve the order of operations

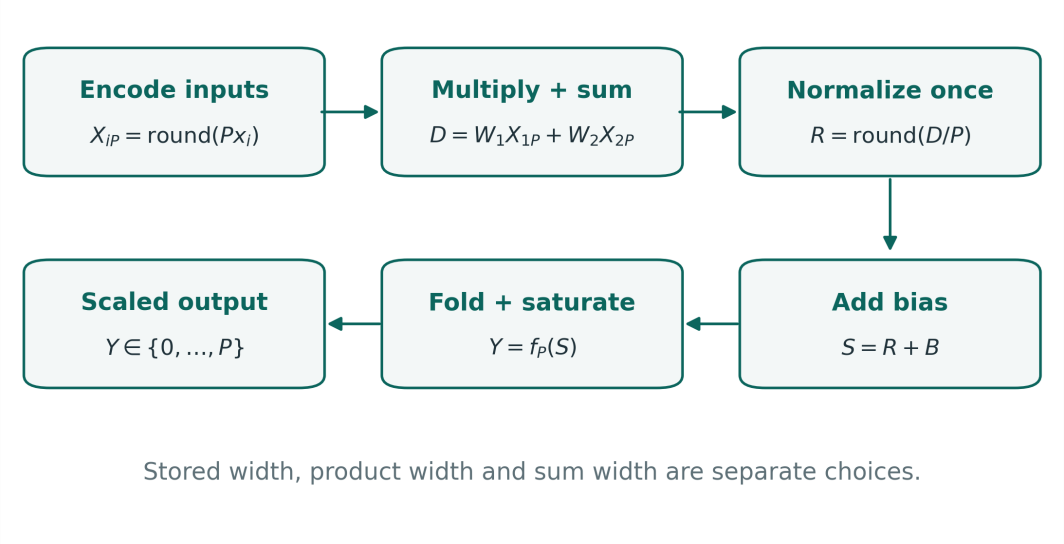


Figure 5. The laboratory integer contract. Products accumulate before one normalization rounding; bias is then added in activation units. Wider arithmetic may be required between narrower stored values.

$$X_{iP} = \text{round}(Px_i), \quad D = W_1X_{1P} + W_2X_{2P}$$

INT mode first rounds W_1, W_2 and B to integers. X_iP is an integer from 0 through P inclusive.

$$S = \text{round}(D/P) + B, \qquad Y = f_P(S), \qquad q = Y/P$$

Do not divide B by P, normalize each product independently, or replace the final rounding with truncation. These changes can move S across an activation boundary.

The reference uses JavaScript Math.round: a half-way value goes toward positive infinity. Thus 1.5 becomes 2, -1.5 becomes -1, and -0.5 becomes a signed zero displayed as 0. A port that uses ties-to-even or ties-away-from-zero must not claim bit-exact agreement. The laboratory uses JavaScript Number in both modes; “INT” describes the arithmetic rules, not a measured native INT8, INT16 or INT32 execution path. [L01]

> **A small order-of-operations counterexample**
> Let P=4, W_1=W_2=1, B=0 and integer inputs X_1P=X_2P=1. The correct sum is round((1+1)/4)=1. Rounding each product separately gives round(1/4)+round(1/4)=0. Both implementations use integers, but they implement different nodes.

The raw output can be passed to another node as a scaled integer when the receiving operator uses the same documented scale convention. Networks with different per-layer scales need explicit conversions, bounds and rounding rules. The two-input laboratory does not define a universal exchange format for arbitrary IGL networks; ask MLiglon to specify that contract for any training or deployment tools offered for evaluation.

2.8 P governs resolution, not datatype or mesh quality

The laboratory offers P=1,2,4,8,16,32,64,128,256,512 and 1024. In INT mode each input has P+1 available levels, spaced 1/P apart, and the output also has at most P+1 levels. A surface mesh is a separate visualization sampling choice: increasing mesh density cannot add numerical states to the integer node. Interpolation can make an integer surface appear smoother than its actual steps. [L01, L02]

$$a_i = W_i/P, \quad b = B/P, \quad q = g(a_1x_1 + a_2x_2 + b)$$

Normalized real-mode equation. If P, W_1, W_2 and B are multiplied by the same positive factor, this expression is unchanged.

The “Preserve ratios when P changes” option follows that scale-invariance idea. Integer snapping can still change the resulting coefficients or coordinates. In particular, after an INT experiment rounds an input, increasing P preserves the stored, already-rounded coordinate; it cannot recover a discarded original measurement. Re-enter the original requested inputs when comparing precision settings. [L02]

$$|q_{\text{INT}} - q_{\text{real}}| \le \min\left(1, \frac{|a_1| + |a_2| + 1}{2P}\right)$$

Derived bound for the same stored integer W_1,W_2,B, comparing exact real inputs with the laboratory’s rounded inputs and rounded normalized sum. It excludes a separate change caused by rounding fractional parameters.

The bound follows because each input error is at most 1/(2P), the final normalization adds at most 1/(2P) in normalized sum units, and g has slope magnitude at most one. It is an absolute worst-case bound, not an expected error or a classification-accuracy forecast. A sampled precision map can demonstrate errors at its samples but cannot prove a smaller global maximum. Physical sensor noise, clipping and distribution shift need their own analysis.

2.9 What the mechanism does and does not establish

The real activation is differentiable except at its joins, with slopes 0,+1,-1,+1,0. Therefore “IGL is not trained by backpropagation” describes the reviewed search approach; it is not a theorem that derivatives can never be used in a relaxation or hybrid model. Integer rounding creates plateaus, while the folded response can produce multiple useful and unhelpful parameter regions. A search method must navigate that landscape under a finite budget.

One IGL node may replace several simple units for a selected Boolean relation. An end-to-end advantage still depends on representation, preprocessing, topology, routing, precision, data and the compiled operator mix. A parameter count should be accompanied by executable size, connection storage, peak live activations and target measurements. The technical attraction is a different design space for compact integer functions, with an inspectable node and direct hardware constraints that can be brought into training.

CHAPTER 03

The knowledge compression effect

3.1 Knowledge compression means retaining the distinctions that matter

Imagine a vibration sensor that must distinguish normal operation from a developing fault. Its training recordings contain waveform detail, environmental variation and repeated examples. A useful deployed model turns those observations into a reusable decision rule. The opportunity is to carry enough of that rule onto a small device that it can make the required decision locally, without retaining or transmitting every recording.

This is the intended meaning of the knowledge compression effect here: compact executable structure that preserves useful task distinctions. IGL offers a candidate representation because its folded response can express several rising and falling regions with a small set of coefficients. When those regions match a real task, one motif may do work that would otherwise require a larger composition. This is an engineering opportunity to test, rather than a claim of lossless example storage or an established IGL-specific compression theorem. [1]

The effect can arise at several levels. A useful response shape may need fewer nodes; shared intermediate features may avoid repeated work; a smaller admissible parameter range may permit narrower storage; and scheduling may reduce the number of activations alive at once. These mechanisms can reinforce each other. They also answer different questions: a short parameter vector can still have expensive preprocessing, while a slightly larger regular model may run efficiently. The practical aim is a compact decision procedure that preserves the distinctions the application needs.

The information bottleneck provides a useful conceptual connection. Tishby, Pereira and Bialek ask how a compressed description of an input can preserve information relevant to another variable. A task-sufficient summary retains the target's conditional distribution: knowing the full input would not improve prediction beyond that summary. Approximate sufficiency is the practical goal when some loss is acceptable. The summary is judged relative to the task and operating distribution; a vibration representation useful for fault detection may discard details needed to identify the machine. [K04]

3.2 The XOR fold makes the mechanism visible

The laboratory gives an exact, inspectable example. Set W_1=W_2=P and B=0. In Real mode the normalized sum is s=x_1+x_2. Across the unit input square, the first two nonzero activation segments produce:

$$q = g(x_1 + x_2) = 1 - |x_1 + x_2 - 1|, \qquad (x_1, x_2) \in [0, 1]^2$$

Here q=Y/P. At inputs 00, 01, 10 and 11, s is 0, 1, 1 and 2; q is exactly 0, 1, 1 and 0. This is the laboratory's XOR construction. [L01]

The sum merges 01 and 10 because the task assigns them the same result. The fold then assigns both extreme sums to zero. One shared rule therefore produces the four distinctions without consulting a stored example list. The same Boolean outputs hold in INT mode at every supported P, since the relevant inputs, products and sums land exactly on the scale. [L01][2]

1. Knowledge is an operational term for useful predictive structure here. Information-bottleneck compression concerns random variables; compressed model storage concerns an executable description. They are related viewpoints, not identical measured quantities. [K04]

2. The XOR example proves representation, not training success or a storage advantage. A truth table for a fixed two-bit address needs four output bits; an IGL implementation also needs its coefficients, scale and evaluation convention. [L01]

The representation has changed: one affine threshold cannot separate these XOR labels, while a folded activation can. A small conventional network can also express XOR. For Boolean XOR alone, a native XOR instruction or four-bit truth table is an excellent comparator. The interesting IGL prospect is broader: a repeated folded motif might reduce intermediate nodes, connections and routing for a useful family of relations. Its continuous interior also supplies behavior between the corners, which must be trained or verified when those inputs matter.

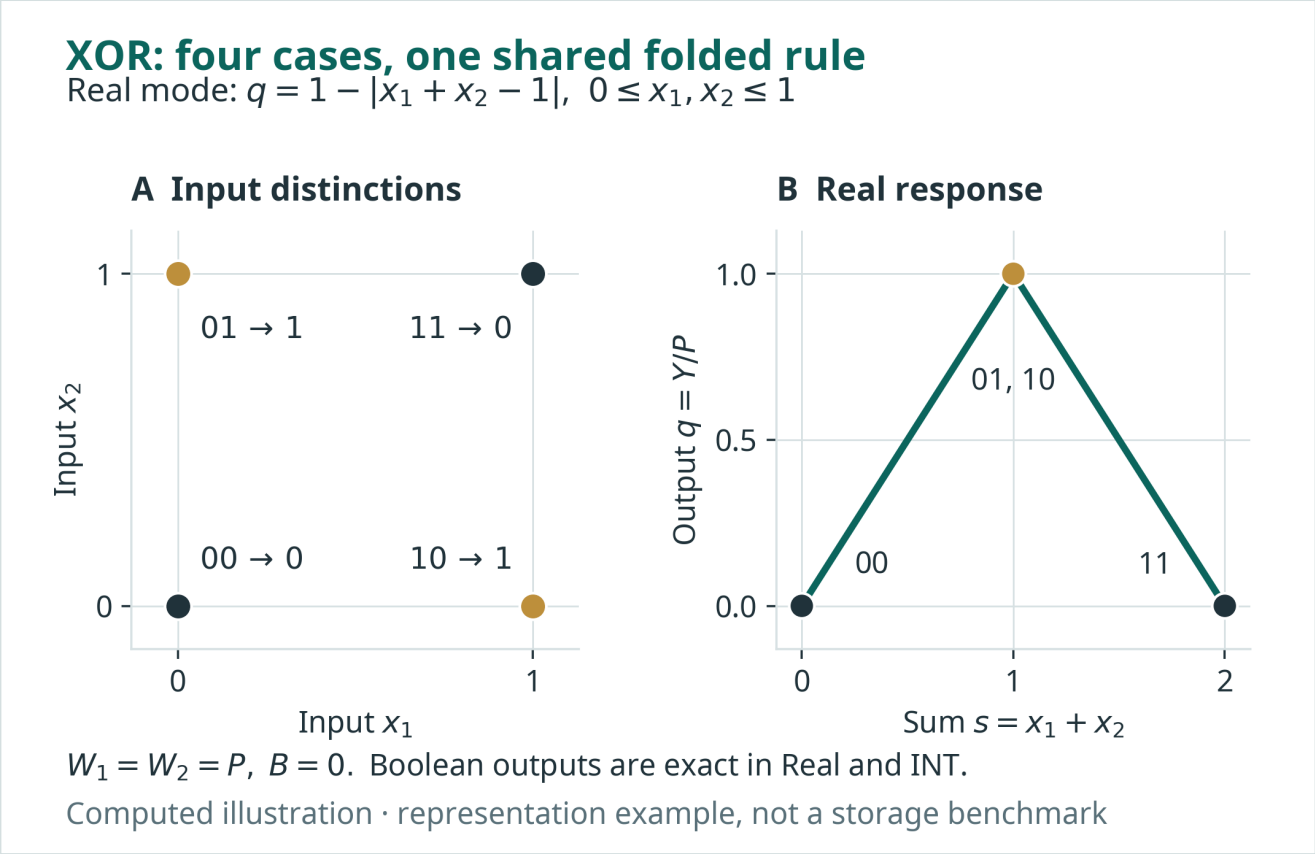


Figure 6. An exact representation example: x_1+x_2 merges the two positive XOR cases, then the triangular fold maps sums 0 and 2 to zero. The rule produces all four Boolean outputs; it does not establish a byte saving over a native XOR instruction or truth table.

3.3 Count the description, including the machinery that interprets it

Minimum description length evaluates how economically a model and its unexplained data can be described together. Rissanen's formulation includes structural and numerical choices. Hinton and van Camp apply the idea to neural weights and residual errors, showing why the information needed to specify weights matters beyond their number. In a deployment, this suggests examining both the learned representation and the conventions needed to execute it. [K01][K05]

A parameter count is a useful inventory, not a unit of knowledge. With a fixed decoder, n independently stored b-bit parameters offer at most 2^(nb) bit patterns; shared values, forbidden combinations and functional equivalences can reduce distinct behaviors. Varying the graph adds choices. A compact architecture code, scales, feature definitions and activation implementation therefore belong in the accounting. Precision, connectivity and the interpreter together define the hypothesis family. [3]

3. The bit-pattern bound assumes a fixed code and decoder. For fixed-length B-bit descriptions there are at most 2^B codes. This is not an asserted VC dimension or a generalization bound for an IGL network.

Occam-style learning results connect sufficiently short consistent hypotheses with learnability under specified assumptions and efficient construction. They support looking for economical explanations, without making small file size a guarantee of field accuracy. A compact rule can still exploit a spurious correlation. Training and validation need examples that exercise the intended conditions. [K02]

Shared computation makes compactness useful. One activation routine can serve many nodes; a computed feature can feed several decisions; repeated parameter values can use a codebook. The routine, feature buffer and codebook still occupy resources, but their costs need not be paid independently for every use. Established neural compression already combines pruning, trained quantization, weight sharing and entropy coding. IGL should be compared with those strong alternatives at the executable level. [K06]

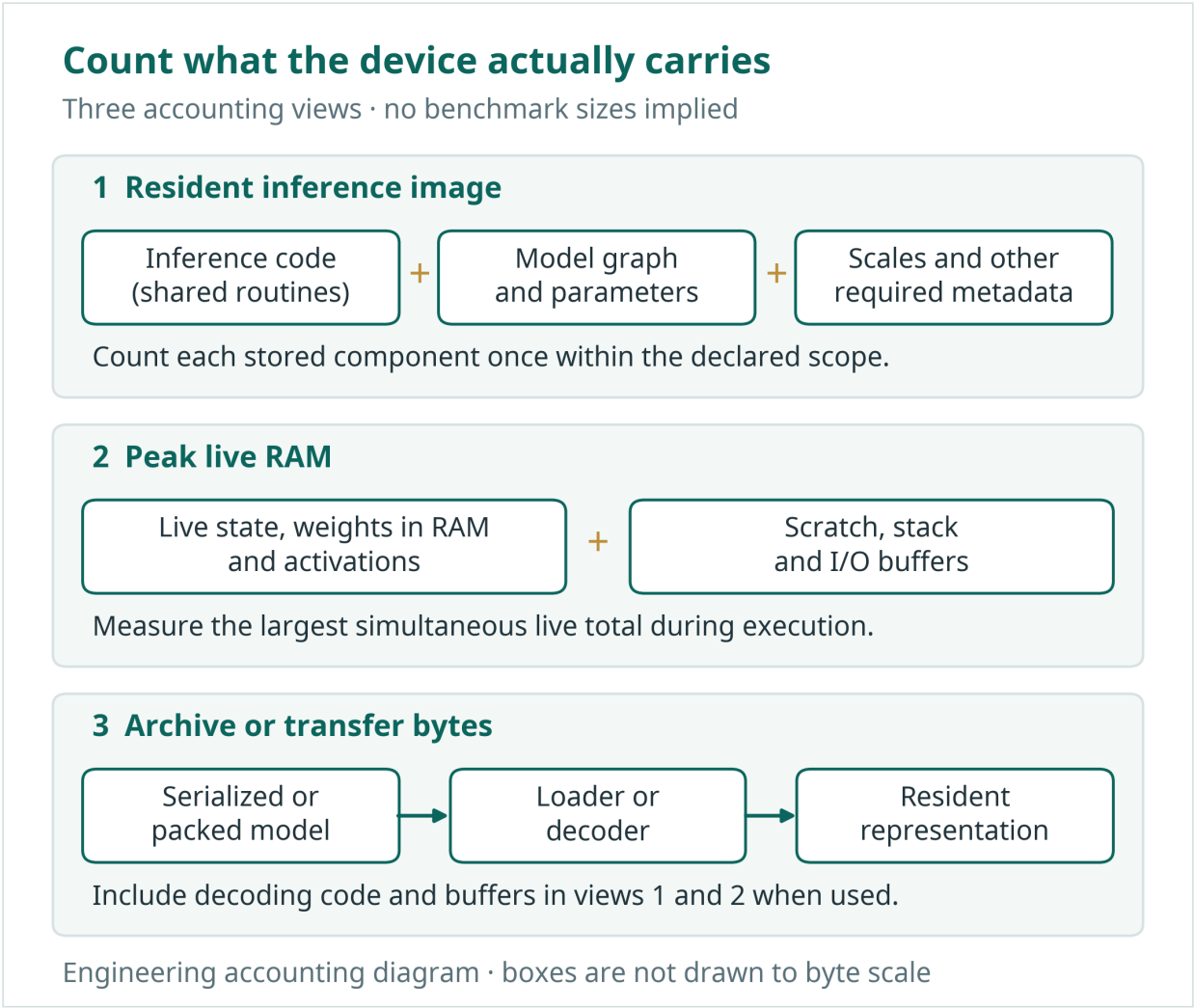


Figure 7. Three accounting views of the same executable model. Resident code and data, peak live RAM, and transfer/archive bytes answer different engineering questions. Shared routines are counted once within each declared deployment scope.

3.4 Turn compression into a loss-versus-bytes design frontier

Shannon's rate-distortion theory formalizes the rate needed to reproduce a source within a specified distortion. Model selection asks a different question, but the design lesson transfers: specify what errors matter before optimizing representation size. For an evaluated collection of implementations, define the following engineering frontier. [K03][4]

4. The measured frontier is empirical over the declared candidate set, not Shannon's rate-distortion function or a global optimum over all programs. Specify the task loss, data split and confidence intervals. [K03]

$$B^*(\varepsilon) = \min_{M \in \mathcal{C}:\, \widehat{R}_{\text{held out}}(M) \le \varepsilon} B_{\text{resident}}(M)$$

C is the declared candidate set; R̂ is task loss measured on held-out examples; ε is the accepted loss. Bresident counts resident model data plus its required inference code under the same accounting convention. An empty feasible set has no qualifying implementation.

Sweep topology, precision and training settings, then plot actual candidates. Reject implementations that exceed peak RAM, deadline or energy limits even when their static image is small. Use validation data to choose a candidate and a separate untouched test set to report its quality. An archived compressed file and a resident executable can have different sizes; unpacking buffers and sparse indices must be counted where they exist.

Include a well-tuned INT8 post-training baseline and quantization-aware training when needed, together with appropriate linear, tree or direct-rule alternatives. Jacob and colleagues demonstrate integer-only inference with a training procedure designed to preserve accuracy, including already efficient architectures. Low precision does not imply a universal large accuracy penalty. IGL's scale P should likewise be selected against task loss and actual storage and accumulator widths, rather than treated as a datatype name. [H17][5]

3.5 Make the compact representation valuable in a field device

A smaller working model can change the product boundary: it may fit into local memory, leave room for sensor history, or enable a decision during a communications outage. Shorter inference may permit a longer sleep interval; a smaller update may reduce transfer time. Memory movement can be a major energy cost, so the benefit can come from where data resides as much as from arithmetic count. These are measurable mechanisms, consistent with Horowitz's system-level energy analysis. [K08]

Engineering opportunity	Measure on the candidate device	Decision supported
Fit the complete inference image	Flash bytes for code, graph, constants, scales and metadata; compare the same scope.	Does the model fit the selected MCU without an extra storage component?
Preserve useful working memory	Peak live activations, scratch, stack and sensor buffers under the real schedule.	Can sensing, communications and inference coexist with required headroom?
Reduce time and energy per decision	Latency distribution and joules including preprocessing, memory traffic and wake-up.	Does the device meet its response deadline and duty-cycle budget?
Keep task distinctions in the field	Misses, false alarms or regression loss across held-out devices and conditions.	Is compactness achieved at the agreed application quality? [6]

The persuasive licensing result is a reproducible point on this frontier: a specified task, a working implementation, an accepted loss and a measured resource advantage. The laboratory demonstrates a representational mechanism; a field pilot establishes its value. A compact IGL model that preserves the needed distinctions while freeing memory, time or energy would give an engineering sponsor a concrete reason to proceed.

5. At P=1024 the laboratory permits integer weights from -3072 through 3072: 6145 values, requiring at least 13 bits in a fixed-width code. Outputs 0...1024 need 11 unsigned bits. Products and sums require separate bounds. [L01]

6. Report whole-image and incremental code costs when a runtime is shared. Measure energy on the selected device; historical memory-energy ratios are motivation, not predictions for the selected MCU or accelerator. [K08]

CHAPTER 04

Laboratory tutorials

4.1 Use the Lab as an experimental instrument

Open https://mliglon.com/igl-lab/ in a separate browser tab. Choose a Boolean preset in the left panel; adjust W_1,W_2,B,P and the Real/INT mode beneath it. The center shows the response field and an analysis chart. The right panel provides a live arithmetic trace, four-point truth table, a training tab and an embedded guide. Resizable splitters, focus view, fullscreen, appearance options and camera controls let the interface use the available browser area. [L02]

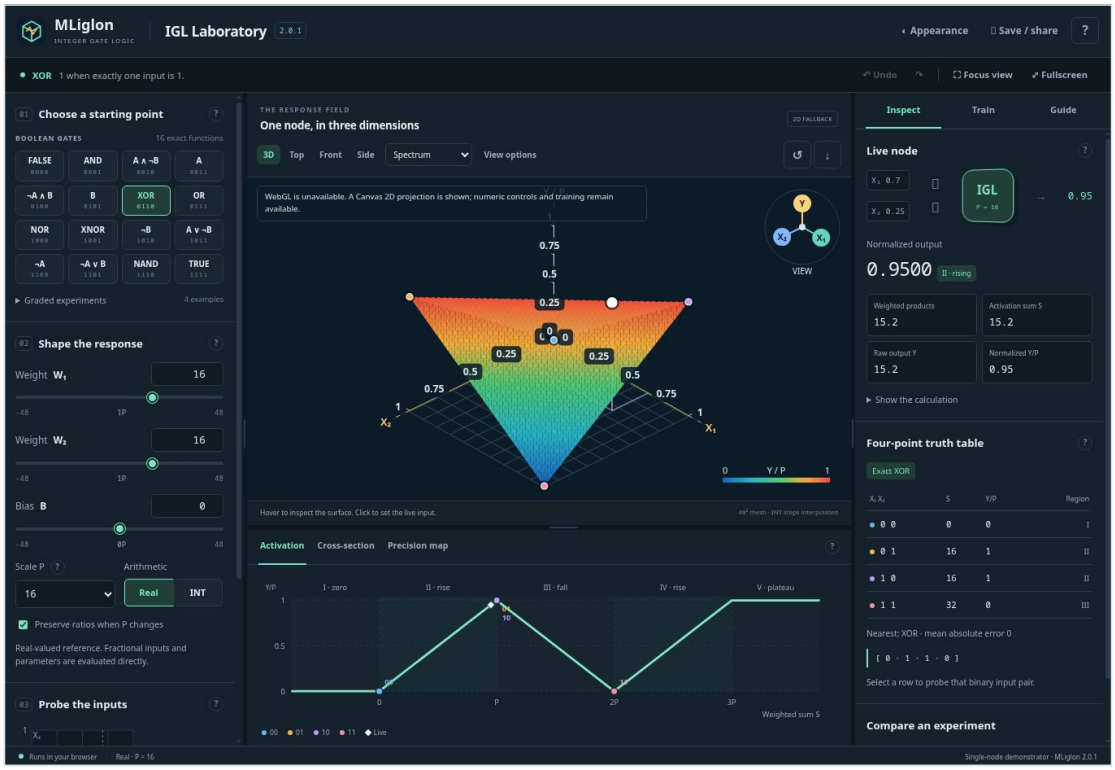


Figure 8. IGL Laboratory 2.0.1 workspace. Capture environment: Canvas2D fallback; GPU/WebGL operation and PNG export were separately confirmed by the installation owner. Numerical controls and teaching search are independent of the renderer.

