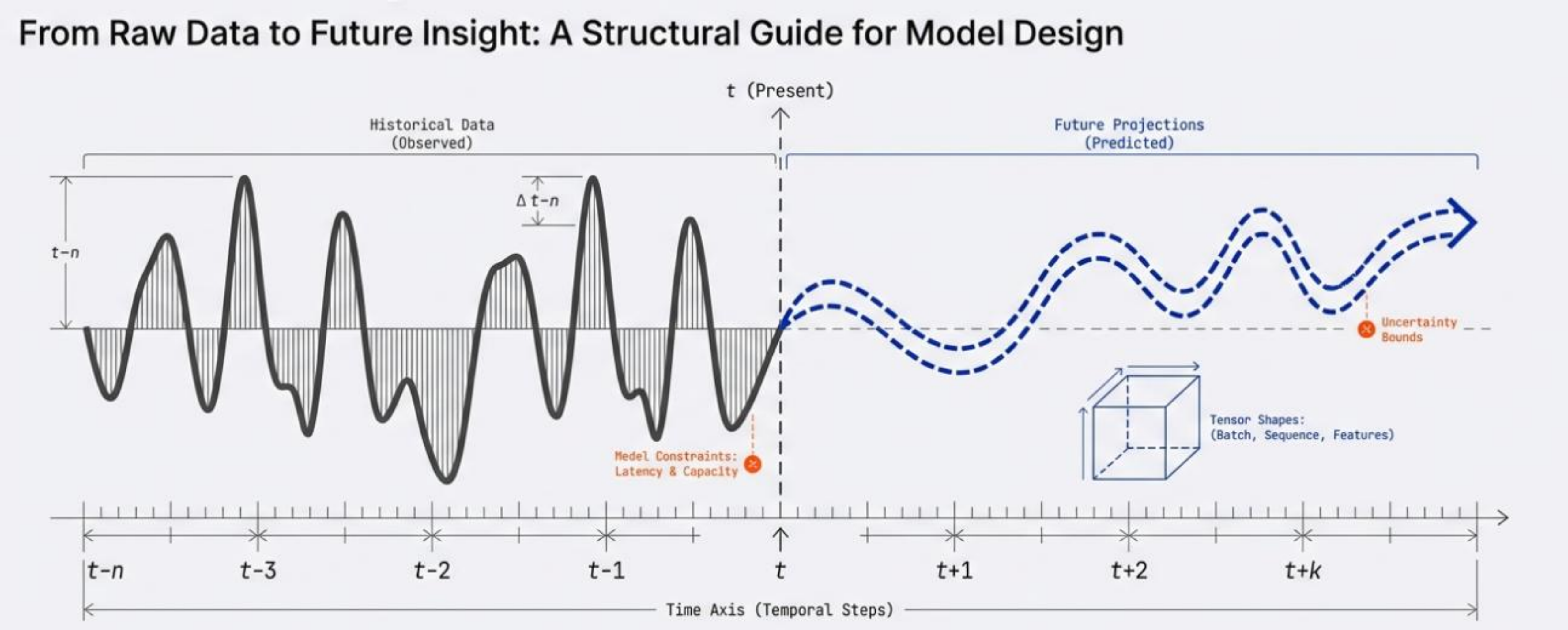
Deep Learning Frameworks

Time Series Forecasting using GRU in Tensorflow, Quantization in Pytorch

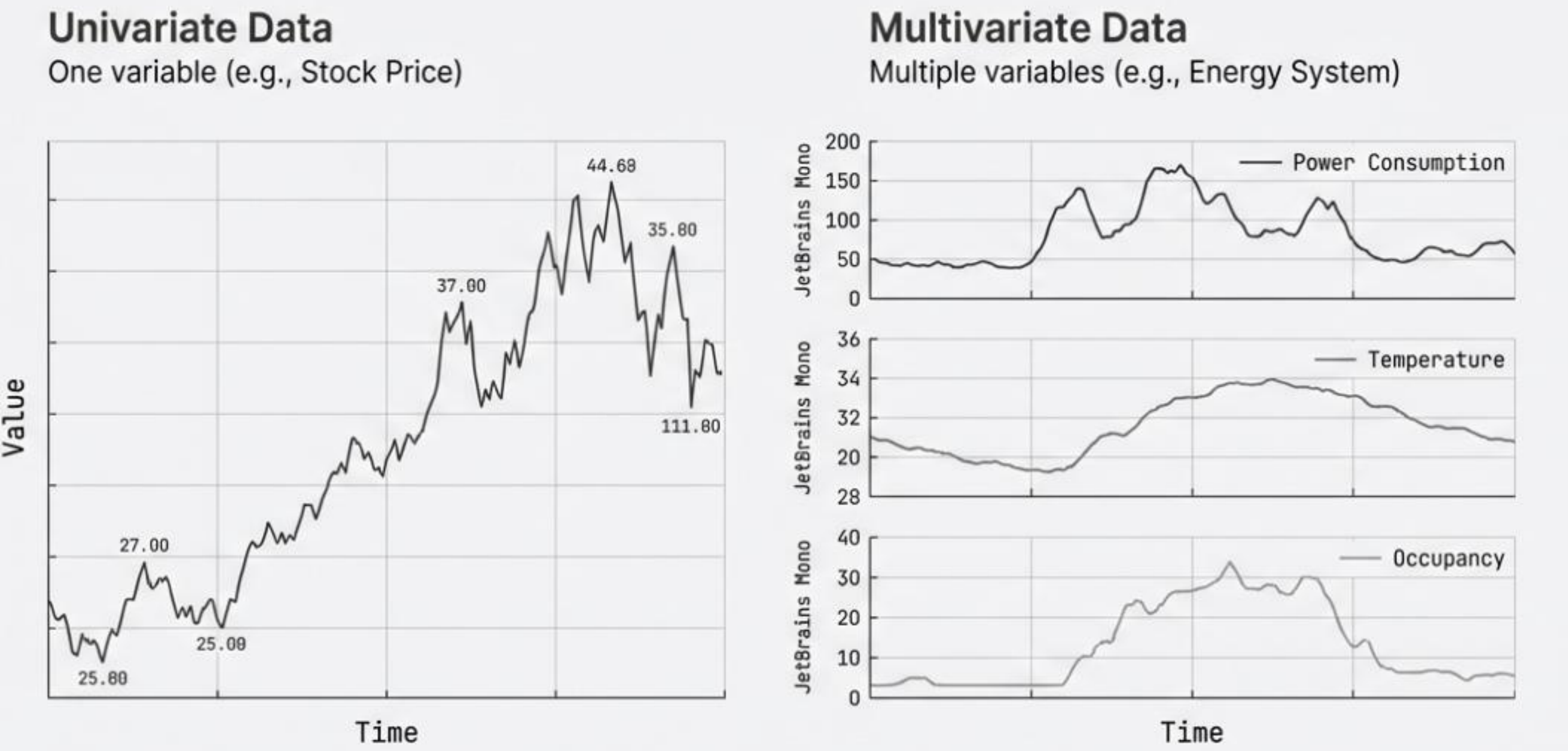
https://tinyurl.com/dlframeworks

https://github.com/sakharamg/DeepLearningFrameworks

Time Series Forecasting



