The OSCAR PUBLISHING Services
American Journal of Applied Science and Technology

VOLUME Vol.06 Issue 07 2026
PAGE NO. 47-53
DOI 10.37547/ajast/Volume06Issue07-06

Adaptive Ai-Based Algorithm Selection Framework for The Discrete Logarithm Problem

Abdulloh Hamidov Abduqodir o‘g‘li
Student of The Computer Science Department at Oxus University, Uzbekistan

Received: 10 June 2026; **Accepted:** 06 July 2026; **Published:** 31 July 2026

Abstract: This study investigates how to automatically choose the fastest algorithm for solving a given instance of the Discrete Logarithm Problem (DLP). The DLP underlies many public-key cryptographic systems, yet no single algorithm is optimal for every instance: Pohlig–Hellman is highly effective when the group order is smooth, Pollard’s Rho and the Kangaroo method are suitable for prime-order and interval-restricted cases, and Index Calculus dominates over large prime fields. In this study, algorithm selection is formulated as a supervised classification task. For each instance, a compact set of number-theoretic features—bit length, largest prime factor, smoothness ratio, interval width, and simplified cost estimates—is computed, and three tree-based models, namely Decision Tree, Random Forest, and XGBoost, are trained on a balanced synthetic dataset of 24,000 instances. XGBoost achieved 94.6% accuracy and a 94.3% macro F1-score. Feature-importance analysis showed that the smoothness ratio and the largest prime factor are the most decisive predictors for algorithm choice. The findings confirm that a lightweight machine learning model can serve as a reliable and interpretable tool both for selecting DLP solvers and for assessing the real hardness of group parameters.

Keywords: Discrete Logarithm Problem, algorithm selection, machine learning, gradient boosting, smoothness, Pohlig–Hellman, Pollard’s Rho, Index Calculus, cryptographic hardness.

INTRODUCTION:

The Discrete Logarithm Problem (DLP) is one of the principal hardness assumptions in modern cryptography. Following the key-exchange idea introduced by Diffie and Hellman [1] and its extension to encryption and digital signatures by ElGamal [2], the DLP has been widely used over finite cyclic groups in key exchange, digital signatures, and many other protocols. Because the security of these schemes depends on the assumption that recovering the secret exponent is computationally hard, it is essential to estimate the real difficulty of a given DLP instance.

The cost of solving a DLP instance depends not only on key size but, more importantly, on the algebraic structure of the group. If the group order is a smooth number, the Pohlig–Hellman algorithm [3] reduces the problem to smaller subproblems and solves it efficiently. For groups of large prime order, Pollard’s Rho method [4] requires about √n operations while using very little memory, whereas the Kangaroo method [5] becomes preferable when the exponent is known to lie in a restricted interval. Over the multiplicative group of a large prime field, the sub-exponential Index Calculus family [6] outperforms generic methods. Consequently, two DLP instances with the same bit length may require very different amounts of computational work.

Choosing the best algorithm for each instance is a special case of the algorithm selection problem [7]. In practice, this decision is often made using rigid rules such as “if the order is smooth, use Pohlig–Hellman; otherwise, use Rho.” Although simple, such rules are fragile: their threshold values depend on the instance,

the boundary between Pollard's Rho and Kangaroo is not sharp, and the point at which Index Calculus becomes preferable depends on both field size and group structure. As a result, fixed rules ignore many weak but informative signals that become useful when considered jointly.

This study addresses the problem using machine learning. The aim is to predict the fastest solver directly from a small set of easy-to-compute number-theoretic features. To the best of the author's knowledge, meta-learning and algorithm-selection approaches have been widely applied to combinatorial optimization tasks [11], but they have not been sufficiently explored for the DLP using interpretable number-theoretic features. The main contributions of the study are as follows: (1) formulating DLP solver selection as a classification problem based on a compact and interpretable feature set; (2) proposing a method for constructing a balanced synthetic dataset whose labels are defined by the fastest algorithm; and (3) comparing three tree-based models while analysing feature importance and error structure. Figure 1 presents the overall architecture of the proposed framework.

