Simulation Framework for Dual-Path Industrial Hydrogen Production v2.0: Architecture, Physics, and High-Frequency Optimization

Technical Engineering Team

December 2025

Abstract

This paper presents the architectural and mathematical specification for the Dual-Path Hydrogen Production System v2.0. The framework provides a high-fidelity, event-driven simulation environment for modeling industrial-scale hydrogen plants combining Grid-powered Electrolysis (PEM/SOEC) and Natural Gas Autothermal Reforming (ATR).

Version 2.0 introduces component-level design with 50+ physics models across 20 categories, replacing the system-level abstraction of v1.0. Key innovations include: (1) a six-layer modular architecture with strict lifecycle contracts, (2) Look-Up Table (LUT) caching on 2000-point (P, T) grids achieving 50–200× speedup, (3) Numba JIT compilation for iterative solvers, and (4) a Split-Layer Control Architecture decoupling economic dispatch from physical outcomes.

The physics models employ rigorous governing equations including the Shomate polynomial for thermodynamic properties, the Ergun equation for packed bed pressure drop, and Stokes-law-based separation efficiency for cyclonic separators. The framework completes annual simulations (525,600 timesteps at 1 min resolution) in under 5 min on standard hardware, enabling rapid techno-economic analysis for capacity planning and hydrogen market participation.

Keywords: Hydrogen, Process Simulation, Techno-Economic Analysis, PEM Electrolysis, SOEC, Python, Numba, Look-Up Tables

Nomenclature

Symbol	Description	Unit
Flow State Variables		
$\dot{m}$	Mass flow rate	kg/h
P	Pressure	Pa
T	Temperature	K
x_i	Mass fraction of species i	–
y_i	Mole fraction of species i	–
$\mathbf{x}$	Composition vector $\{x_{\mathrm{H}_2}, x_{\mathrm{O}_2}, \ldots\}$	–
Thermodynamic Properties		
H	Specific enthalpy	J/kg
S	Specific entropy	J/(kg·K)
C_p	Isobaric heat capacity	J/(kg·K)
ρ	Density	kg/m^3
k	Ratio of specific heats (C_p/C_v)	–
Physical Constants		
R	Universal gas constant	$8.314\,\mathrm{J/(mol\cdot K)}$
R_{H_2}	Specific gas constant (H$_2$)	$4124\,\mathrm{J/(kg\cdot K)}$
T_{ref}	Reference temperature	298.15 K
P_{ref}	Reference pressure	101 325 Pa
Efficiency and Performance		
η	Efficiency (subscripted)	–
η_{is}	Isentropic efficiency	–
η_F	Faraday efficiency	–
Simulation Parameters		
t	Simulation time	h
Δt	Timestep	1 min (1/60 h)
$\mathbf{U}$	Internal state vector	(varies)
$\mathbf{S}$	Stream state vector	(defined below)
Electrochemical Variables		
j	Current density	A/m^2
j_0	Exchange current density	A/m^2
α	Charge transfer coefficient	–
F	Faraday constant	96 485 C/mol
z	Electron transfer number	–
Separation and Transport		
ε	Bed porosity (void fraction)	–
μ	Dynamic viscosity	Pa·s
D_p	Particle diameter	m
d_{50}	Cut-size diameter (cyclone)	m
N_e	Number of effective turns	–
u_s	Superficial velocity	m/s

1 Introduction

1.1 Problem Definition

The global transition to a hydrogen-based energy economy demands sophisticated modeling tools capable of handling the intrinsic complexity of hybrid production pathways. Industrial hydrogen production increasingly combines multiple technologies:

- **Green Hydrogen**: Water electrolysis powered by renewable electricity (PEM, SOEC)
- **Blue Hydrogen**: Autothermal reforming (ATR) with carbon capture

Each pathway exhibits distinct operational characteristics, capital costs, and carbon footprints, creating a multi-objective optimization landscape that plant operators must navigate in real-time.

Annual techno-economic assessments require simulation of 8,760 operating hours—or 525,600 timesteps at 1 min resolution—creating a formidable computational challenge. Each timestep demands:

1. Thermodynamic property calculations for multi-component gas mixtures
2. Phase equilibrium computations for separation equipment
3. Control logic evaluation for power allocation and arbitrage

Naively implemented, each CoolProp equation-of-state evaluation requires approximately 0.1 ms to 1 ms. Multiplied by 50+ components and 26 million annual property calls, simulation time would extend to days or weeks.

1.2 What's New in Version 2.0

Based on the detailed component system guide, v2.0 introduces the following upgrades:

- **Component-Level Design**: Build plants using individual components (PEM stacks, heat exchangers, pumps) rather than system-level abstractions
- **20 Component Categories**: 50+ implementations organized by functional area (electrolysis, separation, compression, thermal, storage, water systems)
- **Environment Manager**: Time-series environmental data (wind power availability from `wind_data.csv`, electricity prices from `EnergyPriceAverage2023-24.csv`) accessible to all components via registry lookup
- **System Assignment**: Context-dependent components (e.g., Rectifier, PSA Unit) can be assigned to specific systems (PEM/SOEC/ATR)
- **Backward Compatibility**: Both v1.0 (system-level) and v2.0 (component-level) configurations are supported

1.3 Optimization Approach

The system addresses the computational challenge through a multi-layered optimization approach:

1. **Pre-computed Thermodynamic LUTs**: 2D interpolation grids on (P, T) replace expensive CoolProp calls, achieving 50–200× speedup
2. **Just-In-Time Compilation**: Numba-decorated numerical kernels compile to native machine code
3. **Pre-allocated Memory**: NumPy arrays sized at initialization eliminate dynamic allocation
4. **Causal Execution Ordering**: Component stepping follows mass and energy propagation paths

2 System Architecture

The software is constructed upon a six-layer modular architecture designed to ensure separation of concerns and scalability.