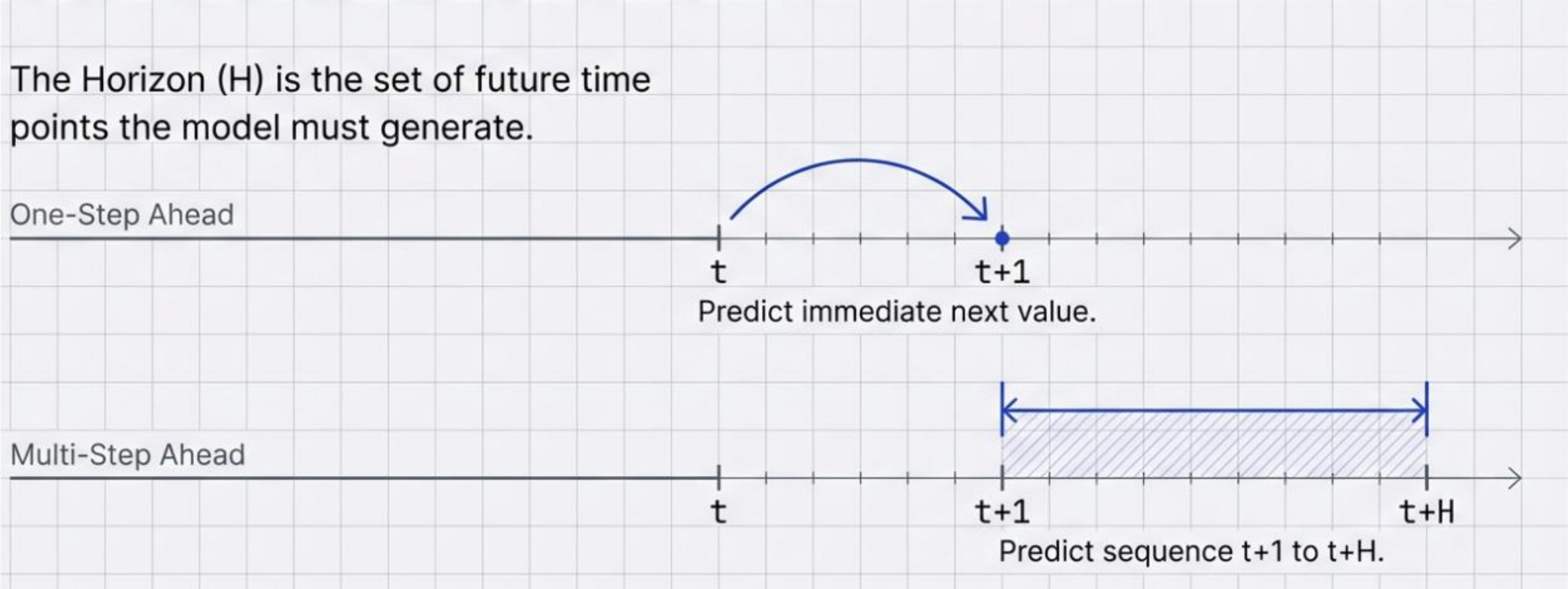
Univariate and Multi variate

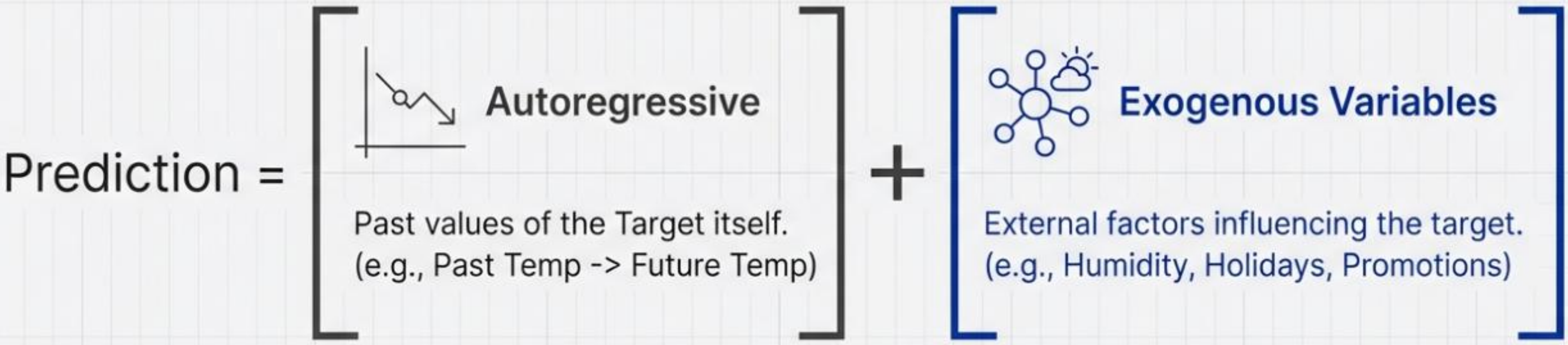


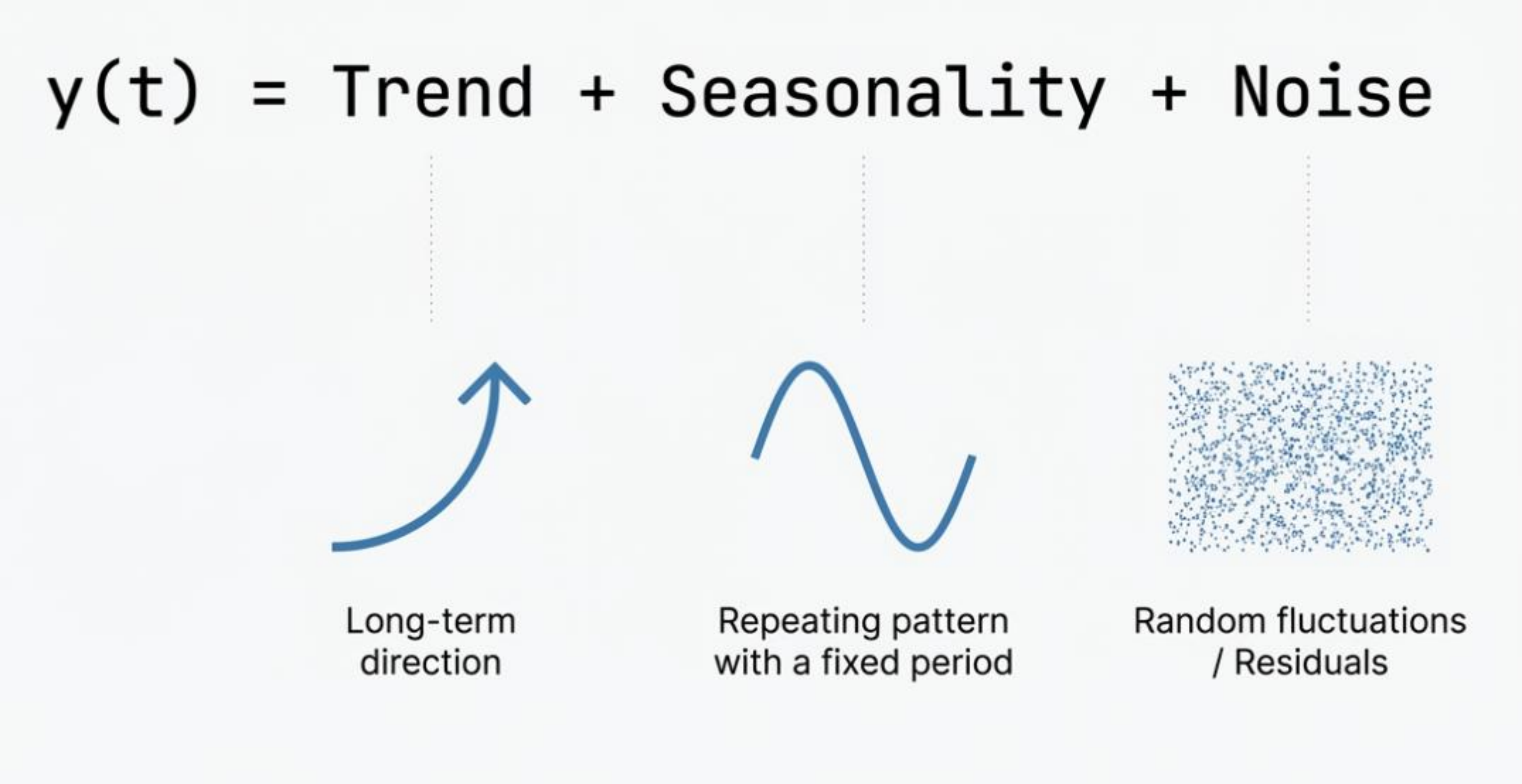
Design Decision: Frequency

Every series requires a fixed sampling frequency (Hourly vs. Daily).
This choice dictates visible patterns and input window size.