For each exercise, set the requested arithmetic mode and P first, then select the preset last and verify the displayed parameters. This ordering avoids inheriting an unintended earlier scale or rounded parameter state. A preset changes the model but does not change the separately selected training target. Click a truth-table row to inspect that corner, then enter the fractional probe specified below. Read S and Y/P rather than inferring an exact value from surface color. Use Activation to connect the four sums to the five regions, Cross-section to inspect a one-dimensional cut, and Precision map for sampled INT-versus-real differences. [L02]

> **Capture provenance**
> New gate screenshots in this brief show the live 2.0.1 interface in its light theme. Crops retain original screenshot pixels and focus on verified numerical fields. Mathematical response surfaces are separately calculated from the released equations and labeled as illustrations. This avoids treating a browser rendering or interpolated mesh as numerical ground truth.

4.2 Tutorial 1: AND and a graded ramp

Set Real mode and P=16, then select AND. Confirm W_1=16, W_2=16 and B=-16. The corners yield sums -16,0,0,16 and normalized outputs 0,0,0,1. The bias delays entry into the rising region until the sum of the inputs exceeds one. At x_1=0.8 and x_2=0.6, the weighted products total 22.4; S=6.4; Y=6.4; q=0.4.



Figure 9. AND at (0.8,0.6), P16 Real: live arithmetic trace from the application.

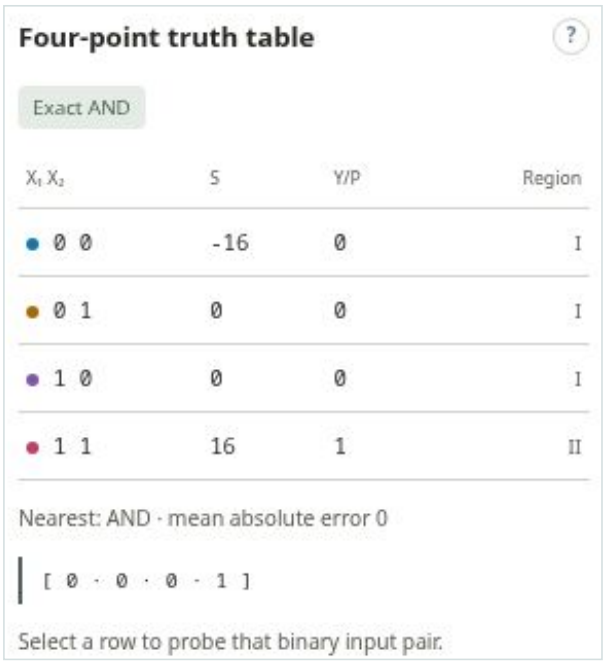


Figure 10. AND at (0.8,0.6), P16 Real: binary-corner verification from the application.

$$q = \max(0, x_1 + x_2 - 1), \qquad (x_1, x_2) \in [0, 1]^2$$

Because x_1+x_2-1 never exceeds one in this domain, only regions I and II are visited. This formula is specific to this parameter construction.

Try x_2=0.6 while sweeping x_1 from 0 to 1. The response remains zero until x_1 exceeds 0.4, then rises linearly to 0.6. This is a useful example of a thresholded conjunction-like feature with a graded output. It is not a probability of both conditions being true unless the surrounding application establishes that interpretation.

4.3 Tutorial 2: XOR, the falling segment and a conventional comparator

Set Real mode and P=16, then select XOR: W_1=W_2=16, B=0. Single active inputs land at S=P and output one. Both active inputs land at S=2P and output zero. Set x_1=0.7,x_2=0.25: D=S=15.2, Y=15.2 and q=0.95. Next set both inputs to 1 and watch the live marker reach the activation trough.

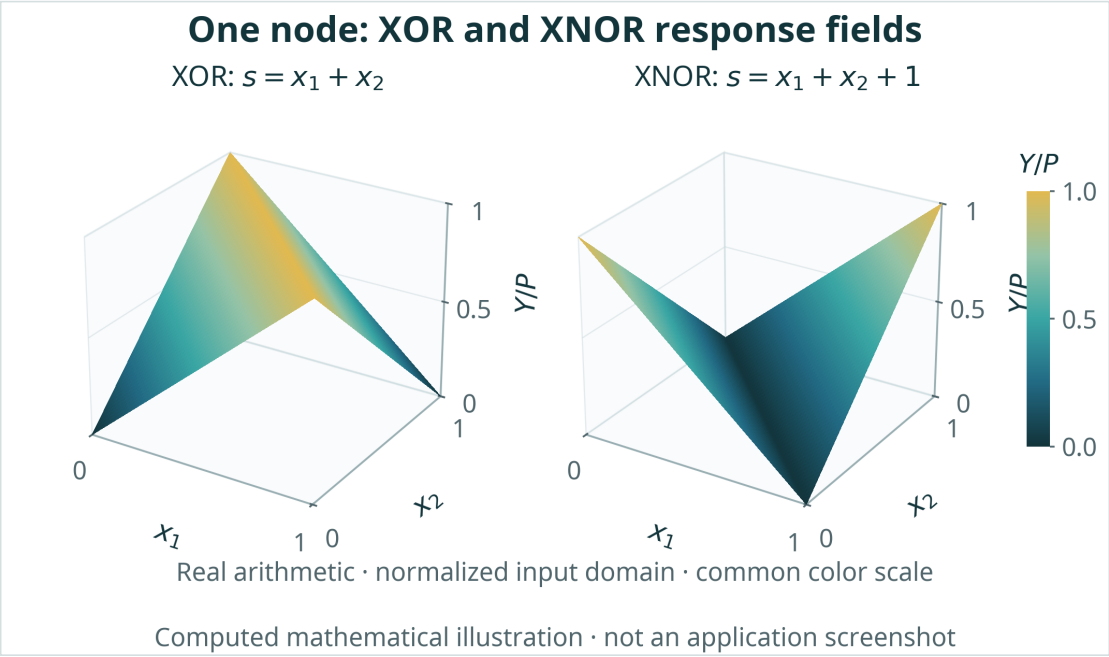


Figure 11. Calculated XOR and XNOR response surfaces. These show the full normalized domain, including graded values between the four Boolean corners. The surfaces come from the equations, not a captured GPU benchmark.

For this XOR construction, $q=x_1+x_2$ when the sum is at most one, and $q=2-x_1-x_2$ when it exceeds one. The conventional two-hidden-ReLU construction in the earlier history section produces the same response on this domain. IGL packages that folded map into one custom activation; this is a meaningful representation choice, not proof that conventional networks cannot solve XOR.

> **Exercise: locate a level set**
> Find all points for which XOR $q=0.75$. The two line segments are $x_1+x_2=0.75$ and $x_1+x_2=1.25$ inside the unit square. Two distinct input sums can produce the same output. When chaining nodes, ask which distinctions must remain available elsewhere in the network.

4.4 Tutorial 3: OR corners do not imply a monotone fuzzy OR

Set Real mode and P=16, then select OR. Its parameters are $W_1=W_2=32$ and B=-16. The four sums are -16,16,16,48, which produce the exact Boolean OR table 0,1,1,1. Now enter $x_1=x_2=0.75$. D=48 and S=32=2P, so the live output is zero even though both inputs are relatively high.

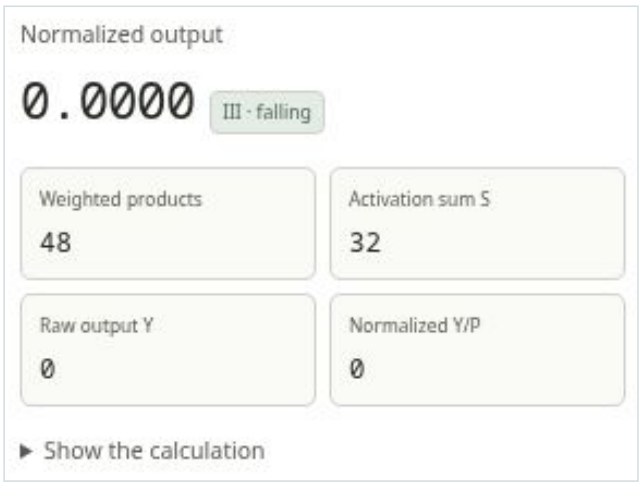


Figure 12. OR at (0.75,0.75), P16 Real: live arithmetic trace from the application.

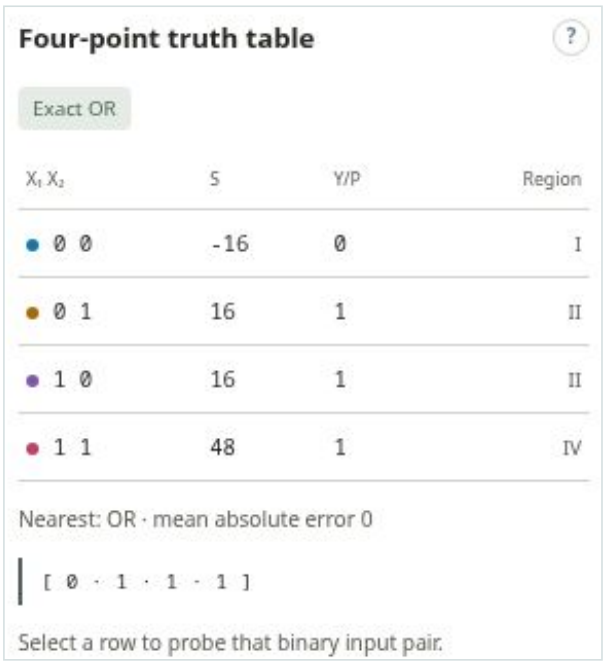


Figure 13. OR at (0.75,0.75), P16 Real: binary-corner verification from the application.

This result is not an arithmetic defect. The preset is designed to reproduce the four binary observations. The interval between the peak at S=P and trough at S=2P remains part of the function. Along the diagonal, q rises, falls, then rises again. If an application needs a monotone response to two intensities, four Boolean training examples are insufficient. A separate application's training set should include representative interior examples. Change the construction or architecture as needed, and evaluate the actual monotonicity requirement; the Lab itself remains a four-example demonstrator.

> **Design implication**
> A logic label describes observed corner behavior. A continuous or quantized sensor application needs a response specification over its relevant domain. This is precisely why interactive probing and dataset design matter more than a successful four-row demonstration alone.

4.5 Tutorial 4: bias shifts and sign reversal

XNOR at P=16 uses $(W_1,W_2,B)=(16,16,16)$. At (0.5,0.5), D=16, S=32 and q=0, while corners 00 and 11 both output one. Compared with XOR, the added bias moves every corner to the next activation feature. NAND uses (-32,-32,48), with corner sums 48,16,16,-16 and outputs 1,1,1,0. At (0.25,0.25), it too reaches S=32 and q=0.

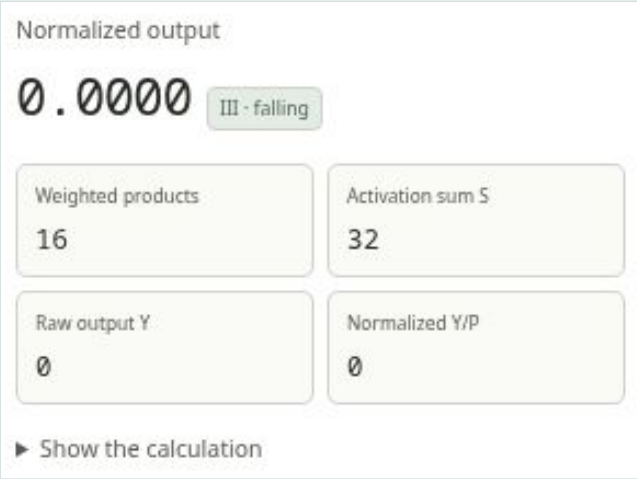


Figure 14. XNOR at (0.5,0.5): the bias puts the live sum at the falling-region endpoint.

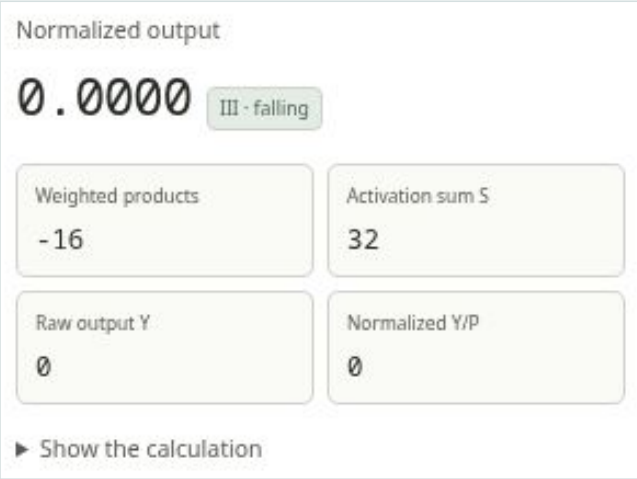


Figure 15. NAND at (0.25,0.25): negative weights and positive bias produce the same live trough.

The two nodes have different parameter signs and different corner truth tables but can share the same output at a chosen interior point. To understand a trained node, inspect its reachable input distribution and activation-region occupancy; parameters alone or a single example are incomplete descriptions. A network visualization could usefully report how often each region is visited, but that would be an additional diagnostic for a broader trainer, not a feature promised by the Lab.

4.6 Tutorial 5: inhibitory weights and asymmetry

Set Real mode and P=16, then select A AND NOT B. Here W_1=16, W_2=-16 and bias B=0. At (0.8,0.2), S=9.6 and q=0.6. Reversing the inputs gives S=-9.6 and q=0. Within the unit square, this construction computes $\max(0,x_1-x_2)$. A positive first weight and negative second weight form a simple competitive feature.

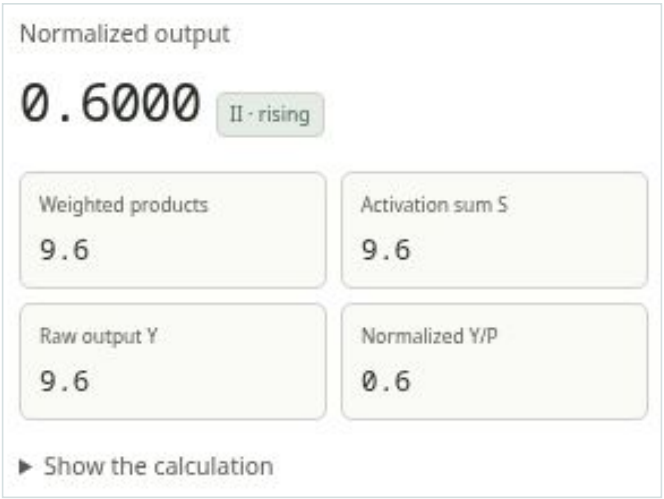


Figure 16. The inhibitory construction: the captured live trace verifies 9.6 raw output and 0.6 normalized output at (0.8,0.2).

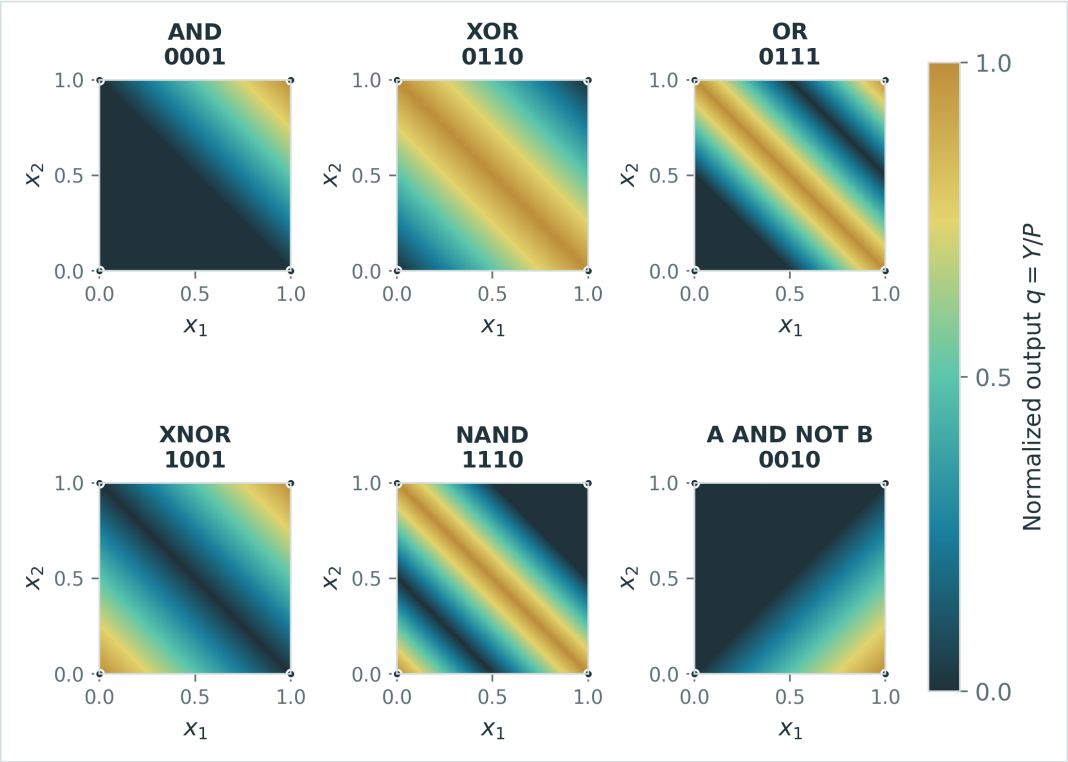


Figure 17. Six calculated response fields on a common input domain and color scale. Exact corners coexist with different interior geometry. Use this gallery to connect parameter signs and bias to response shape.

As an exercise, hold x_1=0.8 and sweep x_2. Predict the kink at x_2=0.8 before moving the input control. Then exchange W_1 and W_2: the useful side of the diagonal reverses. This is a small example of engineering an interpretable primitive; learning a reliable detector from real sensor data still requires coverage of confounders, noise and operating conditions.

4.7 Tutorial 6: measure the effect of P

Set Real mode and P=4, enable “Preserve ratios when P changes,” then choose the graded Asymmetric preset. In Real mode its coefficients W_1/P=1, W_2/P=0.5, B/P=0 give q=0.71 at requested inputs (0.38,0.66). Choose INT and re-enter those requested inputs. The stored coordinates become (0.5,0.75), represented by integers (2,3); with weights (4,2), D=14, S=round(14/4)=4, and q=1. The change illustrates input quantization and normalization rounding together.

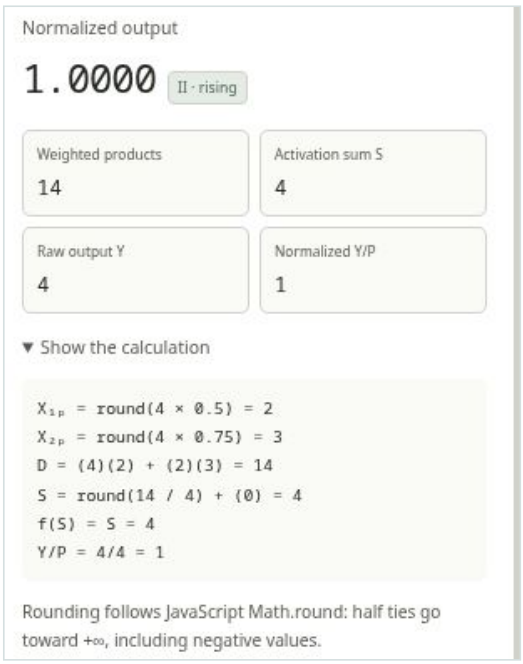


Figure 18. P4 integer trace. The requested coordinates have snapped to the available input levels.

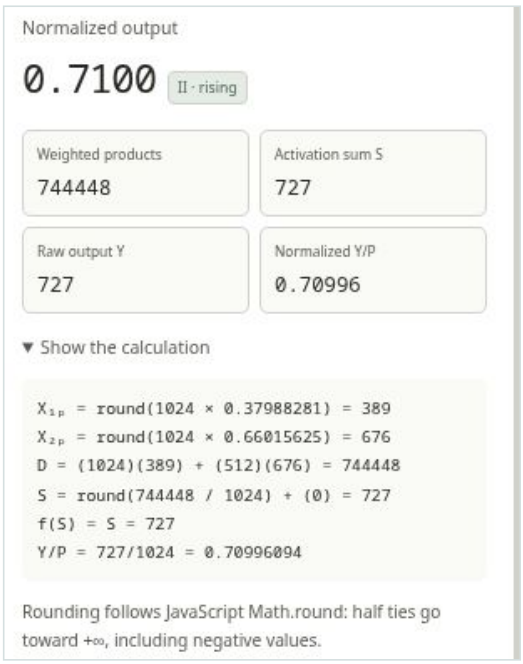


Figure 19. P1024 trace after re-entering the original requests: output 0.7099609375, displayed rounded.

Condition	Effective coordinates	D, S	q
Real, requested values	0.38, 0.66	0.71P, 0.71P	0.71
INT P4	0.5, 0.75	14, 4	1
Raise P to1024 without re-entry	0.5, 0.75	917504, 896	0.875
INT P1024, re-enter requests	389/1024, 676/1024	744448, 727	0.7099609375

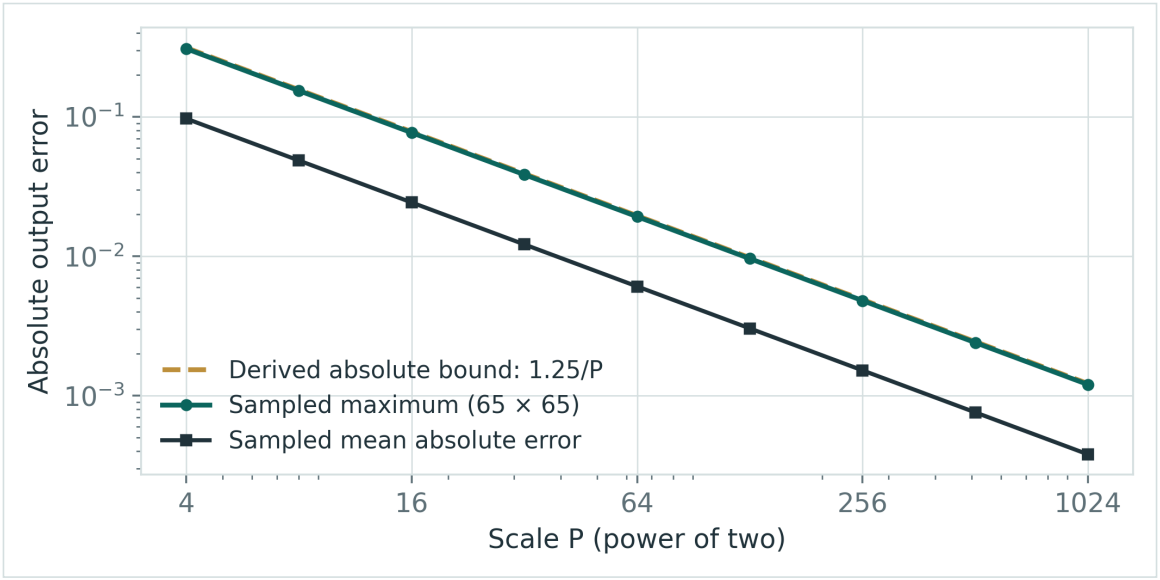


Figure 20. Asymmetric node: sampled absolute error decreases with increasing P. The dashed curve is the derived worst-case bound, not a measured hardware result. Values compare identical stored parameters with real and integer arithmetic.