2.1 Layered Design

The architectural hierarchy progresses from foundational abstractions to user-facing interfaces:

Table 1: Six-Layer Architecture Overview

Layer	Purpose	Key Classes
1	Core Foundation	`Component`, `Stream`, `ComponentRegistry`
2	Performance	`LUTManager`, `numba_ops`
3	Components	50+ physics models (20 categories)
4	Orchestration	`PlantGraphBuilder`, `DualPathCoordinator`
5	Simulation	`SimulationEngine`, `FlowNetwork`
6	UI/Reporting	PySide6 editors, `markdown_report`

2.2 Component Lifecycle Contract

Every simulation entity inherits from the abstract `Component` base class, ensuring uniform behavior. The lifecycle contract formalizes three distinct phases:

Listing 1: Component Lifecycle Interface

```
class Component(ABC):
    @abstractmethod
    def initialize(self, dt: float,
                   registry:
                       ComponentRegistry):
        """Allocate memory, resolve
           dependencies."""

    @abstractmethod
    def step(self, t: float):
        """Advance state by one timestep.
            """

    @abstractmethod
    def get_state(self) -> Dict[str, Any]:
        """Return JSON-serializable state.
            """
```

Initialization Phase: Called once before simulation. Components allocate internal data structures (NumPy arrays sized for simulation duration), resolve dependencies via `ComponentRegistry`, and validate configuration.

Stepping Phase: Called every timestep with simulation time t. Components process inputs via `receive_input()`, execute physics models, and prepare outputs for downstream consumers.

State Access Phase: Components expose internal state through `get_state()`, enabling checkpoint persistence and monitoring.

2.3 Data Flow Patterns

The architecture implements distinct patterns for physics data and control signals:

2.3.1 Push Architecture (Stream Propagation)

Mass and energy flow downstream via the `Stream` dataclass:

$$\texttt{Producer.step()} \rightarrow \texttt{Downstream.receive\_input(stream)}$$

Each producer calculates outputs and pushes them to connected consumers. The `FlowNetwork` orchestrates this propagation after all components have stepped.

2.3.2 Pull Architecture (Control Flow)

Control and monitoring data flow upstream via registry queries:

$$\texttt{Dispatch.record\_post\_step()} \rightarrow \texttt{component.get\_state()}$$

The dispatch strategy queries actual component states after physics execution to record outcomes that may differ from commanded setpoints.

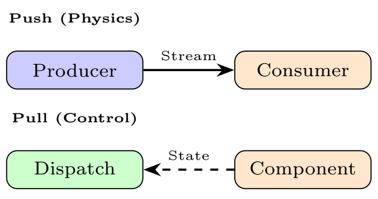

Figure 1: Push vs. Pull Data Flow Patterns

2.4 The Registry Pattern

The `ComponentRegistry` serves as the central orchestrator for dependency injection:

$$T_{\text{lookup}} = O(1) \quad \text{(hash-based dictionary access)} \tag{1}$$

This pattern provides:

- **Decoupled Dependencies**: Components depend on string IDs, not concrete class references
- **Lazy Resolution**: Dependencies resolved at initialization, avoiding circular imports
- **Type-Indexed Queries**: Retrieve components by category (e.g., "storage")
- **State Aggregation**: Single `get_all_states()` call for checkpointing

2.5 Event-Driven Architecture

The simulation combines continuous physics with discrete events via the `EventScheduler`:

$$\text{Timestep} = \underbrace{\text{Events}}_{\text{Discrete}} + \underbrace{\text{Physics}}_{\text{Continuous}} + \underbrace{\text{Flows}}_{\text{Network}} \tag{2}$$

Event types include maintenance scheduling, price updates, setpoint changes, and recurring actions (e.g., SOEC module priority rotation).

2.6 Execution Flow & Orchestration

The `SimulationEngine` does not merely iterate through timesteps; it orchestrates a strict multi-phase cycle that ensures consistency between **Economic Intent** (what the dispatch strategy commands) and **Physical Result** (what the thermodynamics permit).

2.6.1 Anatomy of a Timestep: The 3-Phase Loop

Each timestep t follows a rigorous three-phase execution pattern:

Phase 1: Decision (Pre-Step) The `EngineDispatchStrategy` analyzes current conditions—electricity spot prices, hydrogen selling prices, and wind power availability—to calculate optimal power allocation:

$$\texttt{dispatch\_strategy.decide\_and\_apply(t, prices, wind)}$$

This phase produces *setpoints* (Intent), which are injected into components via `receive_input()`. For example, the dispatch may command the PEM electrolyzer to operate at 2.5 MW, representing the economically optimal split between grid sales and hydrogen production.

Phase 2: Physics (Step) The `ComponentRegistry` orchestrates component execution in causal order:

```
for component in execution_order:
    component.step(t)
```

The *actual* physics happens here. Crucially, physical constraints may override economic commands. For example:

- A 2.5 MW setpoint may result in only 2.1 MW actual consumption if the stack reaches thermal limits
- Compressor throughput may be curtailed if discharge temperature exceeds 150 °C
- Storage tanks may refuse inflow if pressure exceeds $P_{\max}$

After all components have stepped, the `FlowNetwork` propagates streams through the topology:

```
flow_network.execute_flows(t)
```

This "Push" propagation delivers output streams from producers to connected consumers via `receive_input()`.

Phase 3: Recording (Post-Step) The dispatch strategy queries *actual* component states to record outcomes:

```
dispatch_strategy.record_post_step()
```

This phase reads operational metrics via `get_state()` and writes them to pre-allocated NumPy arrays. The recorded values reflect what *actually happened*, not what was *commanded*—enabling accurate techno-economic analysis.

2.6.2 Concrete Example: Wind to Hydrogen

Consider wind power becoming available at timestep t:

1. **Decision**: Dispatch calculates that producing hydrogen yields higher value than grid export. Commands 5 MW to PEM electrolyzer via `pem.receive_input('power_setpoint', 5000, 'kW')`.
2. **Physics**: PEM executes `step(t)`:
   - Calculates $\dot{m}_{H_2}$ from Faraday's law
   - Creates output `Stream(mass_flow=45.2, T=353, P=3e6, composition={'H2': 0.98, 'H2O': 0.02})`
3. **Flow Propagation**: `FlowNetwork` pushes the stream to the Knock-Out Drum:

   ```
   kod.receive_input('inlet', pem_stream,
                     'gas')
   ```

4. **Recording**: Dispatch reads `pem.get_state()['actual_power_kw']` = 4850 (thermal de-rating) and stores in history array.

2.6.3 Algorithm: Main Simulation Loop

Algorithm 2 formalizes the timestep execution using actual method names from the codebase.