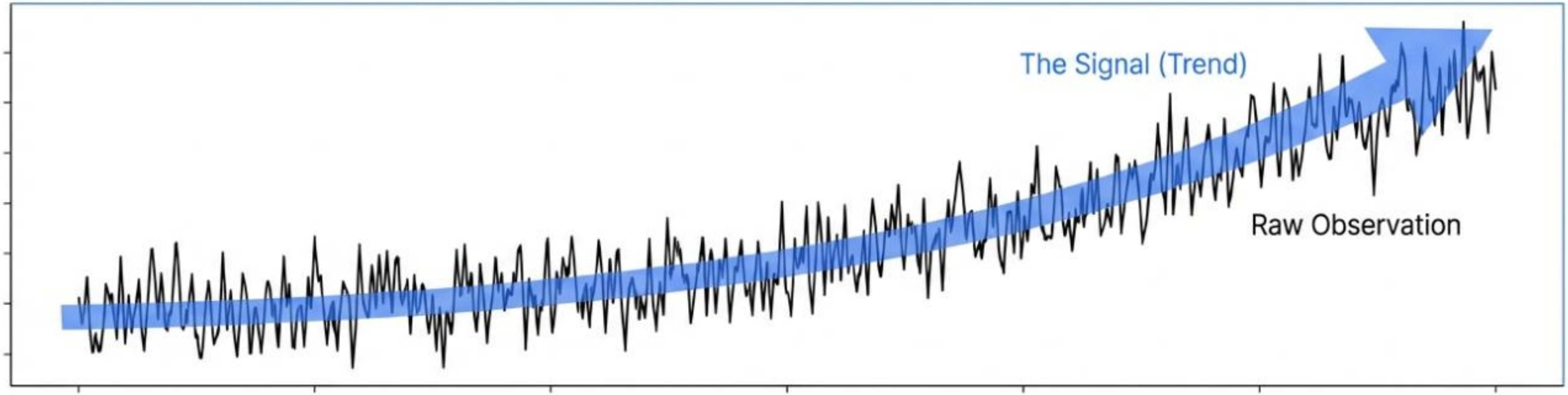
Forecast Horizon




Prediction =
Autoregressive
Past values of the Target itself.
(e.g., Past Temp -> Future Temp)
+
Exogenous Variables
External factors influencing the target.
(e.g., Humidity, Holidays, Promotions)


y(t) = Trend + Seasonality + Noise
Long-term direction
Repeating pattern with a fixed period
Random fluctuations / Residuals

Trend: The long-term Direction



Definition

The slow, long-term movement of the series.

Real-World Examples

- **Commercial**: Sales increasing over months as product gains popularity.
- **Physical**: Sensor readings drifting due to mechanical wear.
- **Environmental**: Gradual warming or cooling over seasons.

Forecasting Intuition

If a series has a strong trend, the model's primary job is learning

"direction". While classical models often require manual detrending, deep models can often learn the trend directly if the dataset is sufficiently large.

Seasonality

Concept

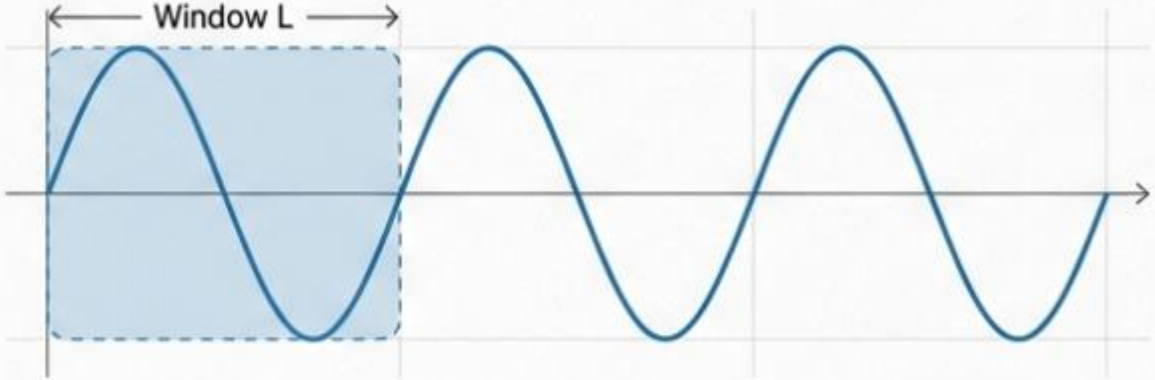
Definition: A repeating pattern with a fixed period.

• Hourly electricity usage	→	Daily cycle (24 hours)
• Website traffic	→	Weekly cycle (7 days)
• Retail sales	→	Yearly cycle (festival seasons)

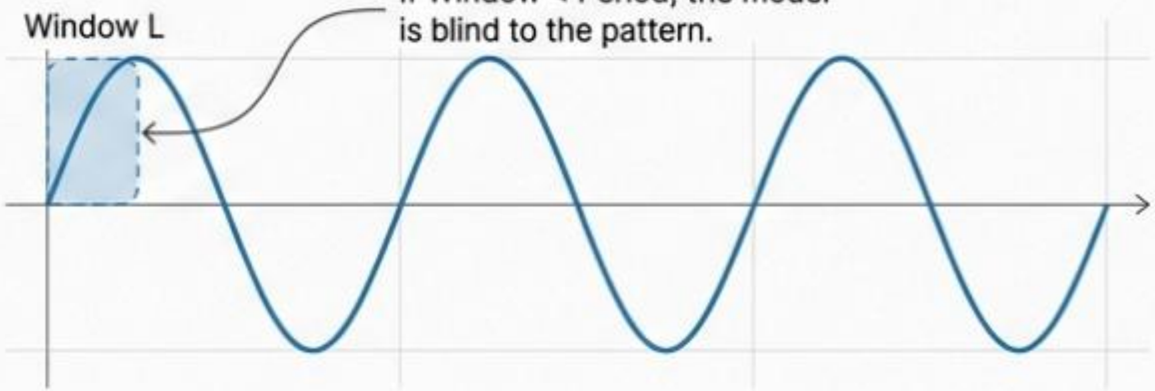
Forecasting Intuition: Seasonality means “the past repeats.” Your lookback window “L” must be at least one full season.

