WEED

A COMPPREHENSIVE ANALYSIS OF MINIMALIST, HIGH-PERFORMANCE C++ INFRASTRUCTURE

Inference and Backpropagation via Functional Equivalence

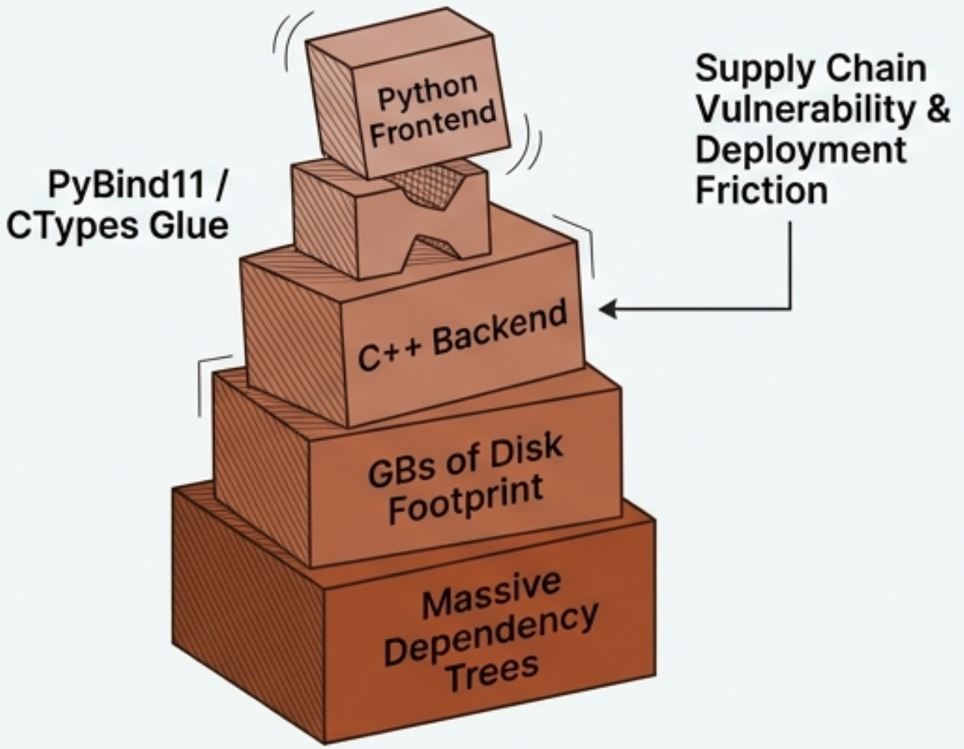
What if sparsity was a storage detail, not a user burden?

