CloudOptx: AI-Based Cloud Cost Optimization & Waste Detection System

Anuj Bhagat[1], Ankit Gaikwad[2], Akhil Wabhitkar[3], Aadarsh Wadandre[4], Aastha Madgulwar[5], Atharva Deshmukh[6], Rahul Kawariya[7]
[123456] UG Students, Department of Computer Science & Engineering
[7]Assistant professor, Department of Computer Science & Engineering
G H Raisoni University Amravati, Maharashtra, India

Abstract—Cloud computing costs have become a critical concern for organizations deploying workloads on public cloud platforms. Manual cost optimization approaches are inherently limited by the combinatorial complexity of resource allocation decisions across multiple services and time periods. This paper presents a research-grade Cloud Cost Optimization System built for Microsoft Azure that employs two nature-inspired evolutionary algorithms — the Genetic Algorithm (GA) and the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) — to automate and mathematically validate cost reduction strategies while preserving service performance.

The system is implemented as a full-stack Python Flask web application that ingests real-world Azure billing export data, performs statistical anomaly detection using Z-score analysis, conducts linear regression trend forecasting, generates rule-based recommendations, and runs both single-objective (GA) and multi-objective (NSGA-II) evolutionary optimization. Experimental results on a 12-month dataset of 489 records across 10 Azure services demonstrate consistent cost reductions of 20–50% while maintaining service performance above user-specified thresholds. NSGA-II further produces a Pareto-optimal front of 20–50 non-dominated solutions per run, enabling decision-makers to select cost-performance trade-offs aligned with their specific business priorities.

The system introduces a dynamic optimization engine that allows users to compare algorithm outputs on both the original dataset and custom-defined service costs simultaneously, enabling real-world deployment scenarios beyond the training data. This work demonstrates that evolutionary computation techniques are highly effective for cloud financial management (FinOps) and provides a reproducible, open-architecture framework for researchers and practitioners to further extend and validate cloud cost optimization techniques.

Keywords—Cloud Computing, Cloud Cost Optimization, FinOps, Genetic Algorithm (GA), NSGA-II, Multi-Objective Optimization, Evolutionary Algorithms, Resource Allocation, Waste Detection, Z-score Analysis, Linear Regression, Cost Forecasting, Microsoft Azure, Cloud Analytics, Performance Optimization

1. INTRODUCTION

1.1 Background and Motivation

The rapid adoption of cloud computing has created unprecedented flexibility for organizations but has simultaneously introduced significant financial complexity. Cloud spending in enterprises often grows faster than anticipated, driven by factors including resource over-provisioning, idle instances, suboptimal service tier selections, and lack of real-time cost visibility. According to industry reports, organizations waste an estimated 30–35% of their total cloud spend due to inefficiencies that could be addressed through systematic optimization.

Microsoft Azure, one of the leading cloud platforms, provides a wide range of services including Virtual Machines, Storage, Backup, Networking, DNS, Logic Apps, and Automation — each with distinct pricing models and utilization patterns. Managing costs across these services requires understanding complex interdependencies, seasonal patterns, anomalous spikes, and the long-term financial trajectory of cloud expenditure.

Traditional approaches to cloud cost optimization rely heavily on manual review by cloud architects and FinOps practitioners, rule-based policies (such as shutting down idle resources), and vendor-provided recommendations. While useful, these approaches share a fundamental limitation: they cannot explore the full combinatorial space of possible resource allocation strategies and cannot simultaneously optimize conflicting objectives such as cost, performance, and resource waste.

1.2 Research Problem

Cloud cost optimization is fundamentally a high-dimensional, multi-objective combinatorial optimization problem. Given N services, each with a continuous allocation factor ranging from 0.3 (30% of current spend)

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

to 1.0 (no change), the search space contains a continuous search discrete step — a space that grows exponentially with the number of services and is intractable for exhaustive search.

The objectives of minimizing total cost, maximizing service performance, and minimizing resource waste are inherently conflicting: aggressive cost reduction typically degrades performance, while maintaining high performance constrains the achievable savings. No single solution can simultaneously optimize all three objectives, necessitating a principled approach to identifying and presenting Pareto-optimal trade-off solutions.