Use Precision map to locate rounding-sensitive areas and cross-check points in the trace. Its midpoint grid samples differences; choose P using task-level validation, sensor noise and numerical tolerance. [L02]

CHAPTER 05

Training and GPU parallelism

5.1 What the single-node teaching search optimizes

The Lab searches only W_1,W_2 and B, with the chosen P, arithmetic mode and four binary input pairs held fixed. Its loss is normalized mean squared error against a selected gate target. It does not train a dataset-driven network, propagate gradients or run a CUDA candidate evaluator. Rendering a surface with WebGL is a separate workload from the JavaScript teaching search. [L01, L03]

$$E(W_1, W_2, B) = \frac{1}{4} \sum_{k=1}^{4} \left(q(x^{(k)}; W_1, W_2, B) - t_k\right)^2$$

The four examples are 00,01,10,11. MSE penalizes large output deviations more heavily than small ones. “Nearest gate” in Inspect instead uses mean absolute error; it is a different metric.

A local step proposes positive and negative changes to one parameter at a time, within the ±3P bounds, and accepts an improving candidate. INT uses unit steps by default; Real uses P/4. The multiscale run searches coarse-to-fine changes, includes joint changes, and can restart from seeded alternatives. A candidate evaluation is not an epoch, and an iteration may evaluate several candidates. The application stops a solved run when MSE$\leq 10^{-12}$. [L01, L03]

A controlled search comparison
Set P1024 INT, target TRUE, start W_1=W_2=1536, B=-1536, seed 2026, candidate budget 100000 and restart limit 32. Restore exactly this start before each method. Keep the same target and count candidate evaluations rather than comparing a variable-duration screen animation.

5.2 Tutorial 7: plateau, restart and reproducibility

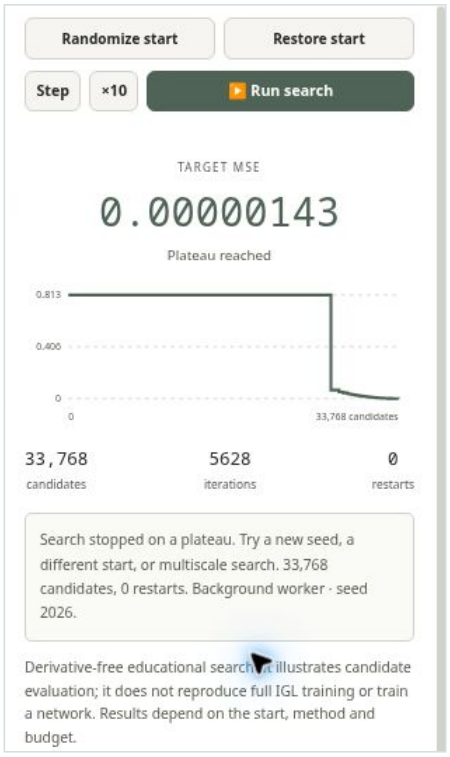


Figure 21. Local search reaches a nonzero plateau after 33,768 candidate evaluations in this reproducible teaching run.



Figure 22. From the same restored start, multiscale search reaches zero MSE after 816 candidate evaluations.

Result	Local	Multiscale
Evaluated candidates	33,768	816
Iterations; restarts	5,628; 0	36; 3
Final (W_1,W_2,B)	(1,2,1022)	(1024,0,3072)
Final MSE	$1.430511474609375\times10^{-6}$	0

The local result has three nearly correct outputs and one exact output; lack of a sufficient one-step improvement is not proof of global optimality. The multiscale result is one valid TRUE solution, not the unique solution. This example explains why search neighborhoods, ties, restart policy and budgets belong in an experimental record. It does not establish a universal ratio between optimizers or a production training speedup. [L01, L03, L04]

Save the starting experiment and final parameters as JSON, the four rows as CSV, and any surface illustration as PNG. Record version, mode, P, target, strategy, seed, limits, start and stop reason. A seed makes this defined search reproducible; it cannot compensate for a changed implementation, arithmetic mode or dataset. A broader GPU trainer also needs deterministic reduction and update rules if exact replay is a requirement.

5.3 From one node to an affected dependency subgraph

For a general directed acyclic network, changing a parameter affects its owner operation and all downstream outputs that depend on it. Other cached activations remain valid under the same model, input and preprocessing state. MLiglon describes using this principle to evaluate proposed changes by recomputing the affected dependencies, then committing a selected improvement and refreshing the affected reference cache. The affected structure can branch; it is not always a single chain. [I05]

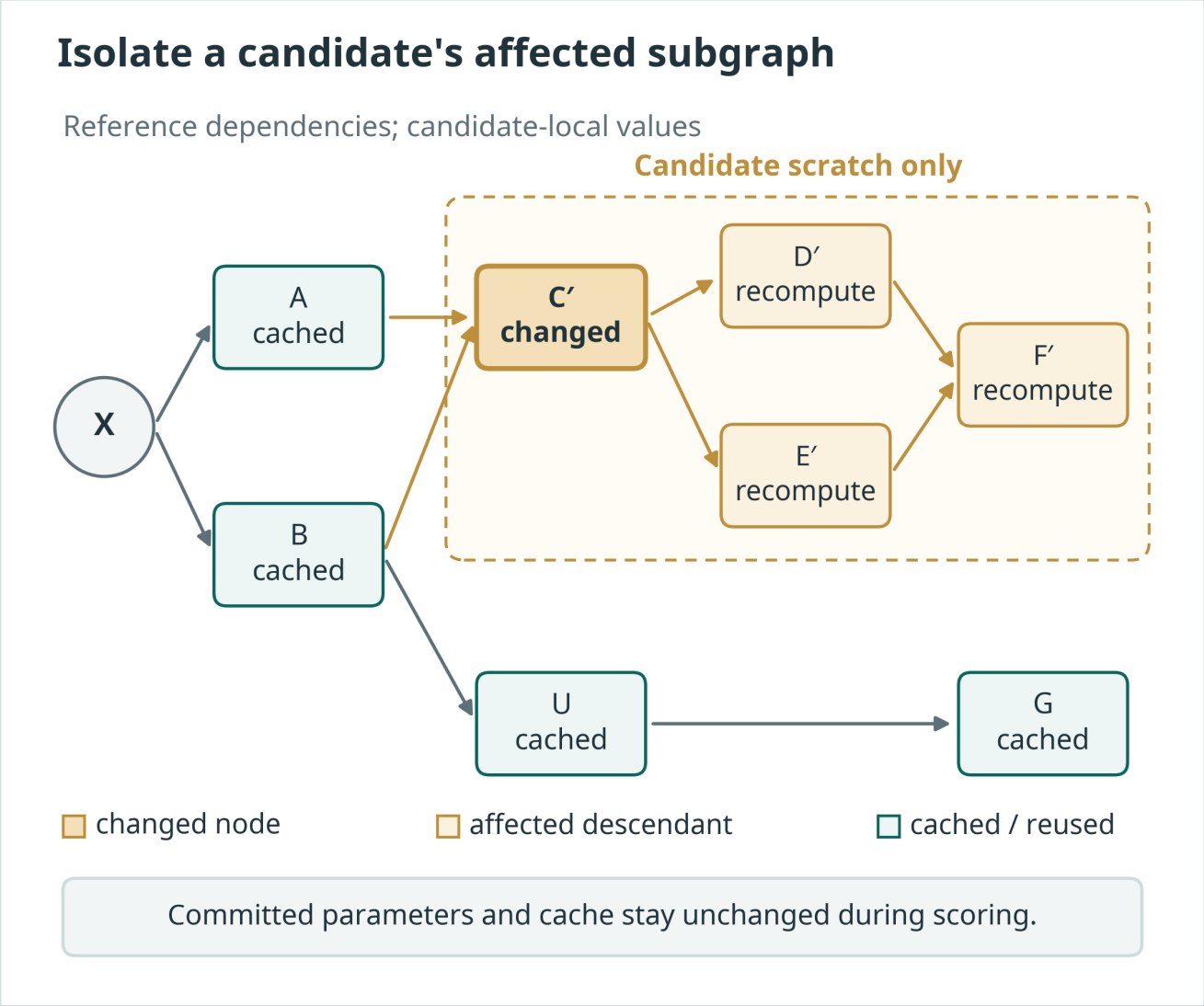


Figure 23. Conceptual dependency reuse. Cached values outside the affected descendants can be reused; every candidate needs state consistent with the same reference model. This illustration is an engineering explanation, not a screenshot of released network-training software.

$$C_{\text{score}} \approx KM\bar{A} + C_{\text{read}} + C_{\text{reduce}} + C_{\text{control}}$$

Illustrative work model: K candidates, M examples, and average affected-operation cost $\bar{A}$ per candidate/example. Memory reads, loss reduction and search control remain real costs; this is not a timing prediction.

Against K full candidate forward passes costing roughly KMN operations, a small affected region $\bar{A}$ relative to full-graph work N creates an opportunity. Caching, indexing and irregular execution can consume that saving. Early nodes, dense connectivity and shared convolution kernels may affect much of the graph. Batching compatible candidates or samples may improve utilization, while excessively large candidate batches can exhaust memory. Measure both useful work and overhead.

Two candidate updates evaluated separately against the same old model can interact after either is committed. Accepting both without re-evaluation is not justified merely because their evaluations ran concurrently. A trainer may serialize commits, prove independence, or validate a joint update; its chosen consistency rule should be documented and tested. A rejected candidate must not alter authoritative parameters or contaminate caches.

5.4 A GPU work decomposition that an engineer can evaluate

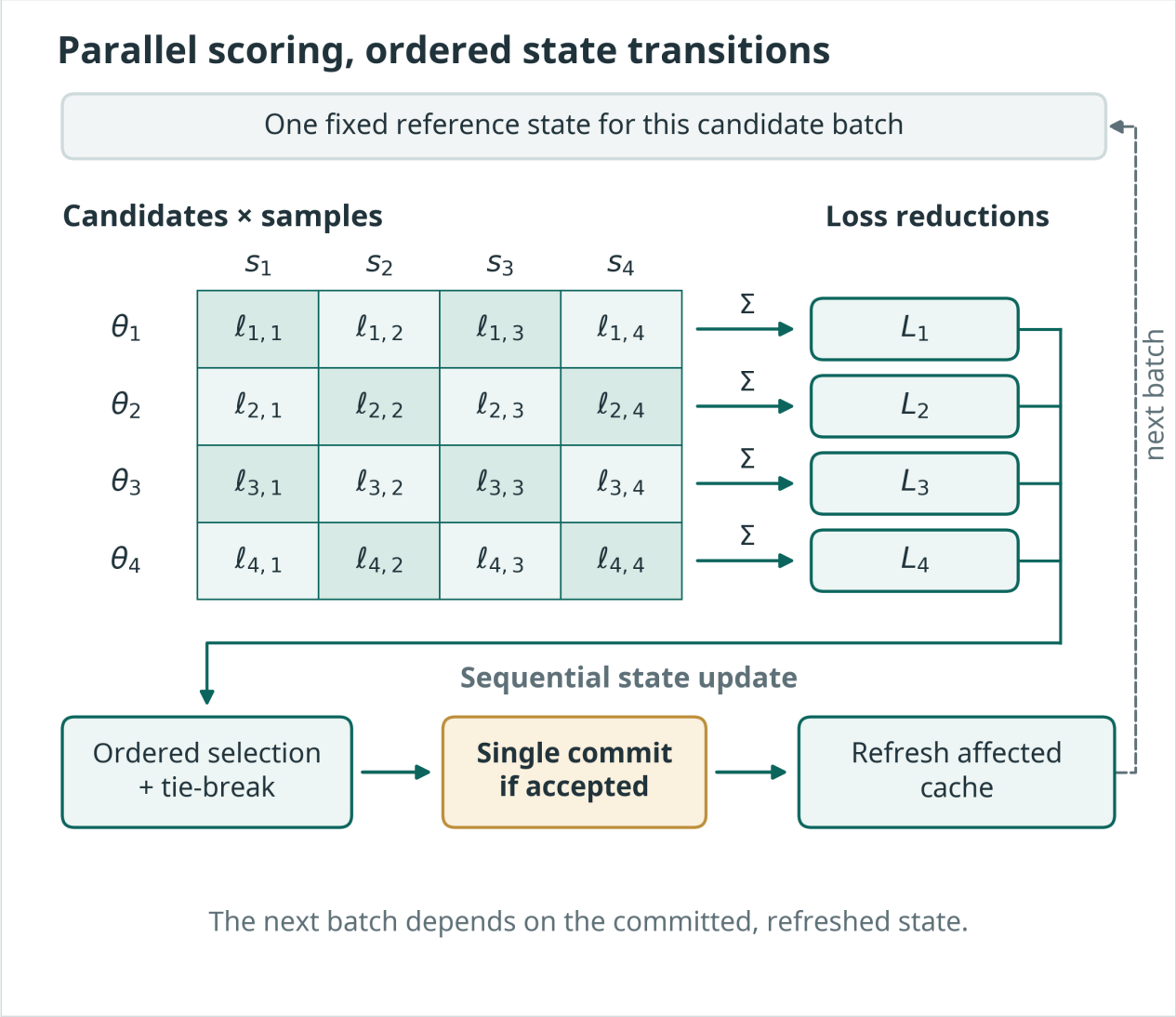


Figure 24. Conceptual candidates-by-examples scoring, reduction and consistent commitment. Parallelism lives inside the scoring stage; the search controller must respect dependencies between accepted model states.

A useful prototype assigns adjacent lanes to contiguous samples for one candidate, or to contiguous candidate/sample tiles when that improves reuse. Candidate descriptors can be small while reference activations are large. Cache design should therefore account for both computation saved and bytes read. Reduce per-example errors into one loss per candidate, select using a documented tie rule, commit, and update affected caches before dependent scoring begins.

For integer output comparisons, accumulating squared integer errors with a sufficiently wide integer accumulator can avoid one source of floating reduction-order variability. That is only valid when all terms share a compatible scale and the accumulated range is proved. Mixed scales, normalized losses or regularization require a carefully defined alternative. GPU efficiency also depends on occupancy, register pressure, branch divergence and launch overhead; the following hardware chapter connects this decomposition to actual ISA and memory constraints.

Training deliverable versus deployment deliverable
A host trainer may retain datasets, candidate buffers and activation caches far larger than the target MCU can hold. The deployed artifact can contain only frozen parameters, topology, scales, preprocessing constants and forward kernels. MCU inference and MCU training are separate requirements. MLiglon has indicated plans for broader training software. Contact MLiglon for current availability, supported targets, model export and licensing. The deployment chapter sets out the interfaces and evidence to request. [I12]

5.5 GPU evaluation: parallel work inside a sequential search

A separate GPU trainer could score independent candidates and examples concurrently. It still has an ordered search process: evaluate a neighborhood, select an accepted result, update step/state, and generate the next work. Preserve candidate order, tie handling, stopping and seeded restarts when claiming the same algorithm. Speculative scoring may perform work the original would stop before evaluating; include that work in accounting and do not silently redefine candidate counts. The released laboratory performs none of this on a GPU. [L01, L03]

CUDA exposes SIMT execution with 32-thread warps. Different per-thread activation branches can reduce active-lane efficiency; explicit comparisons and selection are another implementation to test. Register and shared-memory usage also constrain residency. These architecture facts motivate benchmarking several mappings rather than selecting one thread per mathematical term by habit. For the current two-input/four-example problem, a thread that evaluates a complete candidate may avoid more reduction overhead than a cooperative mapping saves. [T13]

For larger candidate batches, a structure-of-arrays coefficient layout can make candidate-adjacent lanes access neighboring weights. If lanes instead traverse examples, place that dimension contiguously and reuse the candidate parameters. Shared-memory staging is justified only by measured reuse or access improvement. Keep persistent datasets and scratch buffers on the device when appropriate; report transfer-inclusive latency separately from kernel time. NVIDIA’s coalescing guidance supplies the memory principles, while these mappings are proposals to test. [T14]

Tensor Cores are not an automatic destination for integer code. NVIDIA’s WMMA table includes signed/unsigned INT8 multiplicands with INT32 accumulators at supported tile shapes. Its compute-capability table makes datatype support architecture-specific. An IGL node also requires scaling, rounded normalization and a non-monotone piecewise activation; matrix multiplication alone does not implement it. Full-range P1024 weights require more than INT8, and standard INT8 WMMA is therefore not a direct representation. Padding or decomposing a two-term product can erase the expected benefit. [T15][T16]

Reproducibility needs explicit reduction rules. Floating-point summation order and FMA can change results and, near an acceptance threshold, the search trajectory. A deterministic reduction and tie-break scheme are part of the implementation, not just the seed. For fixed-P integer output and Boolean targets, a bounded integer sum of squared raw-output errors can rank candidates before final normalization; that is an exact-arithmetic design opportunity, provided its bound and acceptance equivalence are proved. Validate the result against the scalar oracle. [T17]

CHAPTER 06

Precision, memory and the numerical contract

6.1 Begin with the numerical contract and a fair baseline

A hardware port begins with an executable numerical specification: input encoding, parameter bounds, product width, accumulation width, rounding, activation boundaries and output encoding. The released laboratory evaluates a two-input node and scores four Boolean examples in JavaScript; its worker performs CPU search, while WebGL renders the scene. The hardware mappings below are engineering proposals for separate implementations, not measured capabilities of the browser application. [L01, L03]

Compare two different questions explicitly. A same-function implementation study holds the IGL node, inputs and accuracy requirements fixed and changes its arithmetic/kernel implementation. An algorithm study compares IGL with another model or training method at matched task quality. The first can establish an implementation advantage; it cannot establish superior learning quality. Conversely, a smaller representation that changes the function requires an accuracy comparison as well as a timing comparison.

FP32 is a credible optimized baseline on hardware that supports it. INT32 and FP32 both occupy four bytes per stored scalar, so converting one to the other does not itself reduce array traffic. INT16 and INT8 can reduce stored bytes, but widening, unpacking, rounding, nonlinear activation and output conversion remain part of the kernel. TI explicitly documents 32-bit float and 32-bit int on C7000; use exact-width integer types rather than assuming that long has the same width across platforms. [T02]

Representation	Packed storage for N values	Question that still needs measurement
FP32	4N bytes	Does hardware floating point and compiler vectorization make the full kernel efficient?
INT32	4N bytes	Does exact integer arithmetic simplify this workload despite unchanged storage size?
INT16	2N bytes	Can the operands stay packed while products and sums safely widen?
INT8	N bytes	Can the chosen scale and parameter domain fit without changing the model?

The byte counts are derived payload sizes, excluding alignment, metadata, padding and work buffers. A fair benchmark includes an optimized FP32 implementation with documented contraction/fast-math settings, plus an exact integer reference. Floating-point reassociation and fused multiply-add can change rounding, so numerical equivalence must be checked at activation boundaries and during loss comparisons. The NVIDIA floating-point guide explains those mechanisms; it does not establish a speed advantage for either representation here. [T17]

Binary32 has 24 significant binary digits including its implicit leading bit, so every integer through 2^{24} is exactly representable; it also represents a wide range of fractional and very large values with finite precision. INT32 represents every integer in its signed range but needs an external scale for fractions. For the Lab's two-input P1024 contract, the integer dot bound 6,291,456 is below 2^{24}; integer arithmetic should therefore not be promoted as an automatic accuracy advantage over a correctly ordered FP32 realization of this particular bounded integer calculation. Wider fan-in or other operations require fresh analysis. [T17]

6.2 P is a scale, not a storage type

In the laboratory, P controls quantization and activation scale. It does not select an eight-, sixteen- or thirty-two-bit machine datatype. For the full allowed parameter domain, $|W_1|$, $|W_2|$ and $|B|$ are at most 3P, and scaled inputs X_i are integers from zero through P. A narrower representation is valid only if it preserves the required domain; fitting one chosen preset is a weaker claim than supporting every legal parameter setting. [L01, L03]

$$|D| = |W_1X_1 + W_2X_2| \le 6P^2, \qquad |S| \le 9P$$

Derived two-input bounds. Here S = round(D/P) + B, with the laboratory rounding rule. Bias is added after the dot-product normalization.

For an exact signed w-bit storage type, the positive limit is 2^(w-1)-1. Consequently, storing every allowed weight and bias requires 3P ≤ 2^(w-1)-1. Restricting P to powers of two gives P ≤ 32 for INT8 and P ≤ 8192 for INT16. The latter is an analytical storage limit beyond the laboratory UI ceiling of 1024. Exact-width C types and their limits are specified in the WG14 draft; the P thresholds are our derivation from those limits. [T18]

Scale P	Weights / inputs	Maximum |D|	Implementation consequence
32	|W|, |B| ≤ 96; X ≤ 32	6,144	INT8 storage is possible; INT8 products/accumulation are insufficient.
1024	|W|, |B| ≤ 3,072; X ≤ 1,024	6,291,456	INT16 operands with INT32 products/sum cover the laboratory domain.
8192	|W|, |B| ≤ 24,576; X ≤ 8,192	402,653,184	INT16 operands still fit; S can reach 73,728 and requires a wider intermediate.

The bounds cover the dot product, not every possible reformulation. Folding bias into an accumulator before normalization introduces B·P and changes its bound to $9P^2$. Squared errors, batches and additional channels need their own bounds. In an illustrative nine-tap extension with the same per-weight and input limits, $|D_9| \le 27P^2$; at P=1024 this is 28,311,552. That is an extension calculation, not an operation performed by the two-input laboratory.

Promote operands before multiplying and prove every intermediate, not merely the final stored output. A saturating output activation cannot repair an earlier overflow that moved S into a different region. Standard signed C overflow is undefined; a hardware overflow flag, wrapping instruction and saturating instruction are different contracts. Apply saturation only where the specification requires it. Keep the power-of-two division optimization separate from the proof of signed rounding, especially for negative half-way cases. [T18]

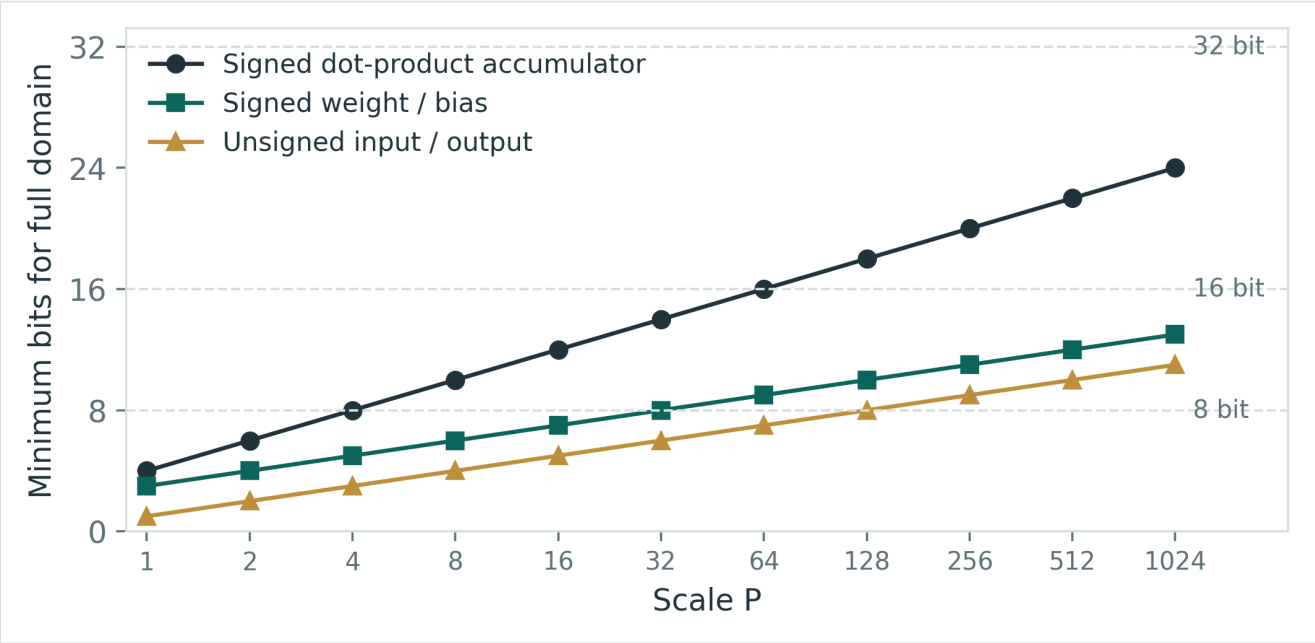


Figure 25. Derived minimum bit counts for the full Lab domain. Input/output counts are unsigned; weights and dot accumulators are signed. The horizontal lines are common storage widths, not speed or energy measurements.

