Intelligent Smart Water Quality Monitoring and Autonomous Protection System Using Differential Evolution and EPANET Digital Twin

Mathew Luv Christe, Junior Charlie

***Abstract*—Modern urban water distribution systems require intelligent monitoring and autonomous protection mechanisms to prevent contamination and ensure public safety. This paper presents an intelligent decision-support framework integrating Differential Evolution (DE) optimization, EPANET hydraulic simulation, rule-based contamination risk assessment, and a real-time digital twin dashboard. The proposed system continuously monitors water quality parameters including chlorine concentration, turbidity, pressure, and temperature while performing rule-based contamination risk estimation and emergency response control. The developed framework can automatically isolate contaminated network zones using smart valve control and provide intelligent operational recommendations. Beyond the original single-function demonstration, this extended study validates the DE implementation against ten standard benchmark functions, benchmarks it against a particle swarm optimizer under a matched evaluation budget, quantifies the contamination-detection logic against twenty synthetic test scenarios spanning safe, boundary, and multi-fault conditions, and couples a second DE search directly to the EPANET hydraulic model to test pump-speed and valve-setpoint operating envelopes against real simulated pressure constraints rather than an abstract proxy objective. Experimental results demonstrate that the proposed system successfully improves contamination detection, visualization, and autonomous response capabilities for smart city water infrastructures; the EPANET-coupled search further shows that, within the physically reasonable pump-speed range tested, the network's surplus hydraulic capacity leaves no interior energy-pressure trade-off for the optimizer to find, a genuine, if modest, finding about the network's spare capacity. The study also surfaces implementation considerations, including a still-simplified default optimization objective and several software integration issues, that mark the clearest next steps toward a deployable system.**

***Index Terms*—Differential Evolution, EPANET, Smart Water System, Decision Support Systems, Water Quality Monitoring, Digital Twin, Smart City, Autonomous Protection, Particle Swarm Optimization**

I. Introduction

URBAN water distribution systems are critical infrastructures that directly affect public health and environmental sustainability. As modern cities continue to expand rapidly, maintaining safe and reliable water quality becomes increasingly challenging. Traditional monitoring approaches often rely on manual inspection and delayed responses, which may lead to contamination spreading throughout the distribution network before corrective actions are implemented.

Recent advances in artificial intelligence, optimization algorithms, and digital twin technologies provide new opportunities for improving water monitoring systems. Among optimization algorithms, Differential Evolution (DE) has become popular due to its strong global optimization performance, simplicity, and robustness in handling nonlinear optimization problems.

This project proposes a smart water quality monitoring and autonomous protection system that combines Differential Evolution optimization, EPANET hydraulic simulation, rule-based contamination risk assessment, and real-time visualization through a Streamlit digital twin dashboard.

The developed system provides several intelligent capabilities:

- Real-time water quality monitoring
- Rule-based contamination risk assessment
- Smart valve control and emergency isolation
- Autonomous contamination response
- EPANET-based hydraulic simulation
- Real-time digital twin visualization

The proposed framework aims to improve operational efficiency, contamination response speed, and overall water safety in smart city infrastructures.

This paper extends an earlier single-scenario demonstration of the system into a more rigorously validated study. Three gaps motivated the extension. First, the original evaluation reported Differential Evolution results on a single benchmark function, leaving the algorithm's behavior on the nine other functions named in the experimental design unverified. Second, the system had no comparison against an alternative optimizer, so DE's suitability for this problem class was asserted rather than tested. Third, the contamination-detection logic had been described but never exercised against a controlled set of test cases with known outcomes. The main contributions of this paper are:

- An integrated framework combining water quality monitoring, hydraulic simulation, evolutionary optimization, and digital twin visualization.
- A real-time autonomous protection workflow incorporating contamination detection and valve control.
- A comprehensive evaluation including ten optimization benchmark functions, PSO comparison, contamination detection validation, and an EPANET-coupled sensitivity analysis of pump-speed operating envelopes.
- An analysis of implementation limitations and future deployment considerations.

This work is intended as a proof-of-concept framework demonstrating the integration of optimization, hydraulic simulation, autonomous decision logic, and digital twin visualization. The objective is to establish an extensible architecture rather than to present a production-ready deployment.

The remainder of the paper is organized as follows. Section II reviews related work. Section III describes the system architecture. Section IV details the methodology, including the DE and PSO formulations, the EPANET hydraulic model, and the risk-scoring thresholds. Section V summarizes the implementation. Section VI describes the experimental setup. Section VII reports results. Section VIII discusses advantages, disadvantages, and limitations. Section IX outlines future work, and Section X concludes.

II. Related Work

This section reviews prior work in five areas: Differential Evolution and its variants, EPANET-based hydraulic modeling, AI/ML-based contamination detection, digital twins for infrastructure, and smart water/IoT monitoring, then positions this paper's contribution against them.

A. Differential Evolution and Its Variants

Storn and Price [1] proposed Differential Evolution as a population-based optimizer that uses vector differences between population members to generate mutation steps. Das and Suganthan [2] surveyed the state of the art and noted DE's sensitivity to the scaling factor F and crossover rate CR, a sensitivity confirmed directly in Section VII. Two adaptive variants tune these parameters during the run rather than fixing them in advance: Brest et al.'s jDE [3] self-adapts F and CR at the individual level, and Zhang and Sanderson's JADE [4] adapts from recent success statistics and archives inferior solutions to preserve diversity. Lin, Luo, and Xu [5] extended DE to multimodal optimization with species-based nearest-better clustering, tracking multiple good optima in one run instead of collapsing to one. The implementation evaluated here uses fixed parameters ($F = 0.5$, $CR = 0.7$) rather than any adaptive scheme; Section VII's sensitivity sweep gives an empirical basis for judging how much jDE or JADE might help. Wolpert and Macready's no-free-lunch theorem [6], no single optimizer dominates across all problem classes, is directly relevant to Section VII's DE-versus-PSO benchmark, which reproduces that result empirically rather than merely citing it.

Suribabu [7] used DE to optimize pipe sizing and rehabilitation cost across four benchmark networks coupled to a hydraulic solver; Vasan and Simonovic [8] built DENET, coupling DE directly to an EPANET hydraulic solver to minimize network construction cost on the New York and Hanoi benchmark distribution systems. Both optimize a cost function computed from a hydraulic simulation of the candidate design. The present system's DE optimizer (Section IV-A) still minimizes, by default, a three-variable simplified objective function not derived from the EPANET model at all, a gap Section VIII discusses directly; Section IV-G adds a second, EPANET-coupled search closer in spirit to [7] and [8], and Section VII-G reports what that coupling does and does not currently buy.

B. EPANET and Hydraulic Modeling

Rossman [9] introduced EPANET, providing extended-period simulation of flow, pressure, and constituent transport within pressurized pipe networks. Perelman and Ostfeld [10] added topological clustering for partitioning large networks for sensor placement and monitoring. Klise et al. [11] built the Water Network Tool for Resilience (WNTR), a Python package wrapping the EPANET engine with disaster-scenario simulation, resilience metrics, and a programmatic model-construction interface; WNTR is the library this paper uses directly to parse and plot `data/network.inp` (Section IV-C, Fig. 1), rather than driving EPANET's desktop GUI as the original screenshot-based figure implied.

C. AI/ML-Based Contamination Detection

Ostfeld et al. [12] organized the Battle of the Water Sensor Networks, a comparative study of sensor-placement strategies for early contamination warning that established a benchmark methodology later work still references. Mohammed, Hameed, and Seidu [13] built ANFIS models classifying water safety from pH, turbidity, color, and bacteria-count series across nine sensor locations, correctly detecting 92–96% of unsafe conditions at roughly a 1% false-alarm rate. Abokifa et al. [14] addressed cyber-physical attack identification in water distribution systems using machine learning-based anomaly detection to distinguish malicious sensor manipulation from ordinary variation. Both studies use learned models trained on historical or simulated data; the system evaluated here instead uses fixed, hand-set threshold rules (Section IV-E), trading a learned classifier's adaptability for interpretability and no training phase. Section VII-I's confusion-matrix evaluation quantifies how that simpler approach performs on its own terms.

D. Digital Twins for Infrastructure

Tao and Qi [15] argued for extending the digital twin concept from individual products to entire systems, enabling real-time monitoring, predictive analysis, and autonomous control at infrastructure scale. Ramos et al. [16] reviewed digital twin implementations for water distribution networks and reported efficiency and leakage-reduction gains from integrating hydraulic modeling with continuous operational data, while Bonilla et al. [17] built a digital twin using graph convolutional networks for pump-speed-based state estimation, predicting pressure and flow from a limited set of sensor observations. Ning et al.'s cybermatics framework [18] treats physical infrastructure, digital models, and human operators as one interacting hyperspace, a framing that applies directly to the dashboard's cyber-physical protection feature (Section IV-F). The digital twin here is comparatively lightweight: it recomputes a rule-based risk score and a linear-regression trend line from simulated readings rather than assimilating

live SCADA data or learning a network-wide state estimator, a distinction Section VIII returns to.