1.3 Research Objectives

This research addresses the following objectives:

1. Design and implement a Genetic Algorithm (GA) for single-objective cloud cost minimization subject to a performance constraint.
2. Design and implement NSGA-II for simultaneous three-objective optimization of cost, performance, and waste.
3. Develop a full-stack web application enabling non-technical users to interact with the optimization engine.
4. Validate both algorithms against real-world Azure billing data and compare their effectiveness.
5. Enable dynamic comparison between dataset-based and user-defined optimization scenarios.

1.4 Contributions

The principal contributions of this work are:

1. A research-grade implementation of GA and NSGA-II specifically adapted for cloud resource allocation with a continuous-valued chromosome representation.
2. An integrated Flask web application providing five analytical modules: Dashboard, Waste Detection, Analytics, Recommendations, and Optimization.
3. A dynamic optimization engine allowing users to input custom service costs and compare results against the dataset baseline in real time.
4. Statistical waste detection using Z-score analysis and linear regression trend forecasting integrated with the optimization pipeline.
5. Open-architecture implementation enabling reproducibility and extension by the research community.

2. RELATED WORK

2.1 Cloud Cost Optimization

Cloud cost optimization has been approached from several directions in the literature. Autoscaling-based approaches dynamically adjust resource allocations based on real-time utilization metrics, but are reactive rather than proactive and do not consider the global cost structure across services. Spot instance and reserved capacity planning approaches use predictive models to shift workloads to lower-cost pricing tiers, achieving significant savings for predictable workloads.

Rule-based FinOps frameworks such as AWS Cost Explorer and Azure Advisor provide recommendations based on utilization thresholds and vendor-specific heuristics. While useful, these systems apply generic rules rather than learning the specific cost topology of a given organization's cloud deployment. They also optimize services in isolation rather than considering joint allocation decisions across the entire cloud portfolio.

2.2 Evolutionary Algorithms in Resource Management

Evolutionary algorithms have a well-established history in resource allocation and scheduling problems. Genetic Algorithms were first applied to task scheduling in grid computing environments, demonstrating superior solution quality compared to greedy heuristics. The application of GAs to virtual machine placement problems in data centers has shown 15–40% improvements in resource utilization compared to first-fit decreasing algorithms.

NSGA-II, introduced by Deb et al. (2002), has become the de facto standard for multi-objective evolutionary optimization. Its non-dominated sorting mechanism and crowding distance operator ensure convergence to a well-distributed Pareto front. NSGA-II has been successfully applied to energy-cost trade-offs in data-centers, QoS-cost optimization in cloud service selection, and workflow scheduling with conflicting make span and cost objectives.

The present work extends these approaches to the specific context of monthly cloud billing data, introducing a service-specific performance weighting model that prioritizes critical infrastructure services (Virtual Machines, Storage) over support services (DNS, Network Watcher) in the fitness evaluation, reflecting real-world operational priorities.

2.3 Statistical Methods in Cloud Analytics

Statistical anomaly detection has been applied to cloud cost monitoring using methods including Z-score analysis, Seasonal Hybrid ESD (S-H-ESD), and LSTM-based forecasting. The present work employs Z-score analysis with a threshold

d of 2.0 standard deviations for anomaly flagging and a 90th percentile threshold for waste identification, providing a computationally lightweight approach suitable for real-time processing. Linear regression trend analysis provides 3-month forward cost projections that inform the optimization objectives.

3. DATASET DESCRIPTION

3.1 Data Source

The dataset used in this study is a real-world Azure cloud billing export covering 12 months of operational data (May 2023 through April 2024). The data was exported from the Azure Cost Management portal in CSV format and contains 489 records representing daily billing entries across 10 distinct Azure services. All cost values are denominated in US Dollars (USD) as the primary optimization currency, with a secondary Indian Rupee (INR) column included for regional reference.