Listing 2: SimulationEngine Main Loop

```python
def run(start_hour, end_hour):
    # Initialization Phase
    registry.initialize_all(dt=dt_hours)
    flow_network.initialize()

    for step in range(start_step, end_step):
        hour = step / 60  # 1-minute resolution

        # --- Phase 1: Decision (Pre-Step) ---
        event_scheduler.process_events(hour, registry)
        if dispatch_strategy:
            dispatch_strategy.decide_and_apply(
                t=hour, prices=prices, wind=wind)

        # --- Phase 2: Physics (Step) ---
        for comp_id in execution_order:
            registry.get(comp_id).step(hour)

        flow_network.execute_flows(hour)

        # --- Phase 3: Recording (Post-Step) ---
        if dispatch_strategy:
            dispatch_strategy.record_post_step()

        monitoring.collect(hour, registry)
```

2.6.4 Causal Execution Order

Component ordering ensures upstream calculations complete before downstream consumption. The default order follows mass/energy propagation:

1. **External Inputs**: `environment_manager`, `energy_price_tracker`
2. **Coordination**: `dual_path_coordinator`
3. **Production**: `soec_cluster`, `pem_electrolyzer`
4. **Thermal**: `thermal_manager`, `chiller`
5. **Separation**: `knock_out_drum`, `psa`
6. **Compression**: `compressor_c1`, `compressor_c2`
7. **Storage**: `lp_tanks`, `hp_tanks`

This ordering minimizes the need for iterative convergence within each timestep.

3 Mathematical Modeling

This section formalizes the mathematical abstraction underlying the simulation framework, establishing notation and governing principles before presenting component-specific equations.

3.1 Thermodynamic State Vector

The fundamental data structure for inter-component communication is the **Stream**, which encapsulates the complete thermodynamic state of a material flow. We define the flow state vector:

$$\mathbf{S} = \begin{bmatrix} \dot{m} \\ P \\ T \\ \mathbf{x} \end{bmatrix} \quad \text{where} \quad \mathbf{x} = \{x_{\mathrm{H_2}}, x_{\mathrm{O_2}}, x_{\mathrm{H_2O}}, \ldots\} \tag{3}$$

The composition vector $\mathbf{x}$ contains mass fractions satisfying the constraint $\sum_i x_i = 1$. Each component acts as a **transfer function** transforming an input stream to an output stream:

$$\mathbf{S}_{out} = f_c(\mathbf{S}_{in}, \mathbf{U}_c, t) \tag{4}$$

where $\mathbf{U}_c$ represents the internal state of component c (e.g., tank inventory, stack temperature) and t is time. This functional abstraction enables modular composition: complex plants are constructed by chaining transfer functions.

3.2 Time Discretization

The simulation advances using a fixed discrete timestep approach. Component internal states evolve according to:

$$\frac{d\mathbf{U}}{dt} = g(\mathbf{U}, \mathbf{S}_{in}, t) \tag{5}$$

This ODE is integrated using an explicit Euler scheme:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t \cdot g(\mathbf{U}^n, \mathbf{S}_{in}^n, t^n) \tag{6}$$

where $\Delta t = 60\,\mathrm{s} = 1/60$ h and n denotes the timestep index. For annual simulations, $n \in \{0, 1, \ldots, 525599\}$.

Dimensional Note: All rate quantities (e.g., $\dot{m}$) are stored in engineering units (kg/h) but are internally normalized to consistent time bases during integration. The timestep Δt is expressed in hours when computing state updates to ensure dimensional homogeneity with mass flow rates.

The explicit scheme is chosen for:

- **Simplicity**: No matrix inversion required
- **Speed**: Each component evaluates $g(\cdot)$ once per timestep
- **Stability**: The 60 s timestep is well below the Courant-Friedrichs-Lewy (CFL) limit for the physical processes modeled

3.3 Conservation Laws

All components in the simulation obey fundamental conservation principles applied to control volumes. These balance equations form the theoretical foundation upon which component-specific physics are built.

3.3.1 Mass Balance

The time-dependent mass balance for a generic control volume with multiple inlets and outlets:

$$\frac{dM}{dt} = \sum_j \dot{m}_{in,j} - \sum_k \dot{m}_{out,k} \tag{7}$$

where M is the mass inventory (kg), and $\dot{m}$ are mass flow rates (kg/h).

3.3.2 Species Balance

For each chemical species i (H_2, O_2, H_2O, etc.):

$$\frac{d(Mx_i)}{dt} = \sum_j \dot{m}_{in,j} x_{i,j} - \sum_k \dot{m}_{out,k} x_{i,k} + r_i V \tag{8}$$

where x_i is the mass fraction of species i, r_i is the volumetric reaction rate (kg/m^3/h), and V is the control volume (m^3). For non-reactive components (mixers, pipes, separators), $r_i = 0$.

3.3.3 Energy Balance

The first law of thermodynamics for an open system:

$$\frac{dU}{dt} = \sum_j \dot{m}_{in,j} H_{in,j} - \sum_k \dot{m}_{out,k} H_{out,k} + \dot{Q} - \dot{W} \tag{9}$$

where U is internal energy (J), H is specific enthalpy (J/kg), $\dot{Q}$ is heat transfer rate (W), and $\dot{W}$ is shaft work rate (W).

3.3.4 Quasi-Steady Approximation

For **quasi-steady** components—such as pipes, mixers, heat exchangers, and separators—the accumulation terms are negligible:

$$\frac{dM}{dt} \approx 0, \quad \frac{dU}{dt} \approx 0 \tag{10}$$

This reduces the differential equations to **algebraic** constraints, significantly simplifying the computational burden. Only **dynamic** components (storage tanks, thermal masses, electrolyzer stacks) retain time-dependent accumulation terms.

3.4 Thermodynamic Property Framework

The simulation employs a **Hybrid Equation of State (EOS)** approach, balancing computational efficiency with physical accuracy.

3.4.1 Equation of State

Low-Pressure Gas Streams For gas-phase streams at moderate pressures ($P < 50\,\text{bar}$), the **Ideal Gas Law** is used:

$$\rho = \frac{PM_{mix}}{RT} \tag{11}$$

where M_{mix} is the mixture molecular weight (kg/-mol) and $R = 8.314\,\text{J}/(\text{mol}\cdot\text{K})$.

High-Pressure Pure Fluids For high-pressure applications (compression, storage at $P > 100\,\text{bar}$), **Look-Up Tables (LUTs)** derived from CoolProp provide accurate real-gas properties including compressibility effects.