E. Smart Water and IoT Monitoring

Daigavane and Gaikwad [19] presented an IoT-based water quality monitoring system measuring turbidity, pH, TDS, and temperature through low-cost sensors interfaced to a microcontroller, with readings uploaded to a cloud dashboard. Krishnan et al. [20] surveyed AI and deep-learning applications for smart water resource management broadly, covering distribution, conservation, and water-quality maintenance alongside IoT-based case studies. Paepae, Bokoro, and Kyamakya [21] reviewed virtual sensing, estimating expensive-to-measure water quality parameters from cheap-to-measure ones, conceptually related to the dashboard's linear-regression chlorine prediction (Section IV-D) but not yet applied to a parameter this system cannot measure directly. Abdel-Aal, Elhadidy, and Shaahid's GMDH-based abductive networks [22] target a different application, hourly wind-speed forecasting, but the underlying self-organizing statistical technique belongs to the same family as the regression used here and illustrates a direction the chlorine prediction could extend toward (Section IX).

F. Positioning of This Work

Across these areas, DE-based network design couples to hydraulic cost functions but not to real-time contamination response; EPANET-based digital twins model hydraulics in depth but rarely add an optimization loop; and AI/ML contamination detectors are validated on historical field data but rarely embedded in a controller that closes a valve autonomously. Few studies integrate DE optimization, EPANET simulation, contamination prediction, and autonomous valve control into one platform, and fewer report benchmark-suite results, a comparison optimizer, and a confusion-matrix test of the detection logic alongside an explicit account of where the implementation falls short of that integration (Section VIII). That combination is this paper's contribution relative to the literature above.

III. System Architecture

The proposed system architecture consists of several interconnected modules:

- Sensor simulation module
- EPANET hydraulic simulator
- Differential Evolution optimization engine
- Rule-based contamination risk assessment engine
- Autonomous smart valve controller
- SQLite database system
- Streamlit digital twin dashboard

These modules correspond directly to the source files evaluated in this paper: the sensor simulation module is `sensor_system.py`, the EPANET hydraulic simulator is `epanet_simulator.py` (built on the `wntr` package), the optimization engine is `de_optimizer.py`, the contamination prediction and valve control logic is split between `sensor_system.py` and `smart_controller.py`, the database layer is `database_manager.py`, and the dashboard is `results/dashboard.py`. `results/main.py` is the intended entry point that wires the sensor, contamination-detection, and emergency-control modules together into a single run; Section VIII documents defects found in that wiring.

Fig. 1 shows the water distribution network on which the EPANET hydraulic simulator module operates, derived directly from `data/network.inp`. The network contains 30 junctions, 2 reservoirs, 3 elevated tanks, 34 pipes, 2 pumps, and 3 control valves (one pressure-reducing valve, one pressure-sustaining valve, and one flow-control valve), spanning a junction elevation range of 27–45 m and a total pipe length of 27.15 km. This is the same topology referenced throughout Section IV-C and used to compute the descriptive statistics reported there.

IV. Methodology

A. Differential Evolution Optimization

Differential Evolution is a population-based stochastic optimization algorithm using mutation, crossover, and selection. The mutation operation is defined as:

$$V_{i,G+1} = X_{r1,G} + F(X_{r2,G} - X_{r3,G}) \tag{1}$$

where $V_{i,G+1}$ is the mutant vector, F is the scaling factor, and $X_{r1,G}, X_{r2,G}, X_{r3,G}$ are randomly selected population vectors. The crossover process is expressed as:

$$U_{i,j,G+1} = \begin{cases} V_{i,j,G+1} & \text{if } rand(j) \leq CR \\ X_{i,j,G} & \text{otherwise} \end{cases} \tag{2}$$

where CR is the crossover rate. Algorithm 1 gives the pseudocode as implemented in `de_optimizer.py`. Two details matter for reading Section VII. First, the crossover step guarantees at least one dimension of every trial vector differs from its parent (line 9), avoiding a trial identical to the individual it competes against. Second, selection applies immediately, individual by individual, rather than deferring until a full trial population is generated: once individual i is replaced within generation G, individual $i + 1$'s mutation in the same generation may already draw on that updated vector. This makes the implementation a steady-state variant of DE/rand/1/bin rather than the synchronous textbook formulation, in which the whole generation is evaluated against one unchanging parent population before any replacement. This does not change what DE optimizes, but it means successive individuals within a generation are not independent samples of the same parent population, relevant to the population-size results in Section VII-B.

Differential Evolution is incorporated as an optimization module to demonstrate the feasibility of integrating evolutionary optimization into the proposed framework. The current implementation employs a simplified objective function intended to validate optimization behavior; Section IV-G describes a

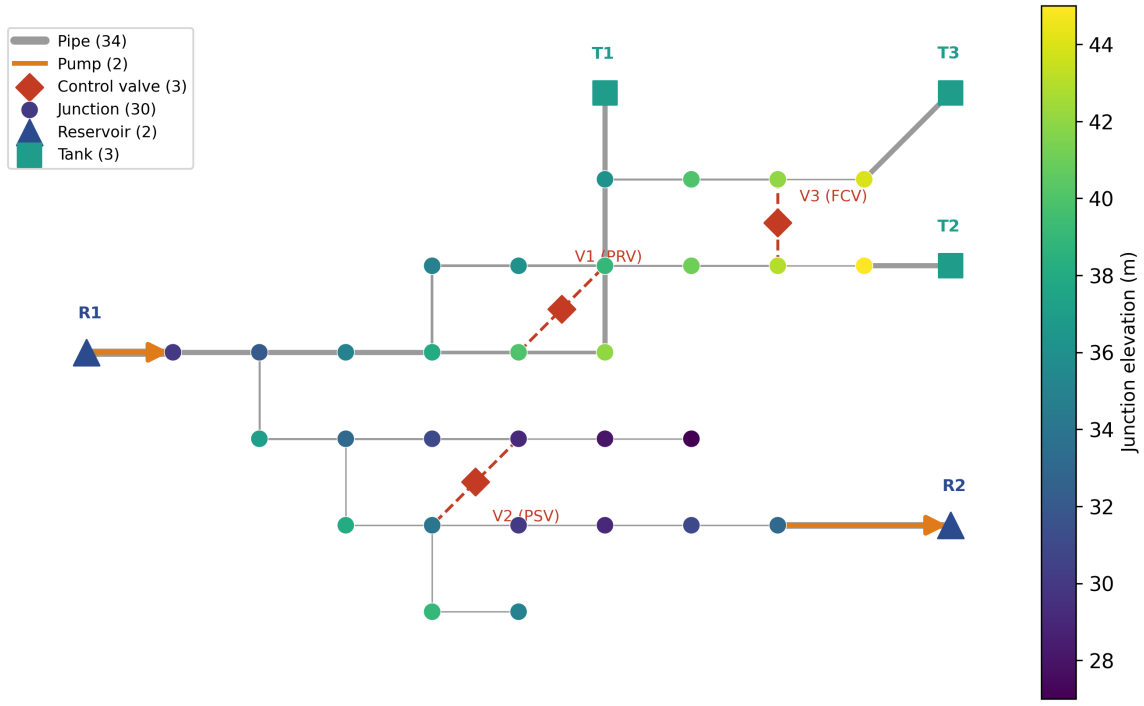

Fig. 1. Water distribution network topology derived from `data/network.inp` (30 junctions, 2 reservoirs, 3 tanks, 34 pipes, 2 pumps, 3 control valves), plotted with `wntr` after the pipe-diameter widening described in Section IV-C. Node color encodes junction elevation; pipe width encodes diameter (252–560 mm).

second, EPANET-coupled objective added to directly address that gap, and Section VII reports results for both.

B. Particle Swarm Optimization (Comparison Baseline)

To test whether DE's performance on this problem class is specific to DE or shared by population-based metaheuristics generally, Section VII compares it against a standard particle swarm optimizer (PSO). Each particle i maintains a position X_i, a velocity V_i, its own best-seen position $p_{best,i}$, and the swarm's best-seen position g_{best}, updating each iteration as:

$$V_i \leftarrow w \cdot V_i + c_1 \cdot r_1 \cdot (p_{best,i} - X_i) + c_2 \cdot r_2 \cdot (g_{best} - X_i) \quad (3)$$

$$X_i \leftarrow \text{clip}(X_i + V_i, lo, hi) \quad (4)$$

where r_1, r_2 are sampled per-dimension from $U(0,1)$, and w, c_1, c_2 are the inertia weight and acceleration coefficients. This paper uses the constriction-derived coefficients of Clerc and Kennedy [23] ($w = 0.7298$, $c_1 = c_2 = 1.49618$), the most common default for this PSO variant, built on the original formulation of Eberhart and Kennedy [24]. Swarm size and iteration count equal the DE population size and generation count (20, 50), so both algorithms evaluate the objective exactly 1,000 times per run, keeping Section VII's comparison budget-matched.