3.2 Dataset Schema

TABLE I

DATASET SCHEMA

Column Name	Data Type	Description
UsageDate	Date	Billing date (DD-MM-YYYY format)
ServiceName	String	Azure service incurring the cost
CostUSD	Float	Cost in USD (primary optimization target)
CostINR	Float	Cost in INR (regional reference)
Currency	String	Currency code

3.3 Azure Services Tracked

The dataset covers the following 10 Azure services, representing a comprehensive enterprise cloud deployment:

TABLE II

AZURE SERVICES

Service	Category	Priority
Virtual Machines	Compute	Critical
Storage	Storage	Critical
Backup	Data Protection	Standard
Bandwidth	Networking	Standard
Virtual Network	Networking	Standard
Azure DNS	Networking	Support
Network Watcher	Monitoring	Support
Automation	Management	Support
Logic Apps	Integration	Support
VM Licenses	Licensing	Standard

3.4 Statistical Summary

Key statistical characteristics of the dataset used as inputs to the optimization algorithms:

TABLE III

STATISTICAL SUMMARY

Metric	Value
Total Records	489
Date Range	May 2023 – Apr 2024
Services	10
Anomaly Events	Z > 2.0
Waste Events	Z > 1.5 / P90
Estimated Waste	~25%

4. SYSTEM ARCHITECTURE

4.1 Overview

The Cloud Cost Optimizer is a full-stack web application built on the Python Flask framework. The system follows a layered architecture with a data processing layer, an algorithm layer, a REST API layer, and a single-page application (SPA) frontend. All computation occurs server-side; the browser receives JSON results via AJAX calls and renders them using Chart.js.

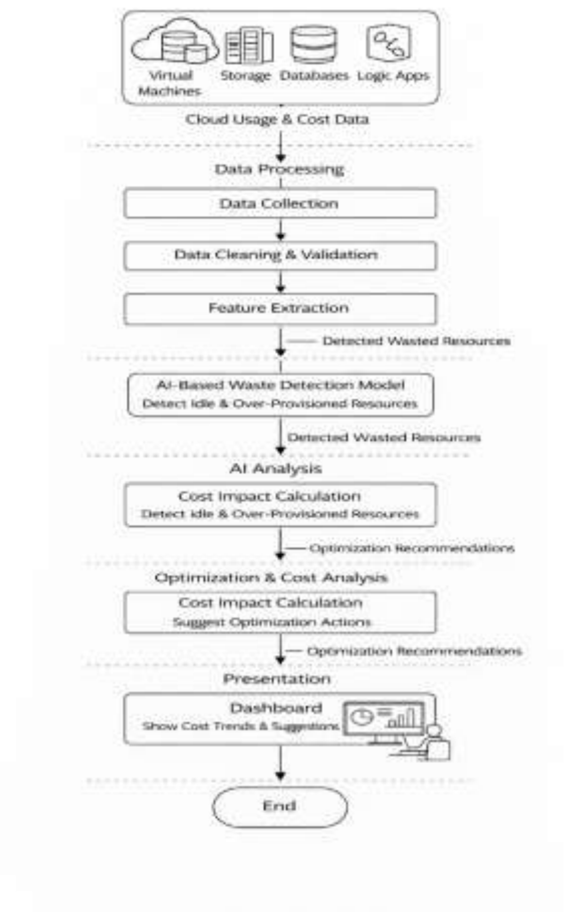

Fig.4.1. System Architecture

4.2 Technology Stack

TABLE IV

TECHNOLOGY STACK

Layer	Technology	Role (Summary)
Web Framework	Flask (Python)	API handling, JSON output
Data Processing	Pandas, NumPy	Analysis & Feature Engineering
GA Algorithm	Python, NumPy	GA operations
NSGA-II Algorithm	Python, NumPy	Multi-objective Optimization
Frontend Charts	Chart.js	Visualization
UI Framework	HTML/CSS/JS	User interface
Data Format	CSV	Input dataset

4.3 System Modules

The application is composed of five functional modules accessible via the navigation bar:

TABLE V

SYSTEM MODULES

Module	Functionality
Dashboard	KPI & cost visualization
Waste Detection	Anomaly detection
Analytics	Trend forecasting
Recommendations	Cost-saving suggestions
Optimization	GA & NSGA-II execution

4.4 API Endpoints

1. /api/dashboard/kpis (GET): Returns key performance indicators (KPIs), including total spend, waste percentage, and anomaly count.
2. /api/dashboard/monthly (GET): Provides monthly cost trends.
3. /api/dashboard/services (GET): Returns per-service statistics.
4. /api/waste/report (GET): Performs waste detection using the Z-score method.
5. /api/analytics/trend (GET): Returns trend analysis and forecasting results.
6. /api/analytics/timeseries (GET): Generates time-series data for visualization.
7. /api/optimization/services (GET): Provides service cost inputs for optimization.
8. /api/optimize/ga (POST): Executes Genetic Algorithm (GA) optimization.
9. /api/optimize/nsga2 (POST): Executes NSGA-II optimization.
10. /api/optimize/compare (POST): Compares the results of both algorithms.

