OPTIMIZATION TIER

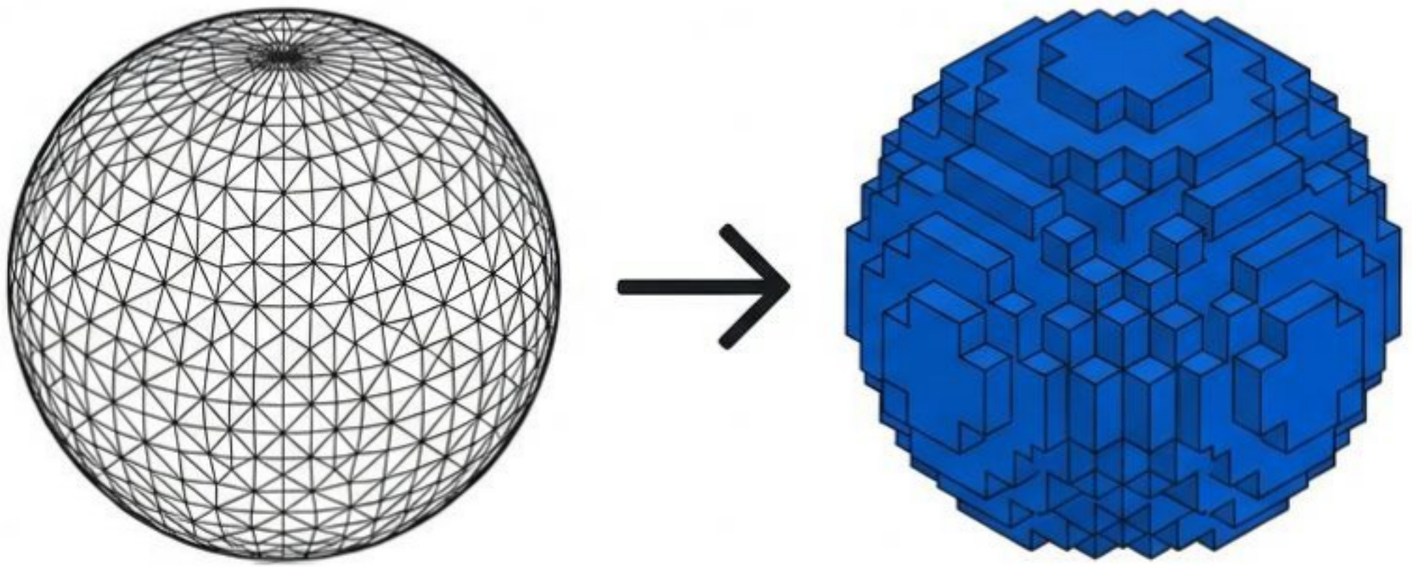
MODULE 15

Quantization

Optimizing memory and computation through reduced precision

Module 15: Quantization

Optimization Tier | Concept → System → Code



Continuous FP32

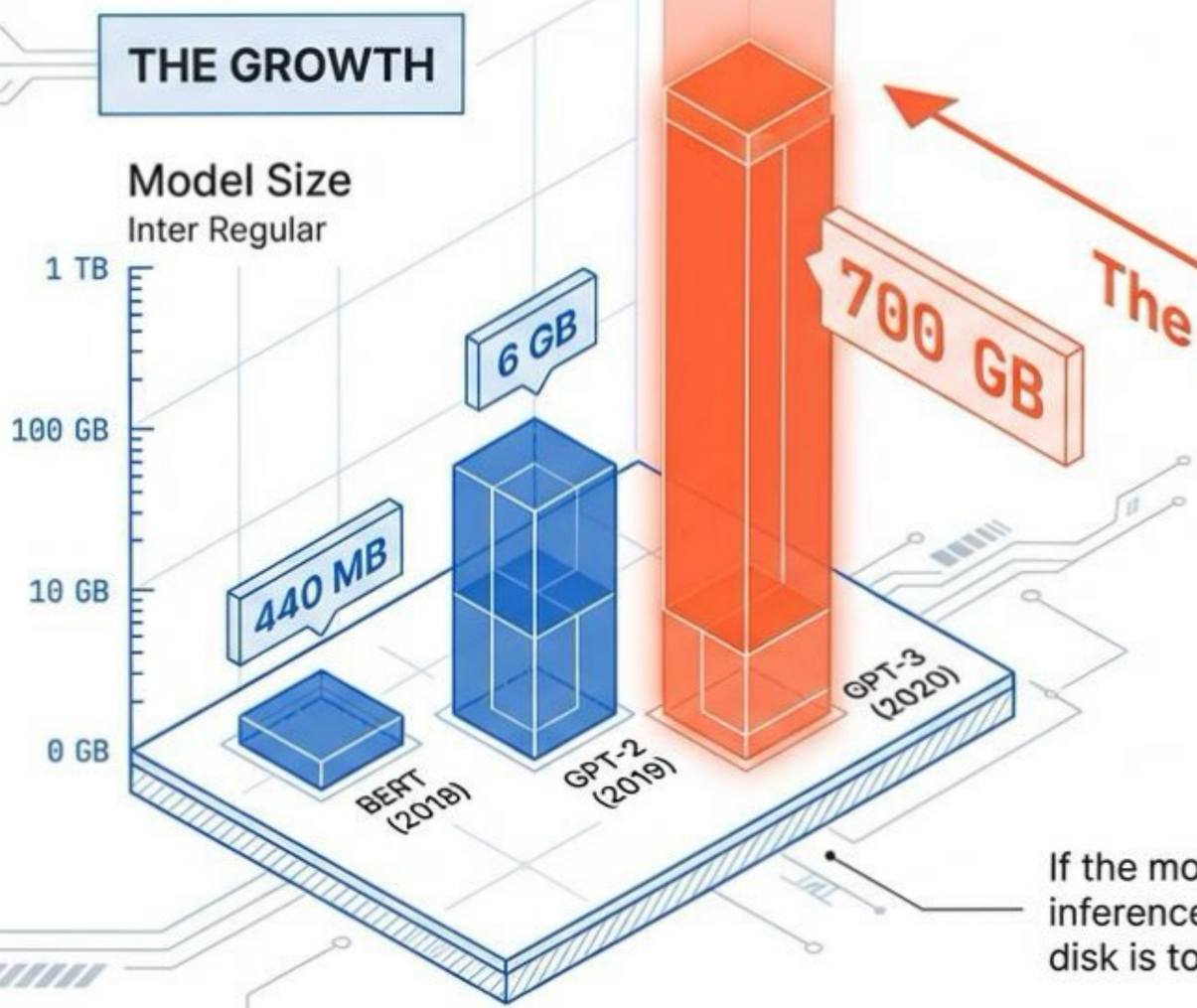
Discrete INT8

Shrinking model memory footprints by 4x using INT8 precision.