6.3 Separate storage, buses, registers and accumulation

There are at least four widths in a deployment: stored element width, memory-transfer width, architectural vector/register width and arithmetic-result width. They need not match. A byte array can feed a word accumulator; a wide vector can arrive through several memory beats; and a smaller on-disk model can require the same accumulator-register capacity. Report each width separately instead of labeling the entire implementation simply “8-bit.”

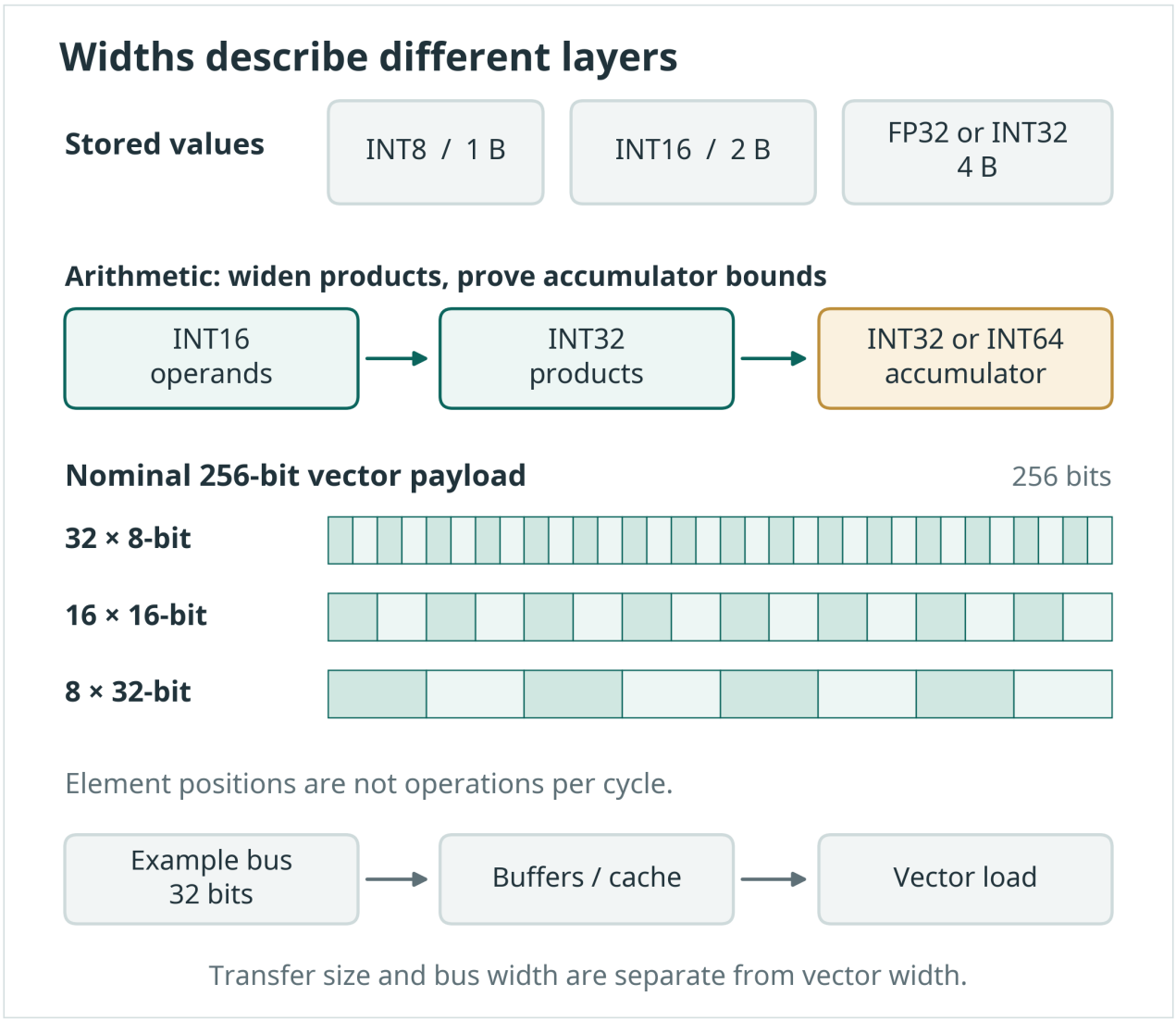


Figure 26. Stored operands, widened arithmetic, vector payload and memory transfers are different widths. A 256-bit register can hold 32 eight-bit, 16 sixteen-bit or eight thirty-two-bit elements; this counts positions, not operations per cycle. [T01-T03]

For an ideal aligned 32-bit payload transfer, four INT8 values, two INT16 values or one FP32/INT32 value fit in the same four bytes. This is a packing calculation, not a statement that every processor performs that many multiplies per cycle. Actual traffic depends on transaction boundaries, cache lines, bursts, reuse and padding. A one-node model is so small that its arithmetic may be dominated by call, scheduling or instrumentation overhead; use realistic batches when evaluating throughput.

Choose layout around the dimension processed together. With many independent node instances, structure-of-arrays storage for W_1, W_2 and B lets adjacent lanes load adjacent coefficients. With one model applied to many samples, keep the model reusable and place the sample dimension contiguously for the selected loop. Tiling should preserve useful reuse rather than repeatedly repacking the same values. Tail elements require predication, scalar cleanup or explicit valid-length masks; padding bytes must never become extra examples.

Treat alignment as an ABI and load-instruction requirement. A packed C structure is not automatically a good SIMD buffer: it can misalign members, increase stride and force extraction. TI documents scalar alignment and structure padding in its compiler guide. Use explicit typed buffers, declared alignment where required, documented strides and size assertions, then inspect the generated loads. When DMA participates, include transfer setup, buffer ownership and cache-maintenance costs in the integration design. [T02]

On CUDA, memory savings require useful transactions, not merely small types. NVIDIA documents 32-byte global-memory transaction units for compute capability 6.0 and later; adjacent four-byte words across a warp can be served by four such transactions. That concrete example motivates contiguous lane access. Our proposed IGL layout should be checked with profiler transaction counts, particularly after introducing candidate-by-example batches or nontrivial strides. [T14]

6.4 A portable rounding and overflow contract

The helper below implements exact integer division rounded to nearest, with ties toward positive infinity. It matches the Lab's Math.round(D/P) over its proven bounded domain; it does not emulate binary64 arithmetic for arbitrary 64-bit integers. C integer division truncates toward zero; converting that quotient to a floor quotient yields a remainder in [0,P). The final comparison below implements ties toward positive infinity. Use an adequately wide type for D, B and all intermediates, and verify the stated preconditions before narrowing. [L01, T18]

```
/* Preconditions: 1 <= p <= INT64_MAX/2;
   d and the later q+b are within proved bounds. */
static int64_t igl_round_div(int64_t d, int64_t p) {
    int64_t q = d / p;
    int64_t r = d % p;
    if (r < 0) { --q; r += p; }
    return q + (2*r >= p);
}
```

Illustrative C reference arithmetic. Include <stdint.h>. This fragment specifies rounding; it is not a complete inference runtime.

D / P case	Required result	Reason
-6 / 4 = -1.5	-1	Tie toward +∞
-2 / 4 = -0.5	0	Tie toward +∞
2 / 4 = 0.5	1	Tie toward +∞
6 / 4 = 1.5	2	Tie toward +∞
Any integer / 1	Unchanged	No fractional remainder

At P1024 with the full Lab bounds, each weight/bias needs 13 signed bits to include ±3072; each input/output needs 11 unsigned bits to include 1024, or 12 signed bits. The two-term dot product needs 24 signed bits to include ±6,291,456. INT16 storage with INT32 intermediate arithmetic is therefore a natural implementation candidate. A small final output does not justify a small intermediate.

On an implementation with 16-bit int, multiplication of two int16_t operands may overflow before assignment to int32_t. Cast an operand to the intended accumulator type before multiplying. A proof should cover the original form, any compiler intrinsic and any optimized reformulation. For $P=2^k$, a signed shift replacement may be worthwhile only after its rounding and overflow equivalence are established for every reachable D. [T18]

6.5 A defensible deployment and measurement plan

Separate training placement from inference placement. A host CPU or GPU can explore candidate parameters and export a fixed model; the MCU or DSP can then execute only the validated inference kernel. Alternatively, local adaptation is a different system requirement with its own memory, scheduling and power budget. Neither arrangement follows automatically from the browser demonstration. Define the intended sample rates, deadlines, batching opportunities and allowed numerical error before choosing a kernel.

Evidence to record	Reason
Exact part, board and enabled core features	Cortex-M, C7x and CUDA family labels omit implementation-specific capabilities.
Compiler/library versions, target flags and assembly reports	A source datatype alone does not reveal the instructions, widening or vectorization used.
Model/input ranges, P, layout, padding and intermediate widths	The kernel must implement the same numerical problem and avoid overflow.
Reference comparisons and boundary/tail tests	Measure correctness at activation transitions, negative ties and non-vector-multiple lengths.
Kernel and end-to-end timings; repeated trials	Expose setup, conversion, DMA/transfer, synchronization and scheduling overhead.
Peak working memory, flash footprint and measured energy	Storage payload, scratch memory and system energy are separate quantities.

Start with a scalar reference, explicit bounds and reproducible vectors. Optimize a measured bottleneck. Compare cold/warm data where relevant, realistic batch sizes, worst-case latency and throughput; include preprocessing, input conversion and output normalization in end-to-end measurements.

State whether the FP32 baseline implements the continuous Real model or emulates the integer contract. State whether narrower integer storage restricts parameters. Report error, latency, memory and energy together; distinguish measurements from analytical bounds and hypotheses. Vendor peak-operations figures do not establish IGL performance.

Deliver a reproducible package: operator contract, typed interfaces, range proofs, reference vectors, source, compiler settings, memory map and measurement scripts. GPU evaluations also require candidate-order and reduction tests. This makes the target implementation reviewable.

CHAPTER 07

Texas Instruments: installed MCUs and the DSP path

7.1 Add a bounded decision to the device already installed

An attractive field opportunity is to add a small, trained decision function to a sensor or controller that already collects the necessary signals. Keeping the installed board can preserve its enclosure, wiring, analogue interfaces, service procedures and established control firmware. The first engineering question is whether a useful decision fits the remaining time, memory and energy budgets. A new processor is not a prerequisite for testing that question. [7]

Candidate applications include two-sensor plausibility checks, temperature-dependent alarms, pump current versus operating state, and slowly changing battery or environmental conditions. A few window statistics may be enough to distinguish useful operating states. These are proposed evaluation cases, not demonstrated IGL products. Establish error costs, representative field data and a baseline using the existing thresholds or a compact conventional classifier. A single two-input node may suit one decision; additional nodes or a network require a separately specified model and runtime.

Training can run on a host workstation. The installed MCU receives frozen parameters and executes preprocessing, inference and decision policy. It need not retain training examples, candidate populations or optimizer state. On processors without an FPU, floating-point calculations use software support; avoiding unnecessary conversions is therefore a credible optimization opportunity. TI documents such helper functions, but this does not prove that IGL outperforms every fixed-point alternative. Compare complete implementations on the same board. [V04, V07]

7.2 Inventory the exact TI part and its arithmetic resources

The following examples distinguish an older 16-bit MCU, a particularly small MCU, and a Cortex-M0+ device. Memory capacities are totals for the named parts, not capacity available to a new feature. None of these three CPUs has a hardware FPU. That statement must not be generalized to all TI microcontrollers. [8]

Exact TI part	CPU / hardware FPU	Documented memory	Integer execution resources
MSP430F1611	16-bit MSP430; no FPU	48 KB main flash + 256 B information flash; 10 KB RAM	16×16 multiplier peripheral with signed/unsigned multiply and accumulate. [V01, V06]
MSP430G2553	16-bit MSP430; no FPU	16 KB main flash; 512 B RAM	No multiplier peripheral in the device map; use the software multiply path. [V02, V08]
MSPM0G3507	32-bit Cortex-M0+; no FPU	128 KB flash; 32 KB SRAM	MATHACL includes division, 32-/64-bit-result multiply and MAC functions. [V03, V04]

7. This opportunity assumes the device can be reprogrammed and the existing firmware can be rebuilt or extended through a supported interface. It does not assume spare memory, a remote update channel, or permission to alter certified behavior. Sensor quality and field labels may be the limiting resources even when the arithmetic fits.

8. The MSP430 conclusions use the named device maps plus the complete MSP430 CPU instruction descriptions and software floating-point ABI. The Cortex-M0+ conclusion is explicit in TI's compiler documentation. A multiplier, MAC or MATHACL peripheral is not an IEEE floating-point unit. Memory figures retain TI's KB notation and do not include available-space claims. [V01, V02, V03, V04, V06, V07, V08]

Older Cortex-M0 and Cortex-M3 systems also deserve consideration: neither core has an FPU option in TI's compiler target documentation. Their instruction capabilities are not identical. Arm documents alternative fast and small multiplier implementations for Cortex-M0, so even an integer multiply needs device-specific interpretation. A 32-bit result on a 16-bit MSP430 can require several instructions and register transfers. A library call, a CPU instruction and a memory-mapped arithmetic peripheral are different execution paths. [V04, V05, V06, V07]

The MSPM0G3507 illustrates why no FPU does not mean no accelerator. Its MATHACL is useful to investigate, but register access, operand formats, synchronization and other firmware users still matter. Confirm the emitted instructions and library selection before timing a kernel. [V03]

7.3 Train off-device and export the whole integer contract

Build the host reference from the sensor's actual output codes. For each input, define calibration offset, gain, valid range, clipping, missing-sample behavior and the scale delivered to the model. Running sums, differences, extrema and fixed-point gain/offset transforms can form a compact preprocessing pipeline. Preserve sufficient width before subtraction and multiplication; a signed residual is not interchangeable with an unsigned ADC reading.

P is the model's scale, not the CPU word length. For an initial experiment, evaluate powers of two such as 32, 64 and 128 against the application's error requirements. A smaller P can reduce ranges and sometimes storage; it also changes the quantized function. Choose it using held-out data and boundary tests, then train or validate with precisely the same quantization and rounding used by the target. [9]

Export calibration constants, P, parameter bounds, activation rules, output thresholds, hysteresis, validity policy and version identifiers together. Include golden vectors with expected calibrated inputs, dot products, rounded sums and outputs. This catches a calibration or scaling mismatch that an output-only spot check can miss. Keep the original alert or control path available during a shadow evaluation, where the new classifier records decisions without immediately taking over actuation.

A compact model does not justify expensive preprocessing. Account for sample acquisition and feature extraction first. If the existing application already maintains a moving average, reuse its documented state and timing instead of creating another raw-sample buffer.

9. The example borrows the Lab's two-input parameter domain to make the arithmetic auditable; it is not a universal IGL-network bound. Inputs are quantized counts after preprocessing. Changing P, fan-in, bias placement, calibration or feature range requires a new range proof and renewed accuracy validation.

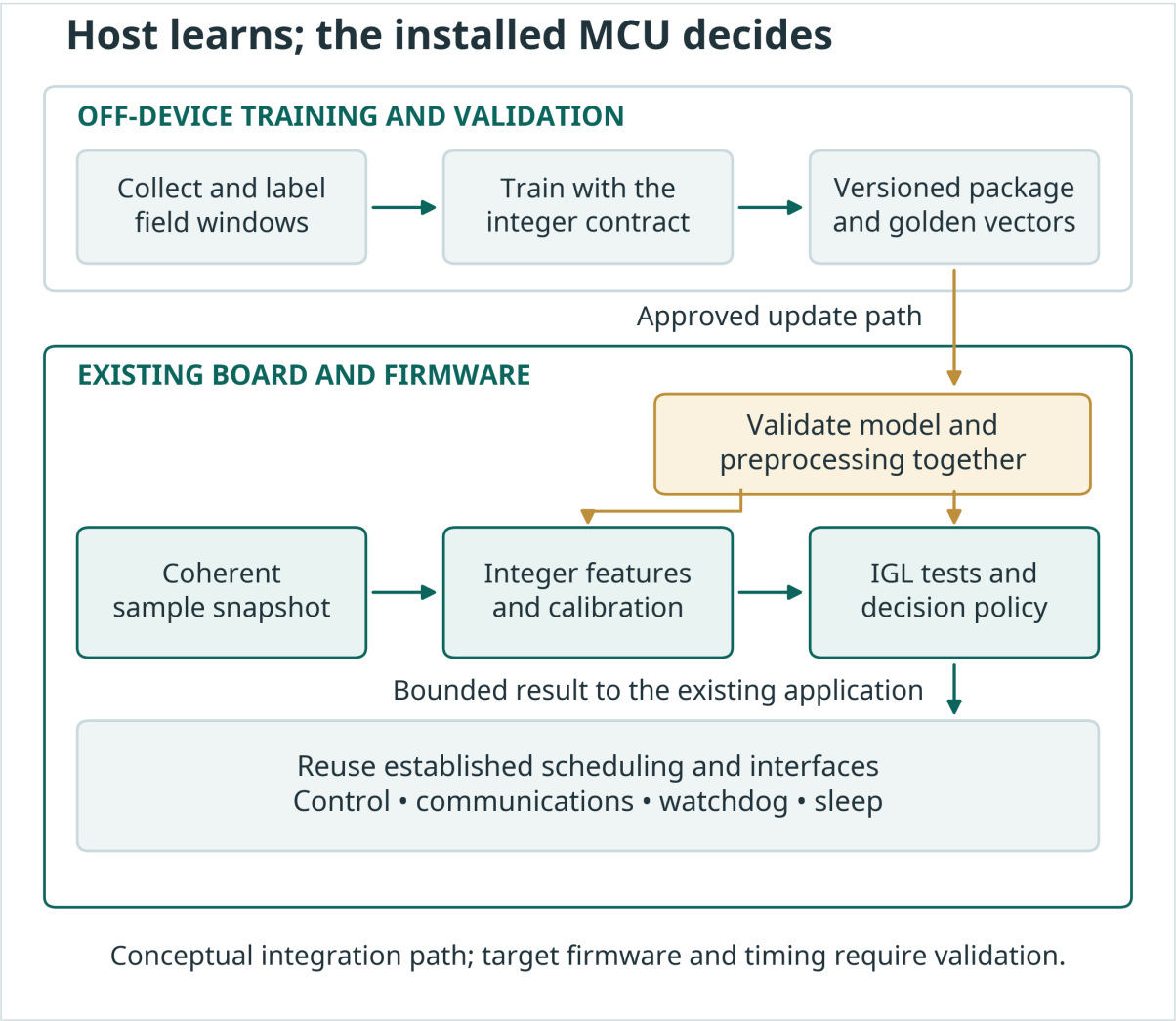


Figure 27. Proposed integration on existing hardware. Training and numerical validation happen off-device; a compatible package supplies both preprocessing and inference. The MCU retains its established scheduler and application interfaces. This is an engineering workflow, not released IGL firmware.

7.4 Use native operations while preserving the rounded function

Keep stored operands narrow where their full range fits, then widen before multiplication and addition. In the MSP430 ABI, ordinary int is 16 bits; declaring the destination int32_t does not by itself widen an earlier int16_t multiplication. Verify an appropriate widening cast or an exact widening-multiply primitive. Inspect the target's helper calls: TI specifies separate software and hardware-multiplier variants. [V07]

A fixed power-of-two P makes normalization a promising specialization. It does not make a bare signed right shift equivalent to the Lab's rounding. The required rule is nearest integer with half-way cases toward positive infinity. For P=128, dot products -192, -64, 64 and 192 must normalize to -1, 0, 1 and 2. Adding half a unit and using C division can also fail because negative C division truncates toward zero. Validate a constant-shift implementation against the scalar oracle. [10]

On the F1611, a multiplier is a shared peripheral, not private register state for each task. TI warns that an interrupt using it between operand writes can corrupt the operation. Give the peripheral a clear owner or use a documented, bounded critical section; do not disable interrupts around an entire inference routine. The same ownership question should be asked when introducing MATHACL use into existing MSPM0 firmware. [V06]

10. For P=2^k with k≥1 and unsigned magnitude U=|D|, an exact bounded specialization is (U+P/2)>>k for nonnegative D and -((U+P/2-1)>>k) for negative D. Handle k=0 separately. Form the magnitude without signed-minimum overflow, prove the added offsets fit, and test all reachable remainders. A compiler's signed-shift behavior must not be assumed portable.

A measured software multiply or divide may still be acceptable at a low decision rate. Optimize the operation that consumes the budget, while retaining an exact reference implementation for every accelerated path.

7.5 An illustrative budget for four independent two-input tests

Assume two ADC channels sampled at 16 pairs per second, a nonoverlapping 16-sample window, and four independent two-input IGL tests evaluated once per window. These are design assumptions, not measurements or released Lab functionality. At P=128, take the Lab's full parameter limits $|W|, |B| \leq 384$ and integer inputs from 0 through 128. Each node then has $|D| \leq 98{,}304$ and $|S| \leq 1{,}152$. INT16 coefficients with INT32 dot arithmetic cover these bounds.

Illustrative storage item	Calculation	Bytes
Retained raw samples	2 channels × 16 × uint16	64
Window sums	2 × uint32	8
Normalized input pair	2 × uint16	4
Four node outputs	4 × uint16	8
Counters and status	Assumed allocation	12
Arithmetic scratch	2 × int32	8
Nominal working payload	Sum of the items above	104
Coefficients in flash	4 × (W_1,W_2,B) × int16	24

The node stage performs eight products, four normalizations and four activations per decision. Sensor accumulation, calibration and output policy are additional work. These operation counts do not predict cycles. The 104-byte payload is already about one fifth of a G2553's 512-byte RAM, before existing globals, communication buffers and stack are counted. Omitting the retained sample window reduces this illustrative payload to 40 bytes if streaming features suffice. [V02] [11]

The useful deliverable is a linker map and measured stack high-water mark from the actual firmware. The coefficient count alone cannot establish fit. Store immutable coefficients in flash where the toolchain supports it, and measure the incremental code and runtime-library footprint.

11. The table counts nominal payload, including eight scratch bytes, without padding or allocator overhead. It excludes other firmware state, call frames, nested interrupt stack, calibration/threshold metadata, diagnostic logs, update staging and executable code. Shared scratch can reduce simultaneous storage only when lifetimes and ownership allow it. Neither a 24-byte coefficient block nor 104-byte payload proves a whole application fits.