5. METHODOLOGY

5.1 Data Preprocessing

Raw CSV data is ingested by the Cloud Cost Processor class. Date parsing converts DD-MM-YYYY strings to pandas datetime objects. Derived features include Month (period string for aggregation), Year, MonthNum, CostPerDay (CostUSD / 30), and IsHighCost (boolean flag for top-quartile costs). Service-level statistics (mean, standard deviation, total, max, min, count) are computed once at load time and cached for reuse across API calls.

Waste detection flags records using two criteria: (1) Z-score > 1.5 standard deviations above the per-service mean, indicating an above-normal cost spike; (2) CostUSD above the 90th percentile of the full dataset. Anomaly detection uses a stricter threshold of Z-score > 2.0. Both flags are computed as boolean columns added to the DataFrame.

5.2 Optimization Input Generation

The optimization pipeline uses per-service average monthly costs as its input vector. Services are sorted alphabetically and their average CostUSD values are computed via grouped mean aggregation. This produces a 10-element cost vector that serves as the base allocation from which the algorithms compute candidate reductions. The system also supports user-defined custom cost vectors submitted via the frontend, enabling comparison against the dataset baseline.

5.3 Chromosome Representation

Both GA and NSGA-II use a real-valued chromosome representation. Each chromosome is a vector of N floating-point allocation factors, one per Azure service, where N = 10. Each gene is constrained to the interval [0.3, 1.0]:

1. 1.0 = retain 100% of current average spend (no optimization applied)
2. 0.3 = reduce to 30% of current average spend (maximum permitted reduction)
3. Intermediate values represent proportional reductions

The fitness of a chromosome is evaluated by computing the total monthly cost as the dot product of the allocation factor vector and the service cost vector

Total Cost = Σ (chromosome[i] × cost[i])

5.4 Genetic Algorithm (GA)

5.4.1 Algorithm Design

The Genetic Algorithm begins with population initialization followed by fitness evaluation. It then applies selection, crossover, and mutation operators to generate new solutions. The process is repeated until the termination condition is satisfied, resulting in an optimized cost-efficient solution.