The Architectural Constraint

Window Rule

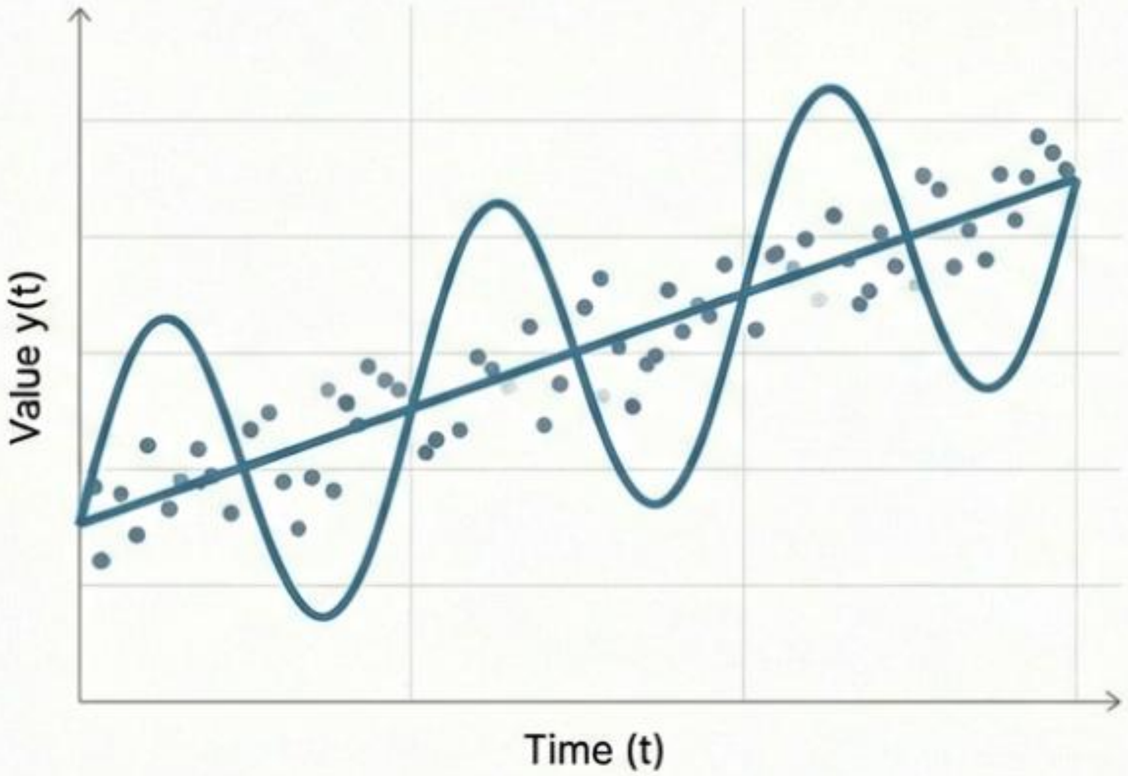


Failure Mode



Additive

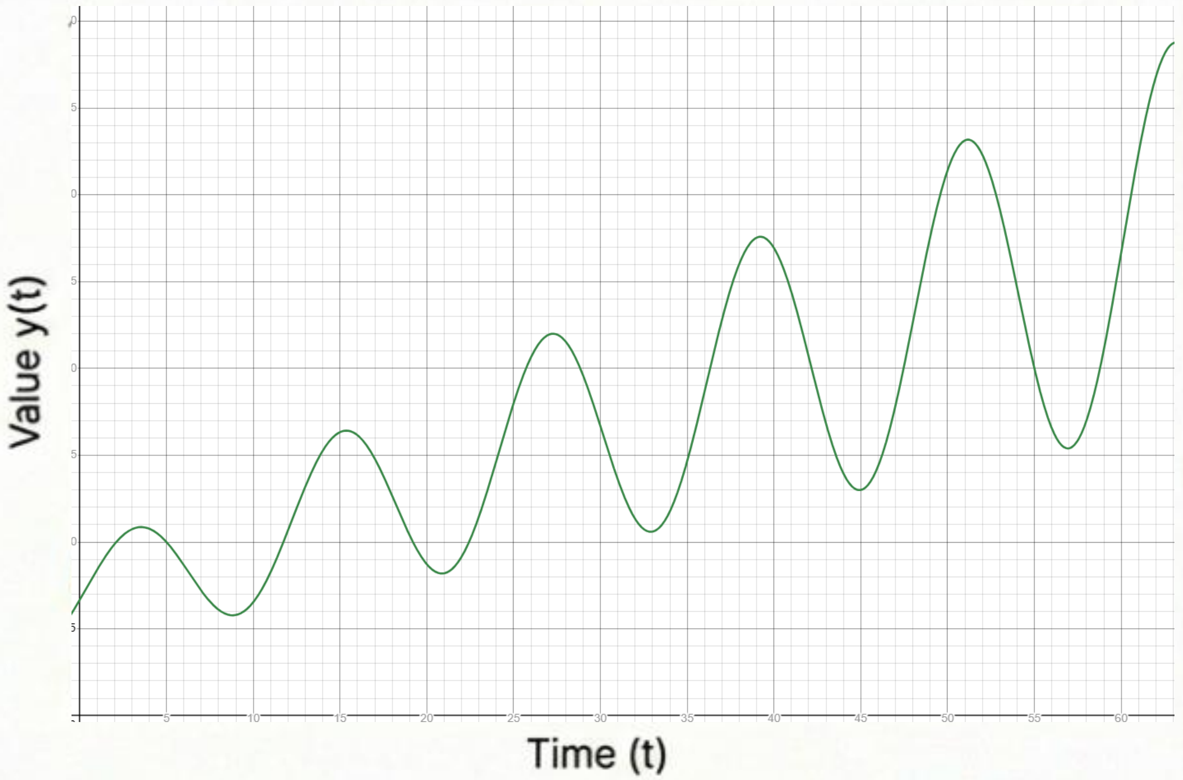
$y(t) = \text{Trend} + \text{Seasonality} + \text{Noise}$



