Chapter 6

Computational Toolbox Implementation

The following sections describe the computational toolbox that implements the models from chapter 5 and the operational strategies from chapter 4.

Section 6.1 presents the design principles guiding the software architecture. Section 6.2 describes the six-layer computational structure. Section 6.3 documents the simulation lifecycle. Section 6.4 addresses performance optimization strategies. Section 6.5 explains layer interactions.

6.1 Design Principles

Five principles guided the architecture of the simulation framework.

1. **Physics-Based Dynamic Modeling.** The absence of operational data for integrated PEM+SOEC+ATR systems precludes data-driven approaches. Dynamic models are necessary because load variations in wind-coupled electrolysis occur on timescales comparable to equipment response times [38]. The toolbox uses literature-derived parameters for offline planning, distinct from digital twins requiring real-time calibration.

2. **Modular Component Architecture.** Each equipment type exists as an independent module with standardized interfaces. This component-based design permits reconfiguration of plant topologies — stack capacities, storage volumes, pathway allocation — without modification to the simulation engine.

3. **Separation of Concerns.** The framework enforces strict boundaries between control (setpoint generation), physics (equipment response), and economics (cost accounting). Control signals propagate downward; physical constraints propagate upward. This separation prevents conflation of economic assumptions with physical capabilities.

4. **High Temporal Resolution.** Grid congestion signals and electricity prices vary at minute-scale intervals. The framework operates at one-minute timesteps, yielding approximately 10.5 million evaluations for a 20-year horizon — a computational load addressed in section 6.4.

5. **Performance-Accuracy Trade-off.** Zero-dimensional lumped-parameter models enable rapid multi-year simulations while higher-dimensional formulations remain computationally prohibitive. Thermodynamic properties are pre-computed on

pressure-temperature grids; runtime queries execute as interpolations rather than iterative equation-of-state solutions.

These five principles materialize in a six-layer computational architecture that separates infrastructure, properties, physics, topology, strategy, and orchestration into distinct software modules.

6.2 Six-Layer Computational Architecture

The framework organizes computation into six hierarchical layers. Each layer encapsulates a specific responsibility, enabling independent modification and testing. Figure 6.1 illustrates the layer hierarchy and data flow.

[Six-Layer Architecture Diagram]

Figure 6.1: Six-layer computational architecture showing data flow between layers.

1. **Infrastructure and Data.** Handles configuration loading, logging, and data persistence. Plant configurations reside in YAML files specifying equipment parameters, process connections, and economic inputs.

2. **State and Properties.** Provides thermodynamic properties via Lookup Tables (LUTs) storing pre-computed values on pressure-temperature grids for six fluids: H_2, O_2, H_2O, N_2, CH_4, and CO_2.

3. **Components.** Each equipment unit exists as a discrete object encapsulating its governing equations from chapter 5. Components respond to a standardized interface: receive setpoints, execute calculations, report outputs.

4. **Plant Topology.** Encodes physical interconnections as a directed graph. Topological sorting via Kahn's algorithm — iteratively processing nodes with no unresolved dependencies — establishes a calculation sequence where production precedes compression precedes storage.

5. **Logic and Strategy.** Interprets external signals (electricity prices, renewable availability) and generates operational setpoints. Implements the dispatch logic from chapter 4: SOEC-first allocation, multi-module coordination, oxygen-constrained ATR scheduling.

6. **Orchestration.** Manages simulation progression: clock advancement, execution coordination, event scheduling, and checkpointing.

The six-layer architecture defines the static structure of the software. The following section describes how this structure executes dynamically through the simulation lifecycle.

6.3 Simulation Lifecycle

Simulation execution proceeds through three phases: Setup, Execution, and Results.

6.3.1 Setup Phase

The setup phase transforms configuration files into a functional computational graph: parsing YAML files, instantiating components, constructing the plant graph, and validating that connections form a valid DAG. Cycles indicate specification errors. The LUT system initializes by loading pre-computed tables or generating them if unavailable.

6.3.2 Execution Phase

Each timestep follows a five-step sequence:

1. **Event Processing.** Check for scheduled maintenance or degradation updates.
2. **Dispatch Evaluation.** Read prices, storage levels, and renewable availability; compute setpoints.
3. **Component Stepping.** Execute physics calculations in sorted order.
4. **Flow Propagation.** Deliver outputs from source to destination components.
5. **Recording.** Store results using memory-efficient structures.

This sequence constitutes an explicit Euler forward integration: control decisions at timestep t use states from $t-1$, preventing circular dependencies.

6.3.3 Results Phase

The results phase aggregates timestep data into annual production totals, capacity factors, efficiency trends, and economic indicators (LCOH, NPV, IRR). The LCOH calculation follows the methodology in subsection 5.5.1.