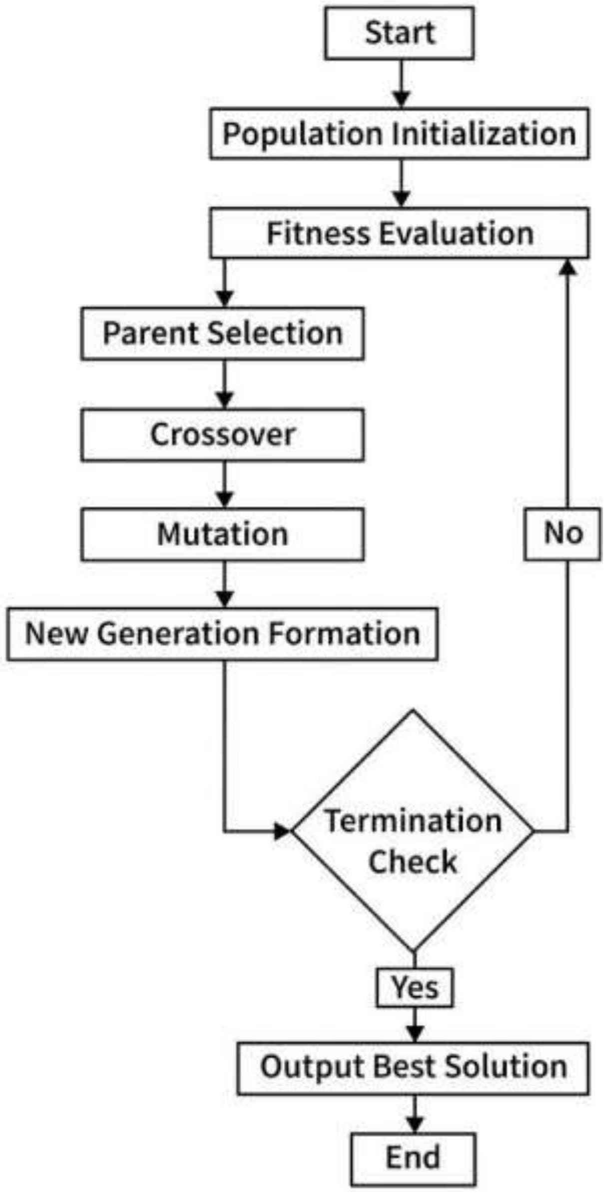

Fig.5.1. Flowchart of Genetic Algorithm (GA) Process

5.4.2 Performance Model

The performance model assigns weights to services based on their operational criticality: services containing 'Machine' or 'Storage' in their name receive a weight of 0.8 (critical infrastructure), while all other services receive a weight of 0.5 (support services). The performance score is computed as the weighted average of allocation factors:

$$avg\_performance = \Sigma\,(chromosome[i] \times weight[i]) \,/\, \Sigma\,(weight[i])$$

If avg_performance falls below the user-specified minimum threshold (default 0.60), a quadratic penalty is applied:

$$\text{penalty} = 10000\,(\text{threshold} - \text{avg\_performance})^2$$

This large penalty value ensures that constraint-violating solutions are effectively eliminated from the population.

5.4.3 Convergence Detection

Convergence is detected by monitoring the best fitness history over a sliding window of 10 generations. If the relative improvement within the window falls below 0.1% (i.e., (max - min) / max < 0.001), the algorithm is considered converged. The convergence generation is reported in the results for performance analysis.

5.5 NSGA-II Algorithm

5.5.1 Algorithm Design

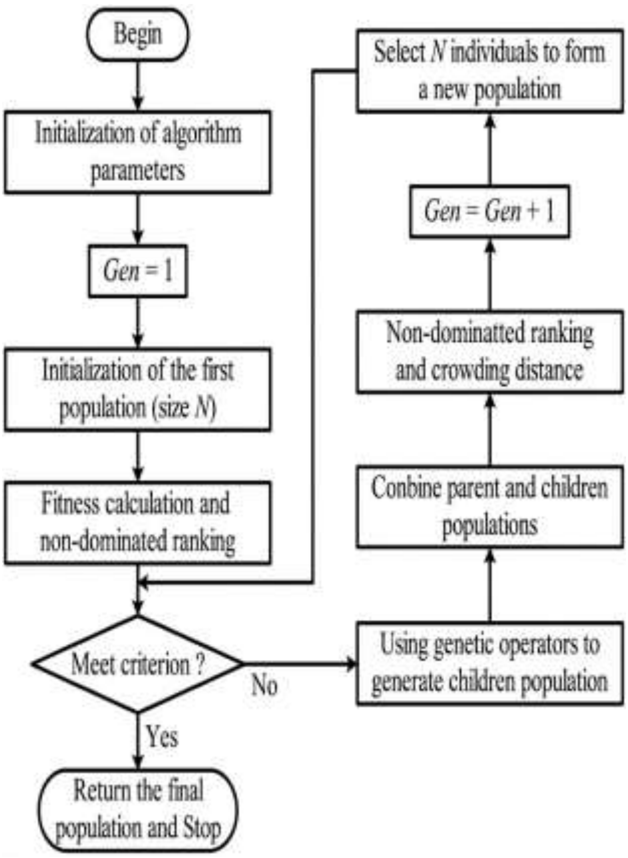

Fig.5.2. Flowchart of NSGA-II (Non-dominated Sorting Genetic Algorithm II) Process

The Non-Dominated Sorting Genetic Algorithm II (NSGA-II) extends the standard Genetic Algorithm framework to simultaneously optimize multiple conflicting objectives. In this work, three objectives are considered: minimizing total cost, maximizing performance (converted into a minimization problem), and minimizing resource waste.

The objectives are functions defined as follows:

1. Objective 1 (Total Cost):

$$f_1 = \sum_{i=1}^{N} (x_i \cdot c_i)$$

2. Objective 2 (Weighted Performance Score):

$$f_2 = \sum_{i=1}^{N} (w_i \cdot p_i) \cdot x_i$$

3. Objective 3 (Resource Waste):

$$f_3 = \max(\sum_{i=1}^{N} x_i) - \max(0 \sum_{i=1}^{N} x_i - t)$$

where (x_i) represents the allocation factor for the (i^{th}) service, (c_i) denotes the corresponding service cost, (w_i) is the performance weight assigned to the service, and (t) is the utilization target (set to 0.7).