Based on the architecture of the Qrack Quantum Simulator. // vm6502q/weed

The Paradox of Modern AI Infrastructure

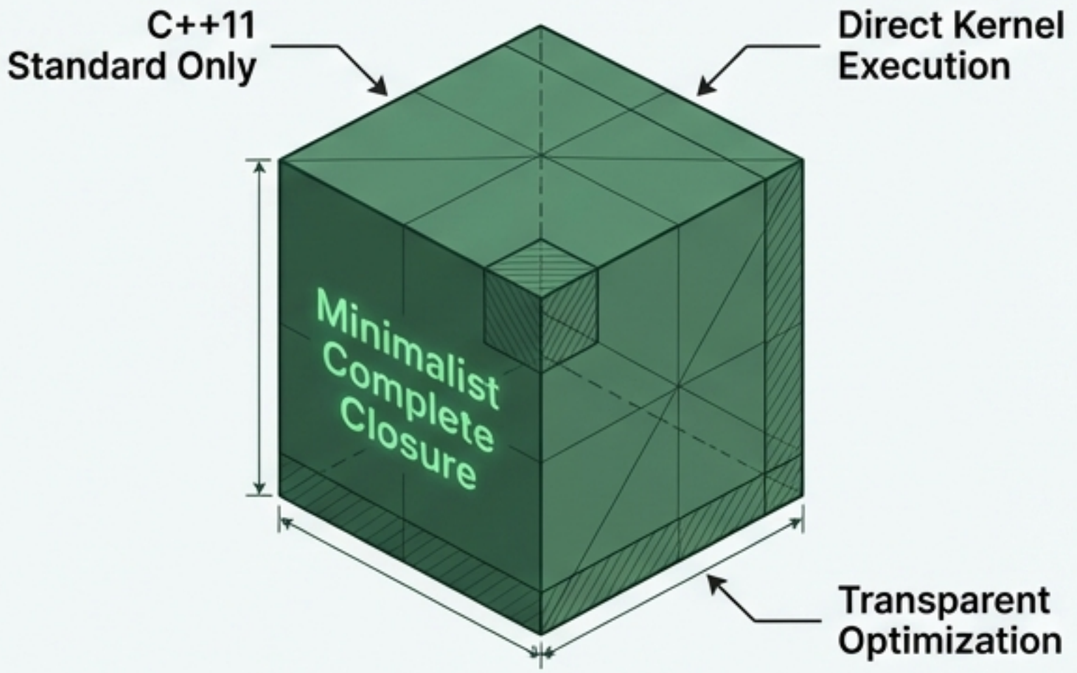
Capability vs. Complexity

THE LEGACY STACK



Legacy frameworks carry a decade of “code debt,” grafting high-performance C++ onto Python.

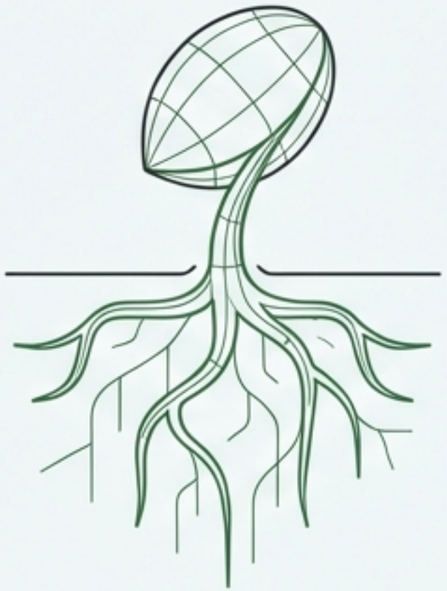
THE WEED IDEAL



A “Fresh Start” hypothesis: Rebuilding from the ground up to avoid dependency hell.

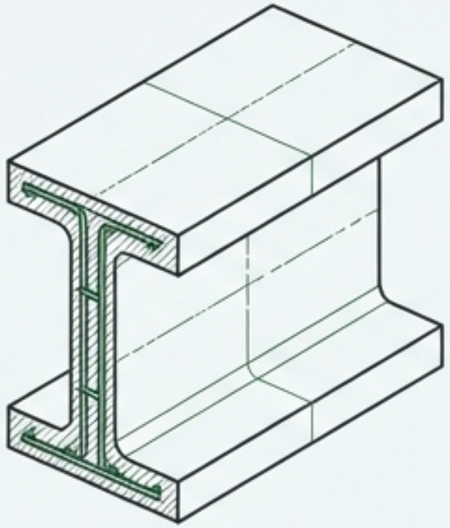
The Philosophy of Weed

Minimalist Complete Closure



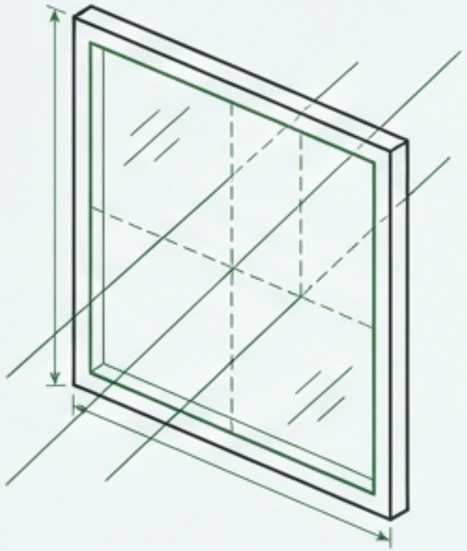
1. Fresh Start

Zero external dependencies. Requires only the C++11 language standard. No pip packages, no massive binaries. A self-contained ecosystem.



2. Production Grade

Distinct from educational toys like Micrograd. Supports n-dimensional arrays, full GPU acceleration, and efficient memory management. Built for deployment, not just demonstration.



3. Transparent Performance

HPC optimizations are defaults, not configs. “Make the correct thing the default—and the expensive thing explicit.”
Sparsity is handled automatically.