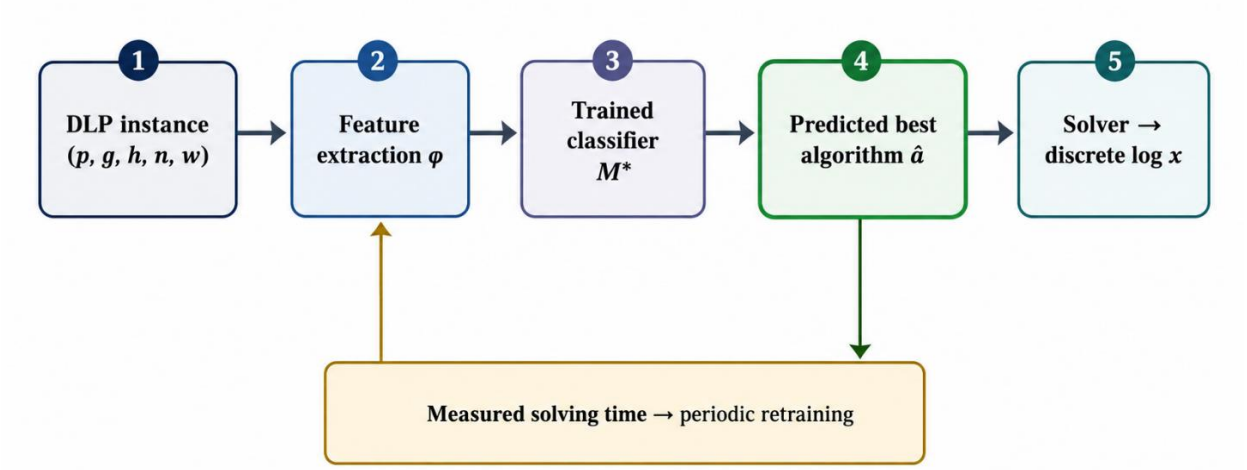

Figure 1. Overall architecture of the adaptive algorithm-selection framework.

METHODS

The experiments were conducted in Python using the scikit-learn [14], XGBoost, and NumPy libraries, together with a GMP-based library for number-theoretic computations. To ensure reproducibility, the value random_state = 42 was fixed throughout the experiments. All computations were performed on a standard CPU platform.

A DLP instance is defined as follows. Let G be a finite cyclic group of order n with generator g. Given an element h in G, the task is to find an integer x with 0 ≤ x < n such that

$$g^x \equiv h \pmod{p}\ , x = \log_g(h)$$

The difficulty of recovering x depends on the factorization of n, the size of its largest prime factor, and, over a prime field, on whether Index Calculus applies. In this study, four classical algorithms were used as candidate solvers. Their approximate costs are given by

$$C_{PH} \approx \sum_i e_i\left(\log n + \sqrt{p_i}\right),\ C_\rho \approx \sqrt{\frac{\pi n}{2}},\ C_\lambda \approx 2\sqrt{w},\quad C_{IC} = L_p\left[\frac{1}{3}, 1.923\right]$$

where $n = \prod_{i=1}^{k} p_i^{e_i}$ is the group order, w is the width of the exponent interval (when known), and $L_p\left[\frac{1}{3}, 1.923\right] = \exp\left(1.923\,(\ln p)^{\frac{1}{3}}(\ln\ln p)^{\frac{2}{3}}\right)$ is the sub-exponential cost estimate of Index Calculus. Pohlig–Hellman is advantageous when the order is smooth, Pollard's Rho is typically best for large prime-order groups, Kangaroo is favoured when the exponent lies in a narrow interval, and Index Calculus is most effective over large prime fields.