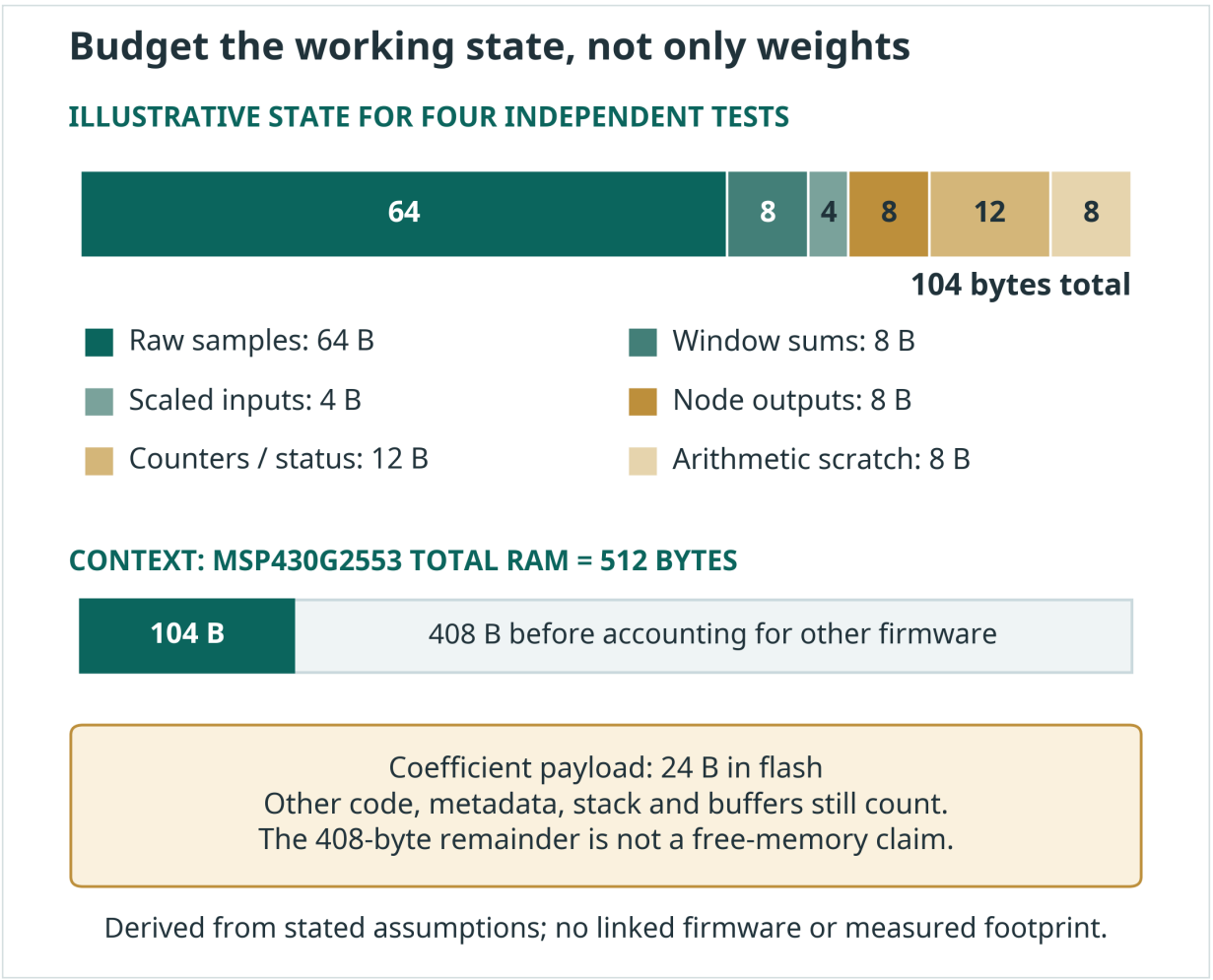


Figure 28. Illustrative storage accounting for the four-test example. The 104-byte working payload includes sample history and scratch; the coefficient payload alone is 24 bytes. The remaining device-wide RAM envelope still has to accommodate all other firmware and stack. Capacities follow V02; payloads are derived assumptions.

7.6 Fit the scheduler, battery budget and field-update path

Integrate inference as a bounded task triggered by completed samples or windows. Keep acquisition interrupts short, snapshot a coherent input pair, and publish results only after evaluation finishes. Define what happens if a window is incomplete or the task misses its deadline. Existing communications, watchdog service, control loops and peripheral ownership remain part of the timing analysis. An average execution time alone does not establish coexistence.

For a battery device, measure a complete sensing cycle: wake-up, sensor settling, conversion, preprocessing, inference, communications and return to sleep. The potential benefit is a useful decision within the available active interval, or less unnecessary transmission. Whether either saves energy depends on the sensor and radio behavior as well as computation. Record decision frequency, supply voltage and sleep state; do not turn an isolated instruction comparison into a battery-life claim.

Version the model and preprocessing as one compatible package. Reuse the product's established update and recovery mechanisms, validate a candidate record before activation, and preserve a known-good record where storage permits. Check interrupted writes and restart behavior. Small coefficient updates may be easier to stage than an entire firmware image, but flash erase granularity, existing bootloader capabilities and available storage determine the design. [12]

12. Where supported, two small parameter records with sequence/version metadata and a commit rule can preserve a previous model without duplicating all firmware. Integrity checking detects corruption; update authenticity still depends on the product's trusted update mechanism. Erase-block layout, power-loss behavior and recovery access must be established on the existing device.

A practical first acceptance gate is modest: reproduce the host vectors, meet application accuracy limits, fit alongside current firmware, and stay within a measured timing and energy allowance. Passing that gate can justify a controlled field trial on unchanged hardware. Broader models, new peripherals or processor replacement can then be considered using evidence from the actual application.

7.7 TI MSPM0G5187: an MCU evaluation entry point

TI's MSPM0G5187 provides a concrete MCU entry point: an active 80 MHz Cortex-M0+ device with 128 KB flash, 32 KB SRAM and TinyEngine NPU. TI identifies the LP-MSPM0G5187 LaunchPad and MSPM0 SDK as development resources. Begin with the portable scalar IGL kernel and golden vectors on that CPU; compile for Cortex-M0+, and measure total firmware, scratch space and inference latency. [T20]

The NPU datasheet documents 8/4-bit input activations, 8/4/2-bit weights, convolution/pointwise/depthwise, max/average pooling, residual layers and ReLU. Those capabilities do not establish support for IGL's five-region activation or its exact normalization/rounding contract. Full-range P=1024 weights also exceed the documented 8-bit weight format. [T21]

An accelerator experiment therefore needs a proven operator decomposition or custom CPU boundary, datatype validation and transfer accounting. A supported conventional model is a useful board-level baseline; any claimed IGL NPU mapping must be demonstrated separately. This is an evaluation path, not evidence of an available IGL firmware package.

7.8 TI C7x: AM62D-Q1 is specifically a 256-bit C7504 system

AM62D-Q1 is a heterogeneous Sitara system-on-chip with Arm Cortex-A53 processors, Cortex-R5F subsystems and a C7504 C7x DSP with an attached matrix accelerator. It is not a Cortex-M microcontroller. The AM62D datasheet explicitly identifies a 256-bit vector core, 1.25 MB of C7x L2 SRAM and a 32-bit LPDDR4 data bus. The vector width and external bus width describe different parts of the memory/compute path. [T01]

The C7x family includes other widths. TI's vector-programming guide distinguishes 512-bit C7100/C7120 implementations from 256-bit C7504/C7524 implementations and describes compiler bisection when a vector expression is too wide for one target register. For portable source, use the scalable C++ vector model and query its element count rather than hard-coding a 512-bit assumption. Compiler reports and assembly should confirm the actual AM62D mapping. [T03]

AM62D 256-bit vector payload	Element positions	What the count does not establish
FP32 or INT32	8	Equal lane count does not prove equal latency or full-kernel throughput.
INT16	16	Products may widen and require additional registers/instructions.
INT8	32	The full P1024 parameter domain cannot be represented directly.

These lane counts are width divisions, not benchmark results. The DSP also supports floating-point vector execution, so integer IGL should be compared with competent FP32 code on the same core. A mapping across independent samples or node instances may use the available lanes better than placing only the two

terms of one dot product in a wide register. Activation comparisons, selection and normalization must be included, along with tail handling. [T01]

Select the correct silicon target explicitly. The cited compiler guide associates C7504 with --silicon_version=7504 and MMA version 2_256; a family default is not sufficient evidence of a correct build. TI's current DSPLIB device table labels AM62D/C7504 as MMAv2, while C7524-based AM275 adds MMAv2f with FP32 MMA support. Do not transfer the newer MMA's FP32 capability to AM62D or confuse DSP-vector floating point with matrix-accelerator floating point. [T02][T07]

7.9 TI streaming engines, address generators and MMA boundaries

The Streaming Engine and Streaming Address Generator solve different problems. TI describes the SE as a read path that prefetches/formats regular streams from memory to the CPU, bypassing L1 for those accesses; L2-resident data is its preferred operating case. The SA generates multidimensional offsets for ordinary loads/stores and can provide vector predicates, including for tails. They do not change IGL's arithmetic precision or make irregular search control disappear. [T04][T05]

A proposed batched inference loop can reuse coefficients, stream contiguous input tiles, compute widened dot products, apply the exact normalization/activation and write a predicated output tail. For multiple instances, a different loop can stream coefficient planes while reusing inputs. This is a design choice to profile: streaming setup and closing costs must be amortized over enough work. Starting with one scalar C reference prevents layout or predication mistakes from being mistaken for numerical behavior.

Automatic use is compiler- and target-dependent. TI's optimization guide documents --auto_stream=saving for C7504/C7524, contrasted with the option used for earlier variants, and states that the compiler normally chooses streaming only when it considers it legal and profitable. Record the exact compiler version and inspect the software-pipeline feedback. A streaming flag is not evidence that the intended loop was vectorized, scheduled efficiently or fed from the intended memory. [T06]

The MMA is most naturally considered when enough dot products can be arranged as a supported matrix tile. Candidate parameters, examples or multiple nodes can supply batching dimensions in a separate implementation, but the laboratory's single two-term dot product is a poor basis for assuming matrix-engine utilization. Include packing, tile padding, accumulator extraction and the nonlinear activation after the matrix product. If bias moves into accumulator units, repeat the headroom proof; if a chosen MMA datatype cannot encode the weights, reject or explicitly transform that mapping.

TI's DSPLIB supplies native-C reference and intrinsic implementation variants for supported kernels, which is a useful validation pattern to adopt. It is not an IGL integration. Maintain a portable reference, a vector path and any MMA path behind the same test interface, and test all region boundaries and tail lengths. Device-specific MMA formats, compiler headers and library constraints are the authority for a proposed kernel; family marketing figures cannot predict this mixed workload's speed or energy. [T07]

CHAPTER 08

STMicroelectronics: from legacy MCUs to acceleration

8.1 Start with the MCU already on the board

An ST evaluation should first ask whether a compact learned rule can fit the product's existing firmware and sampling schedule. That creates a concrete opportunity: add a local decision without making a board redesign the first dependency. Begin with a frozen model and a portable C reference; training can remain on the host. The following are specific evaluation targets, not interchangeable descriptions of every device in their families.

Exact example	Documented resources	Proposed IGL starting point
STM32F103C8	Cortex-M3; up to 72 MHz; 64 KB Flash; 20 KB SRAM. [S01]	Scalar C; count model code, coefficients and live buffers.
STM32G071RB	Cortex-M0+; up to 64 MHz; 128 KB Flash; 36 KB SRAM. [S02]	Scalar C; inspect generated arithmetic and library helpers.
STM8S003F3	8-bit STM8; up to 16 MHz; 8 KB Flash; 1 KB RAM. [S03]	A deliberately small frozen rule or rule bank; measure wider arithmetic.

Neither Cortex-M3 nor Cortex-M0+ includes an FPU or the Cortex-M DSP extension. M3 has hardware integer division; M0+ does not. These differences affect implementation choices, but they do not prevent integer inference. They also do not establish an IGL advantage: optimized quantized networks and direct rules belong in the comparison. [A02][13]

For example, an F103 pilot could classify a motor's operating condition from a normalized current envelope and enclosure temperature. Freeze the feature definitions before fitting a two-input candidate, then compare it with the product's thresholds or lookup rule. Include warm-up, sensor noise, missing samples and conditions outside the training range. The interesting question is whether the folded response captures a useful nonmonotonic relationship with less deployed machinery at the accepted error rate. This is a proposed application experiment; the laboratory's Boolean demonstrations do not establish motor-diagnostic accuracy.

8.2 Prove arithmetic width before selecting a larger part

The STM8 case makes the distinction between a small model and cheap execution especially visible. A coefficient may occupy two bytes while a product or sum needs four; an 8-bit core must implement that wider calculation with its available instructions. Count helper routines, stack, temporary storage and preprocessing as well as parameters. A running feature accumulator may avoid a raw sample window when the task permits, but changing feature computation also changes the contract to verify. [14]

For the reviewed two-input laboratory at P=1024, coefficients bounded by ±3072 fit int16 storage, while the maximum absolute dot product is 6,291,456 and fits int32. This is a useful bounded test kernel, not a

13. Keep the ordering code in the test report: F103C8 has 64 KB Flash, whereas F103CB has 128 KB. Clock ceilings and installed memory are device specifications, not an estimate of available application budget or IGL throughput. [S01][S02]

14. Storage width, promoted expression width and accumulator width must be specified separately. Widen operands before multiplication when the C implementation's integer promotions are insufficient. Preserve the laboratory's round-then-add-bias order and its negative-half tie rule in the reference tests. [L01]

resource certificate for an arbitrary trained network. Additional fan-in, rescaling and preprocessing require their own bounds. [L01]

STM32U575ZI offers a distinct next experiment: a Cortex-M33 implementation with DSP instructions and single-precision FPU, up to 160 MHz, 2 MB Flash and up to 786 KB SRAM. Its datasheet distinguishes SRAM capacity with ECC enabled. Evaluate an integer implementation and a credible floating-point baseline on this same device; its FPU makes “avoids floating point” an insufficient performance argument. M33 does not provide Helium. [S04][A02][15]

8.3 Use current ST tools without assuming custom-operator support

ST now positions STM32Cube AI Studio as the replacement for X-CUBE-AI for new development. Its workflow imports Keras, TFLite or ONNX, validates on host or target and generates integration code. Its published compatibility list excludes STM32F1, among other families. Thus the F103 experiment proposed here is a manual C port, not a claim of supported Studio model import. Existing older toolchains need not be discarded, but should be versioned separately. [S06]

ST Edge AI Core 4.1.0 documents partial Keras custom-layer support. Simple layers may map to supported TensorFlow operations; more complex layers need a Python definition, custom configuration and user-written C. This gives an integration direction for IGL, subject to the exact arithmetic, datatype and target APIs. It does not demonstrate an existing IGL exporter or automatic accelerator mapping. [S07][16]

For an upgrade with acceleration, STM32N657 provides an up-to-800 MHz Cortex-M55 with Helium and 4.2 MB contiguous SRAM. The N6 datasheet assigns Neural-ART specifically to N6x7 variants. Neural-ART supports inference under documented operator, shape and quantization restrictions; its compiler does not accept arbitrary integer code as an NPU instruction. First test a CPU implementation, then an equivalent graph decomposition or custom CPU layer where supported. Inspect compiler placement and validate every boundary. [S05][S08][17]

15. The U575 datasheet lists 786 KB SRAM with ECC off and 722 KB with ECC on. Its DSP/FPU features describe this implementation; optional features in the Arm core catalogue do not prove their presence in another MCU. [S04][A02]

16. The documented complex Keras custom-layer example leaves parameters and learned weights for the developer to supply in the C implementation. A production adapter therefore needs a repeatable export path; hand-copying an example’s constants is not a maintainable model-update mechanism. [S07]

17. Neural-ART is an inference engine. Its documented integer formats and supported operators constrain placement; software fallback is available only where supported. Count CPU/NPU transfers, format conversion and fallback buffers when comparing the complete implementation. N6x5 and N6x7 are not equivalent accelerator configurations. [S05][S08]

8.4 Give each integration partner a measurable deliverable

A practical ST pilot has three owners. MLiglon supplies or specifies the model, arithmetic contract, export representation and reference vectors. The firmware team owns sensor preprocessing, scheduling, memory allocation and the shipping application. An ST tools or ecosystem partner can assess compiler integration and device-specific optimization. These are proposed responsibilities to agree, not evidence of a current commercial partnership.

Pilot deliverable	Acceptance evidence
Installed-device build	Golden-vector agreement; complete Flash image and peak RAM; deadlines under the real interrupt load.
U5 or N6 comparison	Same task and held-out data; measured latency and energy including preprocessing and memory traffic.
Toolchain adapter	Pinned versions; reproducible export; operator-placement report; source and model-update ownership.

Negotiate the software evaluation before a hardware commitment: ask what training/export software is available, which C interface is supported, and who maintains conversion and regression tests. A qualifying result can be modest but commercially useful-for example, an accepted local classifier that fits existing memory headroom. The evidence should show why that product benefits, including integration cost and an explicit comparison with its current decision method.

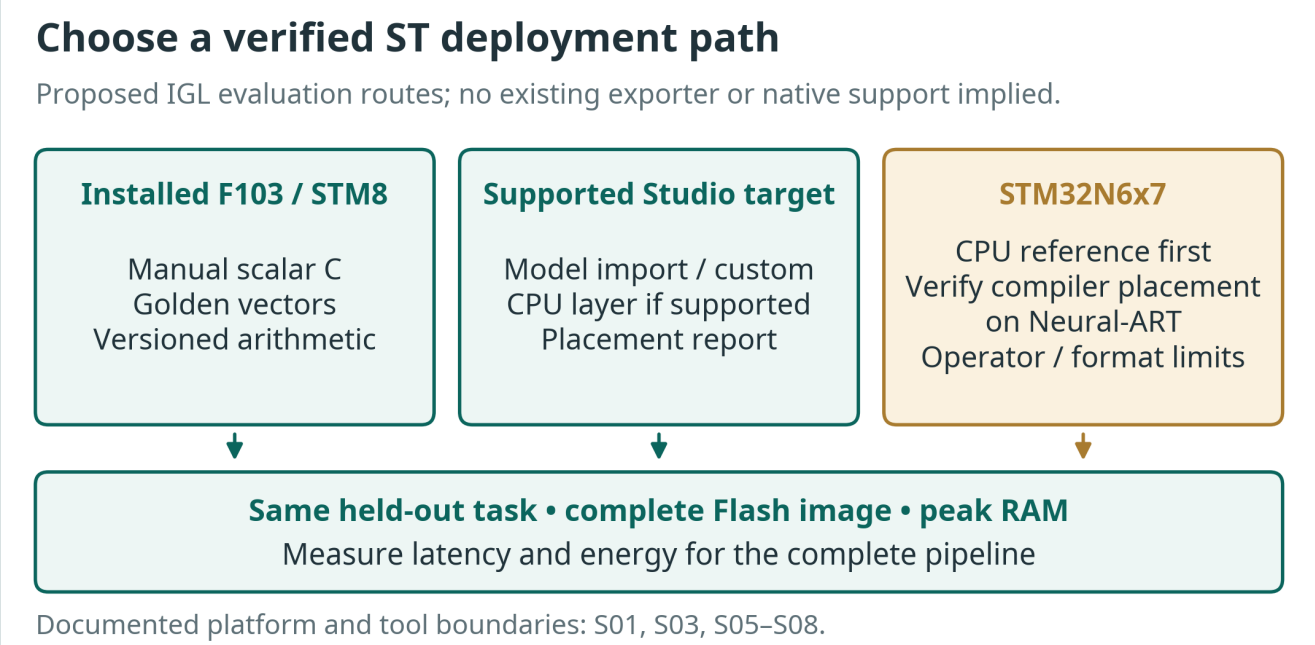


Figure 29. Proposed ST evaluation routes. Installed devices can start with a manual C implementation; tool import and accelerator placement require support in the selected version and target. Compare every route on the same application evidence and complete resource accounting.

CHAPTER 09

Arm Holdings and the Cortex-M ecosystem

9.1 Separate processor IP from the selected MCU

For the Cortex-M route discussed here, Arm supplies processor and system IP that silicon partners integrate; ST and TI supply the selected finished microcontrollers. An IGL discussion can therefore have different sponsors: a firmware owner seeking a compact model, an MCU vendor assessing a supported library, or a chip designer considering acceleration. The deliverable and ownership boundary should be explicit. [A01][18]

Cortex-M0, M0+ and M3 have no FPU, DSP extension or Helium. On other cores, feature presence varies: M33 offers optional DSP and single-precision FPU, while M55 offers optional Helium and FPU configurations. Use the actual chip's datasheet and build configuration, rather than the core name alone, to select kernels. [A02][19]

Prospective sponsor	Useful evaluation package	Integration boundary
Product firmware team	C kernel, frozen parameters and conformance vectors.	Existing MCU instructions; application timing and memory.
MCU or tools partner	Maintained exporter and architecture-specific kernels.	Supported device/tool versions, API and update ownership.
SoC design partner	Profiled workloads and a simulated datapath candidate.	Licensed IP, hardware implementation and silicon verification.

9.2 Make scalar, DSP and MVE implementations interchangeable

CMSIS-NN provides portable C, DSP-extension and Helium/MVE implementations of supported neural operators. That supplies both a useful implementation ecosystem and a credible conventional baseline. Its integer kernels follow LiteRT quantization conventions; sharing an integer datatype does not establish equivalence to the IGL laboratory's scale, rounding and folded activation. An adapter must prove the complete operator contract, or supply a dedicated kernel. [A03][20]

Maintain one scalar reference and validate each optimized implementation against identical vectors. For a packed DSP or vector design, group independent outputs or compatible dot products so data layout serves the selected instructions. Test tails, alignment, sign extension and accumulator bounds. Report speed after packing and memory movement, not only the arithmetic loop. These are proposed implementation checks, not an assertion that any CMSIS function already computes IGL.

Helium is CPU vector execution. The M55 can be configured with or without it, so a scalar fallback remains useful. CPU vector instructions and an NPU are distinct execution routes, even when both reside in one SoC. [A04]

18. Arm's fiscal-2026 filing describes IP licensing as its primary business and also discloses the March 2026 introduction of Arm-designed AGI CPU silicon. The processor-IP distinction here concerns the Cortex-M integration route, not every current Arm business activity. [A01]

19. Optional core features are selected by the chip implementation. Firmware compiler flags must match that hardware. The absence of an FPU makes an integer implementation relevant, but does not establish that IGL alone can perform integer inference. [A02][A03]

20. Check bias placement, zero points, multiplication, requantization, tie handling and saturation separately when considering a CMSIS-NN primitive. Bit-exactness for its documented LiteRT convention is not automatically bit-exactness for a different IGL operator. [A03]

Cortex-M55 adds another distinction: MVE provides 128-bit architectural vectors, while the cited M55 TRM describes a 64-bit arithmetic and load/store datapath operating in beats. MVE and floating-point configuration are optional features in the processor specification. Thus 16 INT8, eight INT16 or four FP32 element positions in 128 bits do not imply those operations all retire in one cycle. Confirm the implemented configuration and measure the compiled loop. [T10]

The fully connected APIs illustrate the datatype issue precisely: s8 uses INT8 input/weights/output and INT32 bias, whereas s16 uses INT16 input/output, INT8 weights and INT64 bias. The s16 name does not mean INT16 weights throughout. At P=1024, the full IGL weight domain therefore does not fit that s16 operator's INT8 weight interface without a changed encoding. A custom kernel or proven decomposition is needed, and its costs belong in the benchmark. [T12]

9.3 Treat Ethos placement as a compiler result

Arm's Vela compiler targets Ethos-U NPUs, converting supported quantized models into accelerator work. Current documentation distinguishes TFLite models, whose unsupported regions remain CPU operations, from TOSA compilation, which has no CPU fallback. A successful TFLite compilation therefore does not prove full NPU execution, and an unsupported custom IGL operation still needs a valid host-runtime implementation. [A05][21]

For an IGL experiment, first inspect whether an exact graph decomposition fits the chosen NPU's operator and quantization constraints. Compare a CPU-only build with any mixed CPU/NPU build using the same task, preprocessing and model. Include tensor transfers, conversion buffers, synchronization and CPU fallback time. If the custom folded response fragments an otherwise efficient graph, a compact CPU kernel may be the better product choice. That is a result to measure, not an architecture-wide verdict.

9.4 Reserve custom instructions for a silicon partner

Arm's current ACI documentation identifies optional Custom Datapath Extension support in Cortex-M33, M52, M55 and M85. It demonstrates defining instructions, using intrinsics and extending simulation models before hardware design. Detailed integration documentation is provided to relevant licensees or under NDA. This is a future-chip design route; firmware cannot add a missing physical datapath to an installed MCU. [A06][22]

Keep the first licensing package useful without new silicon: an arithmetic specification, portable kernel, model-export contract, golden vectors and representative workloads. A subsequent SoC partner can assess whether a recurring bottleneck justifies a custom datapath. Arm's Flexible Access programme is one route for eligible silicon-design participants to evaluate covered IP; exact entitlements and terms need agreement with Arm. IGL rights and maintenance responsibilities need their own agreement. [A08]

The ecosystem opportunity is a verified kernel or exporter that several products can reuse. A hardware proposal becomes persuasive after software profiling establishes the workload, expected utilization and remaining bottleneck; it does not have to be the first commercial milestone.