E.g. Electricity usage, Temperature

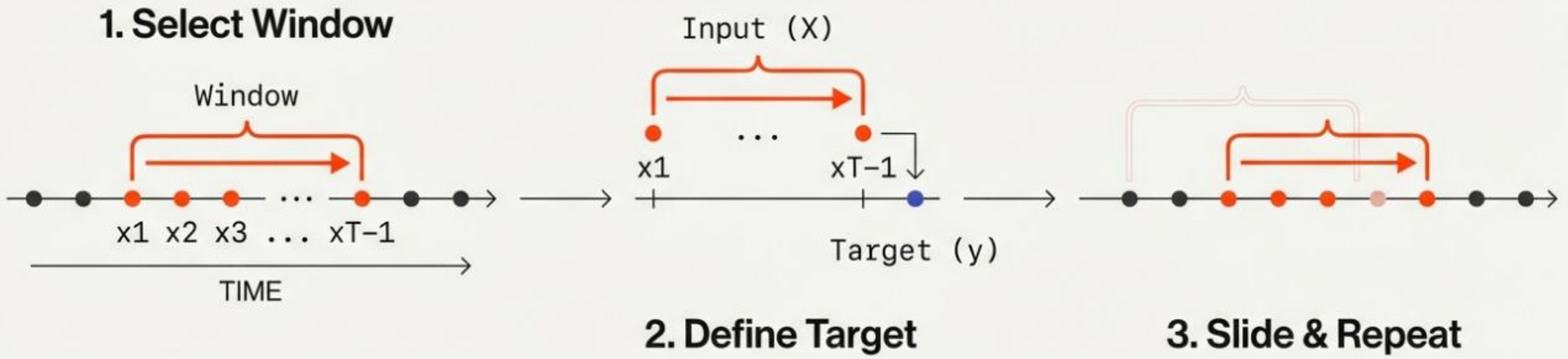
Multiplicative

$y(t) = \text{Trend} \times \text{Seasonality} \times \text{Noise}$



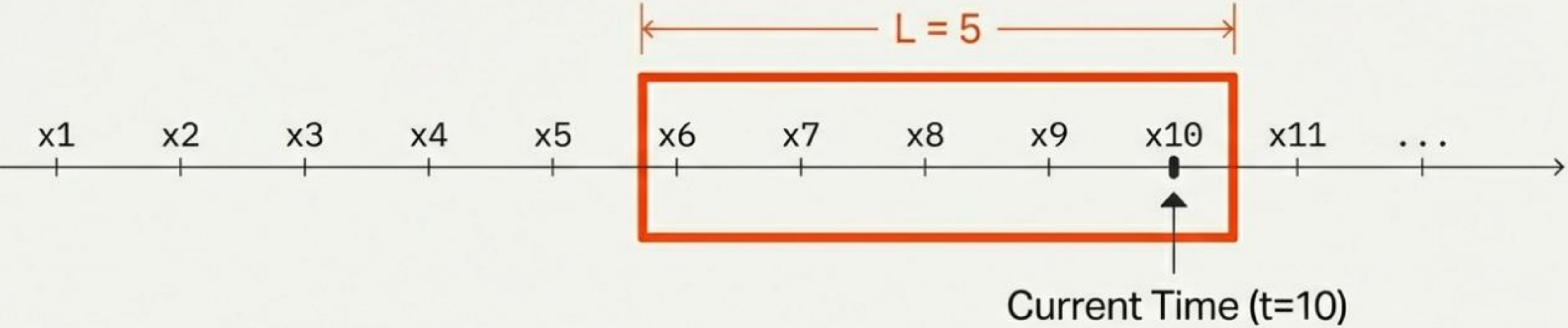
E.g. Retail sales, Website traffic

Windowing



Windowing is the process that converts the raw time series into a standard supervised learning dataset. By taking a specific chunk of the past, we create a context that asks the model to predict the next value.

Lookback



The Lookback determines how far into the past the model sees at any given time step.

Input Equation: Xt = [x_t-L+1, …, x_t]

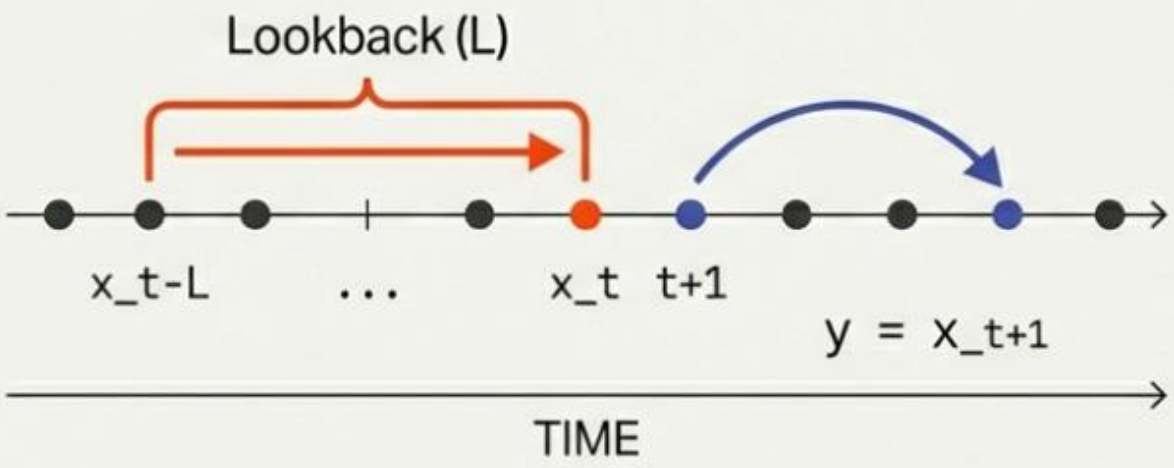
EXAMPLE SCENARIO:

Lookback L = 5
Time t = 10
Resulting Input Vector: [x6, x7, x8, x9, x10]

Horizon

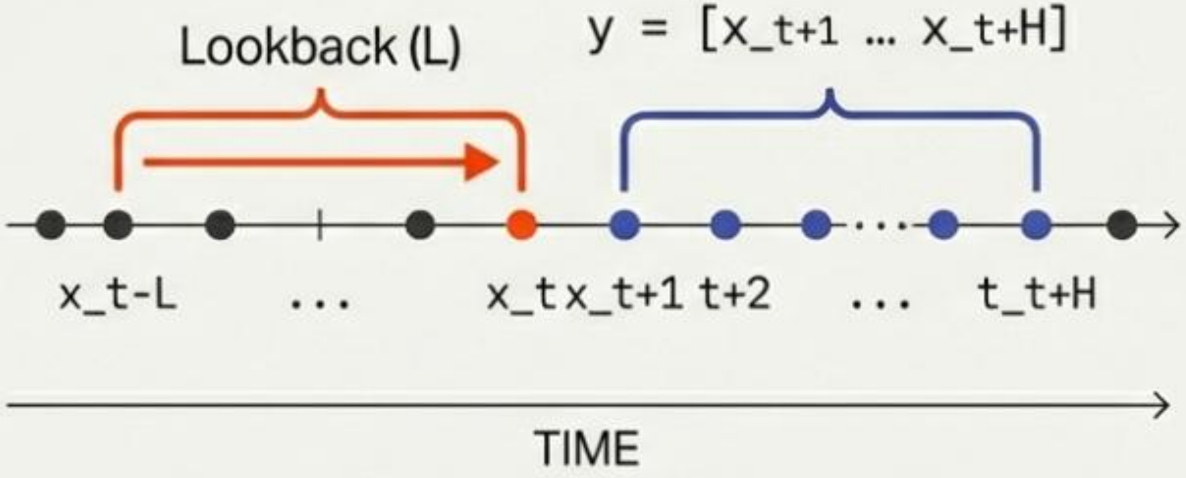
The target (y) is what the model attempts to predict based on the Lookback.

1. One-step Target



Use for immediate forecasts
(e.g., next hour).

2. Multi-step Target



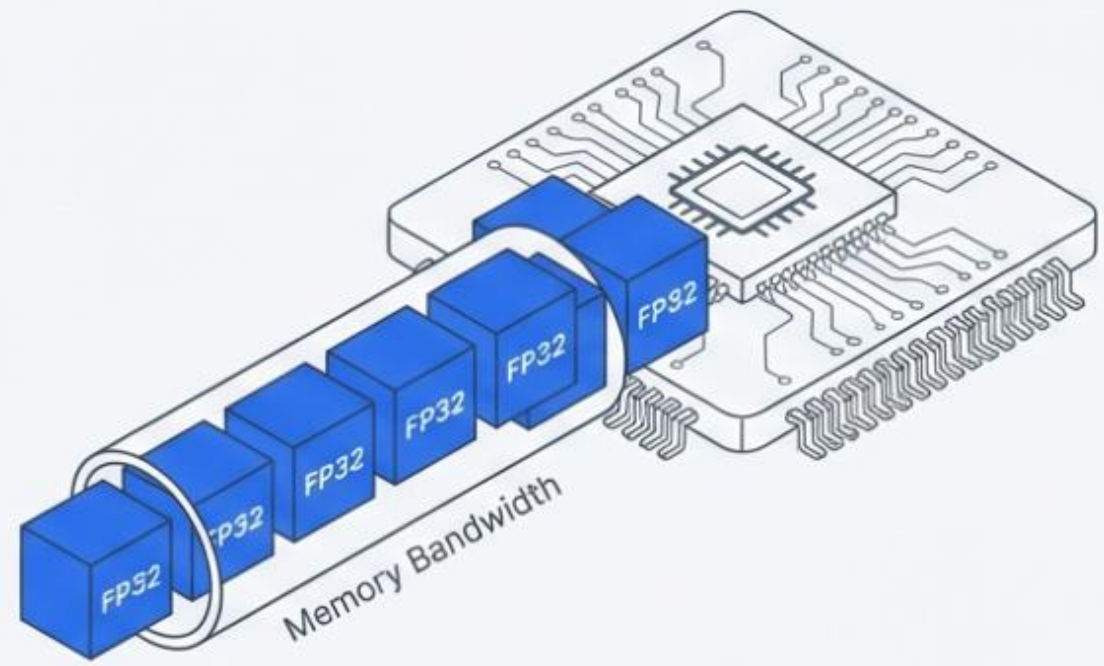
Use for forecast ranges
(e.g., next 24 hours).