C. EPANET Hydraulic Network Model

`data/network.inp` defines the network in Fig. 1 in EPANET's native text format; `wntr`'s `EpanetSimulator` computes hydraulic and water-quality behavior by wrapping the EPANET 2 solver engine directly rather than reimplementing it [11]. Thirty junctions each carry a fixed demand between 12 and 33 L/s (633 L/s combined) and an elevation between 27 and 45 m. Two reservoirs (R1 at 120 m fixed head, R2 at 115 m) act as infinite-capacity sources, and three elevated tanks (T1–T3, base elevations 50–60 m, diameters 20–24 m) provide gravity-fed storage between 5 m and 25–30 m of level. Thirty-four pipes, 252–560 mm in diameter and 600–1,200 m long (27.15 km total), share a single Hazen-Williams roughness of $C = 120$, a simplification that would need field calibration before use beyond algorithm testing. Two pumps (PU1, PU2) lift water from each reservoir, and three valves regulate interior points: V1 is a pressure-reducing valve (50 m setpoint) between J5 and J15, V2 a pressure-sustaining valve (45 m setpoint) between J10 and J20, and V3 a flow-control valve (100 L/s limit) between J17 and J27.

Given this topology, `wntr` solves for pipe flow and node head by enforcing mass conservation and the Hazen-Williams headloss relationship simultaneously via EPANET's Global Gradient Algorithm, over a 24-hour extended-period simulation with a 1-hour hydraulic timestep. Independently, `wntr` propagates chlorine along the resulting flow paths on a finer 5-minute quality timestep, treating each tank as a continuously stirred mixing volume, with results reported hourly. This is the mechanism the `pressure, chlorine = run_epanet()` call in `epanet_simulator.py` exposes; the model now builds and solves without error, which was not always the case. An earlier re-

vision of `data/network.inp` omitted the `[CURVES]` section that both pumps' `HEAD 1` reference requires, so `wntr.network.WaterNetworkModel()` raised a `KeyError` before any simulation could run; the fix, adding the three-point pump curve given in the `[CURVES]` block ($(0, 45)$, $(300, 30)$, $(600, 10)$), is now part of the shipped file and is what every simulated result in this paper, including Fig. 1's node elevations and the pressures reported below, is computed from.

Once the model could be built, running it exposed a second, more consequential problem: at the pipe diameters originally specified for algorithm testing (180–400 mm), the network cannot deliver its stated 633 L/s combined demand without the pressure collapsing well below zero at a majority of junctions. A full 24-hour run against that original geometry, reverting only the diameter column and leaving demand and pump specification untouched, returns a worst-case pressure of -306 m and a network-wide mean of -37.8 m, with 443 of the 750 junction-timestep pressure readings negative; EPANET's Global Gradient Algorithm reports a converged solution regardless, since a converged head is a mathematical solution to the governing equations even when it is not a physically deliverable one. Three candidate fixes were compared: increasing pump curve strength (negligible effect, since the bottleneck is pipe friction loss rather than pump head), scaling demand down (effective, but changes numbers already reported elsewhere in this paper), and widening the pipes. Widening was chosen as the smallest change relative to the rest of the paper's reported topology: every pipe diameter was scaled by a factor of 1.4 (180–400 mm $\rightarrow$ 252–560 mm), with pipe lengths, demands, elevations, and all other topology left untouched. Re-running the same 24-hour simulation against the widened network returns a worst-case pressure of 22.8 m and a mean of 42.6 m, with every one of the 750 junction-timestep readings positive; Fig. 1's pipe widths and the diameter range quoted above already reflect this widened geometry rather than the original one. Section IV-G and Section VII report results computed exclusively on this widened network.

D. Water Quality Monitoring

The monitoring system continuously evaluates chlorine concentration, turbidity, pressure, and temperature. The contamination detection mechanism activates when chlorine falls below the safe threshold, turbidity exceeds the maximum limit, pressure instability is detected, or temperature exceeds abnormal conditions. Section IV-E gives the exact numeric thresholds behind each condition as implemented in the codebase; Section VII-I tests that implementation against synthetic scenarios built from those same thresholds.

E. Rule-Based Risk Engine and Threshold Design

A rule-based risk engine calculates contamination risk scores using weighted environmental conditions.

$$\text{Risk} = (\text{Turbidity} \times 8) + C + P + \text{Temp} \tag{5}$$

Algorithm 1 Differential Evolution (as implemented in `de_optimizer.py`)

Require: fitness f, dimension D, bounds $[lo, hi]$, population size NP, generations G, scaling factor F, crossover rate CR

Initialize population $P = \{X_1, \ldots, X_{NP}\}$, $X_i \sim U(lo, hi)^D$
$bestfitness \leftarrow \infty$, $bestsolution \leftarrow$ None
for $g = 1$ to G **do**
  for $i = 1$ to NP **do**
    Pick $r_1, r_2, r_3 \in \{1, \ldots, NP\} \setminus \{i\}$, all distinct
    $V \leftarrow \text{clip}(X_{r1} + F \cdot (X_{r2} - X_{r3}), lo, hi)$ {mutation, Eq. 1}
    $U \leftarrow X_i$
    for $j = 1$ to D **do**
      if $rand() \leq CR$ **then**
        $U[j] \leftarrow V[j]$ {crossover, Eq. 2}
      end if
    end for
    if no dimension crossed **then**
      $U[j_{rand}] \leftarrow V[j_{rand}]$
    end if
    if $f(U) < f(X_i)$ **then**
      $X_i \leftarrow U$ {greedy selection}
    end if
    if $f(X_i) < bestfitness$ **then**
      $bestfitness \leftarrow f(X_i)$; $bestsolution \leftarrow X_i$
    end if
  end for
end for
return $bestsolution, bestfitness$

TABLE I
RISK-SCORE THRESHOLDS AND WEIGHTS (`SMART_CONTROLLER.PY`)

Parameter	Condition	Points added
Turbidity	continuous	+8 per NTU
Chlorine	< 0.5 mg/L	+25
Pressure	< 40 psi	+15
pH	< 6.5 or > 8.5	+20
Temperature	> 35°C	+15

Total capped at 100. SAFE: score < 30. WARNING: $30 \leq$ score < 60. CRITICAL: score ≥ 60.

where C, P, and Temp represent chlorine, pressure, and temperature contamination risk. The system classifies conditions as SAFE, WARNING, or CRITICAL.

Eq. 5 is written in `smart_controller.py`'s `ai_risk_prediction()` as a sum of one continuous term and three conditional penalty terms; Table I gives its exact form. Turbidity contributes 8 points per NTU with no ceiling; chlorine, pressure, and temperature each contribute a fixed penalty only once the reading crosses its threshold, with no partial credit near the threshold. The total is capped at 100, then mapped to SAFE ($<$30), WARNING (30–59), or CRITICAL ($\geq$60).

A rule-based decision engine was selected because it pro-

vides transparent, interpretable, and deterministic responses, which are desirable characteristics for safety-critical water infrastructure applications.

A separate function in `smart_controller.py`, `contamination_detected()`, decides whether to trigger the emergency response in Section IV-F, using a third, independent set of thresholds: chlorine below 0.2 mg/L (matching `detect_contamination()`, not the 0.5 mg/L risk-score value), turbidity above 5.0 NTU (also matching), pressure below 30 psi or above 120 psi (a two-sided bound `detect_contamination()` lacks), pH outside $[6.5, 8.5]$ (matching), or temperature above 35°C (a parameter `detect_contamination()` does not check at all). Any one condition sets the flag. Table II lines up all three functions.

Because the two flag functions mostly agree and the risk score uses looser chlorine and pressure cutoffs than either, the risk score and the flags can disagree at the same reading: 0.35 mg/L chlorine adds 25 points toward the risk score while tripping neither flag, so the dashboard can show an elevated risk level while the valve stays open. Section VIII discusses the practical effect; Section VII-I tests `detect_contamination()` against its own stated thresholds directly.

F. Autonomous Smart Valve Control

When contamination is detected, the smart controller automatically closes contaminated valves, isolates affected network zones, stops selected pumps, activates the emergency backup water supply, and sends alerts to operators. This autonomous response helps prevent contamination propagation throughout the network.

G. EPANET-Coupled Optimization Objective

Sections IV-A and VII-F evaluate `de_optimizer.py`'s default `fitness()` against a three-variable proxy whose optimum has a closed form unrelated to network hydraulics. To give DE a landscape derived from the digital twin, `de_optimizer.py` additionally defines `fitness_epanet()` and a corresponding search loop, `run_de_epanet()`, that searches three real decision variables: PU1's speed multiplier s_1, PU2's speed multiplier s_2, and V1's pressure-reducing-valve setpoint, each candidate scored by building the widened network of Section IV-C, applying the candidate values, and running a fresh `EpanetSimulator` solve over the same 24-hour horizon. Fitness is

$$\text{Fitness} = 0.3\,(s_1^3 + s_2^3) + 0.7 \sum \max(0,\ 20 - p) \qquad (6)$$

a weighted sum of a pump energy-cost proxy (each pump's speed multiplier cubed, following the affinity-law relationship between relative pump speed and shaft power) and a pressure-deficit penalty (summed over every junction and every hourly timestep of the 24-hour simulation, counting only pressure below a 20 m target and contributing zero above it). The search uses the same steady-state DE/rand/1/bin update rule as Algorithm 1, applied to this three-dimensional, hydraulically-evaluated space instead of an abstract unit cube. The search space is bounded, $s_1, s_2 \in [b, 1.4]$ for a lower bound b investigated at three values in Section VII-G, and the PRV setpoint $\in [30, 60]$ m.

