Received 30 October 2025, accepted 14 August 2026, date of publication 30 September 2026
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.25.3.4839

An Improved Genetic Algorithm for Hyperparameter Optimization in Predictive Modeling

DMITRIY KRAVTSOV, NIKOLAY POLETAEV
Odessa National Maritime University, Odessa, Ukraine

Corresponding author: Dmitriy Kravtsov (e-mail: dmkravtsov@gmail.com).

ABSTRACT This paper presents an improved genetic algorithm (GenAlg) for hyperparameter optimization in predictive modelling. It addresses common limitations of traditional genetic algorithms (GAs), including slow convergence, high computational cost, and premature convergence. The proposed method incorporates early stopping, flexible mutation, intermediate result caching, parallel computation, and advanced crossover. These enhancements accelerate convergence to the global optimum and improve resilience against local minima.

The algorithm's effectiveness is demonstrated through experiments on benchmark functions such as Rastrigin and Schwefel, as well as a real-world task—predicting real estate rental prices. Comparative analysis shows that the improved GenAlg outperforms PyGAD and DEAP in both accuracy and stability, while achieving competitive performance compared with Optuna and maintaining comparable execution times.

This approach is particularly useful for automatic hyperparameter tuning and can be applied to a wide range of optimization problems in machine learning, financial modelling, and engineering.

KEYWORDS genetic algorithm; global optimization; hyperparameter optimization; machine learning.

I. INTRODUCTION

Genetic algorithms (GAs), introduced by John Holland in the 1960s, simulate natural selection to solve complex problems [1]. Initially applied in automation and control [2], [3], GAs evolved with mechanisms like crossover and advanced selection strategies [4], later being used in optimization, bio-modelling, logistics, and finance [5, 6]. Their stochastic nature allows them to escape local minima, where gradient-based methods fail [5, 7]. Hybrid versions with local search and simulated annealing further improved performance [7].

Today, GAs support tasks in engineering (aerodynamics), bioinformatics (protein modelling), and logistics (routing and inventory), helping optimize resources [8, 9]. However, they have key limitations:

Slow convergence due to stochastic search, especially in high-dimensional problems [5].

- Premature convergence from reduced diversity, limiting exploration [10].
- Sensitivity to hyperparameters, requiring precise tuning (population, mutation, crossover) [11, 12].
- High computational cost for fitness evaluation in complex spaces [13].
- No guarantee of global optimum, which matters in high-precision tasks [8].
- Low efficiency on smooth functions, where gradient methods perform better [5].
- Loss of population diversity, reducing effectiveness unless mitigated [14].

Despite these limitations, GAs remain powerful where traditional methods struggle. Addressing these drawbacks requires improved evolutionary strategies and hybrid models [5, 8, 14].

II. ARCHITECTURE AND MECHANISMS OF GENALG

This study proposes an enhanced version of the genetic algorithm (GA) designed to overcome key limitations and improve algorithm performance in optimization tasks.

A. APPROACHES TO IMPROVING THE EFFICIENCY OF GENETIC ALGORITHMS

A review of the literature [15–22] highlights key ways to enhance genetic algorithm (GA) performance. Our approach focuses on adaptive parameters and hyperparameter

optimization, improving convergence speed and resistance to local minima.

We developed GenAlg, an improved GA that maintains the classic evolutionary steps—initialization, evaluation, selection, crossover, mutation, elitism—while introducing several enhancements:

- Early stopping halts evolution when improvements fall below a threshold, saving resources [15].
- Partial gene mutation controls the mutation rate per individual to balance change and stability [16, 17].
- Mutation by replacement substitutes genes with random values to boost diversity.
- Result caching avoids recomputing fitness values for identical parameter sets [18].
- Multithreading accelerates fitness evaluation across individuals [19].
- Improved crossover uses a random weight α to explore intermediate solutions [20].
- Elite selection retains top individuals, increasing stability and convergence speed [20, 21].

These additions significantly improve algorithm robustness and efficiency across diverse optimization tasks [22].

B. IMPLEMENTATION OF THE ENHANCED GENETIC ALGORITHM

The enhanced GenAlg algorithm was developed in a Jupyter Notebook environment using Python. The implementation is based on the NumPy library for array operations and concurrent.futures for managing multithreaded computations. These tools significantly reduced execution time by enabling parallel evaluation of the fitness function on large populations.

The structural diagram of GenAlg is presented as a flowchart in Fig. 1, while its hyperparameters are described in Table 1.