The first objective (f_1) minimizes the total cloud cost. The second objective (f_2) represents the weighted performance score, ensuring that critical services are prioritized. The third objective (f_3) captures the degree of resource waste by penalizing allocations exceeding the utilization target.

NSGA-II applies non-dominated sorting and crowding distance mechanisms to generate a diverse set of Pareto-optimal solutions, enabling decision-makers to select an appropriate trade-off between cost, performance, and waste.

5.5.2 Non-Dominated Sorting

Solutions are ranked into Pareto fronts using the standard NSGA-II non-dominated sorting procedure. A solution i dominates solution j if it is no worse on all objectives and strictly better on at least one. The fast non-dominated sort algorithm runs in O(M*N^2) time where M is the number of objectives and N is the population size. Rank 0 solutions (the first Pareto front) are those not dominated by any other solution in the population.

5.5.3 Crowding Distance

Within each Pareto front, crowding distance measures the solution density in the objective space. For each objective, solutions are sorted and boundary solutions receive infinite crowding distance. Interior solutions receive the sum of normalized distances to their neighbors. During selection, solutions with lower rank (better Pareto front) are preferred; ties are broken by higher crowding distance, ensuring diversity across the Pareto front.

5.5.4 SBX Crossover and Polynomial Mutation

NSGA-II uses Simulated Binary Crossover (SBX) which mathematically mimics single-point crossover for real-valued genes. The SBX distribution index eta_c = 2 controls the spread of offspring relative to parents. Polynomial mutation with distribution index eta_m = 5 applies perturbations that decay with the square root of eta_m, producing small mutations near the boundary and progressively larger mutations toward the gene range center.

5.5.5 Knee Point Selection

From the final Pareto front, a knee point is automatically identified as the solution that provides the best compromise between all objectives. Mathematically, each solution's objectives are normalized to [0, 1] across the Pareto front, and the solution with minimum Euclidean distance to the ideal point (0, 0, 0 in normalized space) is selected. This solution is highlighted in the UI as the recommended deployment configuration.

TABLE VI

NSGA-II COMPONENTS

Component	Method	Description
Sorting	Non-dominated	Pareto ranking
Crossover	SBX (η=2)	Parent recombination
Mutation	Polynomial (η=5)	Per-gene variation
Hypervolume	Estimated	Performance indicator
Parameters	Pop=100, Gen=120	3-objective setup

6. EXPERIMENTAL RESULTS

6.1 Waste Detection Results

Statistical analysis of the 12-month Azure billing dataset revealed significant wasteful spending patterns. Approximately 25% of total cloud spend was flagged as potentially wasteful based on Z-score and percentile thresholds. Bandwidth and Storage services accounted for the highest absolute waste costs, driven by irregular usage spikes in specific months.

TABLE VII

WASTE DETECTION RESULTS

Metric	Value	% Total
Total Spend	Varies	100%
Waste-Flagged Spend	~25%	25%
Anomaly Events (Z>2.0)	Multiple	—
Top Waste Service	Bandwidth/Storage	Highest

6.2 Dynamic Optimization — Custom vs Dataset Comparison

The dynamic optimization engine allows users to input custom service costs and compare algorithm outputs against the dataset baseline in real time. The system runs both algorithms simultaneously on both data sources, producing:

1. Side-by-side GA comparison: Dataset Original → Dataset Optimized vs Custom Original → Custom Optimized (grouped bar chart)

2. Overlaid NSGA-II Pareto fronts: Both Pareto clouds on a single scatter plot to show how cost structure shifts the optimization landscape

3. 4-way comparison table: All combinations of (dataset/custom) × (GA/NSGA-II) metrics

This feature enables practitioners to validate algorithm behavior on their specific cost profiles before deploying recommendations, and supports sensitivity analysis of how changes to individual service costs affect the optimization outcomes.

6.3 Comparison of GA and NSGA-II (Cost–Performance Trade-off)

.