Liquid Water Liquid water density is treated as incompressible: $\rho_{H_2O} = 1000\,\text{kg/m}^3$.

3.4.2 Caloric Properties

Specific heat capacity and enthalpy are derived from the **Shomate polynomial** (NIST database), which captures temperature dependence:

$$C_p(T) = A + B\tau + C\tau^2 + D\tau^3 + \frac{E}{\tau^2} \quad [\text{J}/(\text{mol}\cdot\text{K})] \tag{12}$$

where $\tau = T/1000$. Coefficients A, B, C, D, E, F are species-specific (stored in `constants.py`). This formulation replaces the simplistic constant-C_p assumption with rigorous NIST-based temperature dependence.

3.4.3 Mixing Rules

Mixture properties are calculated using linear combination rules (Dalton's Law approximation for ideal gas mixtures):

Molar Properties

$$H_{mix} = \sum_i y_i H_i(T), \quad S_{mix} = \sum_i y_i S_i(T,P) - R\sum_i y_i \ln(y_i) \tag{13}$$

where y_i is the mole fraction of species i.

Mass-Basis Properties

$$C_{p,mix} = \sum_i x_i C_{p,i}(T), \quad M_{mix} = \left(\sum_i \frac{x_i}{M_i}\right)^{-1} \tag{14}$$

Kay's Rule for Pseudo-Critical Properties When compressibility corrections are needed, pseudo-critical properties are estimated via Kay's mixing rule:

$$T_{c,mix} = \sum_i y_i T_{c,i}, \quad P_{c,mix} = \sum_i y_i P_{c,i} \tag{15}$$

This enables reduced property calculations for real-gas corrections in high-pressure applications.

3.5 Governing Physics Summary

Table 2 summarizes the physical principles governing each component category, referencing both conservation laws and component-specific equations.

Table 2: Component Physics Summary

Component	**Physical Principle**	**Equation**
Generic (all)	Mass conservation	Eq. 7
Generic (all)	Energy conservation	Eq. 9
Compressor	Isentropic work	Eq. ??
KnockOutDrum	Rachford-Rice VLE	Eq. ??
TSA Unit	Ergun pressure drop	Eq. ??
Multi-Cyclone	Stokes settling	Eq. ??
PEM	Tafel kinetics	Eq. ??
SOEC	Operating curve	Table ??
Storage Tank	Ideal gas accumulation	Eq. ??

3.6 Component-Specific Physics

The following subsections detail the physics models for specific component categories. All components inherit the conservation laws from Section 3.3 and property calculations from Section 3.4.

3.6.1 Caloric Property Integration

Enthalpy is computed by integrating the Shomate C_p polynomial (Eq. 12) from reference temperature T_{ref}:

$$H(T) - H(T_{ref}) = A\tau + \frac{B\tau^2}{2} + \frac{C\tau^3}{3} + \frac{D\tau^4}{4} - \frac{E}{\tau} + F \tag{16}$$

where $\tau = T/1000$. This is distinct from simulation time t.

3.6.2 Compression Systems

The framework distinguishes between two compression regimes based on pressure differential and thermal constraints.

Regime A: Adiabatic Single-Stage Implemented in `CompressorSingle`, this model applies to low-pressure differentials ($r_p < 4.0$), such as Balance-of-Plant (BoP) recirculation pumps or transfer compressors. The process is modeled as adiabatic compression corrected by isentropic efficiency η_{is}:

$$W_{compressor} = \dot{m} \cdot \frac{H(P_{out}, S_{in}) - H(P_{in}, T_{in})}{\eta_{is}} \tag{17}$$

The discharge temperature T_{out} is obtained by solving the inverse enthalpy relation:

$$T_{out} = T(P_{out}, H_{out}) \quad \text{where} \quad H_{out} = H_{in} + \frac{W_{compressor}}{\dot{m}} \tag{18}$$

Regime B: Multi-Stage Intercooled Implemented in `CompressorStorage`, this model is used for high-pressure storage applications ($30 \rightarrow 350$ bar). The system minimizes work by approximating isothermal compression through N stages of adiabatic compression followed by intercooling.

The number of stages N is optimized to limit discharge temperature ($T_{out} \leq T_{max}$):

$$N = \left\lceil \frac{\ln(P_{target}/P_{suction})}{\ln(r_{max})} \right\rceil \tag{19}$$

The stage pressure ratio r_p assumes geometric progression:

$$r_p = \left(\frac{P_{target}}{P_{suction}} \right)^{1/N} \tag{20}$$

Total power is the summation of individual stage work, with fluid temperature reset to T_{cool} (typically 40 °C) between stages:

$$P_{total} = \dot{m} \sum_{i=1}^{N} \frac{1}{\eta_{is}} \Big[H(P_{i,out}, S_{in,i}) - H(P_{i,in}, T_{in,i}) \Big] \tag{21}$$

Implementation Note: Enthalpy changes ΔH and entropy S are dynamically calculated using the Shomate polynomial or LUTs (Section 3.4), capturing real-gas behavior where $C_p(T)$ and $Z(P,T)$ vary significantly.

3.6.3 Electrolysis Systems

The plant employs two distinct electrolysis technologies, modeled with fidelities appropriate to their operational dynamics.

PEM Electrolysis (Polarization Model) The Proton Exchange Membrane (PEM) electrolyzer (‘DetailedPEMElectrolyzer‘) utilizes a mechanistic electrochemical model. The cell voltage V_{cell} is calculated by summing the reversible potential and overpotential losses:

$$V_{cell}(j, T, P) = V_{rev} + V_{act} + V_{ohm} + V_{conc} \tag{22}$$

where j is current density (A/cm^2). The reversible potential V_{rev} follows the Nernst equation:

$$V_{rev} = V^0 + \frac{RT}{2F} \ln \left(\frac{P_{H_2} P_{O_2}^{0.5}}{a_{H_2O}} \right) \tag{23}$$

Hydrogen production rate is derived from Faraday's Law, accounting for efficiency losses (η_F) due to crossover:

$$\dot{m}_{H_2} = N_{cells} \cdot \frac{I}{2F} \cdot M_{H_2} \cdot \eta_F(j) \tag{24}$$

SOEC Electrolysis (Operational Model) The Solid Oxide Electrolyzer (‘SOECOperator‘) is modeled as a thermodynamic state machine due to the dominance of thermal transients in ceramic stacks. The system operates in discrete states $S \in \{\mathrm{OFF}, \mathrm{RAMP}, \mathrm{STEADY}, \mathrm{STANDBY}\}$.