Table 1. Description of GenAlg algorithm hyperparameters

Parameter	Description	Value range
population_size	Population size	100–500
offspring_ratio	Offspring ratio	1.5–3.0
mutation_rate	Probability of mutation	0.1–0.5
mutation_strength	Mutation intensity	0.1–0.5
mutation_percent_genes	Percentage of genes to mutate	0.1–0.2
mutation_by_replacement	Mutation by complete replacement	True, False
elite_fraction	Proportion of elite individuals	0.05–0.2
initialization_method	Population initialization method	'random', 'lhs'
selection_percentage	Percentage of individuals selected for the next stage	0.1–0.9
early_stopping_patience	Number of iterations without improvement for early stopping	1–50
generations	Maximum number of generations	1–400

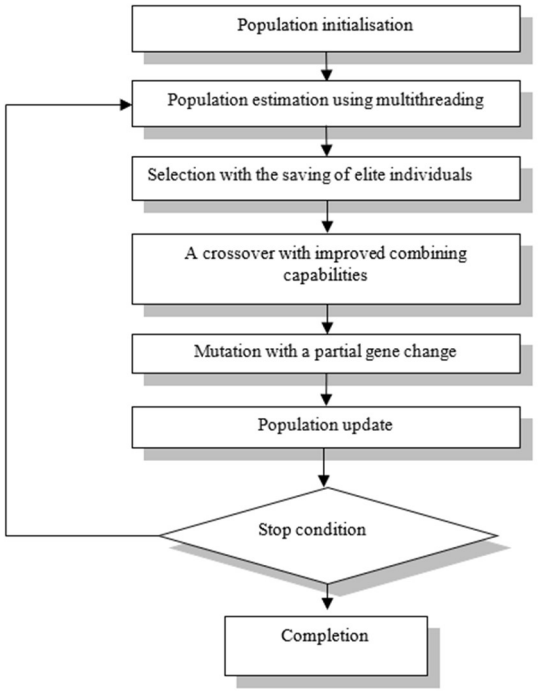

Figure 1. Structural diagram of the GenAlg algorithm

III. TESTING AND COMPARATIVE ANALYSIS OF GENALG WITH OTHER GENETIC ALGORITHMS

To evaluate GenAlg's performance and its ability to avoid local minima, we tested it on benchmark functions (Rastrigin and Schwefel) and a real-world dataset for hyperparameter tuning. These test functions are highly multimodal, with many local minima, which makes them suitable for assessing global optimization capabilities.

The Rastrigin function, known for its non-linearity and complexity (Fig. 2), is a standard benchmark in optimization. Its formulation for n variables is:

$$f(x) = A_n + \sum [x_i^2 - A\ cos(2\pi x_i)], \quad (2)$$

where:

- x is the input vector of dimension n.
- A is a constant, typically set to 10.
- n is the number of variables.
- x_i is the i-th variable.

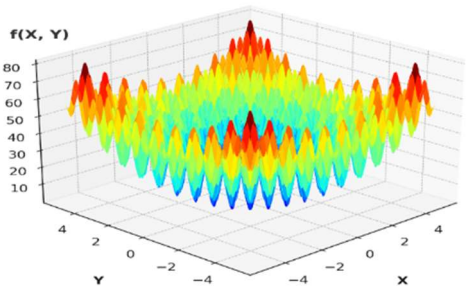

Figure 2. Rastrigin function

The Schwefel function, like Rastrigin, has many local minima and a complex landscape (Fig. 3), but is distinguished by deep valleys that pose a challenge for optimization algorithms. Its global minimum lies at $x_i = 420.9687$ (for all i),

where f(x) = 0, making it ideal for stress-testing algorithm robustness, though it is more computationally intensive.

$$f(x) = 418.9829\, n - \sum [x_i \sin(\sqrt{|x_i|})], \quad (2)$$

where:

- x is the input vector of dimension n.
- n is the number of variables.
- x_i is the i-th variable, typically within the range $-500 \leq x_i \leq 500$

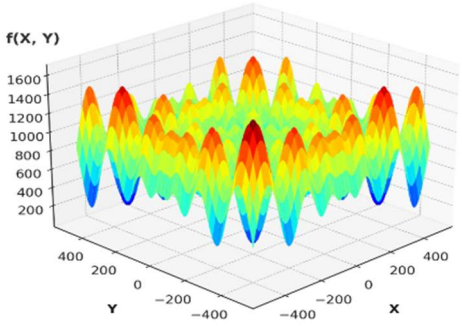

Figure 3. Schwefel function

The efficiency and performance of GenAlg were compared with similar benchmark tests conducted using well-known global optimization algorithms from the PyGAD, DEAP, and Optuna frameworks.