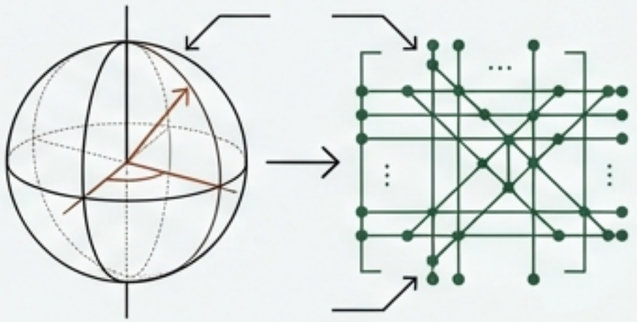
Weed does not seek to supplant established frameworks, but to offer a dependency-free alternative for sovereign AI.

The Quantum Lineage

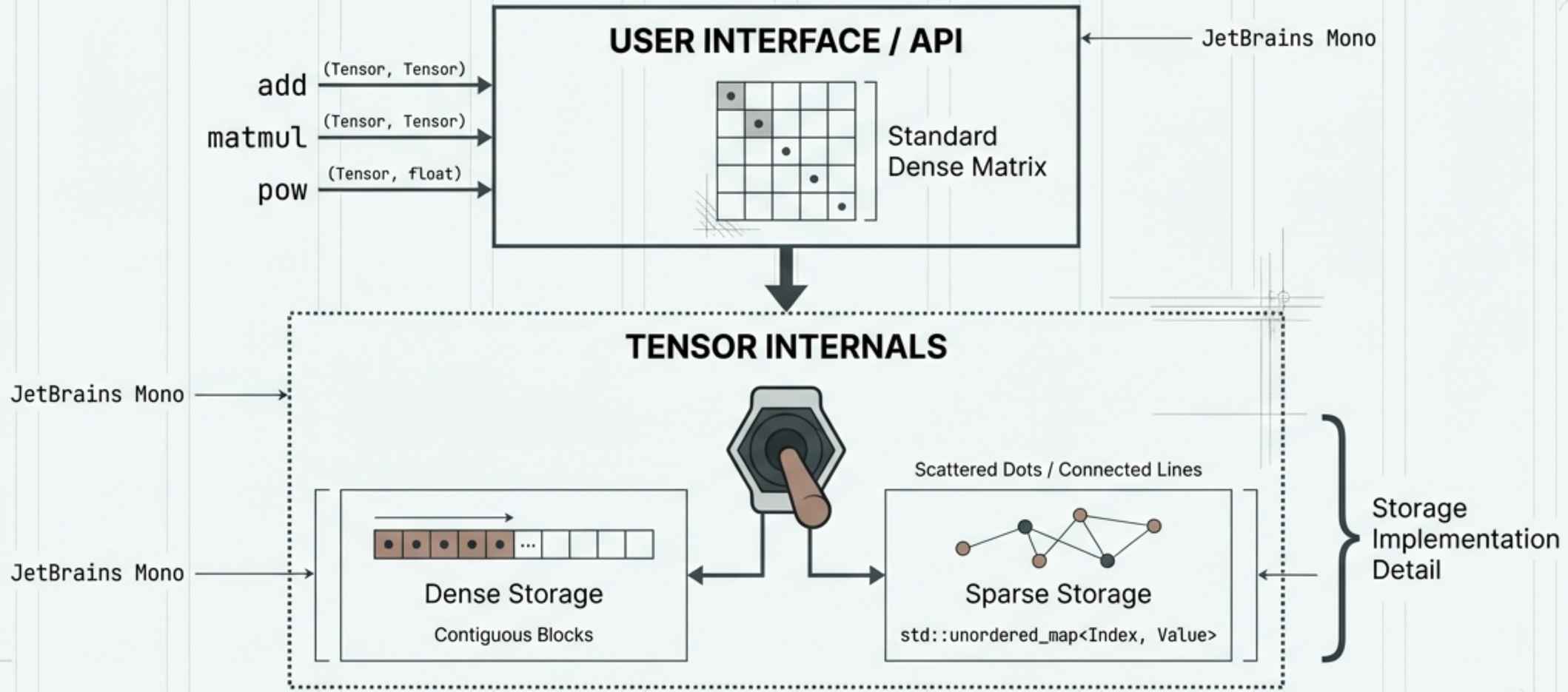
From Qrack Simulation to Classical ML

CONCEPT	QRACK (Quantum)	WEED (Classical ML)
Core Object	Quantum State Vector	Tensor
Data Type	Complex Numbers	Real or Complex (First-class)
Optimization	Stabilizer States	Sparse Tensors (Hash Map)
Philosophy	Transparent Simulation	Transparent Storage

Weed is authored by the creators of Qrack (vm6502q). Just as Qrack optimizes quantum state representation without user intervention, Weed applies ‘Functional Equivalence’ to classical tensors.