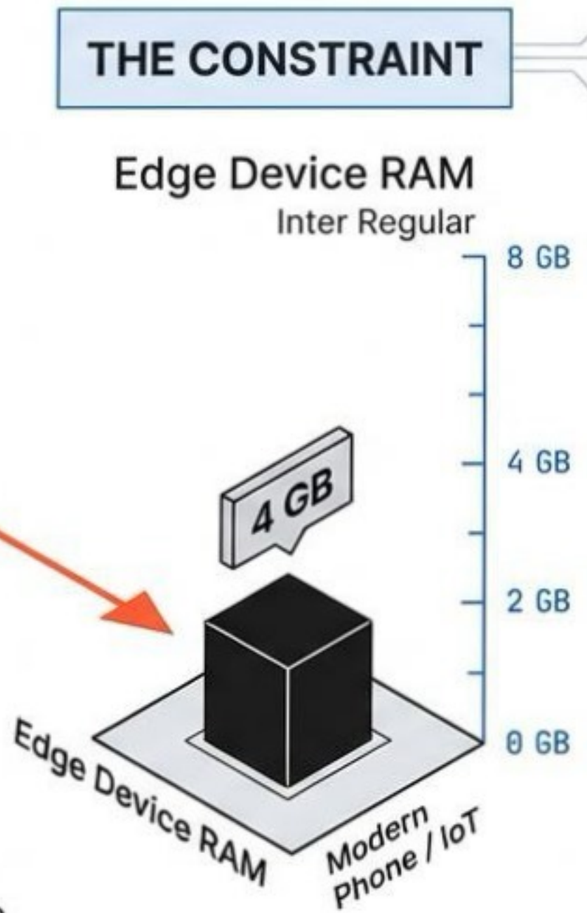
The Memory Wall

Deep Learning models are growing exponentially. Hardware is not.

THE GROWTH



THE CONSTRAINT

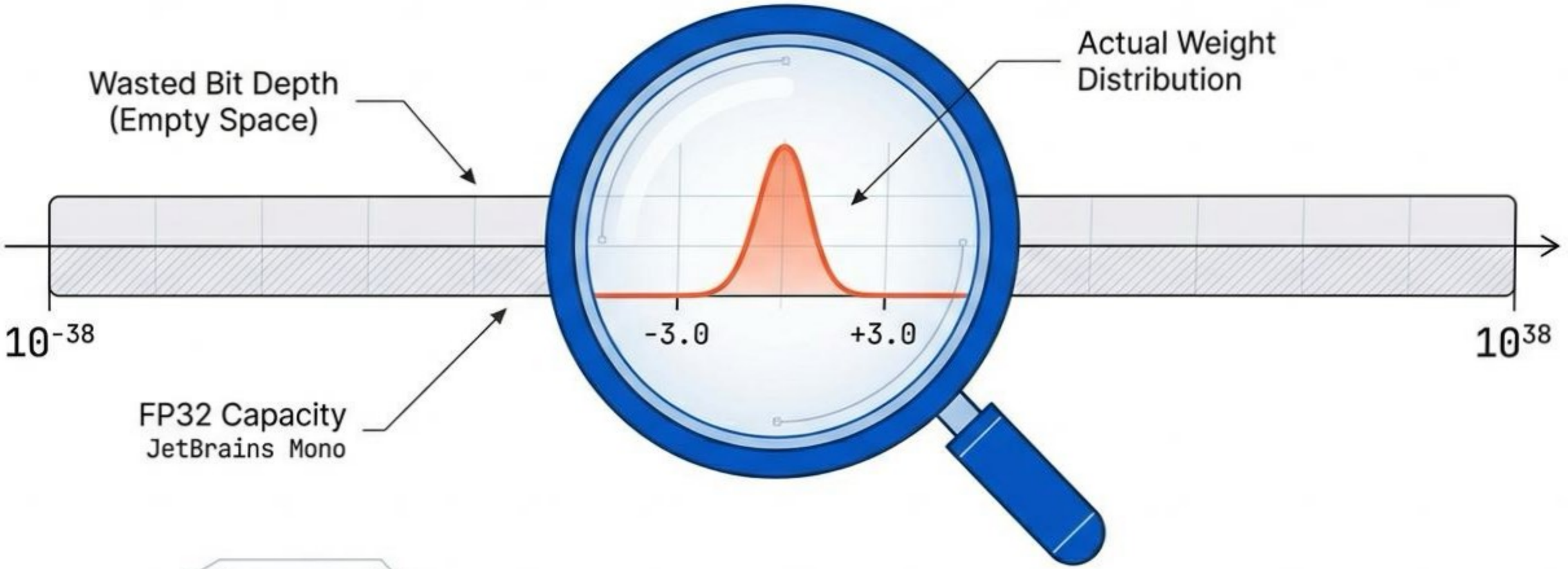


The Deployment Gap

If the model is larger than RAM, inference is impossible. Swapping to disk is too slow for real-time AI.