Production is governed by specific energy consumption maps $E_{spec}(t_{age})$ derived from degradation curves, reflecting the thermodynamic advantage of steam splitting ($\sim$ 37.5 kWh/kg):

$$\dot{m}_{H_2} = \frac{P_{load}}{E_{spec}(t_{age})} \tag{25}$$

Thermal gating logic prevents load changes until stack temperature stabilizes, capturing the high thermal inertia of the modules.

3.6.4 Balance of Plant (Water Systems)

Water circuit components (`WaterPump`, `PumpThermodynamic`) model incompressible flow dynamics. Unlike gas compressors, density ρ is constant ($\approx 1000\,\mathrm{kg/m^3}$):

$$P_{pump} = \frac{\dot{m} \Delta P}{\rho \eta_{pump}} \tag{26}$$

Mass balance closure ensures continuous supply for electrolysis:

$$\dot{m}_{water,in} = \sum \dot{m}_{cons,electrolyzers} + \dot{m}_{purge} \tag{27}$$

3.6.5 Thermal Management (Dry Coolers)

The indirect cooling system (`DryCooler`) rejects process heat Q_{rej} to the ambient environment. The system operates under a rigid thermodynamic constraint: the outlet process temperature T_{out} cannot drop below the ambient air temperature T_{air} plus a minimum approach difference ΔT_{min}:

$$T_{out} \geq T_{air} + \Delta T_{min} \tag{28}$$

Heat rejection is calculated using the ε-NTU method for cross-flow exchangers:

$$Q_{rej} = \varepsilon \cdot C_{min}(T_{in} - T_{air}) \tag{29}$$

Unlike variable-load compressors, the air-cooler fan power is modeled as a fixed parasitic load P_{fan} when active, derived from the nominal design air flow:

$$P_{fan} = \frac{\dot{V}_{air} \Delta P_{air}}{\eta_{fan}} \tag{30}$$

3.6.6 Flow Control (Throttling Valves)

Throttling valves (`ThrottlingValve`) are modeled as isenthalpic expansion devices (Joule-Thomson expansion). Assuming adiabatic operation and negligible kinetic energy changes:

$$H(P_{in}, T_{in}) = H(P_{out}, T_{out}) \tag{31}$$

While enthalpy H is conserved, the outlet temperature T_{out} is determined by querying the Equation of State (LUT or Shomate) to solve the inverse problem:

$$T_{out} = T(P_{out}, H_{in}) \tag{32}$$

Crucially, for Hydrogen at standard conditions ($T > 202\,\text{K}$), the Joule-Thomson coefficient $\mu_{JT} < 0$. Consequently, expansion results in a temperature **increase**, contrasting with the cooling effect seen in standard gases.

3.6.7 Mixing and Junction Systems

Unlike simplified simulators that employ linear temperature averaging ($T_{mix} \approx \sum w_i T_i$), this framework enforces **rigorous enthalpy and energy conservation**. This accounts for non-linear specific heat capacity $C_p(T)$ and phase behavior critical for detecting thermal anomalies.

Gas Phase Mixing The `MultiComponentMixer` operates in two thermodynamic modes depending on the physical context, utilizing **Newton-Raphson iteration** accelerated via **Numba JIT**:

- **Continuous Mode (Pipe Junction)**: Solves a **PH-Flash** where specific enthalpy is conserved. The outlet temperature T_{out} is found such that:

$$H_{mix}(P, T_{out}) = \frac{\sum \dot{m}_i H_i(T_i, P_i)}{\sum \dot{m}_i} \tag{33}$$

- **Batch Mode (Tank Filling)**: Solves a **UV-Flash** where internal energy is conserved within the rigid control volume. The internal energy accumulation equals the incoming enthalpy flow (flow work included):

$$U_{sys}(t + \Delta t) = U_{sys}(t) + \sum (\dot{m}_i H_i)\Delta t \tag{34}$$

Liquid Phase Mixing The `WaterMixer` implements **Adiabatic Isobaric Mixing**, essential for tracking thermal buildup in recirculation loops where C_p varies with temperature:

1. **Mass Balance**: $\dot{m}_{out} = \sum \dot{m}_i$
2. **Enthalpy Balance**: $H_{out} = (\sum \dot{m}_i H_i)/\dot{m}_{out}$
3. **State Resolution**: Invert Equation of State ($T_{out} = T(P_{out}, H_{out})$)

Buffer Dynamics The `OxygenMixer` accounts for the thermal mass of gas buffers, applying First-Order Lag damping to temperature updates:

$$T^{n+1} = \left(1 - \frac{\Delta t}{\tau}\right) T^n + \frac{\Delta t}{\tau} T_{target} \tag{35}$$

Pressure is rigorously derived from density $\rho = M/V$ and temperature using the Real-Gas EOS:

$$P = P_{EOS}(\rho, T) \tag{36}$$

3.6.8 Chemical Reaction Systems

This framework employs divergent modeling strategies for chemical reactors, selecting between mechanistic and surrogate approaches based on the balance between safety-critical dynamics and computational tractability.

Catalytic Purification (Deoxo PFR) [Mechanistic Model] The `DeoxoReactor` removes trace oxygen via the highly exothermic reaction $2H_2 + O_2 \rightarrow 2H_2O$ ($\Delta H_{rxn} = -242\,\text{kJ/mol}$). Due to the safety risk of thermal runaway, a rigorous **Plug Flow Reactor (PFR)** model calculates spatial profiles (dT/dL, dX/dL) to detect hotspots.

The system solves coupled Ordinary Differential Equations (ODEs) using **Runge-Kutta 4th Order (RK4)** integration:

$$\frac{dX_{O2}}{dL} = \frac{k(T) C_{O2} A_c}{\dot{F}_{O2,0}} \tag{37}$$

$$\frac{dT}{dL} = \frac{(-\Delta H_{rxn}) \dot{F}_{O2,0} \frac{dX}{dL} - U \pi D (T - T_{cool})}{\sum \dot{F}_i C_{p,i}} \tag{38}$$

where $k(T)$ follows Arrhenius kinetics. This granularity ensures that catalyst temperature limits ($T < T_{max}$) are strictly enforced during transient startup.