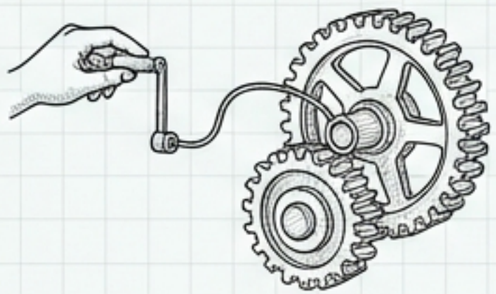
The Thesis of Functional Equivalence

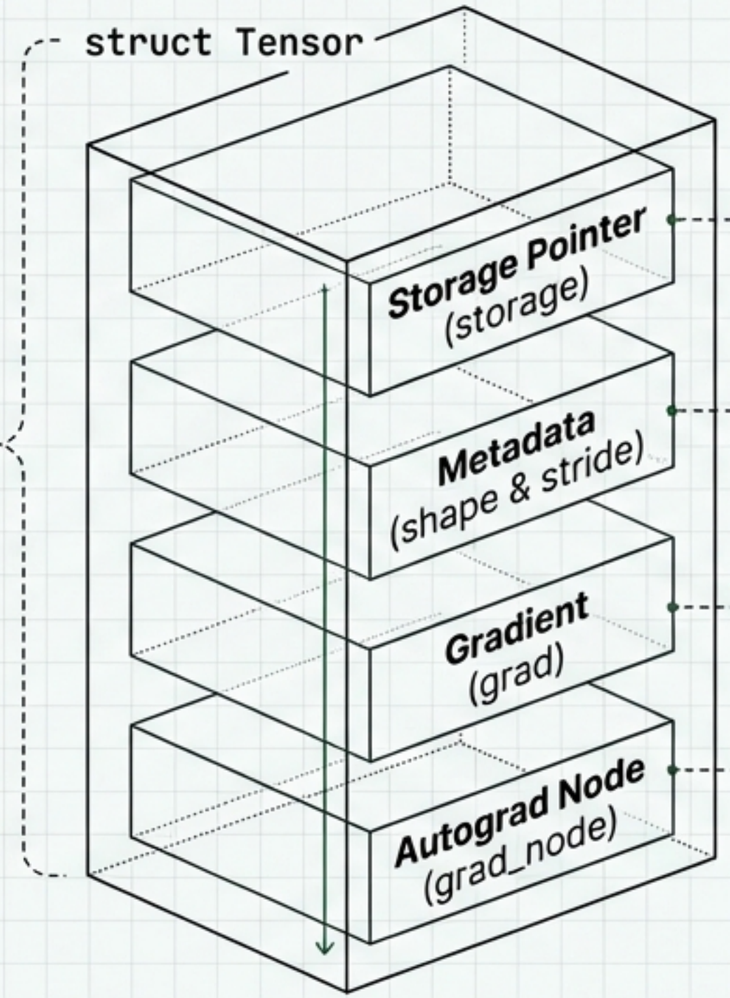


Sparseness is a **STORAGE** concern, not an **INTERFACE** concern.
User code remains identical regardless of data density.

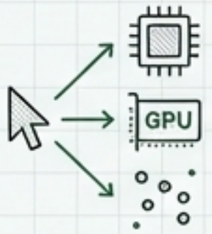
Anatomy of a Primitive: The Tensor Structure

Pimpl Idiom (Pointer to Implementation): Lightweight handles managing heavy resources.

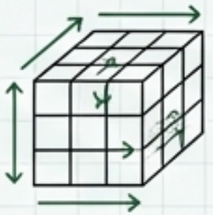




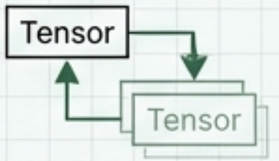
→ Smart pointer (`StoragePtr`). The polymorphic bridge to data (CPU / GPU / Sparse).



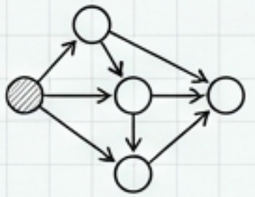
→ Vectors of `tcapint`. Defines dimensions and memory layout. Allows for efficient "views" without copying data.



→ Shared pointer to another `Tensor`. Recursive definition for higher-order derivatives.