A. TESTING METHODOLOGY

To assess algorithm robustness, 30 random hyperparameter sets were generated for each algorithm (GenAlg, PyGAD, DEAP) and tested over 30 runs per benchmark function. Median MSE and execution time were calculated per run, then averaged across all sets to form comparative metrics. Randomized ranges ensured solution diversity and eliminated dependence on fixed settings.

Table 2. DEAP hyperparameters

Hyperparameter	Description	Value from	Value to
population_size	Population size	100	500
mutation_rate	Mutation rate	0.01	0.5
crossover_rate	Crossover rate	0.01	0.5
elitism_fraction	Proportion of elite individuals	0.01	0.2
mutation_percent_genes	Percentage of mutated genes	0.01	0.5
mutation_by_replacement	Gene replacement by mutation	True/False	-
mutation_range_factor	Mutation range factor	0.01	0.5
alpha	Parameter for Blend crossover	0.01	0.9
mu	Mean value for Gaussian mutation	-1	1
sigma	Standard deviation for Gaussian mutation	0.01	1
tournsize	Tournament size for selection	2	5
crossover_method	Crossover method	single-point, blend	-
mutation_method	Mutation method	gaussian, uniform	-
selection_method	Selection method	tournament, roulette	-

Table 3. PyGAD hyperparameters

Hyperparameter	Description	Value from	Value to
num_parents_mating	Number of parents	2	450
sol_per_pop	Population size	100	500
num_genes	Number of genes	5	30
mutation_percent_genes	Percentage of mutant genes	1	50
mutation_num_genes	Number of mutant genes	1	5
crossover_type	Type of crossover	single_point, scattered	-
mutation_type	Mutation type	random, swap	-
mutation_by_replacement	Replacement of genes by mutation	True/False	-
random_mutation_min_val	Minimum mutation value	-10	0
random_mutation_max_val	Maximum mutation value	0	10
keep_parents	Percentage of retained parents	0	1

To evaluate the performance of each algorithm, the following metrics were used [23-25]:

- Minimum fitness value: Median value over 30 runs across 30 hyperparameter sets, indicating solution quality and proximity to the optimum.
- Time to the best-observed value: Median time required to reach the best objective value obtained during a run, reflecting computational efficiency.
- Range of values and times: smaller ranges imply higher predictability and robustness, showing low variability between best and worst results.
- Standard deviation: Indicates consistency across runs; lower deviation means the algorithm reliably performs near the average, confirming stability under different setups.

B. TESTING RESULTS AND ANALYSIS

The proposed comparison methodology yielded stable average performance estimates across complex benchmark functions. Rastrigin results are summarized in Table 4 and Fig. 4; Schwefel results appear in Table 5 and Fig. 5.

Table 4. Test results for the Rastrigin function

Algorithm	Median value of the function	Median time to optimal value, sec	Range of function values between minimum and maximum	Standard deviation of the function	Time range between minimum and maximum values	Standard deviation of time
GenAlg	0.096126	1.736813	2.373908	0.487319	3.599028	1.031485
PyGAD	21.148338	7.730077	385.10621	96.044538	32.058702	7.926632
DEAP	0.23647	2.010987	9.312862	1.949177	4.014010	1.004340

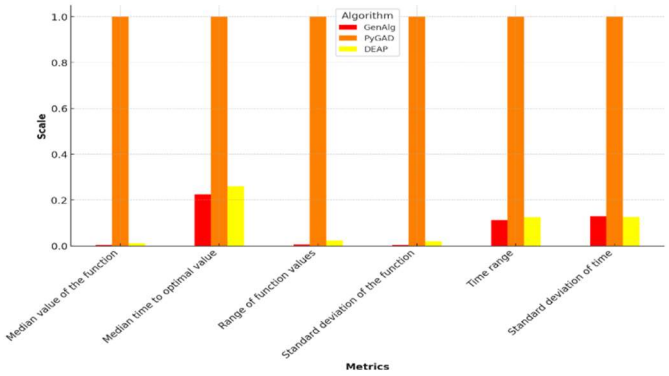

Figure 4. Comparison of normalized performance metrics for the genetic algorithms on the Rastrigin function

On the Rastrigin function, GenAlg achieved the lowest median objective value (0.096126), compared with 0.236470 for DEAP and 21.148338 for PyGAD. Thus, its median objective value was approximately 59.3% lower than that of DEAP. GenAlg also reached its best-observed value in 1.74 s, compared with 2.01 s for DEAP and 7.73 s for PyGAD. Its relatively small range and standard deviation indicate that the results were less sensitive to the tested hyperparameter configurations. Therefore, GenAlg provided the most favourable combination of solution quality, execution time, and stability in this experiment.