Autothermal Reforming (ATR) [Surrogate Model] Simulating the complex kinetics of Autothermal Reforming (Partial Oxidation + Steam Reforming + Water-Gas Shift) is computationally prohibitive for real-time plant control. The `ATRReactor` therefore utilizes a **Data-Driven Surrogate Model**.

Yield curves derived from high-fidelity offline simulations (Aspen Plus) are mapped using **Numba-accelerated interpolation**:

$$\mathbf{Y}_{out} = \mathcal{F}_{interp}(\dot{F}_{O2}, \dot{F}_{feed}) \tag{39}$$

This model captures the net effect of three simultaneous reactions:

- **Partial Oxidation**: $CH_4 + 0.5O_2 \rightarrow CO + 2H_2$ ($\Delta H < 0$)

- **Steam Reforming**: $CH_4 + H_2O \rightleftharpoons CO + 3H_2$ ($\Delta H > 0$)

- **Water-Gas Shift**: $CO + H_2O \rightleftharpoons CO_2 + H_2$ ($\Delta H < 0$)

This approach ensures accurate mass/energy balances ($\pm 1\%$) while reducing calculation time by orders of magnitude compared to mechanistic CFD or Gibbs minimization solvers.

3.6.9 Separation & Purification

Inertial Separation (Multi-Cyclone) The `HydrogenMultiCyclone` employs the **Barth/-Muschelknautz Model** to determine the critical cut-size magnitude. Separation occurs when the centrifugal settling velocity exceeds the radial gas drift in the swirl tubes:

$$v_{settling}(d_p) \geq v_{radial} \tag{40}$$

The model solves for the particle diameter d_{50} (50% efficiency) using **Numba-accelerated iterations** to resolve the coupling between the tangential velocity field and particle drag coefficients. Pressure drop is derived from the geometric **Euler Number** (ξ) correlation:

$$\Delta P = \xi \frac{\rho_g v_{axial}^2}{2} \tag{41}$$

Aerosol Coalescence (Porous Media) For fine mist polishing ($d_p < 0.1\,\mu\text{m}$) where cyclones are ineffective, the `Coalescer` uses fibrous cartridge elements. Pressure drop is modeled via the **Carman-Kozeny Equation** for flow through porous media:

$$\Delta P = K_{perm} \mu L_{elem} U_{sup} \tag{42}$$

where μ follows the Sutherland power law ($\mu \propto T^{0.7}$). Unlike inertial separators, this component strictly accounts for **Dissolved Gas Losses** in the condensate. The mass of H_2/O_2 lost is calculated using Henry's Law constants ($H(T)$):

$$\dot{m}_{gas,loss} = \dot{m}_{liq,rem} \cdot \frac{P_{partial}}{H(T)} \tag{43}$$

This ensures rigorous mass conservation even for waste streams.

Gravity Separation (Knock-Out Drum) The `KnockOutDrum` removes bulk liquid via gravitational settling. To prevent liquid re-entrainment, the superficial gas velocity (u_G) must not exceed the terminal settling velocity derived from the **Souders-Brown Equation**:

$$u_{max} = K_{SB} \sqrt{\frac{\rho_L - \rho_G}{\rho_G}} \tag{44}$$

The component verifies that the vessel Diameter (D) is sufficient such that $u_{real} < u_{max}$. Separation equilibrium works as an **Isoenthalpic Flash**, solving the Rachford-Rice equation to determine the exact vapor-liquid split at inlet conditions.

Pressure Swing Adsorption (PSA) The `PSA` unit provides final purification (99.999% purity) by removing trace impurities (H_2O, N_2, CO_x). Fluid dynamics through the packed adsorbent bed are modeled using the **Ergun Equation**:

$$\frac{\Delta P}{L} = \frac{150 \mu u_s (1-\varepsilon)^2}{D_p^2 \varepsilon^3} + \frac{1.75 \rho_g u_s^2 (1-\varepsilon)}{D_p \varepsilon^3} \tag{45}$$

where ε is porosity and D_p is particle diameter. Mass balance is governed by a fixed **Recovery Rate** (R), with the unrecovered hydrogen and all impurities rejected to the `tail_gas` stream:

$$\dot{m}_{tail} = \dot{m}_{in} - \dot{m}_{product} \tag{46}$$

The unit operates in discrete cycles (Adsorption $\leftrightarrow$ Regeneration) defined by t_{cycle}, integrating power consumption for purge/vacuum operations.

Thermal Swing Adsorption (TSA) For temperature-driven purification, the `TSAUnit` uses dual packed beds where one absorbs moisture while the other regenerates. Pressure drop follows the same packed-bed mechanics as PSA (Eq. ??). The primary constraint is the **Regeneration Energy Balance**:

$$Q_{total} = \underbrace{m_{bed} C_{p,bed} \Delta T}_{Sensible} + \underbrace{m_{H_2O} \Delta H_{ads}}_{Desorption} + \underbrace{\dot{m}_{purge} C_{p,gas} \Delta T t_{cycle}}_{Purge\ Heat} \tag{47}$$

Heated purge gas (typically 250 °C) drives the endothermic desorption process, regenerating the bed capacity for the next cycle.

3.6.10 Storage Systems

The storage module allows selecting between computational speed and physical fidelity depending on the simulation timescale.

Vectorized Tank Arrays (Low-Fidelity) Designed for grid-scale capacity planning, the `TankArray` implementation is a **Zero-Dimensional Mass Balance** model optimized for high-performance computing (HPC). It utilizes **Contiguous Memory Arrays** (NumPy) and **SIMD operations** (via Numba) to update thousands of tank states in a single CPU cycle.

Tanks transition between discrete states ($\mathcal{S} \in \{\text{CHARGING}, \text{DISCHARGING}, \text{IDLE}\}$) based on threshold logic. The global state update is vectorized:

$$\mathbf{P}_{t+1} = \frac{\mathbf{m}_{t+1} RT}{\mathbf{V}} \quad \text{(ideal gas approximation)} \tag{48}$$

Dynamic Pressure Vessels (High-Fidelity) For compressor control loops and transient analysis, the `H2StorageTankEnhanced` acts as a **Dynamic Accumulator**. It solves the pressure derivative based on the Equation of State (EOS):

$$\frac{dP}{dt} = \frac{(\dot{m}_{in} - \dot{m}_{out}) RZT}{V} \tag{49}$$

This model captures transient pressure spikes during filling and rapid depressurization effects, essential for validating safety relief valve (PRV) logic.