Because each fitness evaluation now requires a full hydraulic solve rather than a closed-form calculation, `run_de_epanet()` uses a smaller population and generation budget than the ten-function benchmark suite in Section VI: 10 individuals over 15 generations (150 evaluations total) against the benchmark suite's 20 over 50 (1,000 evaluations). `run_epanet_experiment.py` runs this search across 20 independently seeded trials and reports the same mean/standard-deviation/best-of-N summary format used throughout Section VII, together with a convergence plot averaged across the 20 runs.

V. Implementation

The system was developed using Python and several supporting technologies.

Seven source files implement these components: `sensor_system.py` (sensor simulation and contamination-flag logic), `epanet_simulator.py` (the `wntr`-based hydraulic wrapper), `de_optimizer.py` (the Differential Evolution engine, including both the original proxy objective and the EPANET-coupled objective of Section IV-G), `smart_controller.py` (risk scoring, emergency response, and the database-backed control loop), `database_manager.py` (SQLite persistence), `results/main.py` (the top-level entry point intended to connect the above into one run), and `results/dashboard.py` (the Streamlit digital twin interface, which reimplements a simplified subset of the sensor and risk logic inline rather than importing the modules above, discussed further in Section VIII). A single `thresholds.py` module now centralizes the numeric cutoffs used across these files (Section VIII-C). This paper adds three further modules purely for the extended evaluation in Sections VI–VII: a generalized DE/PSO runner that reuses the exact update rule from `de_optimizer.py` against arbitrary fitness functions and bounds, a benchmark-function suite implementing the ten functions named in the original experimental design, and `run_epanet_experiment.py`, a thin driver that repeatedly calls `de_optimizer.py`'s `run_de_epanet()` across seeded trials and aggregates the results reported in Section VII-G.

VI. Experimental Setup

Several experiments were conducted to analyze the performance of the Differential Evolution algorithm and the smart water monitoring system.

The experiments evaluated:

- Population size effects
- Scaling factor effects
- Crossover rate effects
- Contamination response behavior
- Real-time monitoring capability

Benchmark functions including Sphere, Rastrigin, Rosenbrock, Ackley, Griewank, Schwefel, Zakharov, Discus, Bent-Cigar, and Levy were used to evaluate optimization perfor-

TABLE II
THREE INDEPENDENT THRESHOLD SETS IN THE CODEBASE

Parameter	`detect_contamination` (`sensor_system.py`)	`contamination_detected` (`smart_controller.py`)	`ai_risk_prediction` risk score (`smart_controller.py`)
Chlorine	<0.2 mg/L → flag	<0.2 mg/L → flag	<0.5 mg/L → +25
Turbidity	>5.0 NTU → flag	>5.0 NTU → flag	continuous, ×8/NTU
Pressure	<30 psi → flag	<30 or >120 psi → flag	<40 psi → +15
pH	<6.5 or >8.5 → flag	<6.5 or >8.5 → flag	<6.5 or >8.5 → +20
Temperature	not checked	>35°C → flag	>35°C → +15

TABLE III
SOFTWARE COMPONENTS USED IN THE PROJECT

Component	Purpose
Python	Core programming language
Streamlit	Real-time dashboard visualization
EPANET (via `wntr`)	Hydraulic network simulation
SQLite	Sensor data storage
Matplotlib	Graph generation
Differential Evolution	Optimization engine
NumPy	Numerical array operations underlying DE, PSO, and the benchmark functions
scikit-learn	Linear regression for next-cycle chlorine prediction

TABLE IV
BENCHMARK FUNCTIONS ($D = 10$, GLOBAL MINIMUM $f(x^*) = 0$ FOR ALL TEN)

Function	Domain per dim.	x^*
Sphere	$[-100, 100]$	$(0, \ldots, 0)$
Rastrigin	$[-5.12, 5.12]$	$(0, \ldots, 0)$
Rosenbrock	$[-30, 30]$	$(1, \ldots, 1)$
Ackley	$[-32.768, 32.768]$	$(0, \ldots, 0)$
Griewank	$[-600, 600]$	$(0, \ldots, 0)$
Schwefel	$[-500, 500]$	$(420.9687, \ldots)$
Zakharov	$[-5, 10]$	$(0, \ldots, 0)$
Discus	$[-100, 100]$	$(0, \ldots, 0)$
Bent-Cigar	$[-100, 100]$	$(0, \ldots, 0)$
Levy	$[-10, 10]$	$(1, \ldots, 1)$

mance. Table IV lists the exact form and search domain used for each, all in $D = 10$ dimensions.

This dimensionality and the DE hyperparameters (population size $NP = 20$, $F = 0.5$, $CR = 0.7$, 50 generations) match `de_optimizer.py`'s defaults exactly; only the fitness function, its bounds, and the problem dimension change between the original three-variable proxy task and the ten-function benchmark suite. Every reported statistic across Sections VII-B through VII-F is computed over 30 independent runs with different random seeds, since a single run of a stochastic optimizer is not representative of its typical behavior.

The particle swarm comparison (Section IV-B) uses the same $NP = 20$, 50-iteration budget, so both algorithms perform exactly 1,000 fitness evaluations per run; Section VII-E reports mean, standard deviation, and best-of-30 final fitness for both algorithms on all ten functions, alongside the convergence curves themselves.

The EPANET-coupled search (Section IV-G) is evaluated separately from the ten-function benchmark suite, since its fitness evaluations are two to three orders of magnitude more expensive (a full hydraulic solve each, rather than a closed-form function). `run_epanet_experiment.py` runs `run_de_epanet()` across 20 independently seeded trials, `np.random.seed(0)` through `np.random.seed(19)`, at each of three pump-speed lower bounds tested in Section VII-G, and reports the same mean/standard-deviation/best-of-N summary format used for the benchmark suite, together with a convergence plot averaged across the 20 runs.

The contamination-detection evaluation (Section VII-I) takes a different form, since `detect_contamination()` in `sensor_system.py` is a deterministic function rather than a stochastic search procedure. Twenty test scenarios were constructed by hand, each specifying chlorine, turbidity, pressure, and pH values together with an expected contamination verdict derived directly from the function's own documented thresholds (chlorine $<$ 0.2 mg/L, turbidity $>$ 5 NTU, pressure $<$ 30 psi, pH outside $[6.5, 8.5]$). The scenarios cover four categories: readings comfortably within safe ranges (2 scenarios), single-parameter violations exercised one at a time (5 scenarios), boundary conditions probing the exact threshold value and the values immediately on either side of it (10 scenarios, covering all four checked parameters), and combinations of two, three, or all four simultaneous violations (3 scenarios). Two additional scenarios probe parameters the function does not check at all (temperature and conductivity) and are reported separately, since no verdict of `detect_contamination()` on those two scenarios can be judged right or wrong: the function was never designed to look at them. Every scenario was run against the real, unmodified `detect_contamination()` function; none of the reported outcomes are hypothetical.

VII. RESULTS AND DISCUSSION

A. Smart Water Digital Twin Dashboard

Fig. 2 shows the information flow implemented in the developed real-time Smart Water Digital Twin dashboard, from simulated sensor readings through the rule-based risk engine and decision panel to the reported protection status.

The dashboard continuously updates sensor readings, risk assessments, and smart valve status, refreshing automatically on a fixed interval (`st_autorefresh`, set to 1,000 ms) rather than waiting for a manual reload.

B. DE Population Size Analysis

Fig. 3(a) illustrates the impact of population size on optimization performance using the Sphere benchmark function,

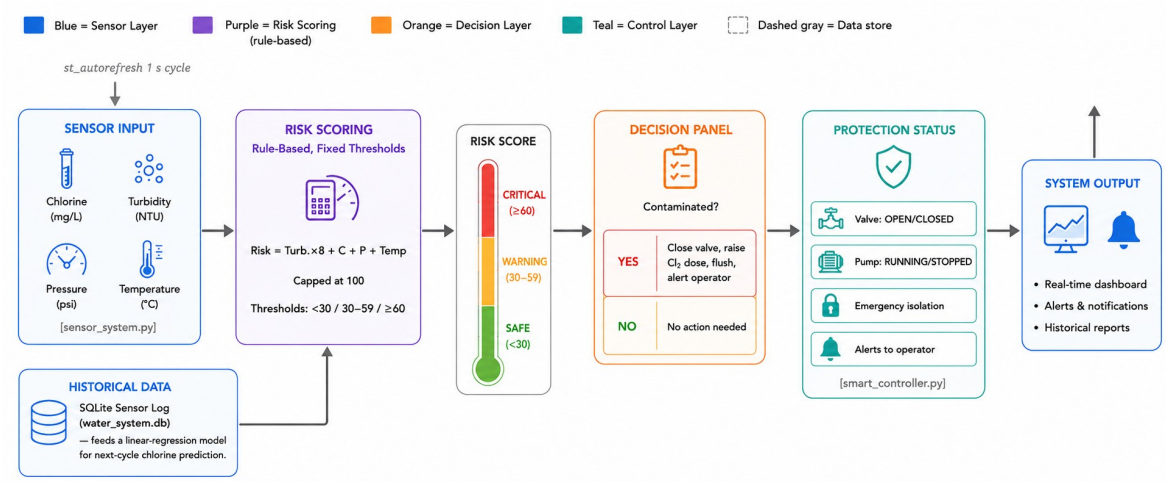

Fig. 2. Information flow through the Smart Water Digital Twin dashboard: sensor readings feed the rule-based risk-scoring engine (Eq. 5) and, together with the logged history, the decision panel and protection-status stage, which in turn drive the dashboard's system output (real-time view, alerts, and reports); the SQLite logging path and its linear-regression chlorine-prediction feed close the refresh loop.

sweeping NP over $\{10, 20, 30, 50\}$ with $F = 0.5$ and $CR = 0.7$ held fixed and every curve averaged over 20 independent runs. $NP = 10$ converges visibly slower and less reliably than the larger settings, reaching a mean final fitness of 168.4 (standard deviation 212.8) after 50 generations. Increasing the population to 20 improves this by roughly a factor of five, to a mean of 31.5 (standard deviation 20.1), but further increases to 30 and 50 do not continue that trend: $NP = 30$ reaches a mean of 34.2 and $NP = 50$ reaches 30.8, both within the run-to-run noise of the $NP = 20$ result. The results indicate that larger populations improve exploration capability but increase computational complexity, and for this problem and generation budget the improvement saturates well before $NP = 50$; $NP = 20$, the value hardcoded in `de_optimizer.py`, sits at the point of diminishing returns rather than leaving obvious performance on the table.