The Precision Paradox

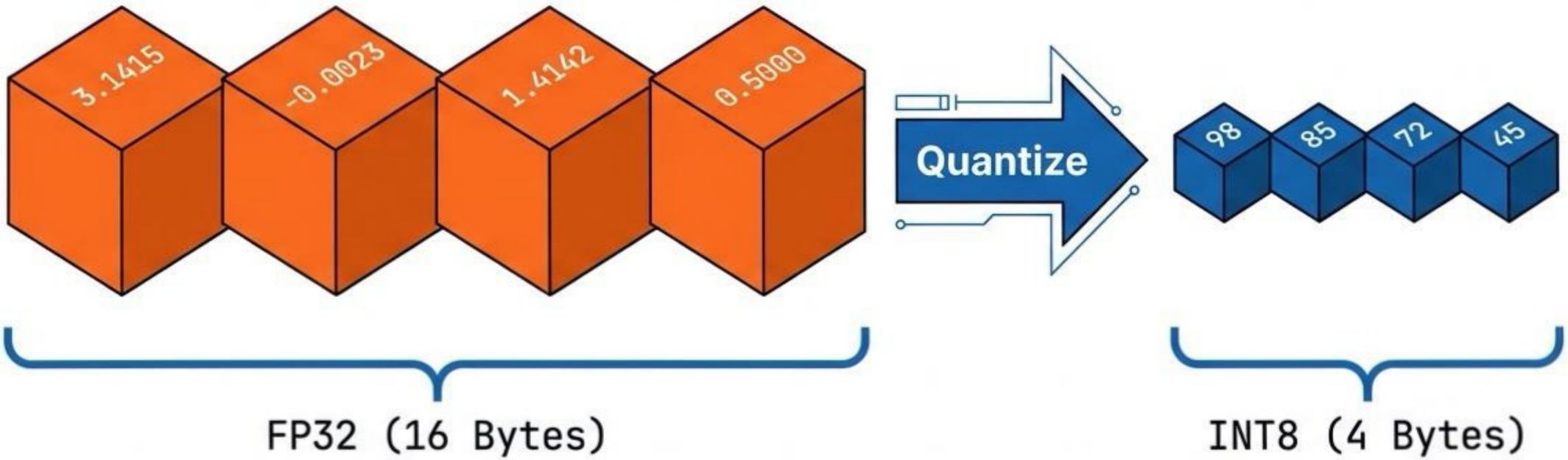
We use galactic scales to measure microscopic values.



We pay 32 bits of storage cost for ~3 bits of actual signal.

The Solution: INT8 Quantization

Deterministic 4x compression.



400 MB Model → 100 MB Model

The Math: Affine Mapping

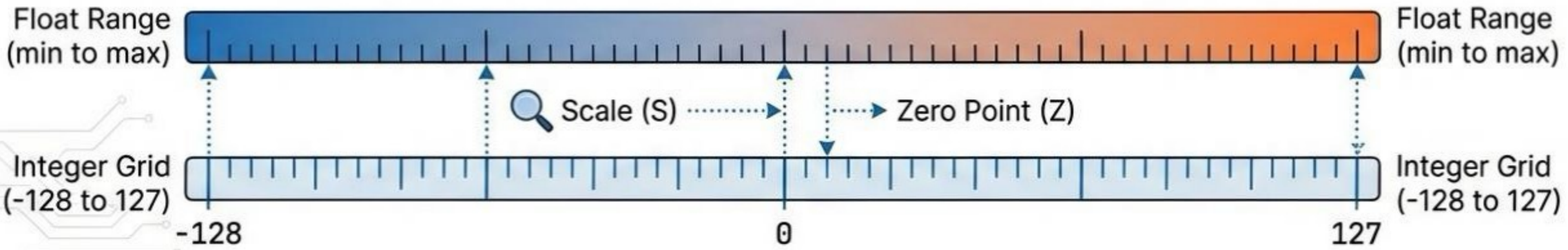
Mapping the continuous to the discrete.

Scale (float). The step size or "resolution" of the grid.

Zero Point (int). The integer value that maps to real 0.0.

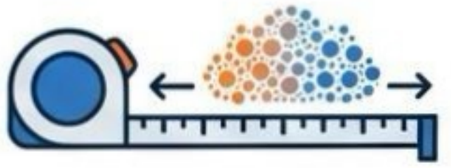
$$x_{float} \approx S \cdot (x_{int} - Z)$$

The stored 8-bit integer (-128 to 127).



Implementation: quantize_int8

```
def quantize_int8(tensor: Tensor):
    data = tensor.data
    # 1. Find dynamic range
    min_val, max_val = np.min(data), np.max(data)

    # 2. Calculate Scale
    scale = (max_val - min_val) / (255)

    # 3. Calculate Zero Point
    zero_point = int(round(-min_val / scale)) + (-128)

    # 4. Quantize
    q_data = np.round(data / scale + zero_point)
    return Tensor(q_data.astype(np.int8)), scale, zero_point
```



Find Range

Calculate Step Size (Scale)

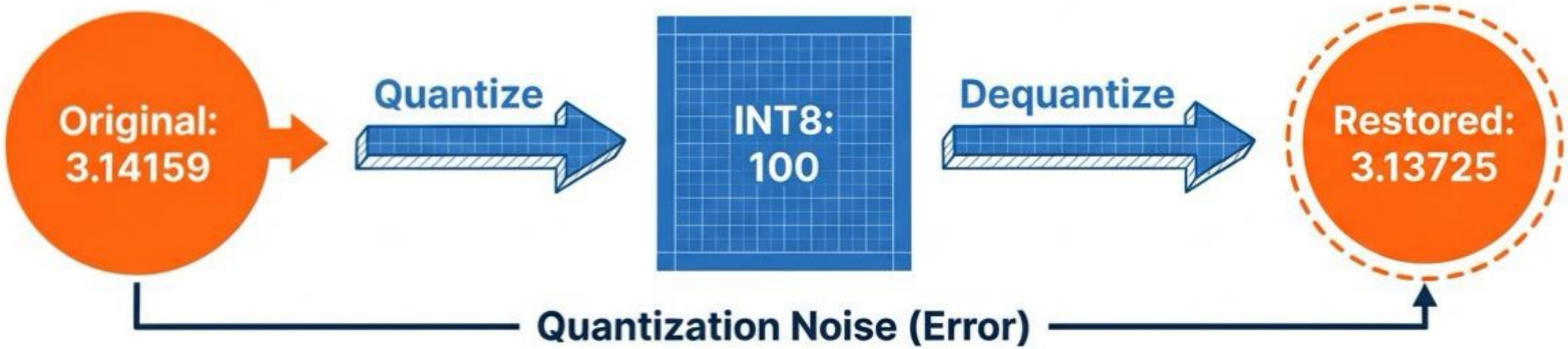
Align Zero

Round to Nearest

Implementation: dequantize_int8

Restoring the data (with noise).