3.6.11 Thermal Management & Heat Recovery

This section details the active heating/cooling units and passive recovery systems, distinguishing between parasitic loads (Boiler/Chiller) and efficiency enablers (Interchanger).

Active Heating (Electric Boiler) The `ElectricBoiler` functions as an **Isobaric Enthalpy Addition** device. Unlike simple temperature integrators, it determines the outlet state based on first-law efficiency (η_{th}):

$$h_{out} = h_{in} + \frac{P_{elec}\eta_{th}}{\dot{m}} \tag{50}$$

By solving the system in the enthalpy domain, the model automatically handles **Phase Transition** (Subcooled Liquid $\rightarrow$ Two-Phase $\rightarrow$ Superheated Steam). The outlet temperature is retrieved via the Equation of State inversion:

$$T_{out} = \text{EOS}^{-1}(P, h_{out}) \tag{51}$$

This eliminates the need for complex "if-then" logic to track latent heat zones.

Active Cooling (Chiller) The `Chiller` calculates cooling load based on a dual-mode physics core to support both real fluids (steam/water) and ideal gases (hydrogen/oxygen):

$$Q_{load} = \begin{cases} \dot{m}(h_{in} - h_{target}) & \text{Real Fluids (Captures Latent Heat)} \\ \dot{m}C_p(T_{in} - T_{target}) & \text{Ideal Gas Fallback} \end{cases} \tag{52}$$

Electrical consumption is derived from the **Coefficient of Performance (COP)**, representing the thermodynamic cost of heat rejection:

$$P_{elec} = \frac{|Q_{load}|}{COP} \tag{53}$$

Passive Recovery (Interchanger) The `Interchanger` is a counter-flow heat exchanger constrained by the **Second Law of Thermodynamics**. The actual heat transfer (Q_{trans}) is limited by the **Minimum Approach Temperature** (ΔT_{min}) at the pinch point:

$$Q_{trans} = \min(Q_{avail}, Q_{demand}) \tag{54}$$

where:

- $Q_{avail} = \dot{m}_h C_{p,h}(T_{h,in} - (T_{c,in} + \Delta T_{min}))$
- $Q_{demand} = \dot{m}_c C_{p,c}(T_{target} - T_{c,in})$

This ensures physical realism by preventing temperature crossovers, modeling the finite surface area of real recuperators.

Centralized Heat Management (Virtual Heat Bus) The `ThermalManager` acts as a plant-wide supervisor, abstracting individual heat streams into a unified energy pool. It maximizes thermal efficiency ($\eta_{recovery}$) by continuously matching aggregate supply with process demand:

$$Q_{utilized} = \min\left(\sum_i Q_{exhaust,i}, \sum_j Q_{sink,j}\right) \tag{55}$$

where sources $Q_{exhaust}$ include electrolyzer stack cooling and compressor intercoolers, and sinks Q_{sink} include feedwater pre-heating and SOEC steam generation. This virtualization decouples heat production from consumption, allowing for flexible "many-to-many" thermal integration without complex piping topology in the graph.

3.6.12 Water Management & Balance of Plant (BoP)

This section details the hydraulic control logic and thermodynamic consequences of the water recirculation loops.

Thermodynamic Pumping The `WaterPump` component extends standard hydraulic models by explicitly calculating the **Parasitic Heat Load** introduced by pump inefficiency. While the shaft power P_{shaft} is defined by head requirements:

$$P_{shaft} = \frac{\dot{m}v\Delta P}{\eta_{is}\eta_{mech}} \tag{56}$$

The thermodynamic model captures the enthalpy rise in the fluid due to viscous dissipation $(1 - \eta_{is})$. For liquid water, this results in a temperature rise ΔT that accumulates in closed loops:

$$\Delta T = \frac{\Delta H_{real}}{C_p} = \frac{v\Delta P}{\eta_{is}C_p} \tag{57}$$

This effect, often neglected in simple flowsheets, significantly impacts the cooling load on downstream heat exchangers.

Inventory Control (Makeup Mixer) The `MakeupMixer` functions as an **Active Control Node** rather than a passive junction. It enforces a "Demand-Driven Injection" interaction to maintain constant loop inventory:

$$\dot{m}_{makeup} = \max(0, \dot{m}_{target} - \dot{m}_{recycled}) \tag{58}$$

Thermodynamically, it solves the adiabatic mixing of cold makeup water (T_{makeup}) with the potentially hot recycled drain stream ($T_{recycled}$):

$$T_{out} = \frac{\dot{m}_{recycled}T_{recycled} + \dot{m}_{makeup}T_{makeup}}{\dot{m}_{target}} \tag{59}$$

The injection of cold makeup water acts as a stabilizing thermal sink, counteracting pump heat addition.

Closed-Loop Mass Accounting To validate the plant's environmental performance, the `WaterBalanceTracker` and `DrainRecorder` components implement rigorous mass conservation tracking. The `DrainRecorder` provides **Source Attribution**, distinguishing between liquid carryover from KODs versus aerosol capture in Coalescers.

System efficiency is quantified via the **Water Recovery Ratio (WRR)**:

$$WRR = \frac{\sum \dot{m}_{recovered}}{\sum \dot{m}_{consumed}} \tag{60}$$

This telemetry prevents "phantom mass generation" in the simulation and verifies that the closed-loop mass balance holds true over transient operation.

4 Computational Optimization

4.1 Look-Up Table Architecture

The `LUTManager` implements a "Generate Once, Cache Forever" strategy for thermodynamic properties.

4.1.1 Grid Specification

Based on the architecture documentation:

Table 3: LUT Grid Specification

Axis	Range	Points
Pressure	0.05 bar – 1000 bar	2000 (log spacing)
Temperature	273 K – 1200 K	2000 (linear)

The grid uses 2D (P, T) indexing with bilinear interpolation. The pressure axis uses geometric spacing (`np.geomspace`) to maintain resolution across four orders of magnitude.

Properties cached: Density (D), Enthalpy (H), Entropy (S), Heat Capacity (C_p), Compressibility (Z).

Fluids: H_2, O_2, H_2O, CO_2, CH_4, N_2.