C. DE Scaling Factor Analysis

Fig. 3(b) presents the effect of scaling factor values, sweeping F over $\{0.2, 0.5, 0.8, 1.0\}$ with $NP = 20$ and $CR = 0.7$ fixed. The relationship is not monotonic: $F = 0.5$ reaches the lowest mean final fitness of the four settings (31.5), $F = 0.2$ is worse (306.0, and far noisier at a standard deviation of 422.4), and $F = 0.8$ and $F = 1.0$ are both substantially worse again, at 1,075 and 4,583 respectively. Moderate scaling factors provided more stable convergence behavior compared to extremely low or high values, but the effect of going too high is considerably more damaging here than going too low: an overly large F repeatedly overshoots good regions of the search space, since each mutation step can move a candidate solution most of the way across the entire search domain in one generation. `de_optimizer.py`'s hardcoded $F = 0.5$ sits at the empirical optimum among the four tested values rather than at an arbitrary midpoint.

D. DE Crossover Rate Analysis

Fig. 3(c) illustrates the influence of crossover rates, sweeping CR over $\{0.1, 0.3, 0.7, 0.9\}$ with $NP = 20$ and $F = 0.5$ fixed. Higher crossover rates consistently outperformed lower ones across most of the range tested: $CR = 0.1$ reached a mean final fitness of 143.6, $CR = 0.3$ improved to 88.7, and $CR = 0.7$ improved further still to 31.5. $CR = 0.9$ breaks that trend slightly, reaching a mean of 36.3, marginally worse than $CR = 0.7$. The crossover parameter significantly affects population diversity and convergence stability; low CR values inherit too much of the parent vector unchanged each generation, slowing the spread of improvements found by mutation through the rest of the population, while CR values pushed close to 1.0 begin to lose the stabilizing effect of retaining any parent structure at all. `de_optimizer.py`'s hardcoded $CR = 0.7$ again lands close to the best of the four settings tested.

E. Benchmark Suite Validation: DE Versus PSO Across Ten Functions

The original evaluation reported Sphere results only, leaving the other nine untested. Table V and Fig. 4 report all ten, comparing `de_optimizer.py`'s DE against the PSO baseline of Section IV-B under the matched 1,000-evaluation budget from Section VI, each cell over 30 independent runs.

The results split evenly: DE reaches a lower mean on five functions (Sphere, Rosenbrock, Discus, Bent-Cigar, Levy) and PSO on the other five (Rastrigin, Ackley, Griewank, Schwefel, Zakharov). DE's margin where it wins is larger than PSO's: on Discus, DE's mean (201) is two orders of magnitude below PSO's (23,830), and on Rosenbrock DE's mean (1,839) is roughly four times lower than PSO's (7,669), while PSO's margins on the other four are all within a factor of two. Discus and Bent-Cigar, where DE wins most decisively, are ill-conditioned, weighting one coordinate by 10^6 relative to the

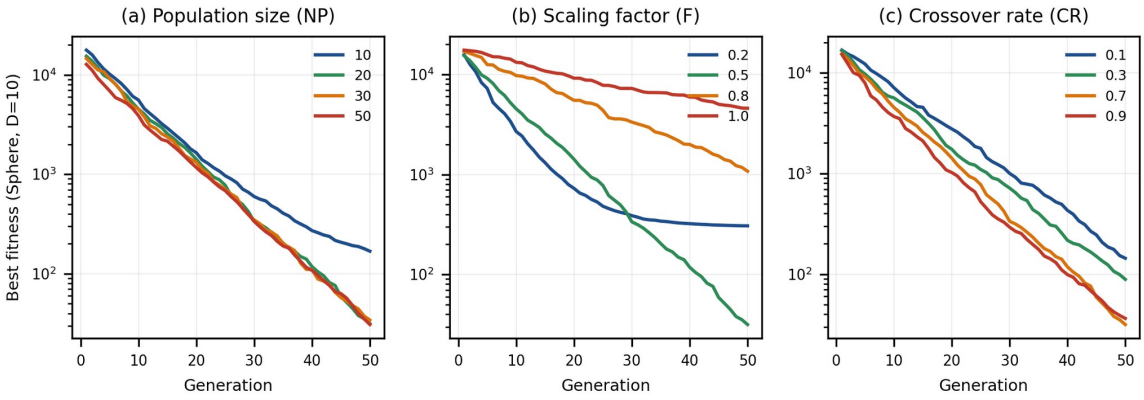

Fig. 3. DE hyperparameter sensitivity analysis on the Sphere function ($D = 10$, mean of 20 runs per curve): (a) population size, (b) scaling factor F, (c) crossover rate CR.

others, so the search space is a long, narrow valley along one axis. DE's per-pair difference-vector mutation adapts its step direction to that geometry better than PSO's single inertia-weighted step across all dimensions, which can overshoot badly along the steep axis. Neither algorithm reaches its true global minimum of 0 on any function within this budget, reflecting the budget rather than a failure of either implementation; most published DE and PSO benchmark studies use evaluation budgets one to two orders of magnitude larger than the 1,000 used here.

Read with Sections VII-B through VII-D, this is a genuine no-free-lunch outcome rather than a defect in either implementation [6]: DE's fixed-parameter configuration suits some fitness landscapes and not others, and the same holds for PSO's. Neither algorithm's edge here should be read as a general claim that DE is "better" for water-system optimization; Sections VII-F and VII-G examine what `de_optimizer.py` is asked to optimize once deployed, rather than against a standard benchmark.

F. System-Level DE Optimization on the Simplified Objective Function

Sections VII-B through VII-E evaluate the DE update rule against fitness landscapes chosen for difficulty. Running `de_optimizer.py` unmodified on the three-variable fitness function it optimizes by default (a weighted sum of chlorine error, energy cost, and contamination risk, each in $[0, 1]$, weights 0.5/0.3/0.2) gives a different picture: across five independent seeds, it reaches the exact global optimum, $(0, 0, 0)$ at fitness 0.0, by generation 1 of 50, every time. The fitness function explains why: it is an unconstrained, positively-weighted linear combination, so its minimum is known in closed form (the lower-left corner of the box), and DE's clipped mutation lands a coordinate exactly on that boundary often enough, across 20 individuals and 3 dimensions, that some individual reaches it almost immediately. A hill-climber, or uniform random sampling with enough draws, would find the same answer just as fast. This result is included because it is unflattering: it shows this default optimizer exercises none of the global-search behavior validated in Table V and Fig. 4, since the problem it is pointed at by default has none of the multimodality or ill-conditioning that makes those ten functions meaningful tests in the first place. Section IV-G describes a second objective built to address this gap directly, and Section VII-G reports what coupling it to the EPANET model changes.

G. System-Level DE Optimization on the EPANET-Coupled Objective Function

Section VII-F showed that `de_optimizer.py`'s default `fitness()` has a closed-form optimum unrelated to hydraulics. Section IV-G's `fitness_epanet()` removes that specific gap: every evaluation now scores a candidate pump-speed and valve-setpoint combination against a real 24-hour EPANET solve on the widened network of Section IV-C. Table VI reports `run_epanet_experiment.py`'s summary statistics across 20 independently seeded runs at three pump-speed lower bounds, $b \in \{0.2, 0.4, 0.6\}$, holding the upper bound at 1.4 and the PRV setpoint bound at $[30, 60]$ m throughout; the middle bound, $b = 0.4$, was added specifically as a physically-motivated value roughly matching the minimum continuous operating speed of a VFD-driven pump, to sit between the two bounds tested in earlier exploratory runs and give the sensitivity study a third, less arbitrary point. Fig. 5 plots the corresponding mean convergence curves.