Lab

https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

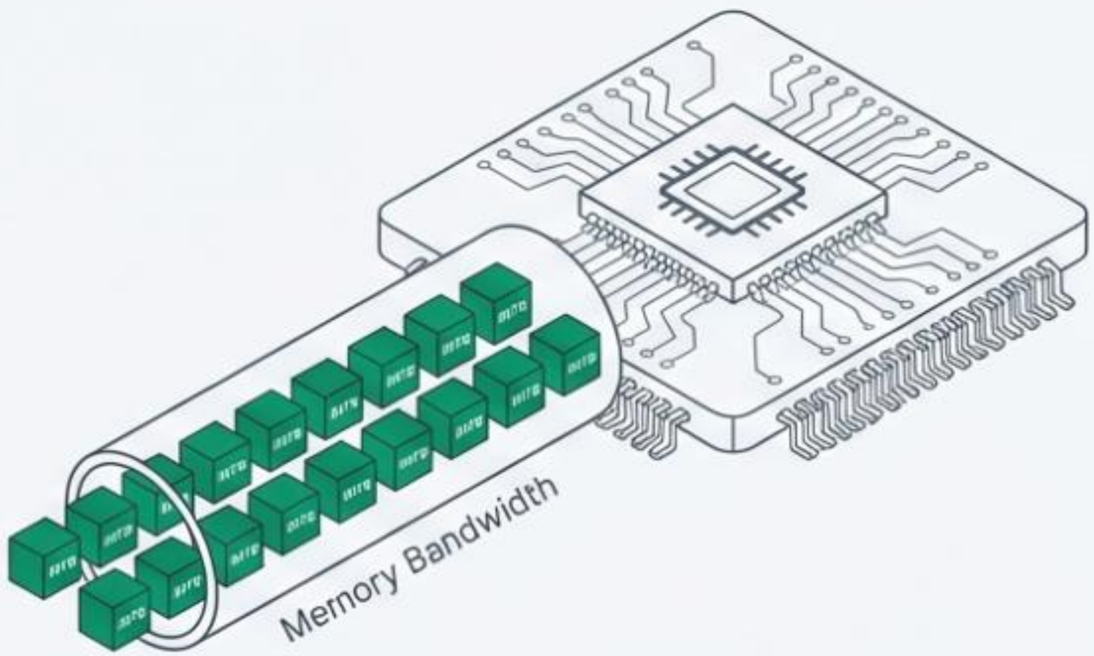
Challenges in Production

The Problem: Heavy Load



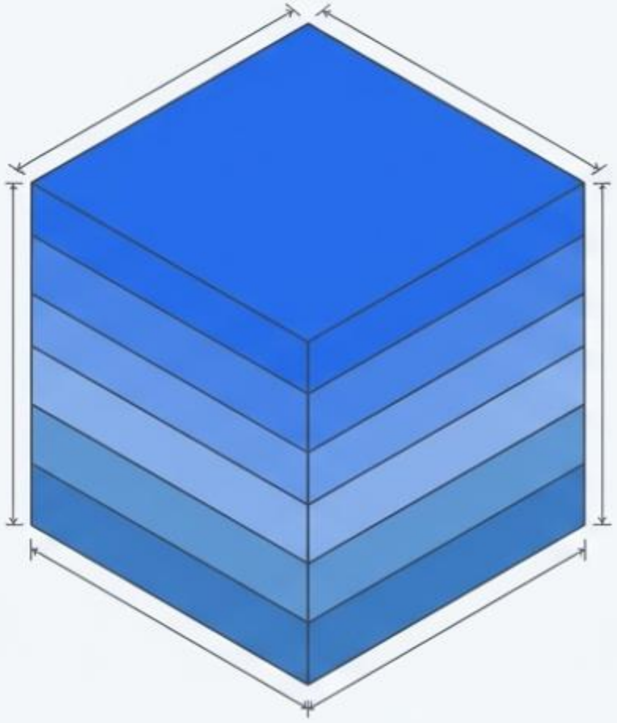
- **Compute Bound:** Massive multiply-add operations in Convolutions.
- **Memory Bound:** Moving 32-bit weights consumes high energy and time.

The Solution: Quantization



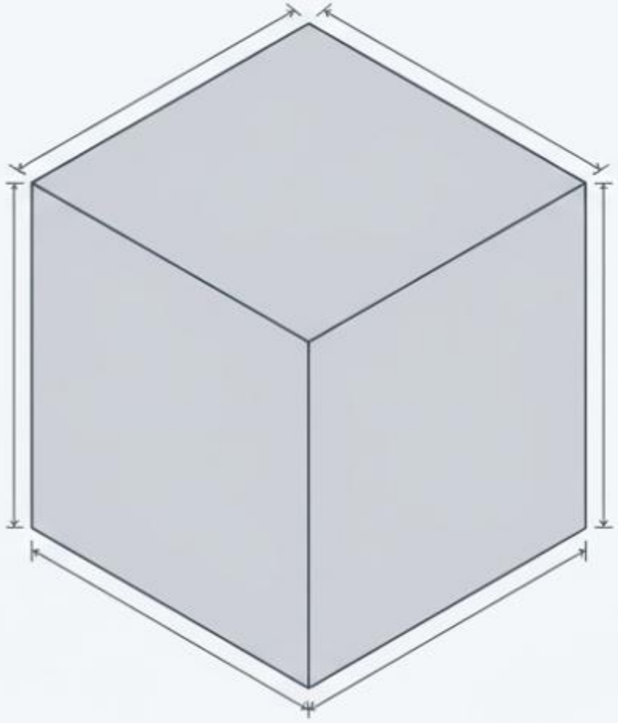
- **Reduced Footprint:** 4x reduction in memory traffic (32-bit → 8-bit).
- **Accelerated Kernels:** Integer arithmetic executes significantly faster on edge CPUs.

Weights (Per-Channel)



- **Applied to:** Model Parameters
- **Strategy:** Independent scale/zero-point per output channel.
- **Benefit:** High accuracy, accommodates varying ranges.

Activations (Per-Tensor)



- **Applied to:** Inputs & Outputs
- **Strategy:** Single scale/zero-point for the entire layer.
- **Benefit:** Performance efficiency for dynamic data.

Quantize and De quantize

01. QUANTIZE (Float → Int)

Discretizer

$$q = \mathrm{clamp}(\mathrm{round}(x / s) + z)$$

Range Limiter

02. DEQUANTIZE (Int → Float Approx)

$$\hat{x} \approx s * (q - z)$$

Function Key

round(.) : Nearest integer conversion

clamp(.) : Restricts to [q_min, q_max]

q_min/max : Typically -128 to 127

Range Inputs

x_min, x_max
q_min, q_max

Scale Calculation (s)

$$s = \frac{x\_max - x\_min}{q\_max - q\_min}$$

Spreads float range over integer codes.