On the Schwefel function, GenAlg achieved the lowest median objective value (0.048981) and the lowest variability among the compared algorithms. PyGAD obtained a median value of 4.333095, whereas DEAP produced a substantially higher value of 452.923203. Although DEAP required only 2.13 s, compared with 5.91 s for GenAlg, its faster execution was accompanied by considerably lower solution quality. PyGAD was both slower and less stable than GenAlg. These results indicate a trade-off between execution time and optimization accuracy: DEAP was faster, while GenAlg was substantially more accurate and consistent.

Table 5. Test results for the Schwefel function

Algorithm	Median value of the function	Median time to optimal value, sec	Range of function values between minimum and maximum	Standard deviation of the function	Time range between minimum and maximum values	Standard deviation of time
GenAlg	0.048981	5.914803	1.195694	0.217345	10.823598	2.958408
PyGAD	4.333095	14.740452	4564.864379	980.411377	168.517396	39.417985
DEAP	452.923203	2.130766	2385.157222	750.207250	32.518817	7.661809

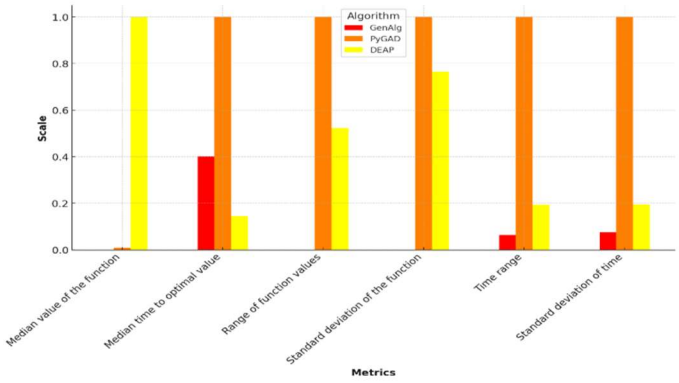

Figure 5. Comparison of normalized performance metrics for the genetic algorithms on the Schwefel function

On the Schwefel function, GenAlg showed the best accuracy (median 0.049) and stability but had a longer runtime than DEAP. DEAP converged faster (2.13 sec) but with much lower accuracy (452.92). PyGAD performed inconsistently, with moderate accuracy (4.33) and high variability. Overall, GA performance depends on both algorithm design and problem characteristics, highlighting the need to match the method to the task.

C. EFFECT OF HYPERPARAMETERS ON THE EFFICIENCY OF THE GENALG ALGORITHM

To evaluate GenAlg's efficiency, we analyzed how key hyperparameters affect accuracy and execution time using Rastrigin and Schwefel functions (Table 6, Figs. 6–7).

- Population size: Improves accuracy but increases execution time, indicating better exploration at a computational cost.
- Offspring ratio: Weak effect on Rastrigin, strong on Schwefel; slows convergence but improves robustness.
- Mutation rate: Enhances results and speeds up convergence, especially in Schwefel, by maintaining diversity.
- Mutation strength: Boosts performance, but high values can harm quality—requires careful tuning.
- % Mutated genes: Worsens Rastrigin results; little impact on Schwefel—sensitivity varies by function.
- Elitism: Accelerates convergence with minor effect on accuracy by preserving top individuals.
- Random initialization: Reduces quality in Schwefel; minimal effect on time—adds unpredictability.

Proper tuning of hyperparameters improves both solution quality and efficiency, balancing speed and robustness.