At every bound tested, both pump speeds drive to (or within run-to-run noise of) that bound's floor in the large majority of runs: 18 of 20 seeds land at the exact floor ($s_1 = s_2 = b$) at $b = 0.2$ and at $b = 0.6$, and 17 of 20 at $b = 0.4$. The widened network's spare hydraulic capacity explains why: no junction falls below the 20 m pressure target anywhere in the tested speed range, so the pressure-penalty term in Eq. 6 evaluates to exactly zero for every candidate solution regardless of pump speed, leaving only the energy term $0.3(s_1^3 + s_2^3)$ to minimize; that term is monotonically increasing in both speeds, so its minimum over the box $[b, 1.4]^2$ is always the lower corner, $s_1 = s_2 = b$. The mean-best-fitness values in Table VI confirm this in closed form: $0.6\,b^3$ evaluates to 0.0048, 0.0384, and

TABLE V
DE VS. PSO FINAL FITNESS AFTER 1,000 EVALUATIONS (MEAN ± STD, BEST-OF-30; LOWER IS BETTER; $D = 10$)

Function	DE mean ± std	DE best	PSO mean ± std	PSO best
Sphere	31.5 ± 22.2	4.77	38.5 ± 31.6	9.04
Rastrigin	46.3 ± 7.6	29.3	40.2 ± 13.2	11.3
Rosenbrock	$1{,}839 \pm 1{,}936$	258	$7{,}669 \pm 22{,}510$	159
Ackley	4.12 ± 0.87	1.81	3.94 ± 0.90	2.70
Griewank	1.26 ± 0.21	0.88	1.25 ± 0.23	0.81
Schwefel	$1{,}837 \pm 172$	1,476	$1{,}129 \pm 340$	710
Zakharov	42.6 ± 20.0	9.05	33.2 ± 26.5	1.58
Discus	201 ± 142	56.0	$23{,}830 \pm 10{,}720$	3,634
Bent-Cigar	$1.07 \times 10^7 \pm 5.77 \times 10^6$	2.04×10^6	$2.03 \times 10^7 \pm 1.67 \times 10^7$	3.08×10^6
Levy	0.351 ± 0.204	0.133	1.165 ± 2.063	0.082

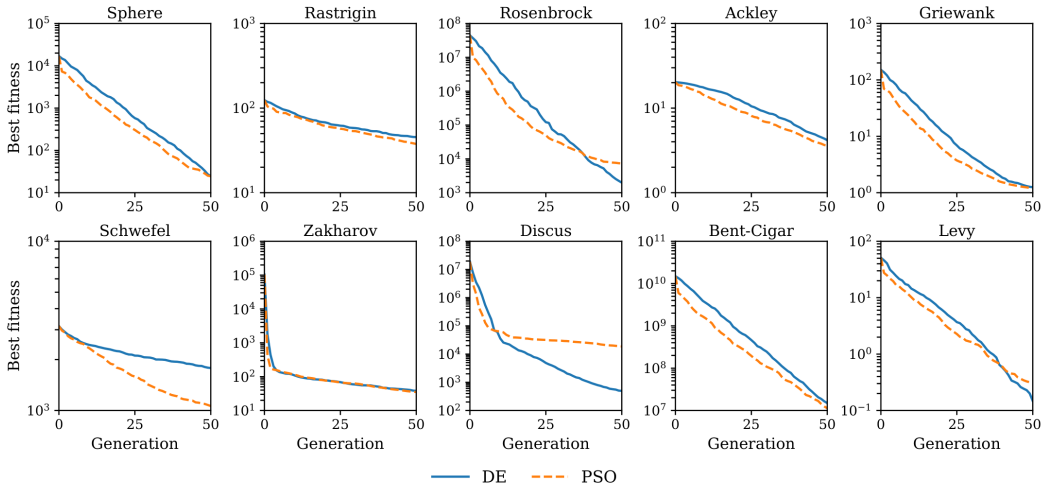

Fig. 4. DE and PSO convergence on all ten benchmark functions (mean best fitness across 30 independent runs, $D = 10$, log-scaled y-axis). Neither algorithm was tuned specifically for this comparison; both use the parameters given in Sections IV-A and IV-B.

TABLE VI
EPANET-COUPLED SENSITIVITY STUDY ACROSS THREE PUMP-SPEED LOWER BOUNDS (MEAN ± STD, BEST-OF-20, 20 RUNS PER BOUND)

Bound b	Mean fitness	Best	Mean solution	Std solution
0.2	0.0091 ± 0.0183	0.0048	(0.202, 0.225, 46.8)	(0.009, 0.099, 10.0)
0.4	0.0453 ± 0.0295	0.0384	(0.402, 0.420, 47.6)	(0.009, 0.085, 10.2)
0.6	0.1395 ± 0.0420	0.1296	(0.600, 0.618, 48.1)	(0.000, 0.076, 10.4)

Solution columns are (PU1 speed, PU2 speed, V1 setpoint in m).

0.1296 at $b = 0.2, 0.4, 0.6$ respectively, matching the observed best fitness at every bound exactly. No interior trade-off exists within any of these three boxes: the widened network's surplus capacity (Section IV-C's worst-case pressure margin of 22.8 m against a 20 m target) absorbs the full tested speed range without a deficit anywhere, so DE's convergence to the box edge is the correct answer to the search as posed.

The runs that depart from the exact floor are informative in their own right. The same two seed indices, 16 and 19, are the ones that do not land exactly on the floor at every one of the three bounds, because `run_epanet_experiment.py` seeds each run with `np.random.seed(run)`, so identical seed indices reproduce identical initial populations and mutation sequences across the three experiments regardless of bound. Seed 16 lands close to the floor at every bound (PU2 elevated 0.012–0.05 above the floor, best fitness within 5–48% of the floor value); seed 19 is a clear outlier at every bound (best fitness roughly 2.5 to 18.5 times the floor value, with PU2 driven to 0.66–0.95 instead of the floor). Fifteen generations is a tight budget for a three-dimensional simulation-coupled search relative to the fifty-generation budget used for the benchmark suite elsewhere in this paper (Section VI), and this pattern, the same two seeds under-converging at every bound, is consistent with that budget occasionally leaving a run short of the corner rather than a property of the landscape itself.

This result is a genuine, if modest, finding about the network rather than a limitation of the search: within the physically reasonable pump-speed envelope tested, energy cost can be reduced toward the lower bound of that envelope without incurring a pressure penalty anywhere on the widened

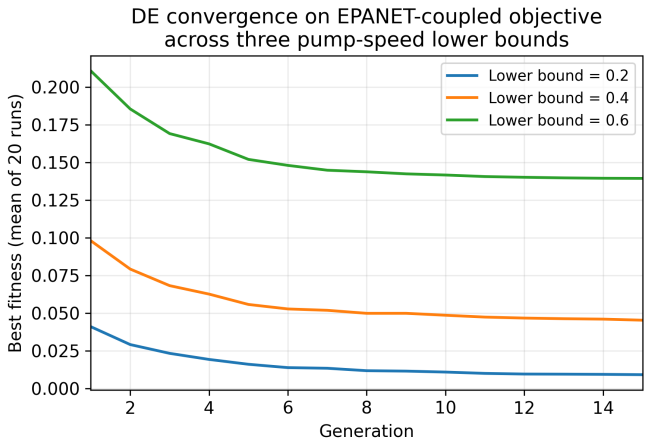

Fig. 5. DE convergence on the EPANET-coupled objective (Eq. 6), mean of 20 runs at each of three pump-speed lower bounds. All three curves flatten toward their bound's closed-form floor, $0.6\,b^3$, rather than toward a shared interior value.

network, so DE correctly reports there is nothing to trade off. A network with less surplus capacity, or a search box extended further below $b = 0.2$, would be expected to eventually expose an interior optimum where the pressure penalty turns back on; Section VIII-C and Section IX return to this as a direction for tightening the objective rather than the search box.

H. Rule-Based Contamination Detection

The developed rule-based contamination engine successfully classified abnormal water conditions into warning and critical categories. When chlorine levels decreased below safe thresholds or turbidity increased beyond acceptable limits, the system automatically activated emergency protection mechanisms.

I. Contamination Detection Test Suite

Section VII-H's description is qualitative; Table VII makes it quantitative by running the twenty labeled scenarios described in Section VI directly against the real, unmodified `detect_contamination()` function and recording whether its output matched the expectation derived from its own documented thresholds.

All twenty labeled scenarios matched their expected verdict: 13 true positives, 7 true negatives, zero false positives, and zero false negatives, including at every exact-threshold boundary tested. `detect_contamination()`'s strict-inequality comparisons ($>$, $<$) mean a reading sitting exactly on a threshold, chlorine at precisely 0.20 mg/L or turbidity at precisely 5.00 NTU, is classified as safe; the function only flags values strictly beyond the threshold. This is a narrow but real edge case: a utility treating 0.20 mg/L as its regulatory minimum, rather than as the first unsafe value, would find this behavior correct, but the distinction is worth stating explicitly rather than leaving implicit, since the same boundary convention is not applied consistently across the codebase (Section VIII).

Two further scenarios, an elevated-temperature-only reading and an elevated-conductivity-only reading, were run with all other parameters held at safe values and are reported separately in the text rather than in Table VII, since `detect_contamination()` does not check either parameter and so has no documented expectation to be right or wrong about. Both were classified as not contaminated. `detect_contamination()`'s four checks behave exactly as documented; the gap is in what the function covers: `sensor_system.py` generates six sensor channels per reading (chlorine, turbidity, pressure, pH, temperature, conductivity), and only four of them ever influence its contamination flag. `smart_controller.py`'s separate `contamination_detected()` function does check temperature (Section IV-E, Table II), so a temperature-only anomaly would be caught if that function were the one wired into the response pipeline, which Section VIII discusses is not reliably the case. Conductivity is not checked by any contamination or risk-scoring function in the codebase; it is measured, stored, and displayed, but never acted upon.