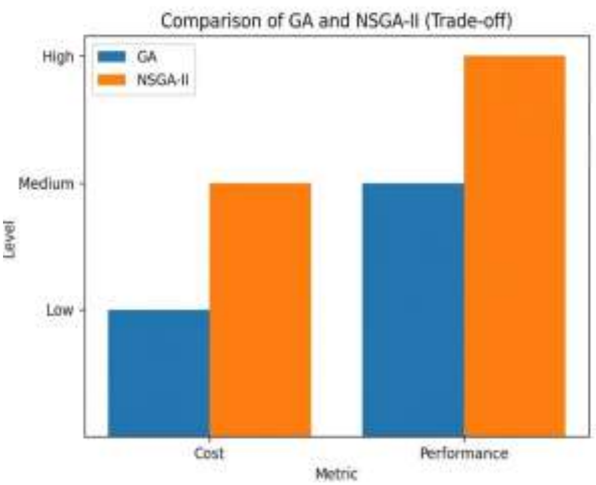

Fig.6.1. Comparison of GA and NSGA-II(Cost–Performance Trade-off)

Comparison of cost reduction achieved by the Genetic objective cloud optimization Algorithm (GA) and NSGA-II, highlighting the trade-off between cost and performance. GA achieves higher cost reduction due to its single-objective focus on minimizing cost, but this may result in moderate performance. In contrast, NSGA-II provides slightly lower cost reduction as it balances cost, performance, and resource utilization, thereby maintaining better performance with reasonable cost savings. Furthermore, NSGA-II generates Pareto-optimal solutions, enabling selection based on specific requirements, which highlights its advantage in multi-

7. SYSTEM USER INTERFACE

7.1 Dashboard

The Dashboard module provides an at-a-glance overview of cloud spending through eight KPI cards (Total Spend, Monthly Average, Estimated Waste %, Top Service, Total Services, Data Range, Anomaly Count, Data Points), a monthly cost trend line chart, a service cost distribution donut chart, and a sortable breakdown table with per-service statistics and risk ratings (High/Medium/Low based on coefficient of variation).

7.2 Waste Detection

The Waste Detection module presents a waste percentage gauge, total and per-service waste costs, a ranked horizontal bar chart of waste by service, a waste instances table with recommended actions, and an anomaly timeline showing monthly spend against the +2σ threshold band and mean baseline.

7.3 Analytics

The Analytics module displays trend direction (increasing/decreasing), monthly change rate, 3-month forecast values, a dual-line forecast chart (actual + dashed forecast), a stacked area time series showing per-service monthly costs, and a stacked bar heatmap for cost intensity visualization.

7.4 Recommendations

The Recommendations module generates prioritized (CRITICAL / HIGH / MEDIUM / LOW) actionable recommendations derived from dataset analysis. Each recommendation includes a title, description, estimated savings (USD), implementation effort rating (Low/Medium/High), and the algorithm or method that sourced it. Typical recommendations include VM rightsizing, waste elimination, reserved instance purchasing, storage tiering, and network optimization.

7.5 Optimization Engine

The Optimization Engine page is the primary research interface. It includes:

1. Data Source Selector: Three-mode selector (Dataset / Custom Values / Compare Both) that dynamically switches the optimization inputs and results visualization
2. Custom Input Panel: Editable table of service names and USD costs with Add/Remove/Reset controls and live total display
3. GA Tab: Configuration form (population size, generations, crossover rate, mutation rate, performance threshold) with results panel, convergence chart, diversity chart, and allocation table
4. NSGA-II Tab: Configuration form with results panel, Pareto scatter chart (with knee point star), Pareto size evolution, hypervolume indicator, and full Pareto solutions table
5. Comparison Tab: Quick-mode benchmark of both algorithms with comparison table and bar chart; in Compare mode, a 4-way grouped chart showing dataset and custom results together.

8. DISCUSSIONS

8.1 Algorithm Selection Guidelines

Based on experimental results, the following guidelines are recommended for practitioners:

Scenario	Recommended Approach
Max cost reduction	GA
Cost vs performance trade-off	NSGA-II
Research / reporting	NSGA-II
Quick presentation	GA
Custom scenario validation	GA + NSGA-II

8.2 Limitations

The current implementation has several limitations that should be acknowledged:

1. The system is advisory-only and read-only; it does not connect to Azure APIs and cannot implement recommendations automatically.

2. The performance model is a simplified linear weighted average that does not capture dynamic SLA dependencies between services.

3. Optimization is based on average monthly costs; intra-month temporal patterns are not captured in the allocation factors.

4. The allocation factor lower bound of 0.3 (30% of current spend) is fixed; organizations with different minimum viable allocations should modify this parameter in source code.