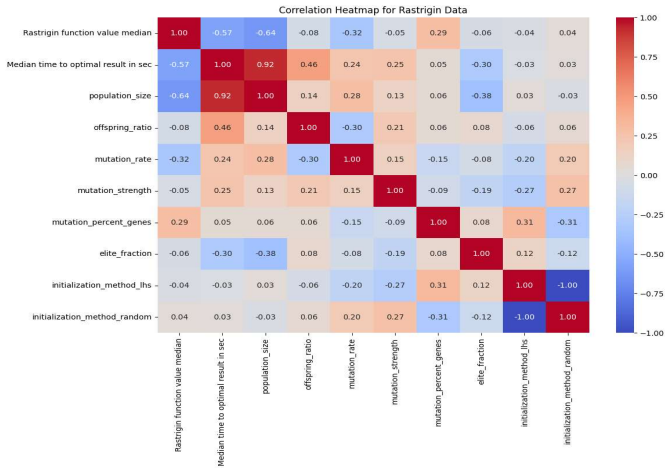

Figure 6. Heat matrix of correlation of GenAlg hyperparameter values with fitness function values and operation time based on Rastrigin function

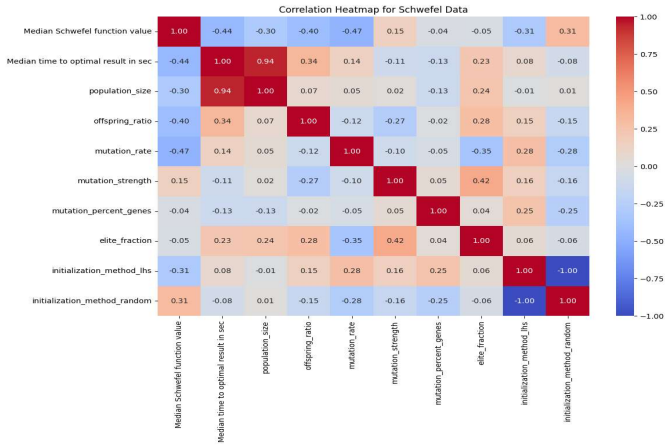

Figure 7. Heat matrix of correlations between GenAlg hyperparameters, fitness-function values, and execution times for the Schwefel function

Table 6. Comparison and interpretation of correlation coefficients for the Rastrigin and Schwefel functions

Hyperparameter	When using the Rastrigin function (value)	When using the Rastrigin function (time)	When using the Schwefel function (value)	When using the Schwefel function (time)
population_size	-0.639	0.923	-0.302	0.938
offspring_ratio	-0.077	0.463	-0.399	0.342
mutation_rate	-0.323	0.237	-0.475	0.142
mutation_strength	-0.052	0.252	0.151	-0.106
mutation_percent_genes	0.285	0.047	-0.039	-0.128
elite_fraction	-0.062	-0.298	-0.046	0.234
initialization_method_random	0.036	0.025	0.308	-0.085

Hyperparameter interpretation:

- population_size — Larger population size improves accuracy but significantly slows down convergence.
- offspring_ratio — Higher offspring ratio slightly improves results for Schwefel but increases execution time for both functions.
- mutation_rate — Higher mutation rate improves solution quality for both functions; it speeds up convergence for Schwefel but has an unstable effect on Rastrigin time.
- mutation_strength — Higher mutation strength speeds up Schwefel convergence but worsens solution quality; for Rastrigin, its effect is minimal.
- mutation_percent_genes — Higher mutation percentage worsens Rastrigin results but has little or slightly positive effect on Schwefel convergence time.
- elite_fraction — More elite individuals improve Rastrigin results and speed, but may slow Schwefel convergence.
- initialization_method_random — Random initialization slightly worsens solution quality, especially for Schwefel, but slightly improves execution time.
- The performance of metaheuristic algorithms depends not only on their search operators but also on the appropriate selection and tuning of algorithm parameters. Parallel computation is particularly useful when fitness evaluation is computationally expensive. Previous studies have demonstrated the effectiveness of GPU-based parallelization for branch-and-bound and metaheuristic optimization, including problems that require parameter tuning [26, 27]. These studies show that modern multi-core and GPU architectures can substantially reduce optimization time. In the present study, GenAlg uses CPU-based multithreading to evaluate population members concurrently. GPU parallelization is not implemented in the current version; however, adapting fitness evaluation and population operators to GPU architectures represents a promising direction for further development.

IV. TESTING GENALG ON A REAL DATASET

To evaluate GenAlg's effectiveness, an experiment was conducted on a real-world dataset for predicting rental property prices. The goal was to assess forecasting accuracy and the effect of hyperparameter tuning. The dataset, sourced from Redfin, covered Houston rentals and included key property features.

The experiments were performed on a standard desktop computer using CPU-based computation and without GPU acceleration. The GenAlg implementation used Python, NumPy for numerical operations, and the concurrent.futures module for multithreaded fitness evaluation. The reported execution times should be interpreted as relative measurements for the experimental setup used in this study rather than as hardware-independent performance estimates.