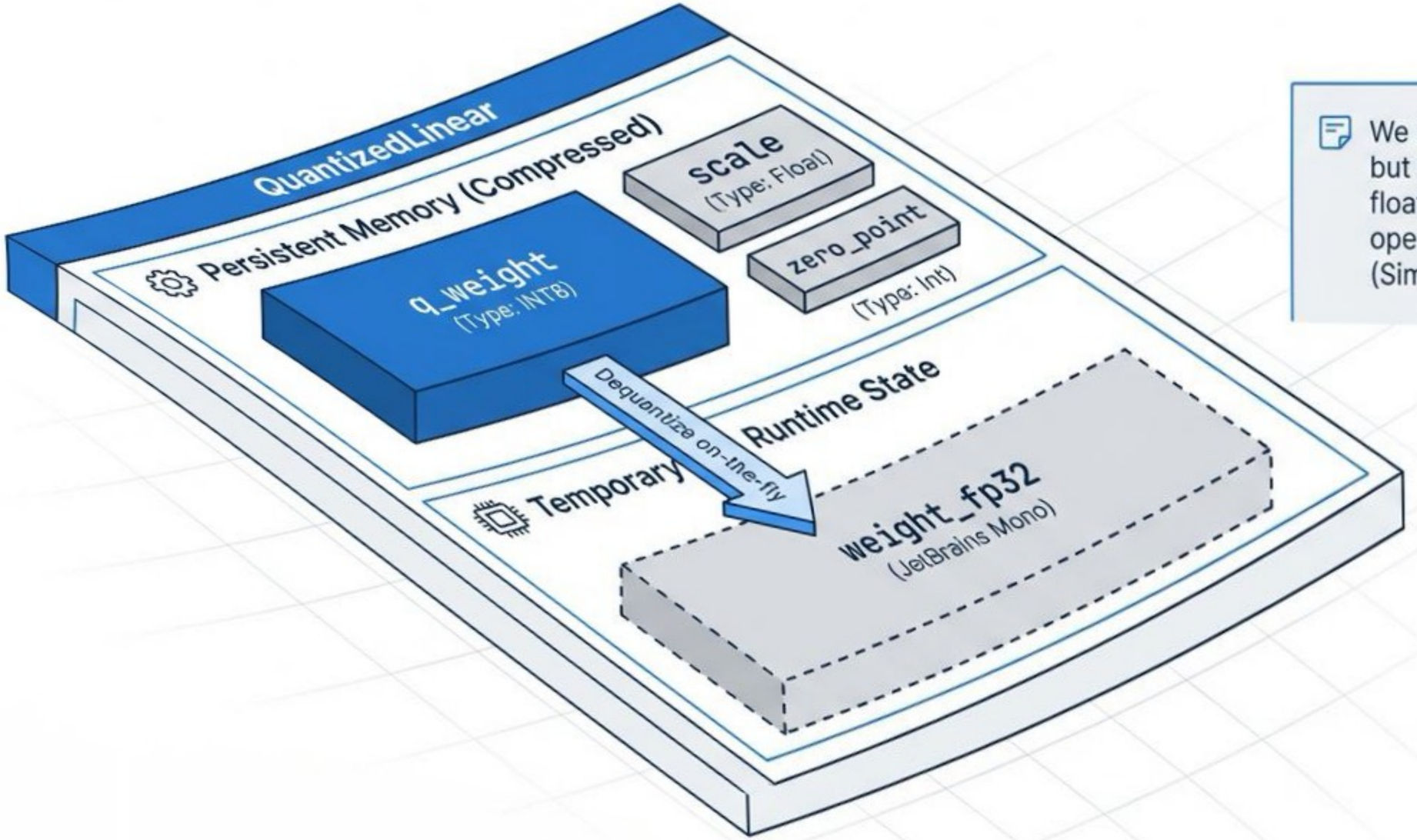
```
def dequantize_int8(q_tensor, scale, zero_point):
    """Restore FP32 from INT8."""
    # FP32 = (INT8 - Zero_Point) * Scale
    fp32_data = (q_tensor.data - zero_point) * scale
    return Tensor(fp32_data)
```



The restored value is never identical to the original. It is an approximation bounded by the Scale.

Systems Constraint: The Layer Wrapper

Simulating hardware constraints in software.