21. Pin the Vela version, NPU configuration, model format and host runtime. Inspect operator placement and warnings. Vela's cycle/bandwidth estimates guide exploration; the official documentation calls for final performance validation on FPGA or target silicon. [A05]

22. ACLE defines CDE intrinsics and feature macros, including a bitmap of available coprocessors. Compiler acceptance describes the configured target; it cannot create absent silicon. Simulation estimates and hardware implementation remain separate validation stages. [A07][A06]

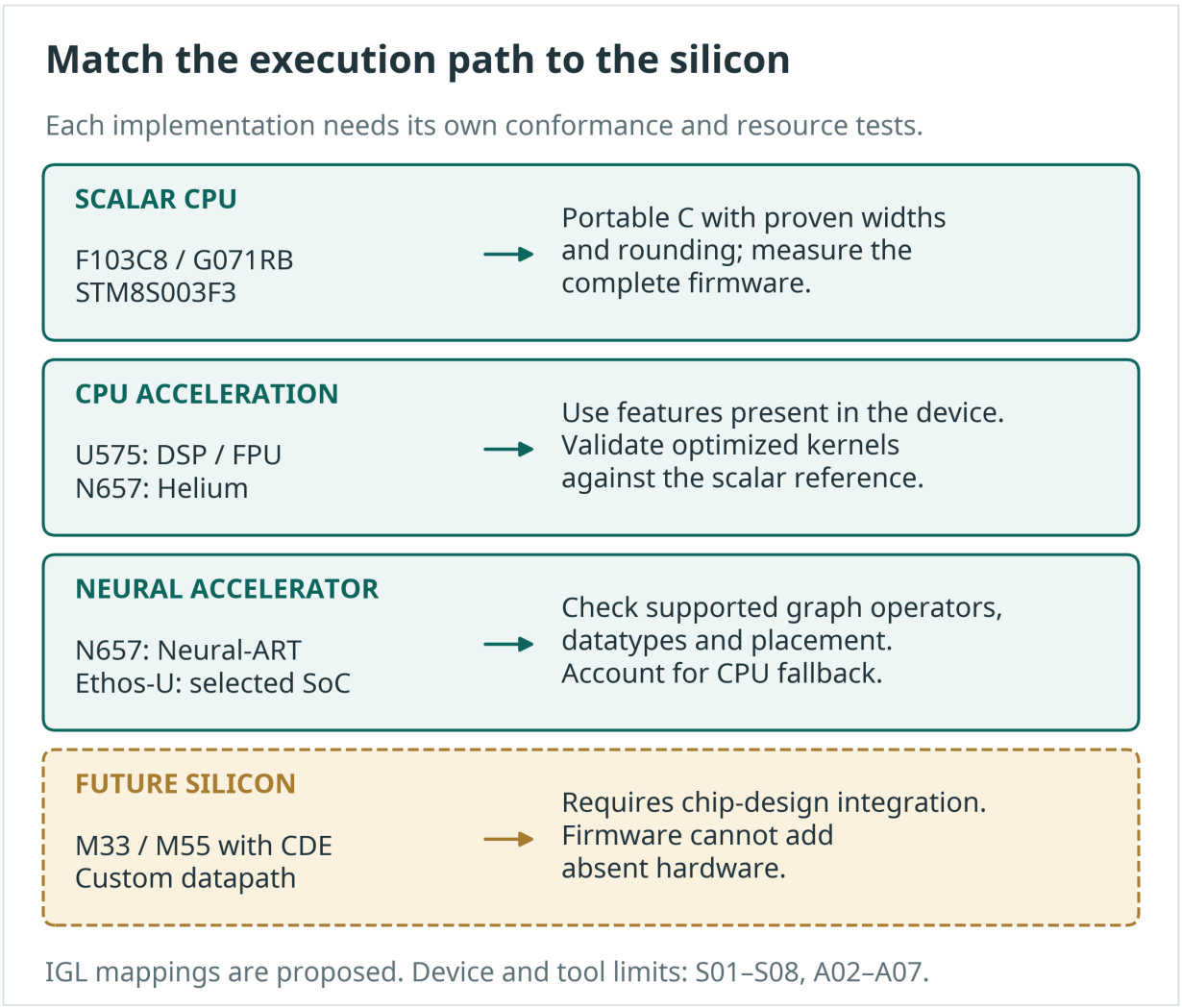


Figure 30. Execution choices follow the selected silicon and toolchain. Portable C, CPU optimization and supported NPU graphs can be evaluated on existing devices. A new custom datapath requires chip-design integration. CPU fallback depends on the model format and runtime: current Vela TOSA compilation has none. Device capabilities do not imply native IGL support. [S01-S08][A02-A07]

CHAPTER 10

Dedicated IGL instructions and silicon

10.1 A progression from software to silicon

IGL creates several possible hardware products, each with a different investment and integration burden. The nearest opportunity is a verified integer library on an existing MCU. A future device could add a narrowly scoped instruction, an MCU-attached accelerator, or a programmable IGL engine with local model memory. A dedicated chip could then combine that engine with sensing, communications or control. These are proposed development paths; the Laboratory supplies an executable node reference, not an existing instruction standard or a characterized silicon implementation.

The folded activation is attractive to examine because its five regions require comparisons, selection and additions or subtractions rather than an exponential. For power-of-two P, normalization can be designed around an exact rounded shift. The surrounding weighted sum uses familiar integer multiply-accumulate hardware. This makes a compact datapath plausible, while the value of specializing it depends on the instructions, memory traffic and wake-up work that it actually removes. [23]

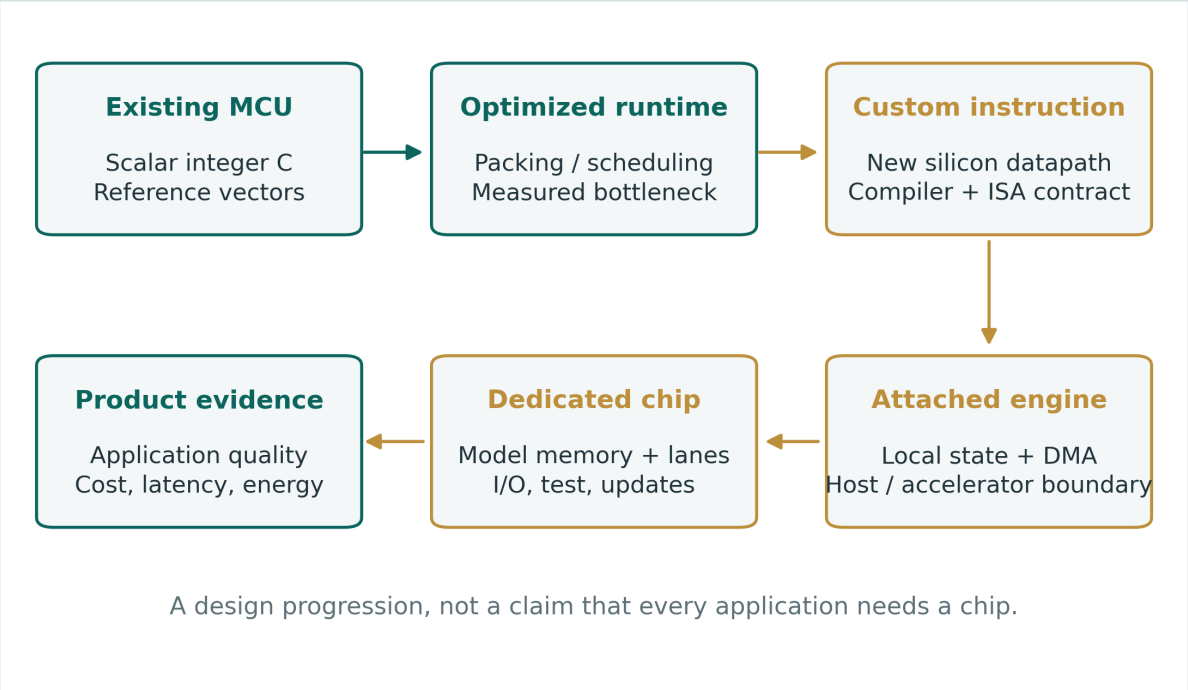


Figure 31. Proposed implementation progression. Existing devices receive software; new instruction logic and integrated accelerators require a hardware design opportunity. Each step should earn its additional integration cost.

23. A piecewise-affine fold need not require a lookup table, but a target may implement comparisons, conditional selection and subtraction in different ways. Gate count, critical path and branch behavior must be evaluated in the selected implementation. Constants P, 2P and 3P must fit their comparison widths.

10.2 Instruction candidates: define the smallest useful operation

Start by profiling the complete application and then identify a repeated primitive. An activation instruction consumes an already normalized signed S and returns raw Y. A rounded-normalization instruction preserves the chosen tie rule. A fused node operation goes further, but two inputs, two weights, bias, scale and a result create operand-delivery and register-pressure questions. Constants, packed operands or a configuration register can reduce instruction fields, at the cost of explicit interface rules. The names below are illustrative proposals, not opcodes available on current TI, ST or Arm products. [24]

Proposed primitive	Precisely specified work	Design question
IGL.FOLD	Y = f_P(S), with the five documented intervals and endpoints.	Scalar or lane-wise vector; P immediate, register or fixed configuration?
IGL.RNORM	R = round(D / 2^k), nearest with ties toward positive infinity.	Signed width, k=0 behavior, rounding carry and overflow handling.
IGL.NODE2	Two widened products, one normalization, bias, then fold.	How are six logical inputs supplied without expensive loads or hidden state?
IGL.STEP	Fetch a compact node record, read source values and write one result.	Useful for an engine; memory exceptions, graph scheduling and bounds are a large contract.

A narrow, stateless fold or normalization primitive is easier to verify, schedule and expose through compiler intrinsics. A graph-step instruction can remove more software work, but binds the ISA to a node format and memory model. A vector fold can process independent nodes or samples in parallel when their scale conventions agree. Its benefit should be assessed with the load, packing, widening and store instructions included.

Arm Custom Instructions provide an optional implementation route for silicon designers using supported Cortex-M IP. RISC-V separately reserves custom opcode spaces; those spaces are not a published IGL extension. Either route requires a real hardware implementation, compiler support, a functional model and fallback code. Existing deployed MCU silicon cannot acquire new execution logic through an ordinary firmware update. [D02, A06]

24. An ISA operation includes operand and result encodings, supported widths, exact arithmetic, illegal values, flags or exceptions, and discoverability. The proposed mnemonics are explanatory labels. They imply neither a vendor commitment nor an available instruction encoding.

10.3 An exact integer datapath and its verification obligations

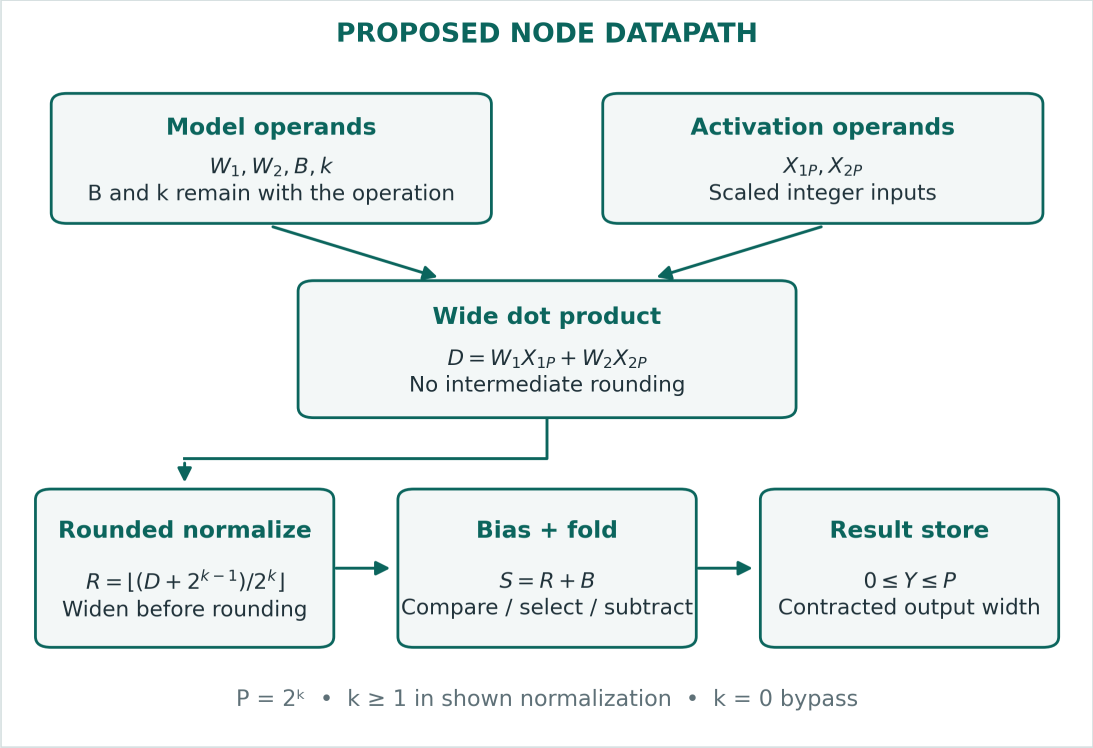


Figure 32. Proposed two-input IGL datapath. Normalization occurs once after accumulation and before bias. Operand widths, intermediate widths and output storage remain independently specified.

For P=2^k with k at least one, nearest rounding with ties toward positive infinity can be written as floor((D + 2^(k-1))/2^k). A hardware design can implement this using a widened rounding addition followed by a signed floor shift. For k=0, the result is D. The addition must preserve carry and the specified signed range; silently overflowing before the shift changes the operator. Firmware should use the portable reference contract until the target language and compiler semantics have been established. [25]

$$R = \left\lfloor \frac{D + 2^{k-1}}{2^k} \right\rfloor, \quad S = R + B, \quad Y = f_{2^k}(S), \qquad k \ge 1$$

Mathematical proposed instruction semantics. The extra rounding term uses widened arithmetic; this formula does not authorize an overflowing C expression or assume a particular language right-shift rule.

Verification vectors should cover every activation boundary from both sides, signed rounding ties, maximum products, bias extremes, all scale encodings, and vector tails. Exhaustive checking is feasible for sufficiently small operand subsets; larger formats need a proved correspondence between the RTL or instruction model and the integer reference. A fused operation must not round the two products separately. The same conformance corpus should exercise scalar C, compiler intrinsics, simulation and programmable logic.

Keep an initial instruction stateless if possible. If scale or format is held in configuration state, define reset values, privilege, task-switch save/restore, interrupts, debug visibility and feature discovery. Predicated vector lanes and exceptions require equally precise behavior. A compact mathematical formula is only part of an instruction-set contract. [26]

25. The rounded-shift identity follows from nearest rounding with halfway values toward positive infinity. For D=-6 and P=4 it gives floor(-4/4)=-1; for D=-7 it gives floor(-5/4)=-2. Signed truncation toward zero would produce a different result in the second case. [L01]

26. Configuration state trades instruction bits for context-management work. Interrupt handlers or RTOS tasks must not accidentally change another inference operation's scale. A stateless immediate can avoid that class of interaction, but may limit the available scales or widen instruction encoding.

10.4 A dedicated engine is also a memory architecture

A programmable IGL engine could contain a command interface, node-record memory, an activation scratchpad, a scheduler and one or more integer datapaths. Independent node instances offer parallelism; dependencies determine when values are ready. A compiler can lay out records and release dead activations to reduce live storage. Reusing a small datapath over many nodes saves arithmetic area, while replicated lanes may improve throughput when memory supplies them. A fixed model synthesized into logic is another option. An FPGA implementation can receive a new bitstream. Changing model logic hardwired into an ASIC generally requires a silicon redesign and replacement devices, unless programmable model storage or configuration was included in its architecture.

Model locality can be more valuable than adding multipliers. If weights, connection information and live activations fit near the execution unit, fewer transfers may be needed over a system bus or to external memory. That is an architectural hypothesis to test with exact access counts and implementation measurements. Conventional accelerators also use local buffers, pipeline stages and specialized data movement; NVDLA is a public example of that design discipline. Its architecture is a reference for integration concerns, not evidence of automatic IGL compatibility. [D01]

Product form	Potential use	Principal engineering burden
MCU-attached IP block	Small sensor graphs; CPU retains communications and control.	Bus arbitration, DMA, interrupts, coherent buffers, sleep-state control.
Programmable dedicated chip	Reusable IGL models with local memory and host interface.	Memory capacity, model format, toolchain, packaging, test and update path.
Fixed-function or model-specific logic	Stable high-volume task with minimal scheduling overhead.	Model changes, distribution drift and silicon respin economics.
FPGA prototype	Validate arithmetic, schedules and interface before committing to silicon.	FPGA power/area is not an ASIC estimate; report clock, resources and board conditions.

The first commercial engine should normally prioritize inference. Training needs candidate storage, example streaming, loss evaluation and a coherent acceptance policy, which are different resources from a small inference datapath. A future training accelerator is a separate architectural study. Host training can already explore precision and topology constraints imposed by the proposed inference engine.

10.5 Performance, energy and commercial decision gates

$$\text{Speedup}_{\text{total}} = \frac{1}{(1 - f) + f/r}$$

Derived serial/accelerated time accounting: f is the original fraction of total runtime affected, and r is its speedup. If f=0.30 and r=4, total speedup is about 1.29, before new overhead.

Use this equation to choose the instruction boundary. Accelerating an inexpensive activation cannot remove sensor acquisition, feature extraction, memory fetches or radio work. Conversely, an instruction that reduces a dominant repeated sequence can also reduce instruction fetch and dispatch. Measure energy for a complete decision cycle and its duty cycle, including wake-up, clocks, memory and communications. A fast prototype is promising only if it improves the application objective under its actual operating conditions.

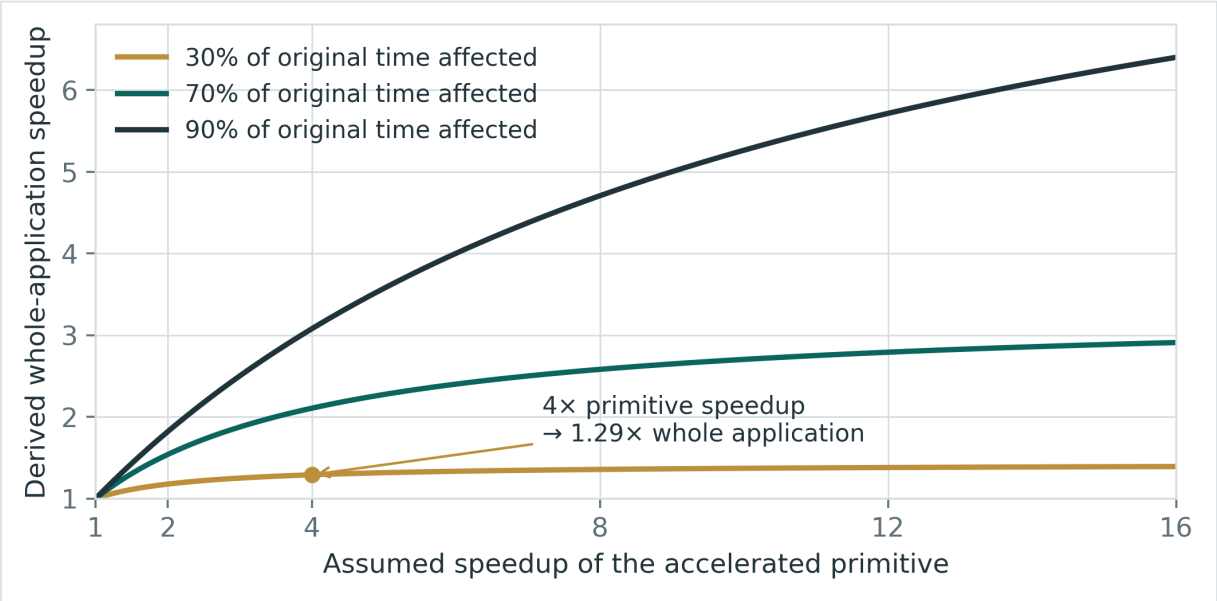


Figure 33. Analytical illustration of acceleration limits. These curves use assumed affected fractions and primitive speedups; they are not IGL device measurements.

Advance through four evidence gates: exact arithmetic conformance; workload simulation and compiler-generated code; FPGA or emulation with representative memory traffic; and technology-specific synthesis, timing and power analysis. Before production, include physical implementation, design-for-test, verification, packaging, characterization and integration support. Area, frequency and energy should always identify the process, voltage, temperature, memory macros and measurement or estimation method. [27]

$$C_{\text{unit}} = C_{\text{recurring}} + \frac{C_{\text{NRE}}}{N}$$

Illustrative cost structure, not a quotation: recurring cost includes die, packaging, test and applicable ongoing charges; NRE includes nonrecurring design and integration work, amortized over shipped quantity N.

A license discussion can start with software evaluation rights and progress to an optimized runtime, accelerator IP or a co-designed instruction. Define who owns the operator specification, compiler/exporter, conformance suite and maintenance obligations. The most convincing silicon proposal joins a useful workload, an exact and stable numerical contract, measured system benefit and a credible volume or strategic case. Contact MLiglon for technology access and applicable software or hardware licensing arrangements. [I12]

27. Simulation establishes behavior under the model's assumptions. FPGA testing establishes behavior of a programmable-logic implementation. Neither by itself establishes ASIC energy, yield, production cost, functional-safety certification or a complete system speedup. Those require their respective implementation and qualification evidence.

CHAPTER 11

From application requirements to MCU deployment

11.1 Requirements: define the decision before selecting the model

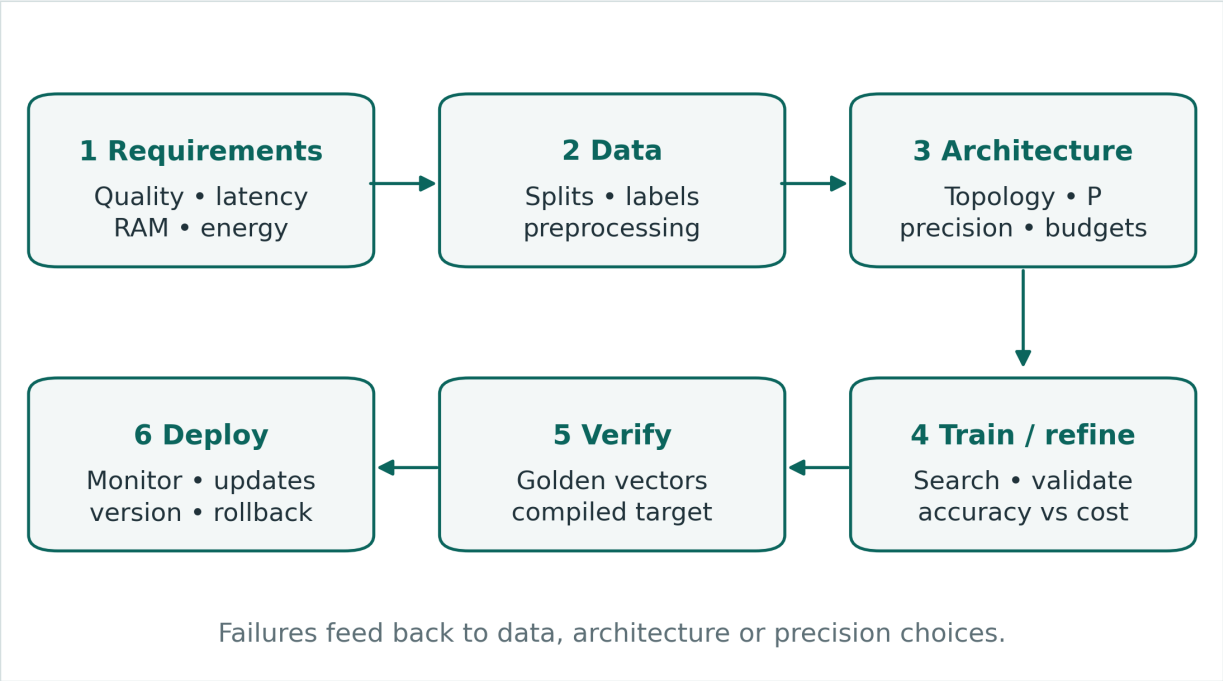


Figure 34. Proposed application-to-firmware lifecycle. Model refinement and failed acceptance tests feed back into the requirements, data and architecture decisions. This is an engineering process, not a product-release claim.