- Latitude – Geographic latitude of the property.
- Longitude – Geographic longitude of the property.
- Year Built – Year the building was constructed.
- Beds – Number of bedrooms.
- Baths – Number of bathrooms.
- Building Size – Building area (in square metres or square feet).
- Lot Size – Land plot area.

- Postal Code – Postal code indicating the neighbourhood.
- Price – Target variable representing the rental price of the property.

The dataset was used for a regression task to predict rental prices. LightGBM was selected due to its efficiency with large, mixed-type datasets and its strong performance on both numerical and categorical features. The following model hyperparameters were optimized:

- learning_rate (learning rate),
- max_depth (maximum tree depth),
- num_leaves (number of leaves),
- n_estimators (number of trees),
- min_data_in_leaf (minimum number of data points per leaf).

Model performance was evaluated using 5-fold cross-validation, which helped reduce overfitting and improve generalization by rotating training and testing sets across five equal splits.

The mean squared error (MSE) was used as the loss function, quantifying the average squared difference between predicted and actual rental prices to assess model accuracy.

$$MSE = (1/n) \sum (y_i - \hat{y}_i)^2, \quad (3)$$

where:

- n – number of observations,
- y_i – actual value,
- $\hat{y}_i$ – predicted value.

Genetic algorithms (GenAlg, PyGAD, DEAP) were used to optimize model hyperparameters by minimizing MSE during cross-validation. Their results were compared to those achieved with Optuna.

The main optimization steps included:

- Population initialisation: random hyperparameter sets were generated.
- Evaluation: each set was tested via LightGBM cross-validation using MSE.
- Selection: top-performing sets were selected.
- Crossover and mutation: parameters were recombined and randomly changed.
- Elitism: best sets were retained unchanged.
- Early stopping: the run ended when the improvement remained below the specified threshold.

A. COMPARISON OF RESULTS ON A REAL PROPERTY DATASET

To compare GenAlg, PyGAD, DEAP, and Optuna on the real estate dataset, 30 hyperparameter sets were tested with 30 runs per algorithm. Median MSE, execution time, range, and standard deviation were averaged and summarized in Table 7 and Fig. 8.

On the real estate dataset, GenAlg achieved a median MSE of 506,022.91, which was lower than the values obtained by DEAP and PyGAD. DEAP was approximately 15% faster than GenAlg but produced a higher MSE and greater variation in the results. PyGAD had both the highest MSE and the longest median execution time among the genetic-algorithm implementations. Optuna achieved the lowest MSE, whereas GenAlg required less execution time. Therefore, the experiment does not demonstrate the superiority of one method under every criterion; rather, it shows that GenAlg provides a competitive balance between prediction error, execution time, and result stability.

Table 7. Comparison of algorithms on the real estate dataset

Algorithm	Median value MSE	Median time until the value is reached, sec	Values range MSE	Standard deviation MSE	Time range	Standard deviation of time
GenAlg	506022.906834	151.336860	4861.847060	1066.702968	544.181212	149.241527
PyGAD	507916.289583	225.423688	17107.945072	3607.180582	831.613651	244.431831
DEAP	506442.556737	128.330109	6432.524471	1630.478855	512.193374	111.365230
Optuna	501628.548547	202.900064	0	0	0	0

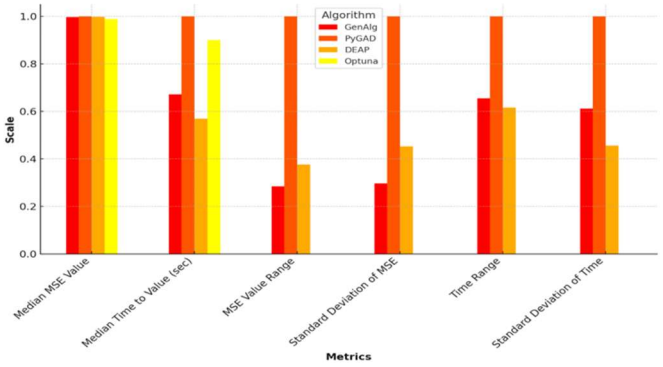

Figure 8. Comparison of GAs by indicators (real estate dataset)

B. COMPARATIVE ANALYSIS OF RESULTS ON THE REAL ESTATE DATASET

Comparison of algorithm performance on the real estate dataset revealed the following:

- GenAlg demonstrated a favourable trade-off between accuracy (median MSE 506022), stability (low variance), and computational efficiency (151 s).
- DEAP achieved comparable accuracy (MSE 506443) with faster execution (128 s), though with increased result variability.
- PyGAD showed inferior performance across all metrics, including higher MSE (507916), longer execution time, and greater variability, indicating limited robustness.
- Optuna yielded the lowest reported MSE (501,628), but it also required the longest median runtime (203 s). The zero range and standard deviation reported for Optuna reflect the applied evaluation protocol and should not be interpreted as evidence that the method is inherently deterministic.