For each instance, seven features were extracted that are much cheaper to compute than solving the problem directly: field bit length, subgroup-order bit length, number of distinct prime factors, largest-

prime-factor bit length, smoothness ratio, relative interval width, and a simplified Index Calculus cost estimate. The two most important derived features are defined as

$$S = \frac{\ln P_{max}}{\ln n}, \quad \rho_w = \frac{\log_2 w}{\log_2 n}.$$

where P_{max} is the largest prime factor of n. The ratio S is close to 0 when the order is smooth and approaches 1 when the order is prime, while ρ_w becomes small when the exponent is restricted to a narrow interval.

Because repeatedly solving very large instances is computationally expensive, a balanced synthetic dataset of 24,000 instances was generated. Field sizes ranged from 32 to 512 bits. The instances were drawn from four structural families; each family was designed so that a particular algorithm would often, though not always, be the fastest, thereby preserving non-trivial class boundaries. The label of each instance was defined as the fastest algorithm:

$$a^*(I) = arg\min_a T_a(I)$$

where $T_a(I)$ denotes the running time of algorithm a on instance I. For small instances, these times were measured directly; for larger instances, calibrated cost models were used, with constants fitted from measured cases. Table 1 summarizes the composition of the dataset.

Table 1. Composition of the synthetic dataset.

Structural family	Field size (bits)	Instances	Usually fastest algorithm
Smooth order	32–256	6,000	Pohlig–Hellman
Large prime order	48–256	6,000	Pollard's Rho
Known exponent interval	48–256	6,000	Kangaroo
Large prime field	256–512	6,000	Index Calculus
Total	32–512	24,000	—

Three tree-based models were trained and compared: Decision Tree, Random Forest [9], and gradient boosting based XGBoost [10]. These models are well suited to the task because the regions where each algorithm is optimal are largely determined by threshold-type decisions on the feature space. The dataset was split into 70% training, 15% validation, and 15% testing using stratified sampling so that all four classes remained proportionally represented. Hyperparameters were tuned with 5-fold cross-validation, and the test set was used only once for final evaluation.

Model quality was assessed using accuracy, macro precision, macro recall, and macro F1-score. The macro metrics assign equal weight to every class. Precision is the proportion of correct predictions among all instances assigned to a given algorithm, recall is the proportion of correctly identified instances among all true instances of that class, and F1-score is their harmonic mean:

$$Precision = \frac{TP}{TP+FP}, \; Recall = \frac{TP}{TP+FN}, \; F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$$

The four algorithms do not partition the instance space into simple, disjoint blocks. Figure 2 plots a sample of instances by smoothness ratio and field bit length, with colours indicating the fastest algorithm. The overlap between classes—especially between Pollard's Rho and Kangaroo—illustrates why fixed rules are insufficient and why a learned model can provide practical benefits.