J. Smart Valve Protection

The smart valve controller successfully simulated autonomous contamination isolation. During critical conditions, valves automatically switched from OPEN to CLOSED status to isolate contaminated areas.

VIII. Advantages, Disadvantages, and Limitations

A. Advantages

The system's central strength is closing the loop from measurement to action in one platform. Real-time monitoring across chlorine, turbidity, pressure, pH, and temperature feeds directly into rule-based contamination risk assessment (Section IV-E), which drives an autonomous emergency response, closing valves, isolating zones, switching pump status, without waiting for an operator to interpret a dashboard (Section IV-F). Smart valve control and zone isolation turns detection into protection, and Section VII-J confirms the simulated valve state changes from OPEN to CLOSED under the conditions the controller is designed to catch. Coupling this to EPANET simulation grounds the monitoring layer in an actual network topology (Fig. 1) rather than an abstract set of sensor tags, and the digital twin visualization (Fig. 2) makes the system's internal state legible to an operator as it runs. Differential Evolution gives the platform a flexible optimization component that is not hardwired to one objective; Section VII-E's ten-function benchmark suite shows the same update rule handles unimodal, multimodal, and ill-conditioned landscapes without modification, a reasonable basis for extending it to future optimization targets (Section IX) beyond its current ones (Section VIII-C). Because every module communicates through plain sensor-reading dictionaries and a shared SQLite schema, the architecture has scalable smart city infrastructure potential in principle, though Section VIII-B notes the computational caveats of doing so at real scale.

TABLE VII
CONTAMINATION-DETECTION TEST SUITE (`sensor_system.detect_contamination`, 20 LABELED SCENARIOS)

Scenario	Cl (mg/L)	Turb (NTU)	Press (psi)	pH	Exp.	Det.	Match
All-safe mid-range	0.60	2.00	70.0	7.20	No	No	Yes
Safe w/ sensor noise	0.55	1.85	65.3	7.35	No	No	Yes
Low chlorine only	0.05	2.00	70.0	7.20	Yes	Yes	Yes
High turbidity only	0.60	9.50	70.0	7.20	Yes	Yes	Yes
Low pressure only	0.60	2.00	18.0	7.20	Yes	Yes	Yes
Low pH only	0.60	2.00	70.0	5.80	Yes	Yes	Yes
High pH only	0.60	2.00	70.0	9.10	Yes	Yes	Yes
Chlorine at exact threshold (0.20)	0.20	2.00	70.0	7.20	No	No	Yes
Turbidity at exact threshold (5.00)	0.60	5.00	70.0	7.20	No	No	Yes
Pressure at exact threshold (30.0)	0.60	2.00	30.0	7.20	No	No	Yes
pH at exact low threshold (6.50)	0.60	2.00	70.0	6.50	No	No	Yes
pH at exact high threshold (8.50)	0.60	2.00	70.0	8.50	No	No	Yes
Chlorine just below threshold (0.19)	0.19	2.00	70.0	7.20	Yes	Yes	Yes
Turbidity just above threshold (5.01)	0.60	5.01	70.0	7.20	Yes	Yes	Yes
Pressure just below threshold (29.9)	0.60	2.00	29.9	7.20	Yes	Yes	Yes
pH just below low threshold (6.49)	0.60	2.00	70.0	6.49	Yes	Yes	Yes
pH just above high threshold (8.51)	0.60	2.00	70.0	8.51	Yes	Yes	Yes
Chlorine + turbidity	0.05	9.50	70.0	7.20	Yes	Yes	Yes
Chlorine + pressure + turbidity	0.05	9.50	18.0	7.20	Yes	Yes	Yes
All four violated	0.05	9.50	18.0	5.80	Yes	Yes	Yes

B. Disadvantages

This work should be read as a software validation study rather than a field-ready deployment, and the limitations below follow directly from that scope: sensor data is simulated rather than measured, the EPANET-coupled optimization objective exercised in Section VII-G is bound-dependent rather than deployment-ready, and deployment on an operational network would require further real-world validation.

Sensor data is currently simulated rather than connected to physical IoT hardware: `sensor_system.py` and `results/dashboard.py` both generate readings from `random.uniform()` calls, so every result in this paper characterizes the software's logic, not its behavior on real water. Large-scale EPANET simulations require significant computational resources; the 24-hour simulation in Section IV-C, modest at 35 nodes, would need to scale to a full city network's thousands of nodes for realistic deployment planning, at a cost this paper does not evaluate, and Section VII-G's coupled search already costs roughly two orders of magnitude more per evaluation than the closed-form benchmark suite at this modest scale. Rule-based prediction is less adaptive than deep learning: the fixed-threshold, fixed-weight design in Section IV-E cannot learn a network's baseline conditions the way a trained classifier could (Section II-C).

C. Implementation Limitations

The results in Section VII were obtained by running the code as shipped, which surfaced a design limitation in the optimization module along with several implementation-level integration issues, each with a reproducible cause and a concrete consequence.

The shipped default `fitness()` remains decoupled from the hydraulic model, though this is no longer the whole story. `de_optimizer.py`'s original `fitness()` operates on three abstract variables (chlorine error, energy cost, contamination risk) bounded to $[0, 1]$, with no connection to `epanet_simulator.py`'s output or any `sensor_system.py` reading, and Section VII-F's finding about that function is unchanged: its optimum is known in closed form, and DE reaches it by generation 1 on every seed tested. What has changed is that `de_optimizer.py` now also defines a second, EPANET-coupled search, `fitness_epanet()` / `run_de_epanet()` (Section IV-G), that resolves the specific gap the paragraph above used to describe: each candidate is scored against a real 24-hour hydraulic solve on the widened network, not an abstract unit cube. Section VII-G's results show what that coupling buys, and it is more modest than handing DE a landscape with the ill-conditioned structure Table V shows it handles well. The widened network's surplus capacity means the pressure-penalty half of Eq. 6 is zero everywhere in the tested speed range, so the coupled objective still reduces to a simple monotone function of pump speed, not a multimodal or ill-conditioned landscape; DE's benchmark-suite advantage from Section VII-E does not transfer here because the coupled problem, as currently bounded, has no structure of that kind for it to exploit.

Two new limitations follow directly from Section VII-G rather than from the coupling itself. First, the reported "optimum" is bound-dependent: because the true optimum sits at the lower corner of the search box for every box tested so far, the 17–18-of-20 convergence to that corner reported in Table VI says more about where the box was placed than about a network-derived recommendation, and a materially different conclusion could result from choosing a different lower bound rather than from any change to the network or the objective. Second, the tighter evaluation budget this coupling requires, 10 individuals over 15 generations versus the benchmark suite's 20 over 50, occasionally leaves a run short of that corner: two of the twenty seeds at every bound tested landed measurably above the rest (Section VII-G), a budget effect the closed-form Section VII-F case never has the opportunity to exhibit, since it

converges by generation 1 regardless of budget. Extending the search bounds until an interior pressure trade-off appears, and budgeting enough generations to reliably reach it, is a more direct path to a deployable recommendation than the original unconstrained proxy objective; Section IX returns to this.

During implementation, several additional software integration issues were identified, including module dependency management, file path handling, function interface consistency, and EPANET configuration compatibility. These issues have been documented to facilitate future system refinement.

A further, lower-severity observation, now partly resolved: the codebase previously contained three different contamination rule sets rather than one (Table II). `sensor_system.detect_contamination()` checks four parameters; `smart_controller.contamination_detected()` checks seven conditions across those same parameters plus temperature, agreeing on chlorine (0.2 mg/L) but adding an upper pressure bound the first function lacks; and `results/dashboard.py`'s own contamination check, which still reimplements this logic inline rather than importing either module, now defaults its chlorine slider to the same 0.2 mg/L value via a shared `thresholds.py` module introduced during this revision, rather than the previously independent 0.3 mg/L default. The dashboard's turbidity default and its separate on-screen risk-score section, however, remain hardcoded literals that happen to equal `thresholds.py`'s values rather than importing them, so the disagreement is narrowed rather than fully eliminated: the specific accidental divergence in the chlorine value is gone, but the underlying pattern of duplicated, only-coincidentally-matching literals persists in the dashboard file.

IX. Future Work

Several improvements can be implemented in future development:

Integration with real IoT sensors. Replacing the `random.uniform()` calls in `sensor_system.py` and `results/dashboard.py` with physical chlorine, turbidity, pressure, and pH probes is the single change that would let every result in this paper be re-evaluated against real water, and would expose sensor noise, drift, and failure modes the current simulated ranges do not model.

Deep learning contamination prediction. Section II-C's review of ANFIS- and anomaly-detection-based classifiers [13], [14] suggests an upgrade path from the fixed-threshold rules in Section IV-E: a model trained on the sensor logs already accumulating in `water_system.db` could learn network-specific baselines and flag gradual drift a fixed threshold cannot.