→ Pointer to computation graph history. Connects the tensor to the operation that created it.



The Storage Abstraction Layer

Root Node

Storage
(Base Class)

DeviceTag

DType

CpuRealStorage

Dense / Contiguous

Optimized for SIMD vectorization.

SparseCpuRealStorage

Sparse Unordered Map

`std::unordered_map<tcapint, real1>`

O(1) Random Access. Mutable & Dynamic.

GpuRealStorage

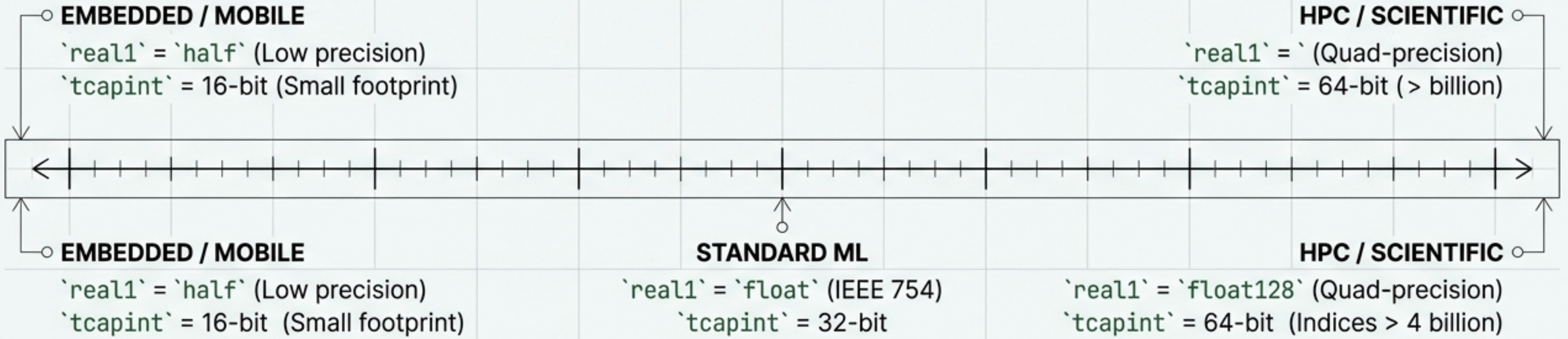
OpenCL / CUDA Buffers

Device pointers.
No direct element access.

Decoupling API from Data allows aggressive memory optimization while keeping the user API unified.

A Configurable Type System

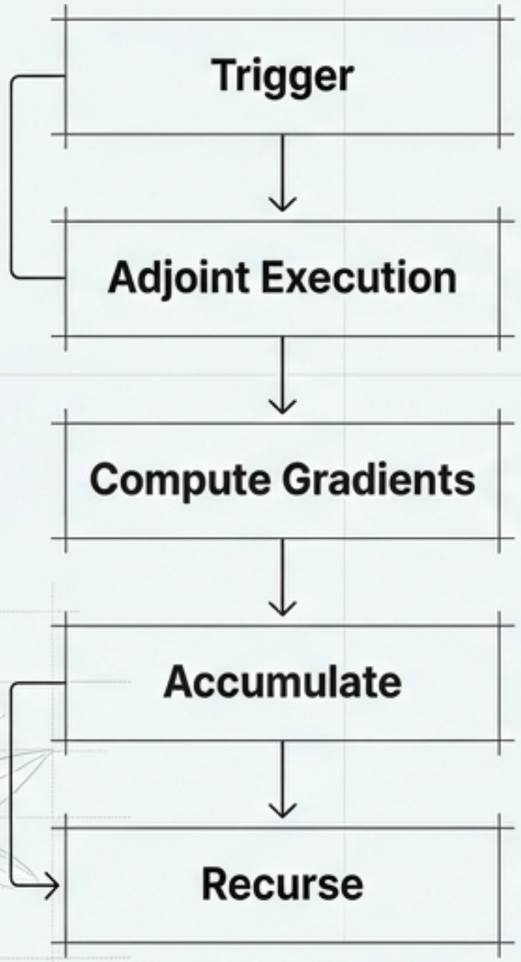
`weed_types.hpp`



Crucial Note:
Complex numbers (`complex<real1>`) are first-class citizens, inheriting the precision configuration of the underlying real type. Essential for Quantum ML.

The Autograd Engine

Reverse-Mode Differentiation (Define-by-Run)



`loss.backward()` called.

Closure (Lambda) execution. Each Node executes stored backward logic.