Zero-Point Calculation (z)

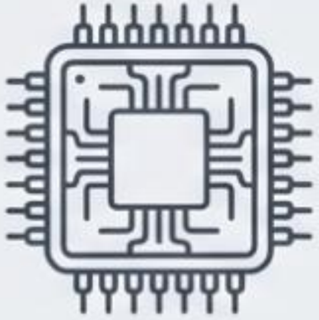
$$z = \text{round}\left(q\_min - \frac{x\_min}{s}\right)$$

Aligns integer code z with float 0.

Output

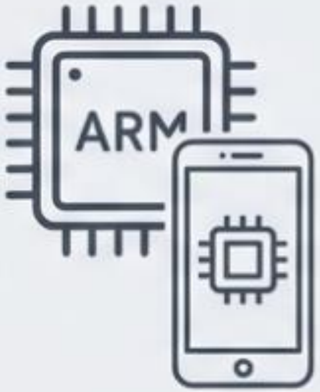
Parameters s and z ready for quantization.

PyTorch 'torch.ao.quantization' relies on specific kernel backends.



Server / Desktop

- Architecture: x86
- Backend Engine: `fbgemm` (Facebook GEMM)
- Optimized for: High throughput server-side inference.



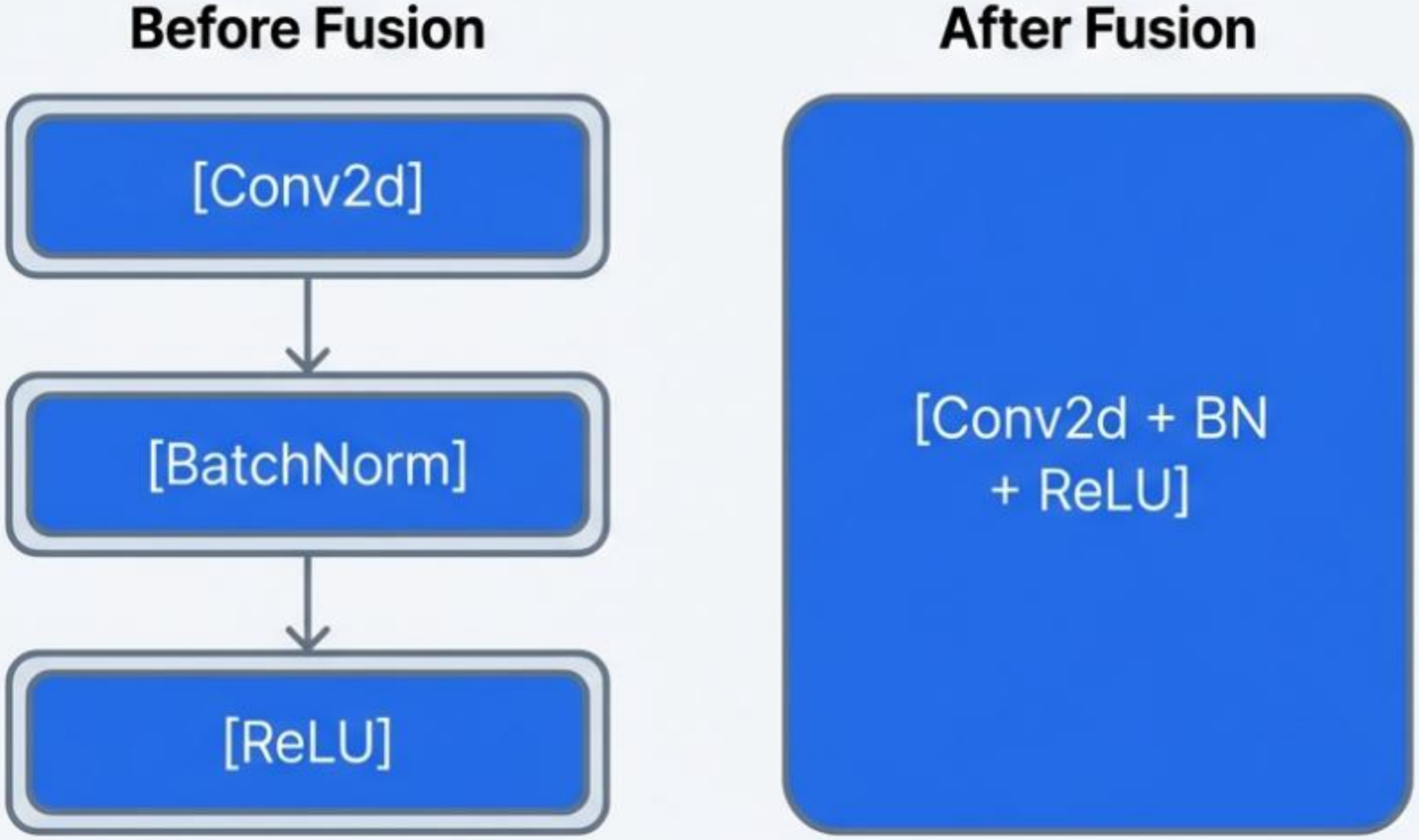
Mobile / Edge

- Architecture: ARM
- Backend Engine: `qnnpack` (Quantized Neural Network PACkage)
- Optimized for: Low power, mobile processors.

Note: The theoretical concepts in this guide apply to both, but specific kernel support may vary.

Fusing Operations

Merging sequential operations for kernel efficiency.



Why Fuse?

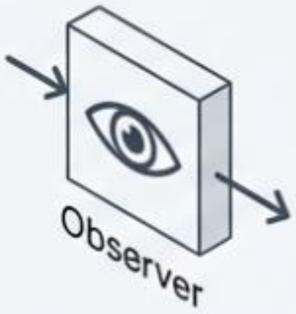
- **Enables Quantization:** Backends like fbgemm require fused operators to access INT8 implementations.
- **Performance:** Reduces memory access overhead. Calculations happen in registers without writing intermediate results to VRAM.