Begin the pilot with one product decision, such as classifying an equipment condition from an agreed sensor window. Specify the required classes, an unknown or rejected-input policy, the consequence of false alarms and missed events, and the deadline between acquisition and decision. Assign owners for the application, model, firmware, and measurements. This proposed workflow evaluates whether IGL meets that product requirement; it does not assume a particular result.

Choose an actual TI device and board configuration, not only a processor family. Record the memory available after existing firmware reservations, the allowed energy budget, input rate, concurrency with interrupts and communications, and update constraints. Agree numerical acceptance limits before optimization begins. The acceptance matrix below is a template: its blanks should become negotiated thresholds, not retrospective descriptions of whatever the prototype achieves.

Define the initial boundary as off-device model development followed by microcontroller inference. Treat training acceleration as a separate evaluation track if suitable software is available. This keeps the first integration small and avoids making on-device training a prerequisite for a useful deployment.

11.2 Data: make preprocessing and calibration part of the model

Build the evaluation set from the intended sensing conditions. Separate training, tuning, and final acceptance data by the source of possible leakage: machine, subject, recording session, site, or time period as appropriate. Preserve a final test set that does not influence feature selection, quantization limits, or model selection. Record class balance and the difficult cases separately so that a favorable overall accuracy cannot conceal an unacceptable failure category.

Specify the complete input contract: sampling cadence, window length, units, channel order, filtering, feature extraction, missing-data handling, clipping, normalization, and conversion into integer values. Fit any data-dependent preprocessing on training data only. Keep sensor calibration constants and model quantization scales distinct; changing either can change the meaning of the values reaching the network.

Produce examples at three boundaries: raw sensor records, derived features, and encoded network inputs. These expose whether a disagreement begins in acquisition, preprocessing, or inference. Include zero input, limits, clipped samples, corrupted records, and representative field disturbances. Store the preprocessing version with the model and repeat the same transformations in the desktop reference and embedded implementation.

Define what happens outside the calibrated range before field use. Clipping, rejection, and an explicit uncertainty indication are different product behaviors. Test sensor offsets, gain changes, missing channels, and out-of-range features against that policy. Keep counters for abnormal input conditions where useful, so that a stable classifier output does not conceal a deteriorating input stream.

11.3 Architecture: establish the hardware and software boundary

Write an executable model description before optimizing kernels. It should identify graph connectivity, node or layer types, parameter sharing, tensor shapes, activation definition, precision, and the final decision rule. The two-input laboratory is a useful semantic reference, but a deployment architecture must describe its actual fan-in and graph. MLiglon's public convolutional description includes shared kernels, channels, and pooling; those features require explicit representation and dependency handling. [I07]

Assign the host responsibilities for dataset management, training, model selection, and generation of a deployment artifact. Assign the MCU responsibilities for acquisition, preprocessing, inference, decision handling, and controlled model loading. A host-side exporter and an embedded runtime are proposed interfaces, not assumed available products. Confirm their availability and permitted use with MLiglon before scheduling integration work. [I12]

Create a live-state memory schedule. List every activation buffer, retained graph value, feature window, input queue, scratch area, model index, stack allowance, and communication buffer. Determine which values remain live until their last consumer and which allocations overlap. Count persistent and temporary memory separately, then validate the peak on the running system. Parameter-file size alone is not this memory budget.

11.4 Training: retain an auditable experiment

Once evaluation access is agreed, record the training implementation version, topology, precision, search neighborhood, initialization, seeds, batch schedule, stopping rule, and validation metric. Preserve configurations and intermediate decisions sufficiently to distinguish a reproducible run from a favorable isolated result. Use several predeclared runs when the learning procedure is stochastic; report variation and unsuccessful runs as well as the selected model.

For a chain isolation implementation, explicitly verify that candidate evaluations share a valid reference state and that accepted changes refresh affected cached outputs. MLiglon describes this reuse-and-refresh mechanism publicly, but the actual evaluation software should demonstrate its correctness under the chosen topology and scheduling. A faster candidate score is useful only if it measures the intended network. [I05]

Record training time, hardware, candidate or update counts, and memory independently of deployment measurements. Keep the final test set sealed while selecting the operating point. Archive the trained model with its preprocessing, label mapping, and validation report. No step in this process requires the production MCU to execute the trainer.

For an implementation comparison, preserve the same feature pipeline and comparable model-selection effort. Include unsuccessful architectures and tuning attempts in the experiment record. Distinguish a change that improves the learned representation from one that merely changes the measurement batch size or excludes preprocessing from the reported time.

11.5 Refinement: build a scalar C reference before accelerating

Implement a small, readable target C reference with explicit data types and separate functions for input conversion, weighted accumulation, rescaling, activation, and output interpretation. Its first purpose is numerical agreement. Use the same model bytes in the desktop harness and on the target, with a documented byte order, signedness, field layout, and version identifier. Avoid relying on a compiler-specific structure layout as an interchange format.

State the rounding rule mathematically. If laboratory compatibility is required, nearest rounding sends half ties toward positive infinity: -3/2 becomes -1, -1/2 becomes 0, and 1/2 becomes 1. Test an explicit quotient-and-remainder implementation instead of substituting a language's default division. Specify whether rounding happens after the complete dot product, and specify the bias scale. Rounding individual products can produce a different model.

Prove bounds for every product, accumulator, bias addition, and activation before choosing widths. Widen operands before multiplication; a wider destination alone is not the design rule. Specify clipping or rejection deliberately and do not allow accidental signed overflow or narrowing to define behavior. After the scalar reference passes, evaluate vectorization, memory layout, lookup tables, or other optimizations one at a time against the same outputs.

Explore precision and topology against an accuracy-resource frontier. A smaller representation may alter decisions and may require retraining; it is not merely a packaging change. Keep a controlled sequence of models and compare their quality, memory, and timing using the same acceptance data and harness. Retain the simplest implementation that satisfies the agreed product constraints.

11.6 Export a small runtime and a versioned model contract

An installed-device package can be generated as ordinary C arrays plus a small evaluation function, with statically allocated working storage. The target need not parse a training framework’s graph format or retain a desktop interpreter. For a larger network, a compact graph interpreter may reduce code duplication; for a fixed tiny graph, generated straight-line C may be preferable. Measure the complete linked image, including arithmetic helpers, rather than assuming that one representation is always smaller.

Contract field	What the exporter and target must agree
Signals and features	Sensor units, calibration, window timing, missing-value and clipping rules.
Graph and arithmetic	Node order, source indices, signedness, stored and accumulator widths, P and activation variant.

Contract field	What the exporter and target must agree
Normalization	Exact rounding ties, bias units, rescaling points and overflow policy.
Output policy	Thresholds, hysteresis, debounce, validity and diagnostic behavior.
Compatibility	Model/schema version, input signature, required runtime features and update record.
Conformance	Reference inputs, expected intermediate values, final outputs and test provenance.

Co-design the model with these constraints during training. Reject architectures that cannot meet storage or arithmetic bounds, then compare feasible candidates on validation quality. A calibrated cost estimate can guide search, but final latency, peak RAM and energy require the compiled application on its target. Sparse connections, mixed widths and feature reuse are useful only when their metadata and execution costs are included. [28]

11.7 Verification: use a pilot acceptance matrix

Create golden vectors from the approved reference before optimizing the target. Include expected intermediate and final values, not just labels. Cover each activation region and breakpoint, adjacent representable values, negative rounding ties, parameter extremes, and representative complete records. Run identical artifacts through desktop and target implementations. Require exact integer agreement unless a specific approximation and tolerance have been approved.

Acceptance area	Evidence to deliver	Pilot gate to agree in advance
Numerical conformance	Golden-vector report; intermediate-value comparison; overflow analysis	Zero unexplained mismatches; every permitted range covered by a bound
Predictive quality	Held-out confusion matrix; class-specific errors; repeated-run variation	Required detection and false-alarm limits for each important class
Memory fit	Program and model map; live-state schedule; measured peak SRAM	Fits the agreed Flash and SRAM allocation with reserved headroom
Timing	Acquisition-to-decision timing under representative concurrent firmware load	Meets the deadline at the agreed rate and measured worst observed case
Energy	Instrumented measurement with board state and workload recorded	Meets the agreed energy per decision and duty-cycle budget
Robustness and recovery	Disturbance tests; invalid-input behavior; update and rollback exercise	Defined safe application behavior and recovery for each tested failure
Reproducibility and integration	Versioned model, build instructions, licenses, and handover package	A second engineer rebuilds and reproduces the acceptance results

A measured worst case is bounded by the test conditions; document those conditions instead of presenting it as a mathematical worst-case guarantee. Record any requirement waiver explicitly. A model that passes numerical conformance may still fail accuracy, memory, or timing, and those are separate engineering outcomes.

28. Generated C and compact binary records are proposed export choices, not a statement that MLiglon currently ships a specific exporter. A human-readable development manifest can be converted off-device; a small MCU need not carry a general JSON parser. Both whole-image and incremental costs should be reported when sharing runtime code.

11.8 MCU deployment, lifecycle, and practical questions

Release a versioned bundle containing the model, preprocessing configuration, runtime compatibility identifier, label map, expected resource limits, and validation results. Integrate inference with the application's existing scheduling and failure handling. Provide an explicit response to invalid input or model-load failure; do not silently substitute uninitialized parameters.

Plan field updates before the first deployment. Validate compatibility and integrity before activation, retain a known working version where the product permits, and test interrupted update recovery. Collect appropriate operational counters without assuming that customer data may be uploaded. Treat a change to features, scaling, parameters, or the decision threshold as a versioned change requiring the relevant acceptance gates to be rerun. Apply the product’s authorization and authenticity policy to every model package, as to other executable configuration.

Must the microcontroller train the model? No. The proposed first pilot trains off-device and deploys an inference artifact. On-device adaptation would be a separate project with its own data, resource, persistence, validation, and rollback requirements. It is not necessary to establish the initial inference value.

Is an MLiglon SDK or exporter already available? This workflow makes no such release claim. Ask MLiglon which training builds, model formats, source or binary runtimes, export paths, documentation, and evaluation terms are currently available. Request a minimal supported example before making delivery commitments. [I12]

What should a model handoff contain? Request the complete graph and operator definitions, parameters, scales, rounding and saturation rules, preprocessing, labels, format version, and golden vectors. Ask whether connectivity and lookup tables are included, generated, or application-owned, and who maintains compatibility between trainer, exporter, and runtime.

Can a compiler or accelerator accept the model automatically? Treat that as an integration question to test. Map each operation and data type to an available execution path, then measure conversions and fallback work. Keep the scalar C implementation as the correctness reference when investigating a faster path.

Does a small parameter file prove the MCU will fit? No. The pilot must include feature buffers, live activations, graph metadata, scratch storage, stack, and other firmware. Precision names also do not replace range analysis: every tensor and intermediate needs its own justified representation.

Do matching golden vectors prove field accuracy? They establish agreement with the reference for the tested inputs. Predictive quality and robustness require separate held-out and operating-condition evaluations. Preserve both reports so that a faithful implementation is not mistaken for a sufficiently capable model.

Can one shared convolutional weight be treated as one local output change? No. The affected work includes every use of that parameter and the downstream dependencies of those uses. Confirm this mapping in both training and verification. The public IGL-CNN description explicitly includes shared kernels, so a two-input isolated-node test is insufficient for that case. [I07]

Who maintains the deployed model? Assign ownership for preprocessing, model regeneration, firmware integration, field diagnostics, and regression testing. Record the support period and the procedure for retiring an incompatible format. A technically successful evaluation still needs these operational responsibilities before product release.

When should licensing discussions advance? A bounded evaluation can establish whether the application benefits and what implementation work remains. Agree rights to the trainer, exported models, runtime distribution, updates, and support separately. MLiglon's published contact is info@mliglon.com; ask for current availability and the appropriate technical contact. [I12]

CHAPTER 12

Evidence, product pilots and licensing

12.1 Two verified United States patent grants

Grant and date	Published title	Inventor; assignee listed in public record
US 12,242,946 B1 4 March 2025	Integer gate logic artificial neural network	Michael J. Pelosi; MLIGLON, Inc. [I03]
US 12,488,222 B1 2 December 2025	Integer gate logic (IGL) artificial neural network with parrallelization and internal visualization capabilities	Michael J. Pelosi; MLIGLON, Inc. [I04]

The first application, 18/776,350, was filed on 18 July 2024. The second, 18/985,282, was filed on 18 December 2024 and is identified by the USPTO Gazette as a continuation-in-part of the first. Both records refer to provisional application 63/667,022 of 2 July 2024. The spelling “parrallelization” above reproduces the official second title. [I03, I04]

These grants establish identifiable intellectual property for a licensing discussion. They do not independently validate benchmark accuracy, speed, energy, or commercial readiness. Before a license is agreed, the parties should verify the current chain of title, relevant claims, jurisdictions, and rights offered. The assignee names above identify the reviewed public records; they are not a separate opinion on current title or claim coverage.

12.2 Reported results worth reproducing

The following figures are MLiglon-reported results, transcribed from its current website. They have not been independently reproduced for this brief. The comparison systems are the specific baselines described by MLiglon; these rows do not establish a general ranking of IGL against all conventional networks. [I10]

Reported experiment	IGL	Reported comparison
MNIST [I10]	99.28% final-test accuracy; 9.537 KB of one-byte parameters	99.22%; 37.908 KB of FP32 parameters
DBpedia [I10]	97.61% test accuracy; 35,408 bytes of parameter storage	97.56%; 1,415,404 bytes for the reported PyTorch baseline

Preserve the website's units when quoting these results: its MNIST summary uses “KB”; the DBpedia row uses exact byte counts. The DBpedia description further separates 5,068 internal nodes with three two-byte parameters from 5,000 one-byte input-node values. That accounting is useful for asking what else must be stored, including connectivity, activations, preprocessing assets, executable code, and work buffers. [I11]

MLiglon also reports a DBpedia inference-time advantage for test-set processing on an RTX 4090. This brief does not convert that report into a single-request latency or TI-device speedup. The reviewed summaries do not establish fully matched software pipelines, tuning effort, precision, or measurement boundaries, and this review did not obtain an independently executable reproduction package. Small accuracy differences likewise need repeated runs and uncertainty estimates before a superiority claim. [I10, I11]

12.3 An evidence ladder for adoption

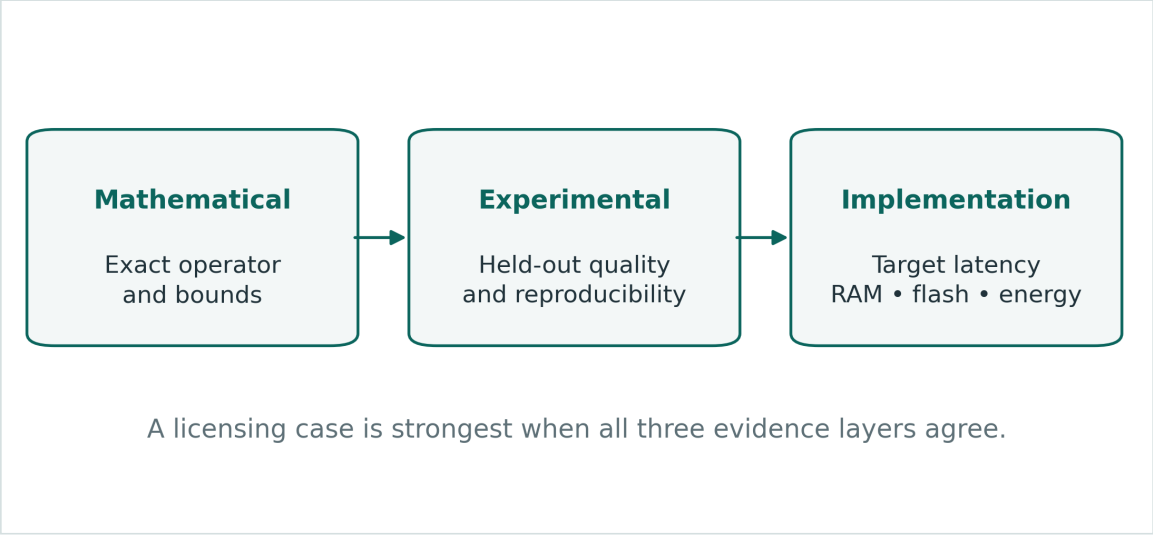


Figure 35. A strong licensing recommendation joins mathematical correctness, repeatable application quality and a measured implementation advantage.

The reviewed public materials support an evaluation opportunity. They do not establish an independent production deployment or a TI endorsement. MLiglon's applications page explicitly treats its examples as potential applications rather than proof of implementation or commercial validation. The absence of such evidence in this review is not a claim that no private deployment exists. [I13]

Evidence level	What it answers	Status or next evidence
Node mathematics	Does the specified node produce the intended responses?	Inspectable public demonstration and equations; validate target arithmetic.
Patent disclosure	Is there identifiable published intellectual property?	Two grants verified; perform transaction-specific rights review.
Developer benchmarks	Has the developer reported promising full-network results?	Yes; retain attribution and experimental conditions.
Independent reproduction	Can another team obtain the result from supplied artifacts?	Request data splits, code, models, environment, and measurement scripts.
Target-device qualification	Does the complete system meet product constraints?	Measure accuracy, memory, latency, energy, and robustness on the chosen device.
Production adoption	Can customers maintain a dependable deployed product?	Establish integration support, updates, operating limits, and field evidence.

12.4 A bounded technical evaluation before commercial scaling

A proposed evaluation should name one workload, one TI target, one accuracy requirement, and explicit resource limits. Agree whether the priority is inference deployment, off-device training efficiency, or a later adaptive-training investigation. Keep those objectives separate so that a successful classifier port is not mistaken for a completed training-system integration.

1 **Establish a reproducible package**
Request the model graph, parameter format, feature pipeline, training configuration, dataset split identifiers, seeds, dependency versions, and expected outputs. Define rounding, saturation, accumulation width, and loss arithmetic before comparing implementations.

2 **Use credible baselines**
Compare against an optimized quantized model on the same input representation and test set. Report accuracy at matched memory or latency budgets as well as the best achievable result. Record preprocessing and model-selection costs.

3 **Measure the whole deployment**
Include model bytes, program Flash, peak SRAM, scratch buffers, data movement, warm and cold timing, energy per inference, and error rates under representative disturbances. Document batch size, clock configuration, compiler options, and measurement boundaries.

4 **Define the licensing decision**
Set quantitative pass criteria and deliverables before work starts. A positive result should lead to an agreed scope covering implementation rights, training access, model distribution, integration support, and maintenance. Commercial terms remain for the parties to negotiate.

This process preserves the potentially valuable result even if IGL wins only within a particular constraint. A smaller classifier meeting a difficult SRAM limit can justify interest without a universal speed or accuracy claim. Conversely, a negative result on one task should identify whether representation, training, numerical format, or implementation was the limiting factor.

12.5 Training software and the next conversation

MLiglon plans to deploy broader IGL training software; this is the stated development direction for this brief, not a verified public release announcement. Contact MLiglon for current availability, evaluation access, supported configurations, and licensing information. The laboratory remains a mathematical demonstration and should not be presented as that future training product.

The current MLiglon contact page publishes **info@mliglon.com** for inquiries and identifies the company as based in Texas, USA. Contact information was verified on 11 November 2026. Website: https://mliglon.com/contact/. [I12]

12.6 A concrete next step for TI, ST and Arm ecosystem sponsors

Choose one application, one target board and explicit quality/resource thresholds. A TI installed-base pilot can start with MSP430 or MSPM0 firmware; an ST pilot can start with a manual scalar C port on an eligible existing device; an Arm ecosystem pilot can focus on a portable kernel, exporter or architecture-specific implementation. Agree who supplies the training evaluation, model contract, reference vectors and baseline builds. The first milestone is a useful product decision within the actual firmware budget.

A second milestone can be a maintained runtime or toolchain adapter used by several products. Custom instructions and accelerator IP follow when profiling establishes a repeated bottleneck and a silicon partner has an integration opportunity. An evaluation should identify software access, source/runtime rights, model-export rights, applicable patent licensing, support, maintenance and the responsibilities for any hardware work. This brief proposes a technical collaboration; it does not imply an existing vendor partnership.

Contact **info@mliglon.com** through MLiglon's contact page for training-software availability, evaluation access, technical support and licensing. Ask about supported architectures and precision, exported model format, compiler integration, model updates, production support, and the rights appropriate to software or silicon deployment. [I12]

CHAPTER 13

Extended technical glossary

13.1 IGL and laboratory mechanics

The definitions below consolidate the mathematical, software and hardware terminology used in the preceding chapters. Hardware details inherit the cited target-specific sources; general definitions are explanatory background, not additional performance claims.

Term	Definition and engineering significance
Activation function	A rule that transforms a weighted sum into a node output. The laboratory uses a continuous five-region map with rising, falling and saturated portions.
Activation region	One of the five intervals of S used by the IGL map. Region membership explains which local formula produced an output; a boundary label follows the implementation's interval convention.
Bias B	An additive shift of the activation input. In Lab INT arithmetic it is added after dot-product normalization, in the same scale units as S.
Boolean gate	A function on binary inputs. There are 16 possible two-input Boolean truth tables; the Lab provides one parameter construction for each.
Candidate	A proposed parameter configuration evaluated by a search. A candidate is distinct from an accepted model update and from an entire training epoch.
Chain isolation	MLiglon's approach to scoring proposed changes using cached unaffected computations and recalculating affected downstream dependencies. An affected region can branch rather than form a single path.
Corner / truth-table row	One of the four binary input pairs 00,01,10,11. Exact agreement on these observations does not define behavior at nonbinary coordinates.
Cross-section	A one-dimensional slice of the response field with the other input held fixed. It exposes slopes, plateaus, kinks and integer steps hidden by a viewing angle.
Exact gate	The Lab's label when all four corner outputs match one Boolean function within its numerical tolerance. It is not a claim about robustness, generalization or physical logic timing.
Gate replication	Choosing or learning parameters that reproduce a gate's output pattern. The replicated truth table need not imply the same transistor implementation or analog interpolation.
Graded response	An output with intermediate levels rather than only zero or one. In Real mode it varies continuously; integer mode restricts it to a finite grid.
IGL	Integer Gate Logic: the logic-capable node and network approach discussed by MLiglon. Separate its activation/representation, search method and hardware implementation when evaluating a claim.
INT mode	The Lab's integer arithmetic contract: rounded parameters and scaled inputs, one rounded dot normalization, then bias and activation. JavaScript still executes the reference arithmetic.
Live probe	The input pair currently selected for inspection. Changing it changes the live output; the four-row truth table continues to use binary corners.
Nearest gate	The Boolean table with the smallest mean absolute deviation from the four current outputs. A nearest match can still have substantial error.

Term	Definition and engineering significance
Non-monotonic	Having intervals where increasing an input quantity decreases the output, alongside intervals where it increases. IGL's folded activation is non-monotonic in its sum S.
Normalized output q	The raw node output divided by P. It lies in [0,1], but requires separate calibration evidence before being interpreted as a probability.
P / scale	The positive activation and integer encoding scale. It determines breakpoints and grid spacing; it does not specify the machine datatype, bus width or surface mesh density.
Precision map	The Lab's sampled absolute difference between INT and real evaluation with the same stored model values. Its sampled maximum is not an exhaustive global maximum.
Preset	A stored parameter construction for a gate or graded experiment. Loading a preset does not automatically change the separately chosen training target.
Real mode	Direct evaluation of fractional inputs and parameters in the Lab's reference arithmetic. It provides a comparison function, not a native FP32 performance benchmark.
Reference cache	Stored activations or other reusable results for an authoritative model state. A change invalidates every cached value whose dependencies changed.
Restart	A deliberate move to a different starting point when local exploration is unproductive or a search policy calls for broader coverage.
Response field	The output over all pairs in the selected input domain. A 3D surface or heatmap visualizes this function; plotting interpolation can conceal integer steps.
Weight W	A learned or selected coefficient multiplying an input. Its magnitude and sign affect sensitivity and the activation regions reached by the sum.