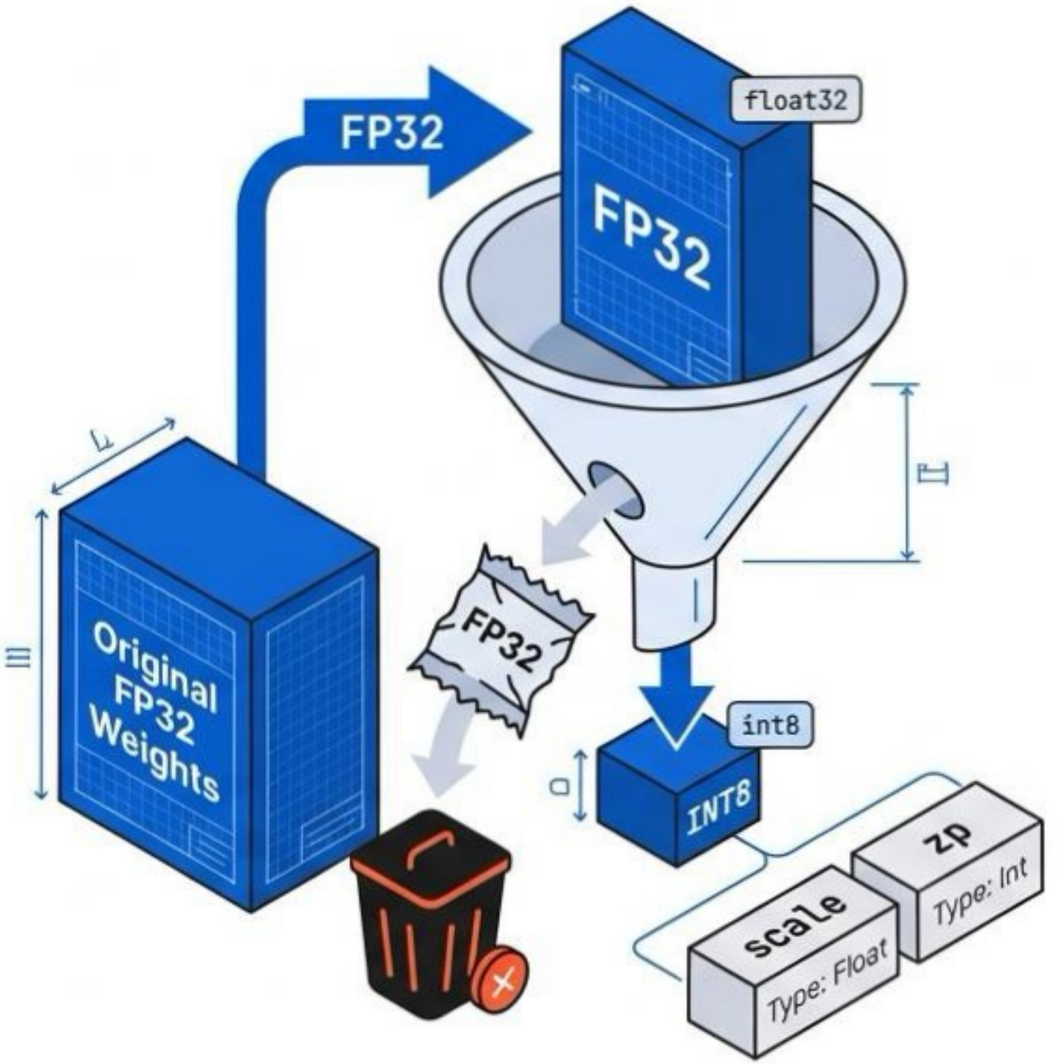


We store cheap integers, but convert to expensive floats just for the math operation (Simulated Quantization).

Code: Layer Initialization

Compressing at the moment of creation.

```
class QuantizedLinear:
    def __init__(self, linear_layer: Linear):
        # 1. Take original FP32 weights
        original_w = linear_layer.weight

        # 2. Quantize immediately and discard original
        self.q_weight, self.scale, self.zp = \
            quantize_int8(original_w)

        # 3. Repeat for bias...
        if linear_layer.bias is not None:
            self.q_bias, ... = quantize_int8(...)
```

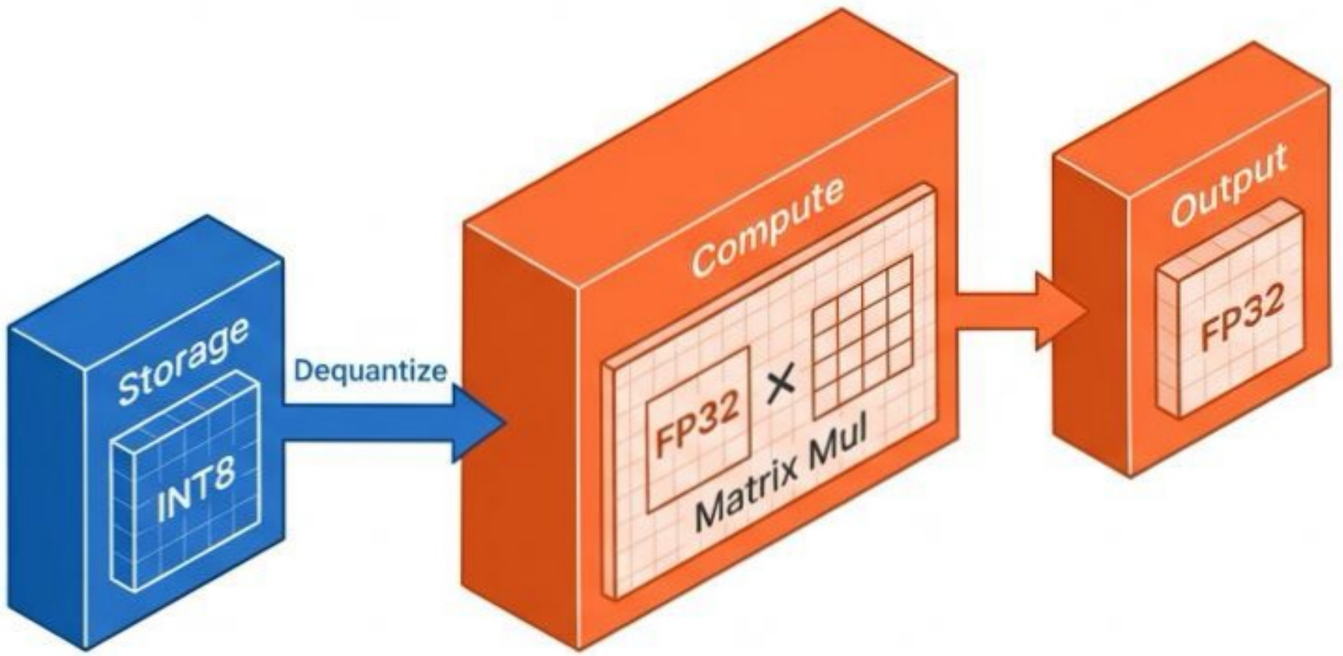


Code: The Forward Pass

The Dequantize-Compute Pattern.

Code Block

```
def forward(self, x: Tensor) -> Tensor:

    # 1. Expand INT8 weights to FP32
    w_fp32 = dequantize_int8(
        self.q_weight, self.scale, self.zp
    )

    # 2. Standard Matrix Multiplication
    # x is FP32, w_fp32 is FP32
    return x.matmul(w_fp32)
```

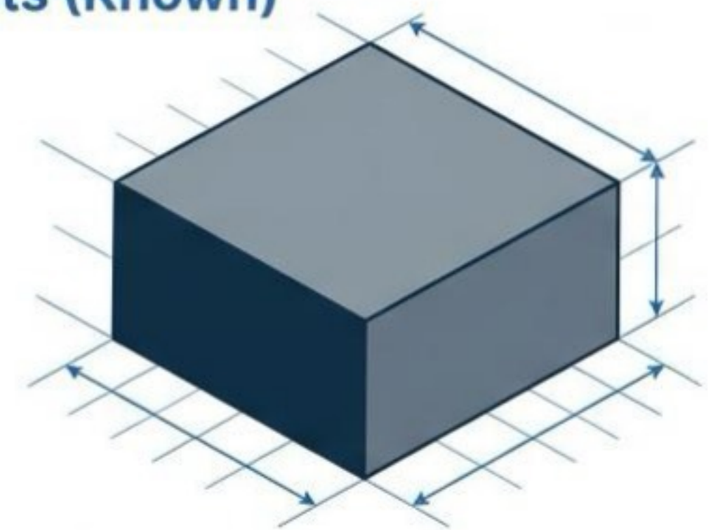


Note: In dedicated hardware (VNNI/Tensor Cores), steps 1 and 2 are fused into a single INT8 instruction.