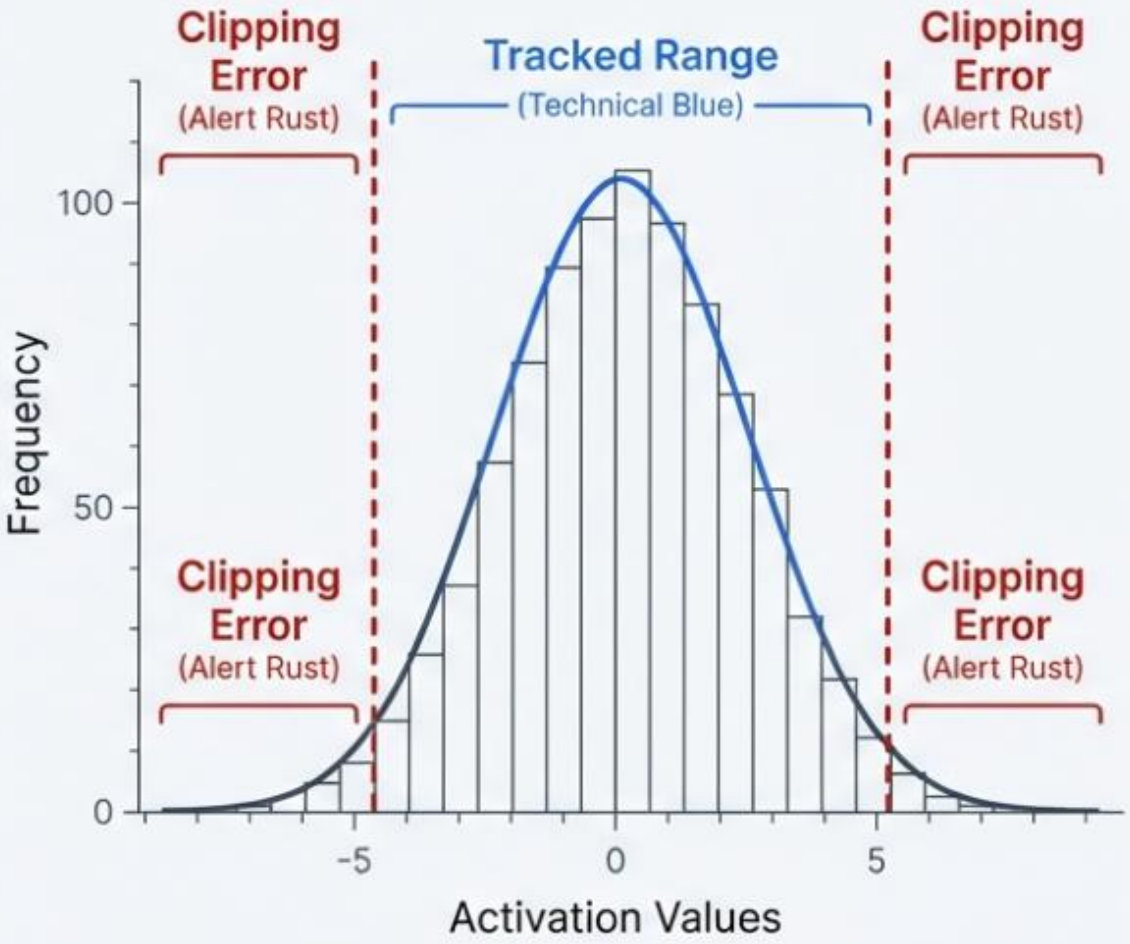
Quantizing Activations

The Observer

A module inserted into the network graph.

It passively records statistics (Min/Max) of the tensors passing through it.



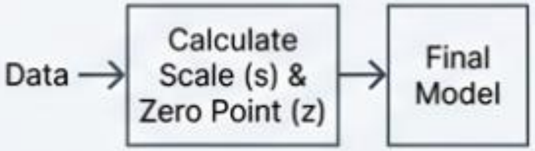


The Calibration

We run a forward pass with representative data.

No backpropagation occurs.

The system uses these statistics to calculate the static Scale (s) and Zero Point (z) for the final model.



A technique to quantize a pre-trained FP32 model without further backpropagation training steps.

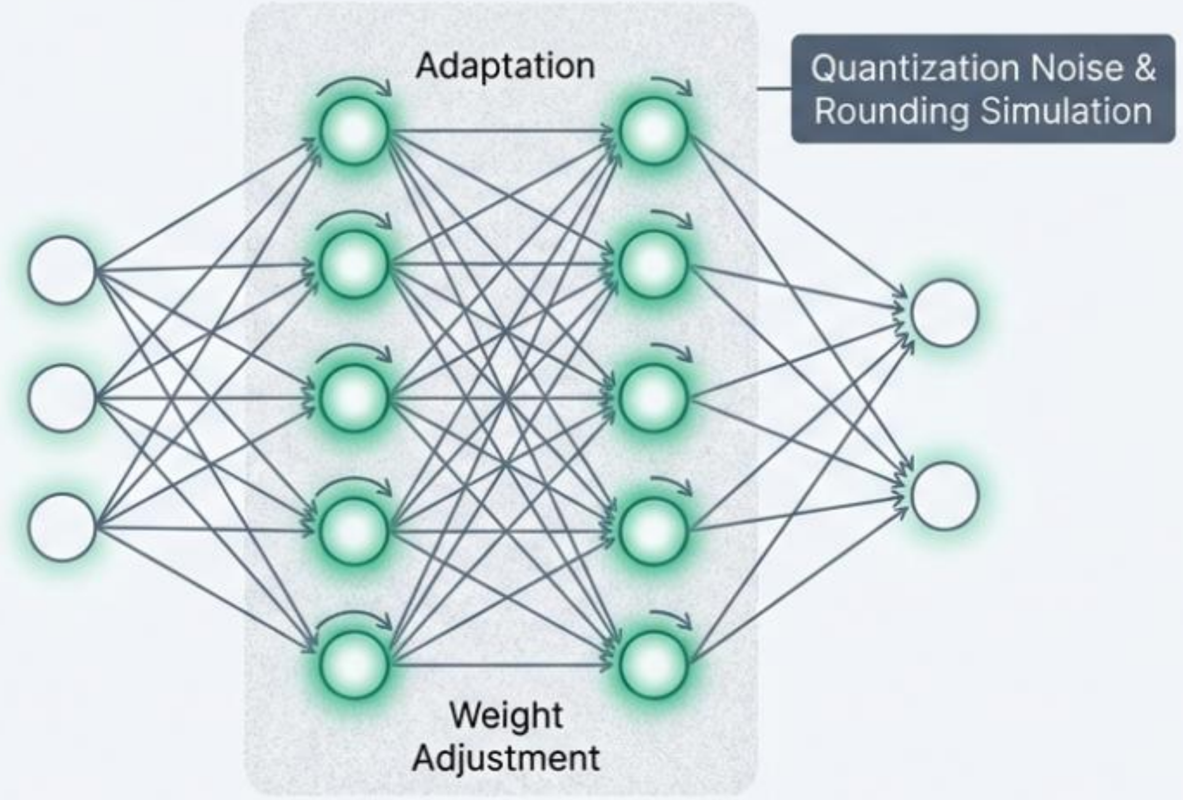
Weights

Quantized once offline. Permanently stored as INT8 to reduce model size.

Activations

Quantized at runtime. Ranges (Min/Max) are estimated beforehand via the calibration step.

Quantization Aware Training

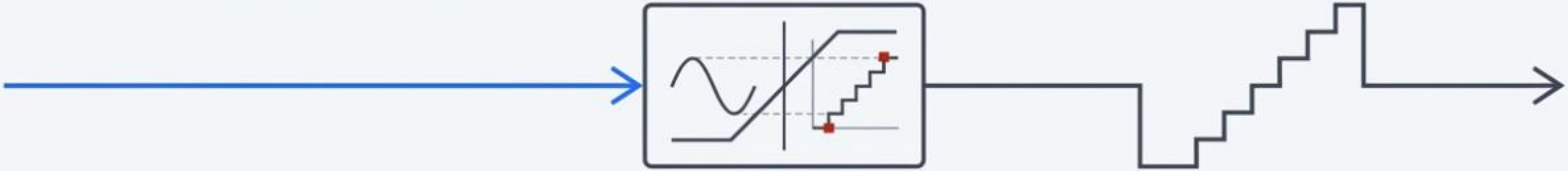


Core Concept: Adaptation.

- Instead of quantizing AFTER training (blindly), we simulate quantization DURING training.
- The network learns to adjust its weights to survive the noise and rounding errors of INT8 representation.

Forward Pass (Simulation)

FakeQuant



Simulates rounding/clamping. The loss function “sees” the quantization error.

Backward Pass (Learning)

Gradients update the high-precision FP32 weights. The master weights cluster into robust values.

Lab

https://tinyurl.com/dlframeworks
https://github.com/sakharamg/DeepLearningFrameworks

Thank You

Appendix

The Mechanics of Asymmetry

Why do we need a zero-point offset?