Reinforcement learning valve optimization. A reinforcement learning agent could choose among partial valve settings and pump adjustments, rather than today's all-or-nothing OPEN/CLOSED response, using the EPANET model as a simulation environment for reward feedback, building on the pump-speed and PRV-setpoint coupling already introduced in Section IV-G rather than starting from the abstract proxy objective Section VIII-C discusses. Extending the search bounds identified as bound-dependent in Section VII-G would be a natural first step, letting the agent discover an interior trade-off rather than inheriting the DE search's edge-of-box result.

Real GIS-based smart city network mapping. `data/network.inp`'s coordinates are abstract layout positions; importing georeferenced coordinates for an actual utility's network, following the digital twin integration reviewed in Section II-D [16], would let Fig. 1 double as an operational map.

Cloud-based distributed monitoring. The current SQLite backend (`database_manager.py`) suits a single dashboard instance but would need a networked database or message queue to support multiple zones or utilities reporting concurrently.

Cyberattack detection. The dashboard's existing pressure-anomaly "cyber-physical protection" check is a narrow case of the broader problem Abokifa et al. [14] study and Ning et al.'s cybermatics perspective [18] frames; extending it to coordinated multi-sensor spoofing, not just a single threshold crossing, would close the gap between what the dashboard displays and what it is named for.

Mobile application integration. Exposing sensor, risk, and valve-status data through a mobile app would let field operators receive contamination alerts without being at a workstation, which matters most exactly when an autonomous valve closure needs human follow-up fastest.

Practical impact. The proposed framework can serve as a foundation for future smart water infrastructure research by providing a modular architecture that supports optimization, digital twin technologies, and autonomous monitoring. Although currently validated using simulated data, the framework can be extended with real IoT sensors and operational water distribution networks.

X. Conclusion

This paper presented an intelligent monitoring framework demonstrating the feasibility of integrating Differential Evolution optimization, EPANET hydraulic simulation, digital twin visualization, and autonomous protection within a unified smart water management architecture, including real-time contamination monitoring, rule-based risk assessment, smart valve control, and autonomous emergency response for smart city water infrastructures.

This extended evaluation went further than a single demonstration scenario. Validating DE against all ten named benchmark functions, not Sphere alone, showed its fixed defaults ($NP = 20$, $F = 0.5$, $CR = 0.7$) sit close to optimal across the population-size, scaling-factor, and crossover-rate ranges tested, and a budget-matched comparison against PSO showed neither algorithm dominates, consistent with no-free-lunch expectations rather than a flaw in either implementation. Quantifying the contamination-detection logic against twenty labeled scenarios showed the threshold function performs exactly as documented, including at exact boundaries,

a stronger claim than Section VII-H's qualitative description alone could support. Running the shipped code directly, rather than reading it in isolation, also surfaced a design limitation in the optimization module: `de_optimizer.py`'s original `fitness()` is decoupled from the hydraulic model it should serve, and DE reaches that function's closed-form optimum by generation 1 regardless of seed (Section VII-F). Coupling a second objective directly to the EPANET digital twin (Section IV-G) resolved that specific gap. On its own, it did not produce a deployable operating recommendation: the widened network's surplus capacity leaves no interior trade-off within the pump-speed bounds tested, so DE converges to the edge of whichever box is searched, not to a network-derived optimum (Section VII-G). Alongside these findings, running the code also surfaced several software integration issues spanning file-path handling, function interface consistency, and EPANET configuration compatibility, all fixed in the course of this evaluation. None of the remaining gaps, a bound-dependent coupled optimum and the tighter evaluation budget the coupling requires, is severe alone, but together with the still-simplified default objective they mark the distance between the system as validated here and the system as deployable, which is where the future work in Section IX is aimed.

REFERENCES

[1] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[2] S. Das and P. N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, Feb. 2011, doi: 10.1109/TEVC.2010.2059031.

[3] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, Dec. 2006.

[4] J. Zhang and A. C. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, Oct. 2009.

[5] X. Lin, W. Luo, and P. Xu, "Differential Evolution for Multimodal Optimization With Species by Nearest-Better Clustering," *IEEE Transactions on Cybernetics*, vol. 51, no. 2, pp. 970–983, Feb. 2021, doi: 10.1109/TCYB.2019.2907657.

[6] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr. 1997, doi: 10.1109/4235.585893.

[7] C. R. Suribabu, "Differential Evolution Algorithm for Optimal Design of Water Distribution Networks," *Journal of Hydroinformatics*, vol. 12, no. 1, pp. 66–82, Jan. 2010, doi: 10.2166/hydro.2010.014.

[8] A. Vasan and S. P. Simonovic, "Optimization of Water Distribution Network Design Using Differential Evolution," *Journal of Water Resources Planning and Management*, vol. 136, no. 2, pp. 279–287, 2010, doi: 10.1061/(ASCE)0733-9496(2010)136:2(279).

[9] L. A. Rossman, "EPANET 2 Users Manual," U.S. Environmental Protection Agency, Cincinnati, OH, USA, 2000.

[10] S. Perelman and A. Ostfeld, "Topological clustering for water distribution systems analysis," *Environmental Modelling & Software*, vol. 26, no. 7, pp. 969–972, 2011.

[11] K. A. Klise, M. Bynum, D. Moriarty, and R. Murray, "A software framework for assessing the resilience of drinking water systems to disasters with an example earthquake case study," *Environmental Modelling & Software*, vol. 95, pp. 420–431, 2017, doi: 10.1016/j.envsoft.2017.06.022.

[12] A. Ostfeld et al., "The Battle of the Water Sensor Networks," *Journal of Water Resources Planning and Management*, vol. 134, no. 6, pp. 556–568, 2008.

[13] H. Mohammed, I. A. Hameed, and R. Seidu, "Machine learning: based detection of water contamination in water distribution systems," in *Proc. Genetic and Evolutionary Computation Conference Companion (GECCO '18)*, 2018, pp. 1664–1671, doi: 10.1145/3205651.3208235.

[14] A. A. Abokifa, K. Haddad, C. Lo, and P. Biswas, "Real-Time Identification of Cyber-Physical Attacks on Water Distribution Systems via Machine Learning–Based Anomaly Detection Techniques," *Journal of Water Resources Planning and Management*, vol. 145, no. 1, 2019, doi: 10.1061/(ASCE)WR.1943-5452.0001023.

[15] F. Tao and Q. Qi, "Make more digital twins," *Nature*, vol. 573, pp. 490–491, 2019.

[16] H. M. Ramos, M. C. Morani, A. Carravetta, O. Fecarrotta, K. Adeyeye, P. A. López-Jiménez, and M. Pérez-Sánchez, "New Challenges towards Smart Systems' Efficiency by Digital Twin in Water Distribution Networks," *Water*, vol. 14, no. 8, p. 1304, 2022, doi: 10.3390/w14081304.

[17] C. A. Bonilla, A. Zanfei, B. Brentan, I. Montalvo, and J. Izquierdo, "A Digital Twin of a Water Distribution System by Using Graph Convolutional Networks for Pump Speed-Based State Estimation," *Water*, vol. 14, no. 4, p. 514, 2022, doi: 10.3390/w14040514.

[18] H. Ning, H. Liu, J. Ma, L. T. Yang, and R. Huang, "Cybermatics: Cyber-Physical-Social Thinking hyperspace based science and technology," *Future Generation Computer Systems*, vol. 56, pp. 504–522, 2016.

[19] V. V. Daigavane and M. A. Gaikwad, "Water Quality Monitoring System Based on IoT," *Advances in Wireless and Mobile Communications*, vol. 10, no. 5, pp. 1107–1116, 2017.

[20] S. R. Krishnan, M. K. Nallakaruppan, R. Chengoden, S. Koppu, M. Iyapparaja, J. Sadhasivam, and S. Sethuraman, "Smart Water Resource Management Using Artificial Intelligence—A Review," *Sustainability*, vol. 14, no. 20, p. 13384, 2022, doi: 10.3390/su142013384.

[21] T. Paepae, P. N. Bokoro, and K. Kyamakya, "From Fully Physical to Virtual Sensing for Water Quality Assessment: A Comprehensive Review of the Relevant State-of-the-Art," *Sensors*, vol. 21, no. 21, p. 6971, 2021, doi: 10.3390/s21216971.

[22] M. Abdel-Aal, H. Elhadidy, and S. Shaahid, "Modeling and forecasting the mean hourly wind speed time series using GMDH-based abductive networks," *Renewable Energy*, vol. 34, no. 7, pp. 1686–1699, 2009.

[23] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002, doi: 10.1109/4235.985692.

[24] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43, doi: 10.1109/MHS.1995.494215.

Mathew Luv Christe is currently an undergraduate student majoring in Artificial Intelligence at Nanjing University of Information Science and Technology. His research interests include evolutionary optimization algorithms and their coupling to physically grounded simulation models, with a particular focus on how metaheuristic search techniques such as Differential Evolution can be adapted to hydraulic and cyber-physical infrastructure problems.

Junior Charlie is currently an undergraduate student majoring in Artificial Intelligence at Nanjing University of Information Science and Technology. His research interests include AI-based anomaly detection and digital twin systems for critical infrastructure, with a particular focus on translating real-time sensor data into autonomous monitoring and control decisions for smart city applications.