Calculate partial derivatives relative to parents.

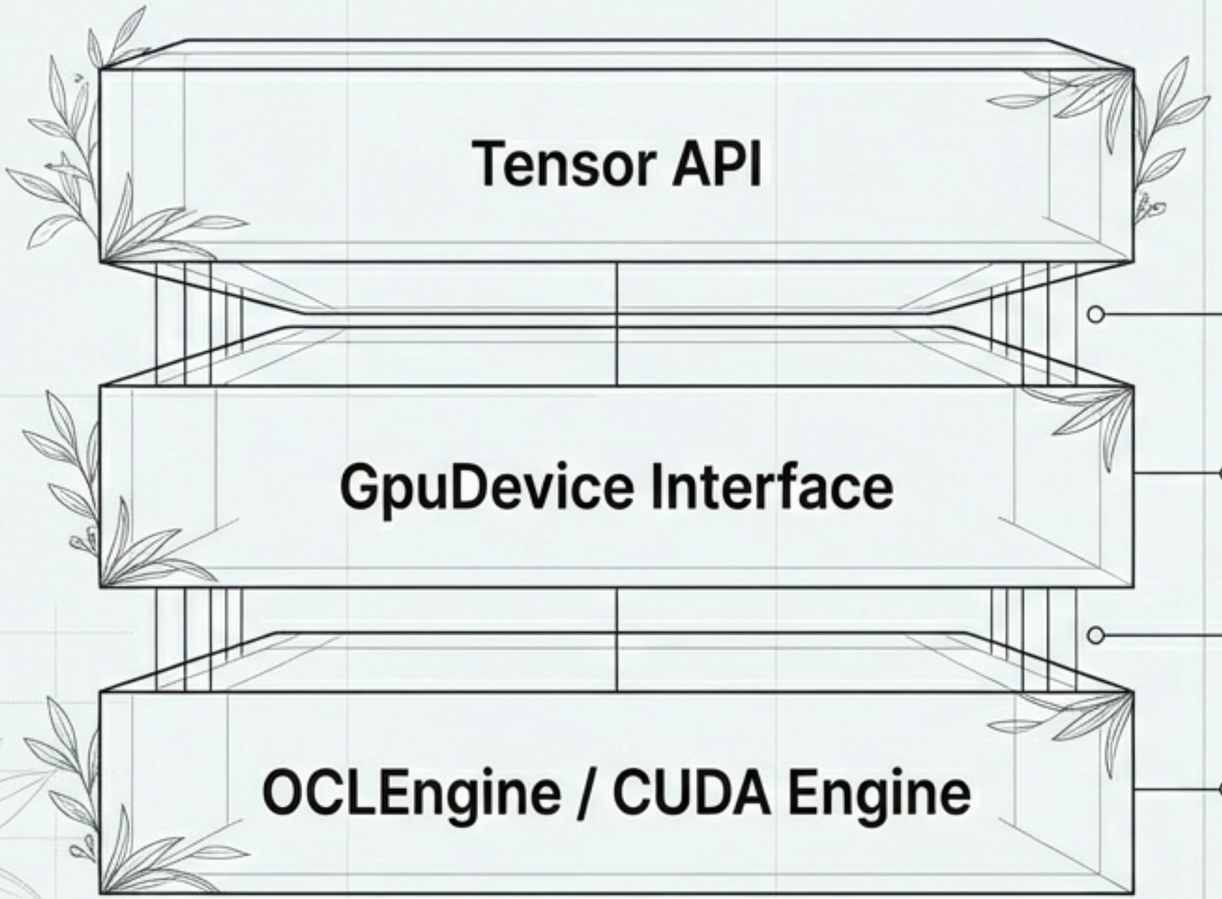
`parent.grad += calculated_grad`.
Uses `shared_ptr` for branching paths.

Repeat up the graph to Parameters.

The Closure

```
// The Closure
node->backward = [input, weight](Tensor& grad) {
    input.grad += grad >> weight.t();
    weight.grad += input.t() >> grad;
};
```

Device Agnosticism & Acceleration



Abstracts backend. Handles Contexts, Command Queues, Buffer Allocation.

OpenCL JIT Compilation & Kernel Execution.

Lazy Migration:
Data moves between CPU and GPU only when necessary (`cpu()`, `gpu()` virtual methods).

Synchronization:
Explicit `LockSync` and `UnlockSync` for memory consistency.