The lifecycle description assumed that each timestep executes within acceptable time bounds. Achieving this performance at scale requires specific data structures and optimization strategies.

6.4 Data Structures and Performance Optimization

A 20-year simulation at one-minute resolution generates approximately 10.5 million data points per tracked variable. Managing this data volume requires specific optimization strategies.

6.4.1 Lookup Table Mechanism

Thermodynamic property calculations represent the primary computational bottleneck. A single timestep may require property evaluations for multiple streams at various conditions. Direct calls to libraries such as CoolProp would dominate execution time.

The LUT mechanism pre-computes property values on a 2000×2000 pressure-temperature grid per fluid, with geometric spacing in pressure and linear spacing in temperature. At runtime, property queries execute as bilinear interpolations. The interpolation kernel is compiled to machine code using Numba through Just-In-Time (JIT) compilation.

Table 6.1 presents performance comparison against direct CoolProp evaluation.

Table 6.1: Thermodynamic property calculation performance comparison.

Method	Speed (calls/s)	Speedup Factor
CoolProp direct	4,967	1×
LUT + JIT kernel	787,441	158×
LUT + Python overhead	243,000	49×

Interpolation errors remain below 0.1% relative to CoolProp across the operating envelope (0.05 bar to 100 bar and 0 °C to 227 °C for hydrogen streams; 0.05 bar to 50 bar and 0 °C to 927 °C for reforming streams).

6.4.2 State Management

The framework manages two categories of state data.

Transient flow states represent instantaneous conditions (mass flow rates, temperatures, pressures) at each timestep. These states serve as inputs to downstream components and are overwritten at each timestep.

Persistent component states track equipment condition over time. Electrolyzer stacks accumulate operating hours and start-stop cycles that affect degradation. Storage tanks maintain inventory levels. These states carry forward through simulation, allowing early-life operating patterns to influence late-life performance.

Degradation modeling uses empirical curves relating efficiency loss to cumulative operating hours and thermal cycles, derived from manufacturer data.

6.4.3 Memory Management

The framework uses pre-allocated NumPy arrays rather than dynamic Python lists, eliminating memory reallocation overhead during the execution loop.

For long simulations, the framework uses a chunked storage approach: data accumulates in memory buffers until reaching a threshold (10,000 timesteps), then flushes to disk in Parquet format. This maintains approximately constant memory usage (∼100 MB) regardless of simulation length.

The preceding sections described individual layers and their optimization. The following section examines how these layers coordinate during execution, focusing on the interfaces between control decisions, physical constraints, and economic accounting.

6.5 Layer Interactions

The framework enforces explicit boundaries between decision-making (control), physical response (physics), and cost accounting (economics). This section describes mechanisms coordinating these layers.

6.5.1 Lifecycle Contract

All components implement a common interface defined by the lifecycle contract. This contract specifies three methods every component must provide:

Initialize: Receive setpoints from control layer and boundary conditions from connected components. Prepare internal state for current timestep.

Step: Execute component's governing equations. Compute output states (mass flows, temperatures, power consumption) based on inputs and internal physics.

Report: Broadcast outputs to flow network for delivery to downstream components. Record results for post-processing.

This standardized interface allows the orchestration layer to manage all components uniformly, regardless of their internal complexity. A simple mixer and a complex SOEC stack both respond to the same method calls.

6.5.2 Flow Architecture

Mass and energy propagate through the plant graph via a push mechanism. When a component completes its step calculation, it makes outputs available at output ports. The flow network delivers these outputs to input ports of connected downstream components.

For demand-driven behavior (e.g., a storage tank signaling fullness), the framework propagates control signals as a special flow type. A downstream component experiencing a constraint (tank at maximum level) outputs a demand signal, a multiplier indicating reduced acceptance capacity. This signal propagates upstream through the flow network. Upstream components receive the signal and adjust production in the subsequent timestep.

The one-timestep delay in responding to demand signals is a deliberate design choice. A component cannot modify its own output within the same timestep in response to downstream conditions; this would create circular dependency. The delay ensures causal consistency at cost of a one-minute response lag, acceptable for the dynamics being modeled.

6.5.3 Economic Reporting

Economic calculations operate on achieved physical outputs, not requested setpoints. If the control layer requests a power level exceeding equipment ramping capability, the physics layer delivers only what equipment can achieve. The economic layer then calculates costs based on actual energy consumption and actual production.

This approach ensures economic projections reflect physical constraints. A simulation cannot overestimate revenue by assuming production rates that violate equipment limits. Similarly, degradation effects propagate to economics: as efficiency declines over plant lifetime, energy costs per kilogram of hydrogen increase.

The architecture and interactions described above enable simulation of complex plant behavior. Validation of the framework, its operational envelope, and performance benchmarks are presented in chapter 7.