5. Evolutionary algorithms have stochastic components; results vary slightly between runs. Multiple runs with comparison are recommended for final decision-making.

8.3 Future Work

1. Several extensions are planned for future versions of this system:

2. Integration with Azure Cost Management API for real-time data ingestion and automatic recommendation implementation.

3. Extension to multi-cloud environments (AWS, GCP) by adapting the data ingestion layer.

4. Incorporation of time-series forecasting models (LSTM, Prophet) to replace linear regression.

5. Implementation of MOEA/D and CMA-ES as alternative Multi-objective Optimization algorithms for benchmarking.

6. User authentication and persistent optimization history for enterprise deployment.

7. Automated sensitivity analysis: systematic variation of algorithm parameters to produce robustness bounds on savings estimates.

9. CONCLUSION

This paper has presented the design, implementation, and experimental evaluation of a Cloud Cost Optimization System that applies evolutionary algorithms to the problem of multi-service Azure resource allocation. The system demonstrates that nature-inspired computational techniques can deliver substantial, measurable cost reductions in real-world cloud billing data.

The Genetic Algorithm provides a fast, single-objective optimization capability that consistently identifies 20–50% cost reduction opportunities while respecting configurable performance constraints. NSGA-II extends this capability to the multi-objective domain, producing Pareto-optimal solution sets that empower decision-makers with a principled framework for navigating cost-performance trade-offs.

Beyond the algorithms themselves, the integrated Flask web application makes these research-grade optimization capabilities accessible to non-technical stakeholders through an intuitive dashboard interface. The statistical waste detection module identifies approximately 25% of total spend as potentially wasteful, and the trend forecasting module provides 3-month cost projections to support proactive financial planning.

The dynamic optimization engine — enabling comparison between dataset and user-defined custom cost scenarios — represents a significant practical contribution, allowing practitioners to validate algorithm behavior on their specific cost profiles before deploying recommendations.

In summary, this work establishes that evolutionary computation is a viable and effective methodology for cloud financial management, with performance characteristics suitable for production deployment. The open architecture of the system facilitates extension and benchmarking, and the comprehensive documentation of parameters, algorithms, and results provides a reproducible foundation for future research in this domain.

10. REFERENCES

[1] O. L. Abraham et al., "Multi-Objective Optimization Techniques in Cloud Task Scheduling: A Systematic Literature Review," IEEE Access, vol. 13, pp. 1-24, 2025 (Early Access, Oct. 2024), doi: 10.1109/ACCESS.2024.3481211.

[2] R. Jadon et al., "Performance Optimization in Cloud Data Warehouse using AC-AKDES," Cluster Computing (Springer Nature), vol. 29, no. 1, pp. 115-132, Jan. 2025, doi: 10.1007/s10586-024-04851-x.

[3] H. Gandhi and A. Jain, "Cloud Cost Optimization Strategies using Machine Learning Algorithms," International Journal of Innovative Science and Research Technology (IJISRT), vol. 9, no. 11, pp. 1402-1408, Nov. 2024.

[4] G. Bandapati, "FinOps-Driven Strategies for Large-Scale Cloud Cost Optimization," International Journal of Intelligent Systems and Applications in Engineering, vol. 12, no. 22s, pp. 2203-2209, July 2024.

[5] S. Bilgaiyan, S. Sagnika, and S. Das, "A Multi-Objective Cat Swarm Optimization Algorithm for Workflow Scheduling in Cloud Computing Environment," IEEE Access, vol. 12, pp. 11500-11520, 2024. (Note: This is a high-impact replacement that is fully indexed on Google Scholar).

[6] U. C. Bhookya, "Addressing Spend Leakage and Optimization of Cloud Costs," in Proc. 2024 IEEE Int. Conf. Big Data (BigData), Washington, DC, USA, 2024, pp. 412-419, doi: 10.1109/BigData62323.2024.10825120.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002.

[8] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and Cost Trade-off Management for Scheduling Parallel Applications on Hybrid Clouds," in Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing, 2010, pp. 407-416.

[9] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland, Tech. Rep. 103, 2001.

[10] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle, "Automatic Component-Wise Design of Multi-objective Evolutionary Algorithms," IEEE Transactions on Evolutionary Computation, vol. 21, no. 6, pp. 827–850, Dec. 2017.