The Input Problem

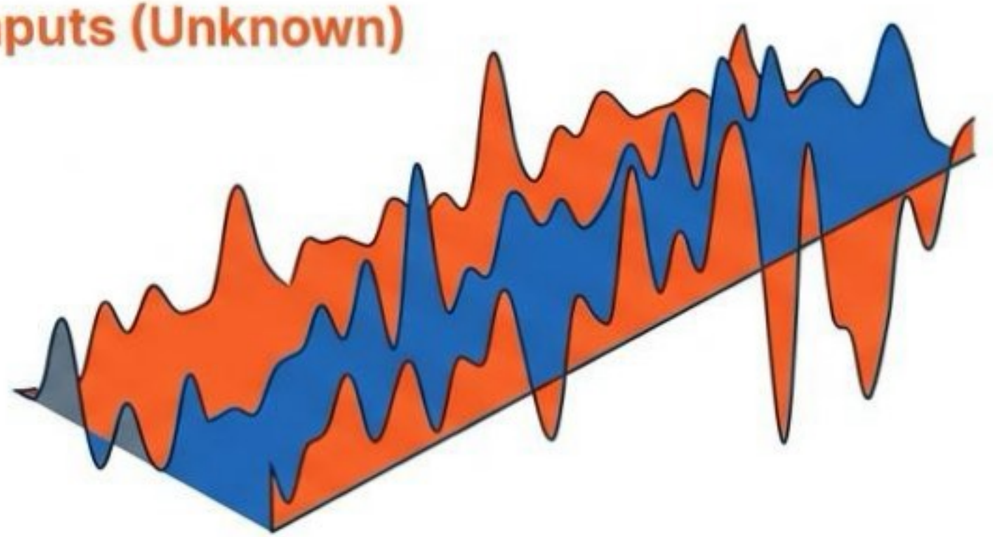
Weights are static. Inputs are dynamic.

Weights (Known)



Range is fixed after training. We calculate Min/Max once.

Inputs (Unknown)



Range changes with every user. Is the max 1.0? 50.0? 1000.0?

How do we choose a Scale for the input if we don’t know its future values?

Calibration.

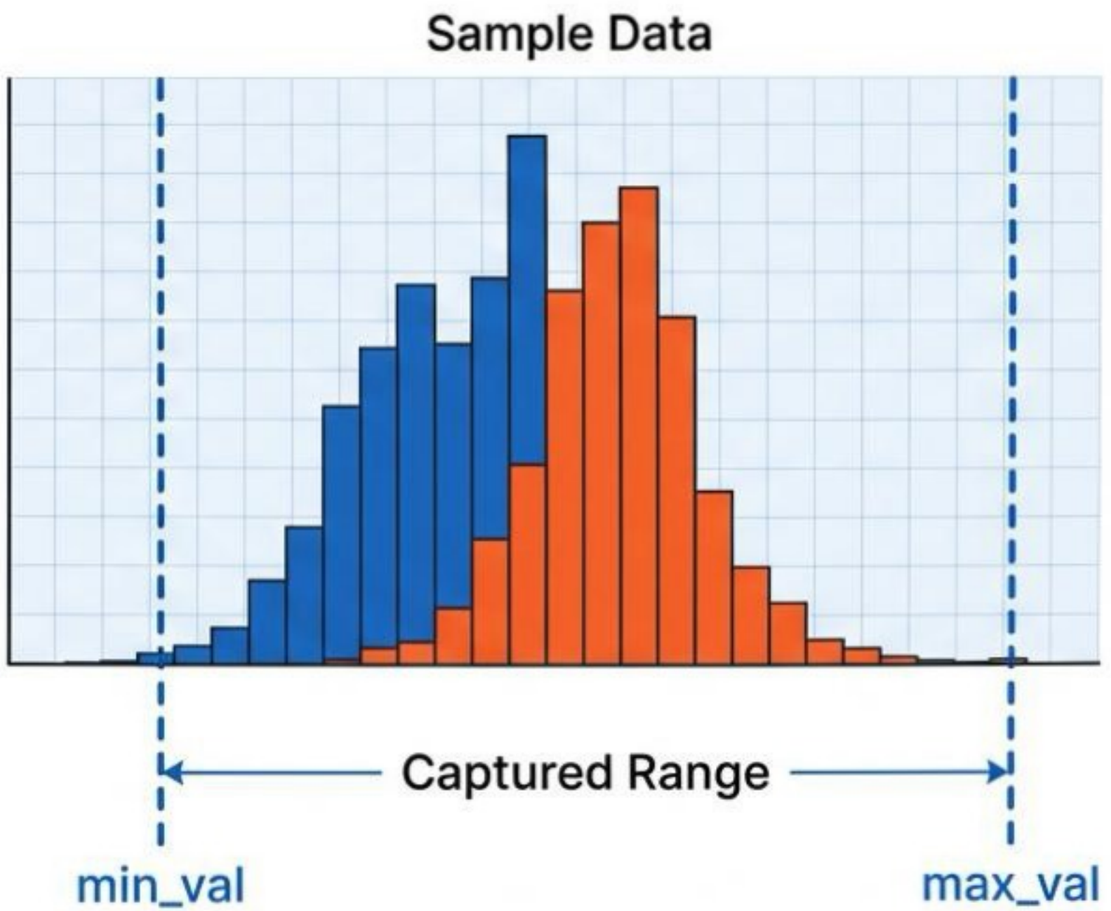
Engineering Editorial

Implementation: calibrate()

Sampling the world to set boundaries.