Pre-compilation:
`weed_cl_precompile` utility caches kernels to disk to solve JIT latency.

Operator Interface & Kernel Dispatch

Lifecycle of an Operation

```
auto y = weights >> input;
```

Syntactic Sugar: `>>` for Matrix Multiplication (Pipe), `^^` for Power.

Dispatcher

Check DeviceTag & DType

Path A (CPU)

Route to `MatMulKernel` -> Iterate dense vector OR sparse map.

Path B (GPU)

Route to `OCL_API_MATMUL` -> Enqueue OpenCL Kernel.

Includes support for broadcasting, type promotion (Real + Complex -> Complex), and in-place optimization.

Composing Neural Networks: The Module API

```
// Define Layer
Linear model(2, 1);

// Forward Pass
auto output = model.forward(input);

// Loss Calculation
auto loss = mse_loss(output, labels);

// Backpropagation
loss.backward();

// Optimization
adam_step(model.parameters(), ...);
zero_grad(model.parameters());
```

Matches **`torch.nn.Linear`** familiarity.

Expressive C++ operators.

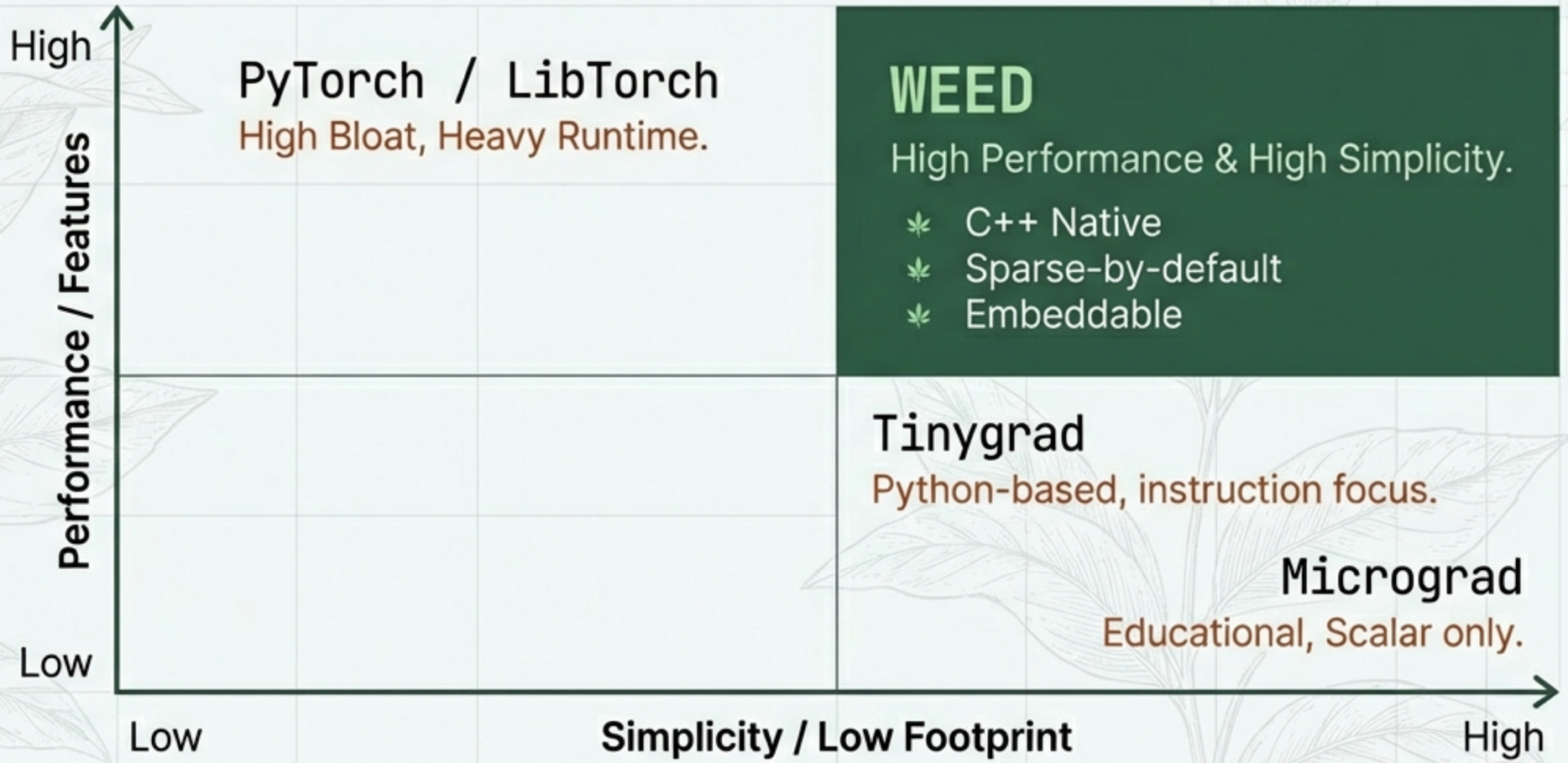
Built-in Loss functions.