13.2 Learning, optimization and model structure

Term	Definition and engineering significance
Adam	A gradient-based optimizer that uses moving first- and second-moment estimates with bias correction. Its optimizer state adds training memory beyond parameters and activations.
Adjoint / reverse sensitivity	The derivative of a final scalar objective with respect to an intermediate value. Reverse-mode differentiation reuses these sensitivities through the computational graph.
Architecture	The graph of operators, connectivity, widths, channels and state that defines a model. It determines expressivity, memory lifetimes and hardware mapping before individual parameters are trained.
Backpropagation	Application of the chain rule in reverse through a model to compute loss derivatives. An optimizer then uses those derivatives; the two operations are not synonyms.
Batch / mini-batch	A set of examples processed together. It may improve computational reuse and supplies a sample-based estimate of training loss or gradients.
Calibration	Either estimation of quantization ranges from representative data or adjustment/evaluation of probabilistic confidence. Specify which meaning applies; they solve different problems.
CNN / convolution	A network using shared kernels over local receptive fields. Sharing reduces independent parameters but a changed kernel can affect many spatial outputs.
Computational graph / DAG	Operations connected by data dependencies. An acyclic graph admits a forward ordering; dependency reachability determines which computations must change after an update.

Term	Definition and engineering significance
Epoch	A pass through a designated training dataset. The Lab's candidate evaluations and iterations should not be relabeled as epochs.
Generalization	Performance on new examples drawn from the relevant deployment conditions. It must be assessed separately from fitting the training observations.
Gradient	A vector of partial derivatives of a scalar objective with respect to parameters. It describes local sensitivity, not a guaranteed route to a global optimum.
Inductive bias	Structural or procedural assumptions favoring some functions over others, such as spatial weight sharing, sparse connectivity or a logic-capable primitive.
Learning rate	The scale applied to a gradient-based parameter update. It belongs to the optimization policy, not to the definition of backpropagation itself.
Loss function	A numerical penalty comparing model outputs and targets, possibly including regularization. It should support, but need not equal, the final application acceptance metric.
MAE / MSE	Mean absolute error averages |prediction-target|; mean squared error averages its square. MSE emphasizes larger deviations. The Lab uses MAE for nearest-gate ranking and MSE for training.
Optimizer / search policy	The rule for proposing and accepting parameter changes. Examples include SGD, Adam, coordinate search and coarse-to-fine candidate exploration.
Overfitting	Fitting idiosyncrasies of training observations so well that performance on relevant unseen data is worse. More accurate training fit alone is not a remedy.
Plateau / local optimum	A region with little measured improvement, or a point with no improvement in a specified neighborhood. Neither proves the best possible solution over all parameter settings.
ReLU / rectifier	The activation max(0,z). Multilayer rectifier networks can construct non-monotone functions even though each individual rectifier is monotone.
Regularization	A penalty, constraint or procedure designed to control model complexity or encourage desired behavior. Its value must be checked against held-out performance.
SGD	Stochastic gradient descent: parameter updates using gradients estimated from sampled data. Scheduling, batch choice and momentum are separate implementation decisions.
Shared parameter	One value reused by several operations, as in a convolutional kernel. Its derivative or candidate change must account for all of those uses.
Training / validation / test	Training fits parameters; validation guides choices; the held-out test estimates final performance. Repeatedly tuning to a test set compromises that estimate.

13.3 Numerical representation and hardware

Term	Definition and engineering significance
Accumulator	An intermediate sum of products or errors. Its required range can be much larger than either the operand range or final output range.
Alignment / padding	Placement of data at required address boundaries and unused bytes inserted to meet a layout rule. Nominal element size does not determine complete structure size.
Bandwidth / coalescing	Bandwidth is data transferred per time; coalescing combines neighboring GPU lane accesses into efficient memory transactions. Small datatypes alone do not guarantee useful bandwidth.

Term	Definition and engineering significance
Bit-exact	Producing identical specified integer results for the same inputs. It requires matching rounding, saturation, overflow handling, boundaries and model representation.
C7x / C7504	TI DSP family and a specific member used in AM62D. The cited C7504 vector width is 256 bits; verify the actual device instead of assigning one width to the whole family.
Cortex-M	Arm's microcontroller processor family. DSP, FPU and MVE capabilities vary by core and implemented configuration; an instruction available on one target may be absent on another.
DMA	Direct memory access, which transfers data without the CPU executing every scalar move. Buffer ownership, coherency, setup cost and overlap still require explicit design.
DSP / FPU	A digital signal processor emphasizes suitable signal-processing operations; a floating-point unit executes floating arithmetic. A system may have both and support efficient integer and floating paths.
Fixed-point / scaled integer	An integer interpreted through an external scale, with a defined relationship between stored bits and represented value. Rounding and overflow remain part of its specification.
FP32	IEEE-style binary32 floating point: a 32-bit stored format with finite precision and a wide exponent range. It occupies the same bytes as INT32 but represents a different set of numbers.
INT8 / INT16 / INT32	Signed integers stored in 8,16 or32 bits respectively. Their finite ranges differ; an INT8 model may still require INT32 or wider accumulation.
ISA / ABI	The instruction-set architecture defines machine operations; the application binary interface defines calling, layout and linkage conventions. Both influence a correct portable deployment.
Lane / SIMD / SIMT	A lane is an element position in parallel execution. SIMD applies an operation to vector elements; GPU SIMT schedules threads in groups. Nominal lane counts are not throughput measurements.
MMA / Tensor Core	Matrix-multiply acceleration facilities with device-specific datatypes, tile sizes and accumulation rules. A sparse two-input primitive does not automatically map efficiently to a dense tile.
Mixed precision	Using different numerical widths or formats for weights, activations, products, accumulators or layers. Each transition needs an explicit range and conversion contract.
MVE / Helium	Arm's M-profile vector extension. Architectural register width and the implementation's datapath beats are distinct, as illustrated by the cited Cortex-M55 documentation.
Overflow / saturation	Overflow occurs when an exact result exceeds its representation. Saturation clamps to a limit. Wrapping, saturating and undefined signed C overflow are different behaviors.
PTQ	Post-training quantization: converting a trained representation to quantized inference, often with calibration or additional optimization. Quality depends on method, model and data.
QAT	Quantization-aware training: including quantization effects while fitting a model. It is a relevant embedded baseline and does not imply that every training operation uses native integers.
Rounding / truncation	Rounding chooses a nearby representable value under a specified tie rule; truncation discards the fractional part; C integer division truncates toward zero. They differ at many inputs.
SE / SA	TI C7000 streaming engine and streaming address generator. They support structured data access; use is constrained by the target and documented access patterns.

Term	Definition and engineering significance
SRAM / flash	SRAM commonly holds mutable working state; flash commonly holds persistent firmware and model data. Peak working memory and persistent storage are separate deployment budgets.
Warp / divergence	A CUDA execution group and the loss of uniform execution when its threads follow different control paths. Candidate/activation layout can influence divergence and utilization.

13.4 Evaluation, delivery and commercial decisions

Term	Definition and engineering significance
Acceptance criterion	A measurable threshold agreed before evaluating a candidate system, such as bounded latency, false alarms, peak SRAM or energy per decision.
Baseline	A relevant comparison implementation with declared data, preprocessing, precision, tuning and measurement conditions. A poorly matched baseline cannot establish the claimed advantage.
Candidate budget	The maximum number of scored proposals permitted in a search. It is separate from wall-clock time, restarts, iterations and dataset size.
Deployment artifact	A versioned package of parameters, topology, scales, preprocessing constants, runtime interfaces and verification vectors needed to execute the frozen model.
Determinism / reproducibility	Determinism gives the same result under the defined execution conditions; reproducibility additionally requires enough recorded information for another run or team to repeat the experiment.
Distribution shift	A change between training/validation conditions and deployment inputs. Sensor replacement, noise, operating ranges or user behavior can alter model performance.
Energy per inference	Energy within an explicitly stated measurement boundary for one decision or equivalent average. Include workload, idle accounting and preprocessing/transfer costs.
Golden vectors	Inputs with trusted intermediate and final expected outputs. They test the numerical contract across scalar references, optimized kernels, export/import and target firmware.
Latency / throughput	Latency is time for an individual request; throughput is completed work per unit time. Batching can increase throughput while increasing per-request latency.
Licensing evaluation	A bounded technical and commercial assessment of technology access, target value, integration work and rights. Patent identification alone does not establish measured product advantage.
Pareto frontier	Candidate designs for which no listed metric can be improved without worsening another. Accuracy, memory, energy and latency form a useful multi-objective design space.
Peak live state	The maximum simultaneously needed activations, scratch buffers, stacks and other mutable state. It is more informative for SRAM sizing than total parameter count alone.
Sensitivity / margin	Sensitivity describes output change under perturbations; margin describes distance from a decision or activation boundary. Neither is completely summarized by average accuracy.
Software-in-the-loop / target test	Evaluation using the deployed software model on a host, followed by execution on the actual hardware/compiler configuration. Both are needed to separate numerical and integration defects.
Version manifest / rollback	A manifest identifies model, preprocessing, runtime, compiler and data lineage; rollback restores a previously accepted compatible package when an update fails.

13.5 Compression, field deployment and specialized silicon

Term	Definition and significance
Knowledge compression effect	Compact executable structure that preserves the distinctions required by a specified task. In this brief it is an engineering interpretation, not a claim that a model losslessly stores its training set.
Minimum description length (MDL)	A model-selection viewpoint that balances the length of a model description with the description of the data or errors left unexplained by it.
Task sufficiency	A representation retains the target information needed for prediction when the original input would add no further predictive information under the stated distribution.
Rate-distortion	An information-theoretic relationship between representation rate and an explicitly defined reproduction distortion. The measured model-size frontier here is an engineering analogue, not that theorem itself.
Resident image	Code and data required while the deployed inference implementation is resident. Its accounting differs from compressed archive size and peak working RAM.
Operator placement	The compiler-selected execution location for each graph operation, such as the host CPU or an accelerator. Verify the placement report and account for transitions between locations.
Soft-float	Floating-point operations implemented through software helpers on the selected execution path. This is distinct from an ABI choice that can affect argument passing.
Widening multiply	An operation or implementation that preserves a product in a wider type than its stored operands. A wide destination alone does not guarantee wide evaluation in C.
Peripheral ownership	A rule controlling which task or interrupt may use shared hardware state, such as a memory-mapped multiplier, during an operation.
Shadow evaluation	Run a new decision function alongside the existing product behavior and record its decisions before granting it control authority.
High-water mark	The largest observed resource use, such as stack depth, during a defined test workload. It is not automatically a proved worst-case bound.
Model manifest	Versioned metadata describing the graph, numerical rules, input preprocessing, compatibility and output interpretation expected by a runtime.
Hysteresis	Different transition thresholds for entering and leaving a state, used to reduce repeated switching near a decision boundary.
Debounce	A rule requiring persistence or repeated agreement before accepting a state change; it affects latency and error behavior.
Custom operator	A computation supplied outside a toolchain’s built-in operator set. Its arithmetic, parameters, implementation and placement must be supported explicitly.
CPU fallback	Execution on the host CPU for work not placed on an accelerator. It requires a valid host implementation and can add transfer and synchronization costs.
Custom Datapath Extension (CDE)	An optional Arm processor integration facility for custom compute operations in supported IP. Actual instructions require silicon implementation and toolchain support.
Instruction-set architecture (ISA)	The visible contract of machine operations, operands, encodings, state, results and exceptions that software targets.
RTL	Register-transfer-level hardware description used to specify and implement clocked digital logic. Simulation and synthesis answer different questions about that description.

Term	Definition and significance
FPGA bitstream	Configuration data that programs supported logic and routing on an FPGA. It cannot rewire a conventional fixed-silicon microcontroller.
ASIC	An application-specific integrated circuit. Fixed logic is manufactured into silicon; any model reprogrammability must be designed into it.
PPA	Power, performance and area, evaluated under stated technology and operating conditions. FPGA resource counts do not directly establish ASIC PPA.
Nonrecurring engineering (NRE)	One-time development and integration cost, often amortized across shipped devices in a product-cost analysis.
Amdahl’s law	Whole-workload acceleration is limited by the fraction left unchanged. The brief uses exact time accounting with affected fraction f and primitive speedup r.

CHAPTER 14

References and verification notes

14.1 Historical learning and quantization

Source review: 11-12 November 2026. Titles are clickable. Primary papers and official documentation support the background and hardware claims; MLiglon material supports attributed company descriptions and reported results. New field budgets, instruction candidates and silicon diagrams are explicitly labeled engineering examples or proposals. Native target latency, power and firmware fit were not measured for this brief.

[H01] Warren S. McCulloch; Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 1943.

[H02] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. 1958.

[H03] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. 1976.

[H04] Paul John Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. 1974.

[H05] David E. Rumelhart; Geoffrey E. Hinton; Ronald J. Williams. Learning representations by back-propagating errors. 1986.

[H06] Yann LeCun; Bernhard Boser; John S. Denker; Donnie Henderson; Richard E. Howard; Wayne Hubbard; Lawrence D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. 1989.

[H07] Yann LeCun; Léon Bottou; Yoshua Bengio; Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. 1998.

[H08] Alex Krizhevsky; Ilya Sutskever; Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. 2012.

[H09] Kaiming He; Xiangyu Zhang; Shaoqing Ren; Jian Sun. Deep Residual Learning for Image Recognition. 2015 preprint; 2016 conference.

[H10] Ashish Vaswani; Noam Shazeer; Niki Parmar; Jakob Uszkoreit; Llion Jones; Aidan N. Gomez; Łukasz Kaiser; Illia Polosukhin. Attention Is All You Need. 2017.

[H11] Herbert Robbins; Sutton Monro. A Stochastic Approximation Method. 1951.

[H12] Diederik P. Kingma; Jimmy Ba. Adam: A Method for Stochastic Optimization. 2014 preprint; 2015 conference.

[H13] Yoshua Bengio; Patrice Simard; Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. 1994.

[H14] Corinna Cortes; Vladimir Vapnik. Support-vector networks. 1995.

[H15] Leo Breiman. Random Forests. 2001.

[H16] Tianqi Chen; Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. 2016.

[H17] Benoit Jacob; Skirmantas Kligys; Bo Chen; Menglong Zhu; Matthew Tang; Andrew Howard; Hartwig Adam; Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. 2017 preprint; 2018 conference.

[H18] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. 2018.

[H19] Markus Nagel; Rana Ali Amjad; Mart van Baalen; Christos Louizos; Tijmen Blankevoort. Up or Down? Adaptive Rounding for Post-Training Quantization. 2020.

[H20] Liangzhen Lai; Naveen Suda; Vikas Chandra. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. 2018.

[H21] Yoshua Bengio; Nicholas Léonard; Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. 2013.

[H22] Xavier Glorot; Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010.

14.2 Compression theory and representation

[K01] Jorma Rissanen. Modeling by shortest data description. 1978.

[K02] Anselm Blumer; Andrzej Ehrenfeucht; David Haussler; Manfred K. Warmuth. Occam's Razor. 1987.

[K03] Claude E. Shannon. Coding Theorems for a Discrete Source With a Fidelity Criterion. 1959.

[K04] Naftali Tishby; Fernando C. Pereira; William Bialek. The information bottleneck method. 1999.

[K05] Geoffrey E. Hinton; Drew van Camp. Keeping Neural Networks Simple by Minimizing the Description Length of the Weights. 1993.

[K06] Song Han; Huizi Mao; William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. 2016.

[K08] Mark Horowitz. Computing's Energy Problem (and what we can do about it). 2014.

14.3 Numerical implementation, TI DSP and GPUs

[T01] Texas Instruments. AM62Dx Sitara Processors datasheet, SPRSPB5A, Rev. A (May 2025).

[T02] Texas Instruments. C7000 Optimizing C/C++ Compiler User's Guide, v5.0.0.LTS, SPRUIG8K.

[T03] Texas Instruments. C7000 Optimization Guide §4.12: Vector Programming and Scalable Vector Programming Model.

[T04] Texas Instruments. C7000 Optimization Guide §6.1: Streaming Engine.

[T05] Texas Instruments. C7000 Optimization Guide §6.2: Streaming Address Generator.

[T06] Texas Instruments. C7000 Optimization Guide §4.3: Automatic Use of SE and SA.

[T07] Texas Instruments. C7x DSPLIB official repository: README and supported-device table.

[T08] Arm. Cortex-M4 Processor Datasheet.

[T09] Arm / CMSIS. CMSIS-Core v6.3.0: Intrinsic Functions for SIMD Instructions.

[T10] Arm. Cortex-M55 TRM r1p1, 101051_0101_03_en, Issue 03 (2024).

[T11] Arm / CMSIS. CMSIS-NN library documentation.

[T12] Arm / CMSIS. CMSIS-NN Fully-connected Layer Functions.

[T13] NVIDIA. CUDA Programming Guide §3.2: Advanced Kernel Programming.

[T14] NVIDIA. CUDA C++ Best Practices Guide.

[T15] NVIDIA. CUDA Programming Guide §5.4: C/C++ Language Extensions.

[T16] NVIDIA. CUDA Programming Guide §5.1: Compute Capabilities.

[T17] NVIDIA. Floating Point and IEEE 754 Compliance for NVIDIA GPUs.

[T18] ISO WG14. N1570 C11 committee draft (April 2011).

[T20] Texas Instruments. MSPM0G5187 product information and development resources.

[T21] Texas Instruments. MSPM0G5187 Mixed-Signal Microcontrollers With TinyEngine NPU datasheet, SLVSJB5B, Rev. B (March 2026).

14.4 TI legacy microcontrollers

[V01] Texas Instruments. MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller datasheet, SLAS368G. 2011.

[V02] Texas Instruments. MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller datasheet, SLAS735J. 2013.

[V03] Texas Instruments. MSPM0G350x Mixed-Signal Microcontrollers With CAN-FD Interface datasheet, SLASEX6C. 2025.

[V04] Texas Instruments. TI Arm Clang Compiler Tools User's Guide, Commonly Used Options: Floating-Point Support.

[V05] Arm. Arm Cortex-M0 Processor Datasheet. 2020.

[V06] Texas Instruments. MSP430x1xx Family User's Guide, SLAU049F. 2006.

[V07] Texas Instruments. MSP430 Embedded Application Binary Interface, SLAA534A. 2020.

[V08] Texas Instruments. MSP430F2xx, MSP430G2xx Family User's Guide, SLAU144K. 2022.

14.5 STMicroelectronics devices and tools

[S01] STMicroelectronics. STM32F103x8/STM32F103xB datasheet, DS5319 Rev 20. 2025.

[S02] STMicroelectronics. STM32G071x8/xB datasheet, DS12232 Rev 5. 2025.

[S03] STMicroelectronics. STM8S003F3/STM8S003K3 datasheet, DS7147 Rev 10. 2018.

[S04] STMicroelectronics. STM32U575xx datasheet, DS13737 Rev 10. 2024.

[S05] STMicroelectronics. STM32N6x5xx/STM32N6x7xx datasheet, DS14791 Rev 11. 2026.

[S06] STMicroelectronics. STM32Cube AI Studio documentation.

[S07] STMicroelectronics. Keras custom layer support - ST Edge AI Core 4.1.0.

[S08] STMicroelectronics. Neural-ART - Supported operators and limitations, ST Edge AI Core 4.1.0.

14.6 Arm IP and tools

[A01] Arm Holdings plc. Arm Holdings plc annual report, fiscal year ended 31 March 2026. 2026.

[A02] Arm Limited. Arm Cortex-M Processor Comparison Table. 2023.

[A03] Arm Limited / ARM-software. CMSIS-NN software pack documentation.

[A04] Arm Limited. Arm Cortex-M55 Processor Datasheet. 2020.

[A05] Arm Limited / ARM-software. Ethos-U Integration for Cortex-M: Vela.

[A06] Arm Limited / ARM-software. Get Started with Arm Custom Instructions (ACI).

[A07] Arm Limited / ARM-software. Arm C Language Extensions - Custom Datapath Extension.

[A08] Arm Limited. Arm Flexible Access: Frequently Asked Questions.

14.7 Dedicated hardware architecture

[D01] NVIDIA. NVDLA Primer.

[D02] RISC-V International. Unprivileged ISA, 20260120: RV32/64G Instruction Set Listings. 2026.

14.8 MLiglon public evidence and intellectual property

[I01] MLiglon Corporation. MLiglon: Integer Gate Logic.

[I02] MLiglon Corporation. IGL Lab Introduction.

[I03] Michael J. Pelosi; assignee MLIGLON, Inc. US12242946B1: Integer gate logic artificial neural network. 2025-03-04.

[I04] Michael J. Pelosi; assignee MLIGLON, Inc. US12488222B1: Integer gate logic (IGL) artificial neural network with parrallelization and internal visualization capabilities. 2025-12-02.

[I05] MLiglon Corporation. IGL Chain Isolation Training on NVIDIA GPUs Using CUDA.

[I06] MLiglon Corporation. Integer Gate Logic Training on NVIDIA GPUs Using CUDA.

[I07] MLiglon Corporation. Convolutional IGL Networks.

[I08] MLiglon Corporation. How IGL Learns: Chain Isolation Optimization Versus Conventional Backpropagation.

[I09] MLiglon Corporation. Integer Gate Logic vs. INT8 Neural Networks: Beyond Quantization to Architectural Efficiency.

[I10] MLiglon Corporation. Integer Gate Logic Benchmark Results: Accuracy, Memory, and Computational Efficiency.

[I11] MLiglon Corporation. Integer Gate Logic: Benchmark Results and the Case for a New Class of Efficient AI Architectures.

[I12] MLiglon Corporation. Contact MLiglon Corporation.

[I13] MLiglon Corporation. Integer Gate Logic for Microcontrollers, TinyML, IoT, and Edge AI.

[I16] MLiglon Corporation. How Integer Gate Logic Emulates Digital Logic Gates Using Weights, Bias, and a Piecewise Activation Function.

14.9 Laboratory and reproduction sources

[L01] MLiglon laboratory source. IGL Laboratory 2.0.1 deterministic numerical core. 2.0.1. Released local source inspected; activation, presets, integer arithmetic and local search.

[L02] MLiglon laboratory source. IGL Laboratory 2.0.1 user interface and analysis. 2.0.1. Released local source and live UI inspected; numerical panel screenshots captured from the public app.

[L03] MLiglon laboratory source. IGL Laboratory 2.0.1 teaching search worker. 2.0.1. Released local source inspected; single-node CPU candidate search, bounded budgets and seeded restarts.

[L04] Technical-brief verification. Reproduced laboratory experiments for this brief (internal verification record). 2026. Worked outputs, all 352 gate/mode/P combinations and the two stated search runs checked against the released 2.0.1 source. Hardware performance was not benchmarked.

14.10 Illustration and verification notes

The retained screenshots were captured from IGL Laboratory 2.0.1 for the first edition. The documentation browser used Canvas2D fallback; the installation owner separately confirmed GPU/WebGL operation and PNG export. Numerical panels retain original screenshot pixels. New diagrams and analytical plots are original technical illustrations. The source-based gate and training checks remain the reference; added rounding identities and illustrative memory calculations were checked separately. Target deployment requires functional, performance, robustness and integration validation on the selected hardware.

MLiglon's planned broader training-software deployment was supplied as commissioning context. Readers should contact MLiglon for current availability, supported formats and licensing. This edition proposes software and hardware evaluation paths; it does not announce a TI, ST or Arm partnership, a native IGL accelerator, or a released dedicated instruction set.

Reviewed source	SHA-256
core.js	38ddde13e90805bdb2c6a68539c203d1f6cc54e83f293d6208457a1c495ede75
app.js	a03672e43400c4e739dd64c7474fc4ad18b29f658a7da1a4c538b85ac1d2332d
training-worker.js	46b73181980971c721ab7251924a4f847427ca1046dcd2daeca74d6cd7c410e1