4.1.2 Cache Hit/Miss Mechanism

- **Cache Hit**: Query within bounds uses bilinear interpolation (0.5 µs – 2 µs)
- **Cache Miss**: Out-of-bounds falls back to CoolProp (100 µs – 1000 µs), with logged warning

4.2 JIT-Compiled Numerical Kernels

The `numba_ops` module provides Numba-decorated functions for hot paths:

Listing 3: JIT Flash Calculation

```
@njit
def solve_rachford_rice_single_condensable(
    z_condensable: float,
    K_value: float
) -> float:
    if K_value >= 1.0:  # Superheated
        return 1.0
    if z_condensable <= K_value:  # Undersaturated
        return 1.0
    # Two-phase solution
    beta = (1.0 - z_condensable) / (1.0 - K_value)
    return np.clip(beta, 0.0, 1.0)
```

5 Integrated Control Architecture

The system implements a Split-Layer Control Architecture separating economic intent from physical outcomes.

5.1 Architecture Diagram

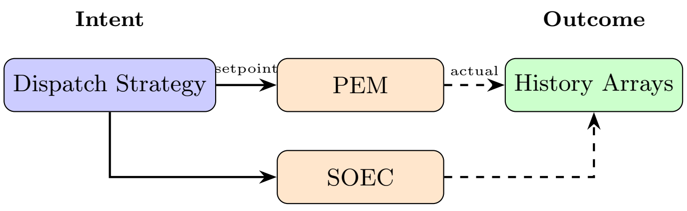

Figure 2: Split-Layer Control: Intent vs. Outcome

5.2 Arbitrage Decision Logic

The dispatch strategy compares selling electricity vs. producing hydrogen:

$$P_{\text{threshold}} = P_{\text{PPA}} + \frac{1000}{\eta_{\text{ref}}} \times P_{\text{H}_2} \tag{61}$$

When $P_{\text{spot}} > P_{\text{threshold}}$, selling power yields higher profit than hydrogen production.

6 Assumptions and Validity Range

6.1 Thermodynamic Assumptions

- **Ideal Gas Mixing**: Gas mixtures follow Kay's rule for pseudo-critical properties
- **Shomate Validity**: Equation ?? is valid for 273 K – 1500 K
- **Isothermal Storage**: Tanks assume constant temperature (no compression heating)
- **Single-Condensable VLE**: Flash calculations assume only H_2O condenses in H_2/O_2 carriers

6.2 Numerical Assumptions

- **LUT Bounds**: Pressure 0.05 bar–1000 bar, Temperature 273 K–1200 K
- **Bilinear Interpolation**: Assumes smooth property variation between grid points
- **Explicit Euler**: Timestep 1 min is sufficient for stability (CFL condition satisfied)

7 Performance Benchmarks

7.1 Baseline Simulation Parameters

Table 4: Standard Simulation Configuration

Parameter	Value
Timestep (Δt)	1 min (1/60 h)
Total Steps (annual)	525,600
Total Hours (annual)	8,760
Component Count	50+
LUT Grid Size	2000 $\times$ 2000

7.2 Speedup Summary

Table 5: Performance Optimization Results

Technique	Baseline	Speedup
LUT Property Lookup	CoolProp EOS	50–200×
JIT Flash Solver	Python iterative	100–500×
Pre-allocated Arrays	Dynamic lists	10–50×

Execution Times:

- JIT Flash calculation: 10 ns–50 ns per call
- LUT property lookup: 0.5 µs–2 µs per call
- Annual simulation: $<$ 5 min on standard hardware

8 Conclusion

The Dual-Path Hydrogen Production System v2.0 demonstrates that high-fidelity thermodynamic simulation can be reconciled with the performance requirements of annual techno-economic analysis.

Key Technical Achievements:

1. **50+ Component Physics Models** organized into 20 functional categories with strict lifecycle contracts
2. **Rigorous Governing Equations**: Shomate thermodynamics, Ergun packed bed pressure drop, Stokes-law cyclone separation, Rachford-Rice VLE
3. **LUT-based property lookups** on 2000 $\times$ 2000 (P, T) grids achieving 50–200× speedup over CoolProp
4. **JIT-compiled solvers** for flash equilibrium achieving 10 ns–50 ns per call
5. **Split-Layer Control Architecture** with Push (physics) and Pull (control) data flow patterns
6. **Real-World Data Ingestion**: Environment Manager loads historical price and wind availability time series

Annual simulations (525,600 timesteps) complete in under 5 min on standard hardware, enabling rapid scenario analysis for capacity planning, grid integration studies, and hydrogen market participation strategies.

Acknowledgments

The development team acknowledges the CoolProp library for thermodynamic property data, the Numba project for JIT compilation, and the NumPy project for numerical array operations.

References

[1] Bell, I.H., Wronski, J., Quoilin, S., Lemort, V. (2014). Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp. *Industrial & Engineering Chemistry Research*, 53(6), 2498–2508.

[2] Lam, S.K., Pitrou, A., Seibert, S. (2015). Numba: a LLVM-based Python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC.*

[3] Hauch, A. et al. (2020). Recent advances in solid oxide cell technology for electrolysis. *Science*, 370(6513).

[4] Rachford, H.H., Rice, J.D. (1952). Procedure for Use of Electronic Digital Computers in Calculating Flash Vaporization Hydrocarbon Equilibrium. *Journal of Petroleum Technology*, 4(10).

[5] Ergun, S. (1952). Fluid flow through packed columns. *Chemical Engineering Progress*, 48(2), 89–94.

[6] Hoffmann, A.C., Stein, L.E. (2008). *Gas Cyclones and Swirl Tubes: Principles, Design and Operation.* Springer, 2nd Edition.

A System Configuration Example

The following YAML snippet demonstrates how a PEM stack is declared in the v2.0 configuration format:

Listing 4: PEM System Configuration (YAML)

```yaml
pem_system:
  stacks:
    - component_id: "PEM-Stack-1"
      max_power_kw: 2500.0
      cells_per_stack: 100
      parallel_stacks: 4
  rectifiers:
    - component_id: "RT-1"
      max_power_kw: 2500.0
      system: PEM
  heat_exchangers:
    - component_id: "HX-1"
      max_heat_removal_kw: 500.0
      system: PEM
```

The `system` assignment enables context-dependent components to be assigned to specific production pathways (PEM=0, SOEC=1, ATR=2).