Standard Autograd trigger.

Optimizer Step & Reset.

Provides the look-and-feel of PyTorch (`torch.nn.Module`) within a zero-dependency C++ environment.

The Landscape of Minimalism



Weed fills the gap for a production-grade tensor library that fits on a 64MB RAM embedded Linux board.

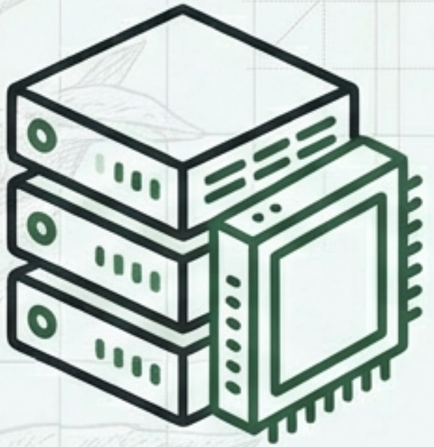
The Case for Sovereign AI

Supply Chain Defense

Zero external dependencies reduces the attack surface to just the compiler and OS. No hidden vulnerability in transitive packages.

Vendor Independence

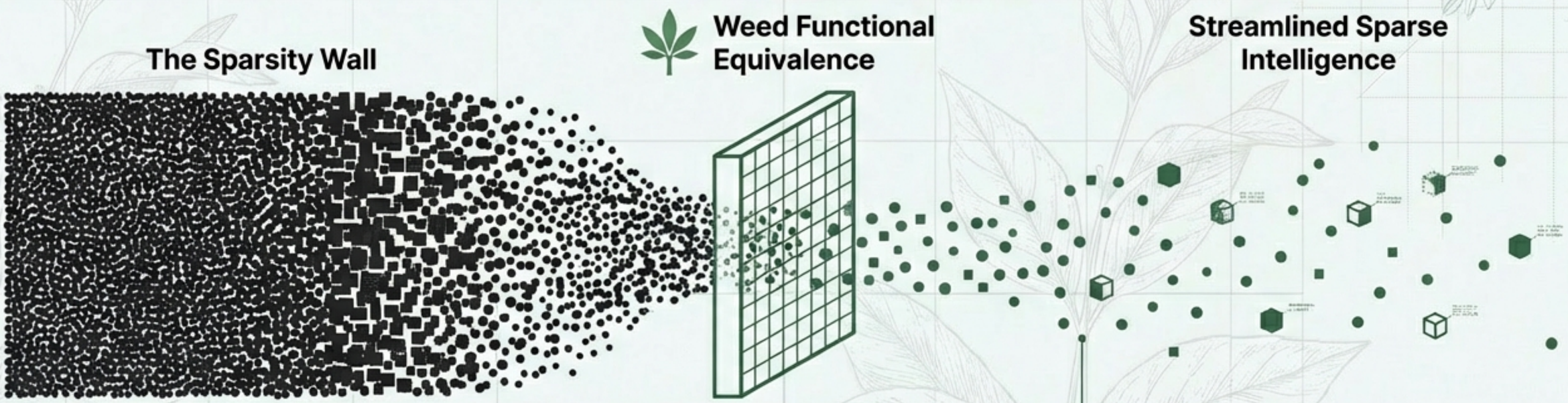
Built on Open Standards (C++11, OpenCL). Never locked into a hardware vendor or proprietary binary blobs.



Cloud Repatriation

Ideal for secure enclaves, defense, and legacy hardware repatriation.

Future Outlook: The Sparsity Wall



- As models grow to trillions of parameters, dense storage becomes physically impossible.
- **Legacy frameworks** require painful conversion. **Weed** makes sparsity a storage detail.
- Models naturally '**shrink**' in footprint and '**speed up**' in compute as they are pruned, without rewriting code.

"Refreshed infrastructure for the next generation of intelligence—where the correct thing is the default, and the expensive thing is explicit."