Code Block

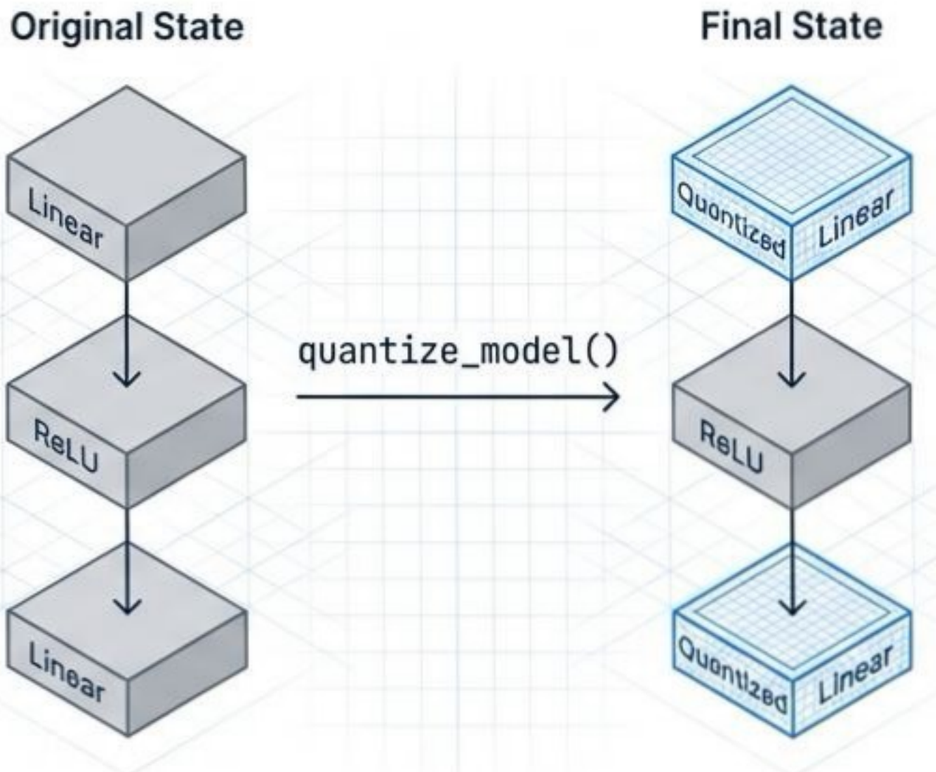
```
def calibrate(self, sample_inputs: List[Tensor]):
    """Observe data to fix input parameters."""
    all_values = []

    # 1. Collect statistics from representative data
    for inp in sample_inputs:
        all_values.extend(inp.data.flatten())

    # 2. Find the 'horizon'
    min_val, max_val = np.min(all_values),
        np.max(all_values)

    # 3. Lock the parameters
    self.input_scale = (max_val - min_val) / 255
    self.input_zp = ...
```



Full Model Integration: quantize_model

In-place graph replacement.

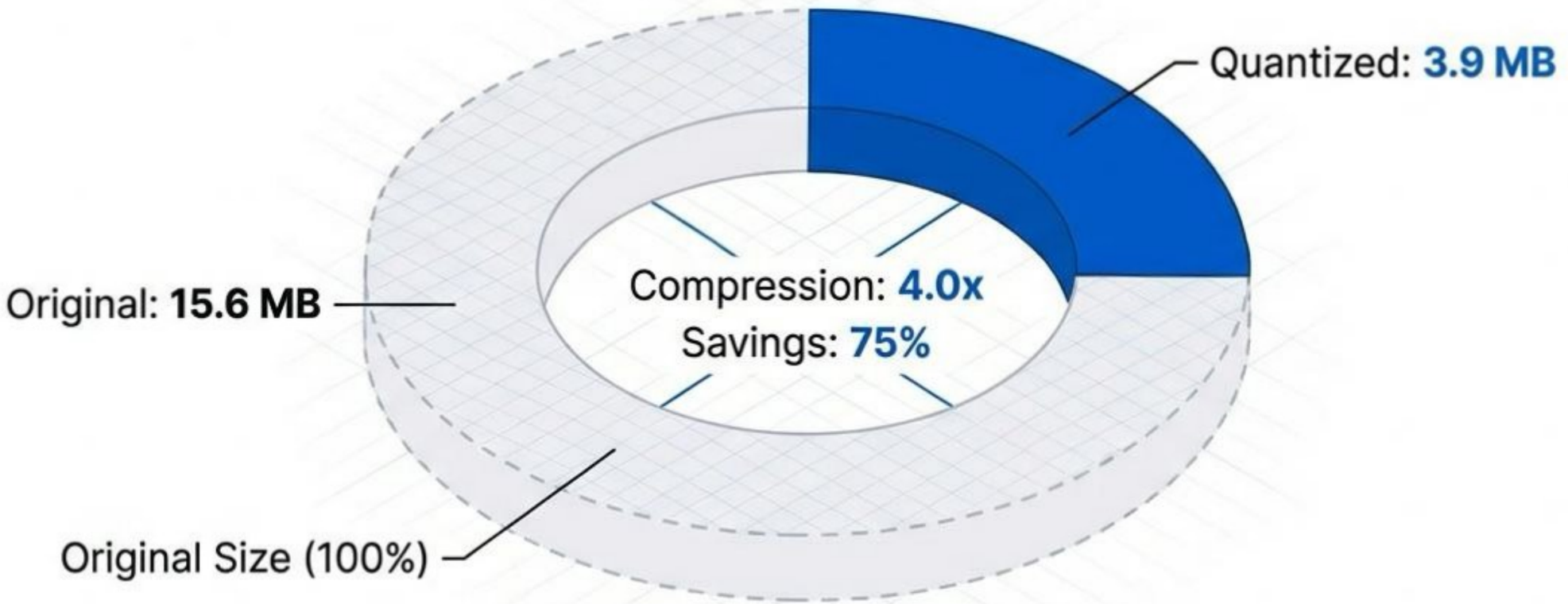
```
def quantize_model(model, calibration_data):
    for i, layer in enumerate(model.layers):
        if isinstance(layer, Linear):
            # 1. Swap Layer
            q_layer = QuantizedLinear(layer)

            # 2. Calibrate
            if calibration_data:
                q_layer.calibrate(...)

            # 3. Replace in-place
            model.layers[i] = q_layer
```



ReLU layers have no weights, so they remain untouched.

Verifying the Savings

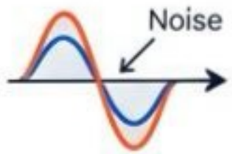
The result of `analyze_model_sizes()`.



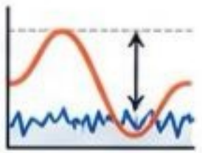
Overhead (Scales/Zero Points) is < 0.1% of total size.

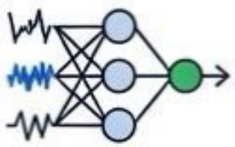
Why Accuracy Survives

Robustness and Signal-to-Noise Ratio.

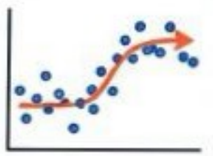


SNR: INT8 provides ~48dB of Signal-to-Noise ratio.