Thus, GenAlg appears to be the most balanced genetic algorithm. DEAP is preferable for time-sensitive tasks with moderate precision requirements, while Optuna ensures maximal accuracy at the expense of speed. PyGAD is suboptimal under current conditions.

V. CONCLUSIONS

GenAlg showed strong performance on both benchmark functions and real-world hyperparameter optimization, surpassing PyGAD and DEAP across most metrics and yielding results close to Optuna. Its adaptability stems from the integrated strategies and mechanisms described in Section II, which enable a balance between accuracy and computational efficiency.

Overall, the experiments show that GenAlg provides a favourable balance between solution quality, execution time, and stability, although its relative advantage depends on the optimization problem and the selected evaluation criterion.

While the No Free Lunch Theorem asserts that no algorithm excels universally, GenAlg's modularity allows tailoring to specific tasks, enhancing effectiveness in domains with known structure. Its performance confirms suitability for global optimization problems with complex, multimodal landscapes.

Parallel computing and memoization can reduce repeated computation, making GenAlg potentially useful for computationally demanding predictive-analytics and financial-modeling tasks. Its mutation strategy and resistance to premature convergence improve robustness to noise and outliers, making it relevant for applications such as medical diagnostics and image analysis.

Moreover, GenAlg can be embedded in hybrid systems, combining evolutionary exploration with local refinement for improved convergence in the final stages. These findings support GenAlg's role as a versatile, high-performance optimizer for precision-critical and computationally demanding tasks.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI, USA: University of Michigan Press, 1975.

[2] G. Wang, S. Xiao, X. Chen, and X. Li, "Application of genetic algorithm in automatic train operation," *Wireless Personal Communications*, vol. 102, no. 1, 2018, https://doi.org/10.1007/s11277-017-5228-6.

[3] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008, https://doi.org/10.1016/j.cor.2007.02.014.

[4] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals," *University Computing*, vol. 15, no. 2, pp. 58–69, 1993.

[5] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994, https://doi.org/10.1007/BF00175354.

[6] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm—A literature review," *in 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, Faridabad, India, 2019, pp. 380–384, https://doi.org/10.1109/COMITCon.2019.8862255.

[7] M. Lozano and C. García-Martínez, "Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report," *Computers & Operations Research*, vol. 37, no. 3, pp. 481–497, 2010, https://doi.org/10.1016/j.cor.2009.02.010.

[8] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2021, https://doi.org/10.1007/s11042-020-10139-6.

[9] Y. Wang and Q. Zhu, "A hybrid genetic algorithm for flexible job shop scheduling problem with sequence-dependent setup times and job lag times," *IEEE Access*, 2021, https://doi.org/10.1109/ACCESS.2021.3096007.

[10] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Computing Surveys*, vol. 45, no. 3, Art. no. 35, 2013, https://doi.org/10.1145/2480741.2480752.

[11] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986, https://doi.org/10.1109/TSMC.1986.289288.

[12] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, 1999, https://doi.org/10.1109/4235.771166.

[13] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, no. 1, pp. 3–12, 2005, https://doi.org/10.1007/s00500-003-0328-5.

[14] G. Squillero and A. Tonda, "Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization," *Information Sciences*, vol. 329, pp. 782–799, 2016, https://doi.org/10.1016/j.ins.2015.09.056.

[15] T. H. Nguyen, X. H. Nguyen, B. McKay, and Q. U. Nguyen, "Where should we stop? An investigation on early stopping for GP learning," *in Simulated Evolution and Learning (SEAL 2012), Lecture Notes in Computer Science*, vol. 7673. Berlin, Heidelberg, Germany: Springer, 2012, pp. 216–227, https://doi.org/10.1007/978-3-642-34859-4_39.

[16] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999, https://doi.org/10.1109/4235.771163.

[17] I. De Falco, A. Della Cioppa, and E. Tarantino, "Mutation-based genetic algorithm: Performance evaluation," *Applied Soft Computing*, vol. 1, no. 4, pp. 285–299, 2002, https://doi.org/10.1016/S1568-4946(02)00021-2.