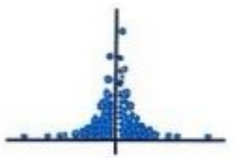




Robustness: Neural networks are trained with noise (Dropout, SGD) and are resilient to small perturbations.



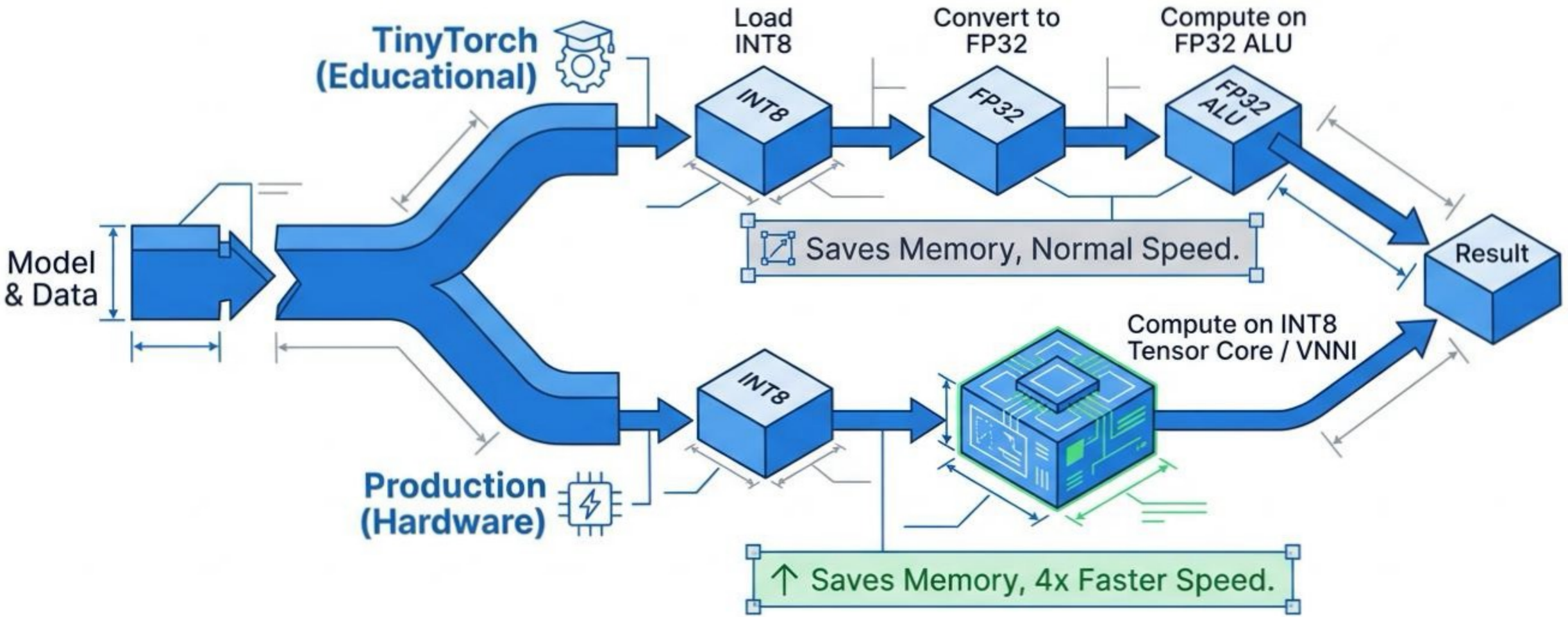
Distribution: Most weights are near zero, where affine mapping is effective.



Typical accuracy loss is < 1%. ✅

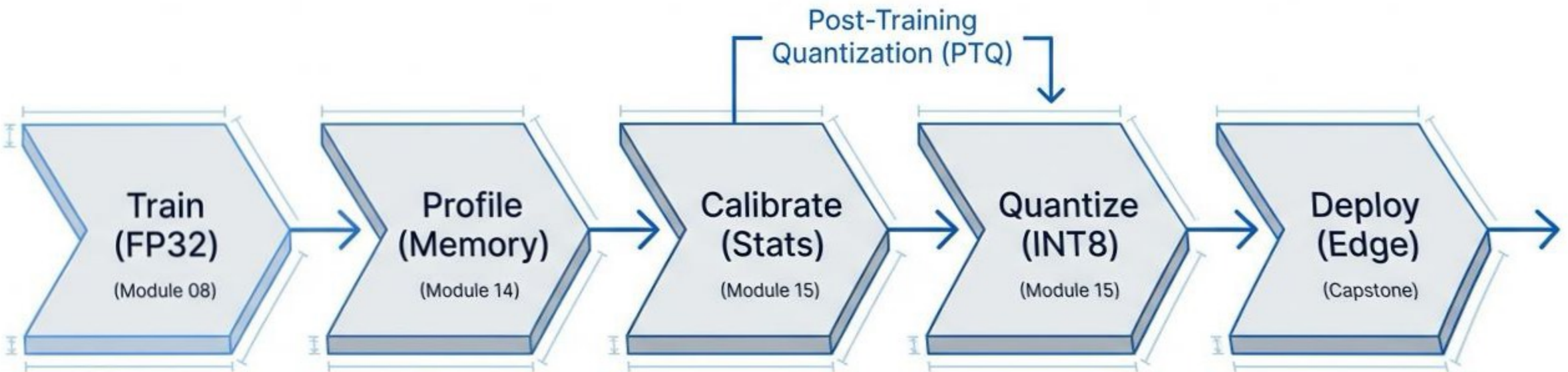
Production Reality: Hardware Acceleration

How silicon executes what we just built.



TinyTorch simulates the logic and accuracy. Production hardware delivers the speed.

The Quantization Pipeline
From Training to **Deployment.**



Concept → Code Map

Where to find these ideas in `tinytorch/perf/quantization.py`.

Concept	Implementation
Affine Mapping Formula	`quantize_int8`, `dequantize_int8`
Storage vs. Compute	`QuantizedLinear` class
Unknown Input Ranges	`QuantizedLinear.calibrate()`
Graph Transformation	`quantize_model()`
Verification	`analyze_model_sizes()`

What's Next?

Optimization Tier.

Module 15: Quantization

Complete (Memory Solved)

Module 16: Compression

Next Up (Pruning)

Reducing parameter count.

Module 17: Acceleration

Upcoming

Optimizing compute kernels.

You now possess the tools to take massive models and fit them into the real world.