[18] M. Schubotz, A. Satpute, A. Greiner-Petter, A. Aizawa, and B. Gipp, "Caching and reproducibility: Making data science experiments faster and FAIRer," *Frontiers in Research Metrics and Analytics*, vol. 7, 2022, https://doi.org/10.3389/frma.2022.861944.

[19] I. G. Tsoulos and V. Charilogis, "Introducing a parallel genetic algorithm for global optimization problems," *AppliedMath*, vol. 4, no. 2, pp. 709–730, 2024, https://doi.org/10.3390/appliedmath4020038.

[20] M. Furqan, E. Hartono, and M. I. Ongko, "Performance of arithmetic crossover and heuristic crossover in genetic algorithm based on alpha parameter," *IOSR Journal of Computer Engineering*, vol. 19, no. 5, pp. 31–36, 2017.

[21] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments," *Evolutionary Computation*, vol. 16, no. 3, pp. 385–416, 2008, https://doi.org/10.1162/evco.2008.16.3.385.

[22] B. Alhijawi and A. Awajan, "Genetic algorithms: Theory, genetic operators, solutions, and applications," *Evolutionary Intelligence*, vol. 17, no. 3, pp. 1245–1256, 2024, https://doi.org/10.1007/s12065-023-00822-6.

[23] M. Ayob and F. Al-Dhief, "Genetic algorithm based on natural selection theory for optimization problems," *Symmetry*, vol. 12, no. 11, p. 1758, 2020, https://doi.org/10.3390/sym12111758.

[24] A. Jenkins, V. Gupta, A. Myrick, and M. Lenoir, "Variations of genetic algorithms," arXiv preprint, arXiv:1911.00490, 2019, doi: https://doi.org/10.48550/arXiv.1911.00490.

[25] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimization problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013, https://doi.org/10.1504/IJMMNO.2013.055204.

[26] A. Borisenko, M. Haidl, and S. Gorlatch, "A GPU parallelization of branch-and-bound for multiproduct batch plants optimization," *The Journal of Supercomputing*, vol. 73, no. 2, pp. 639-651, 2017. https://doi.org/10.1007/s11227-016-1784-x.

[27] A. Borisenko and S. Gorlatch, "Efficient GPU-parallelization of batch plants design using metaheuristics with parameter tuning," *Journal of Parallel and Distributed Computing*, vol. 154, pp. 74-81, 2021. https://doi.org/10.1016/j.jpdc.2021.03.012.

***DMITRIY KRAVTSOV,** Is s Ph.D. candidate in Computer Science at Odessa National Maritime University, Odessa, Ukraine, enrolled in the doctoral program at the Faculty of Computer Science since 2023. Research activity is oriented toward the development of advanced machine learning methods for predictive analytics with applications in real estate valuation.*

Academic contributions include presentations and publications at international conferences. The paper "Machine learning model for house price predicting based on natural language text data analysis" was presented at the 6th Modern Machine Learning Technologies Workshop (MoMLeT 2024),

published in CEUR Workshop Proceedings, Vol. 3711, pp. 319–332. Another paper, "A machine learning method for real estate operation projects forecasting", was presented at the 5th International Workshop on IT Project Management (ITPM 2024) in Bratislava and published in CEUR Workshop Proceedings with DOI: 10.23939/IW_itpm2024.077.

Current research interests include explainable artificial intelligence (XAI), feature engineering from textual and geospatial data, optimization techniques for predictive modelling, and the design of automated valuation models.

***NIKOLAY POLETAEV,** received his M.Sc. degree in Physics from the Faculty of Physics, Odesa I. I. Mechnikov National University in 1975, and later obtained his Ph.D. (1993) and D.Sc. (2013) degrees in Physics, specializing in chemical physics and the physics of combustion and explosion.*

Since 1993, he has been leading the Department of Technological Combustion at the Institute of Combustion and Advanced Technologies, Odesa I. I. Mechnikov National University (ICAT ONU), where research was focused on the scientific and technological aspects of producing metal oxide nanoparticles by combustion methods. In 2015, he became Head of ICAT ONU.

Since 2019 he has served as a Full Professor of the Department of Technical Cybernetics and Information Technology named after prof. R. V. Merkt at Odesa National Maritime University. His current research interests include combustion in dispersed systems, dusty plasma, synthesis of metal oxide nanostructures via combustion methods, optical and spectral flame diagnostics, materials science, machine learning, and data mining.