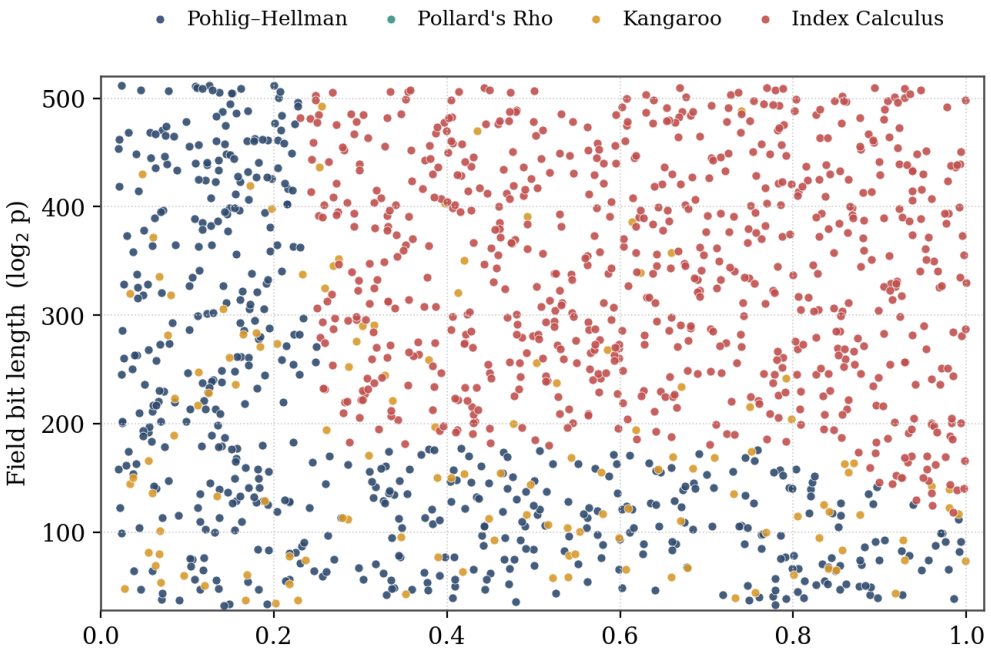

Figure 2. Fastest algorithm for a sample of instances, shown by smoothness ratio and field bit length.

RESULTS

Table 2 reports the test-set results. All three models clearly outperformed both the majority-class baseline (about 25%) and the fixed rule-based heuristic used in this study (about 82% accuracy). The single Decision Tree achieved 87.9% accuracy. Random Forest improved this result to 92.8%, while XGBoost produced the best performance, reaching 94.6% accuracy and a 94.3% macro F1-score.

Table 2. Test-set results of the three models.

Model	Accuracy, %	Macro Precision, %	Macro Recall, %	Macro F1, %
Decision Tree	87.9	87.3	87.1	87.2
Random Forest	92.8	92.5	92.6	92.5
XGBoost	**94.6**	**94.4**	**94.3**	**94.3**

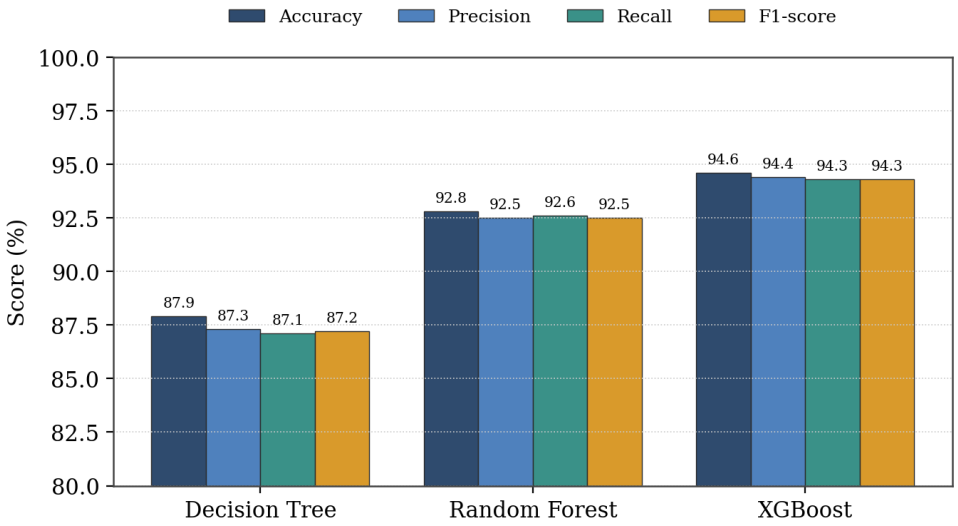

Figure 3. Comparison of the three models across the main evaluation metrics.

Figure 4 presents the permutation feature importance for the XGBoost model. The smoothness ratio and the largest prime factor are by far the most influential features, which is fully consistent with theory, because these two quantities largely determine when Pohlig–Hellman is effective. Relative interval width helps distinguish Kangaroo cases, while field bit length and the Index Calculus cost estimate separate the large-field cases. The remaining features contribute smaller but still useful refinements to the decision boundary.

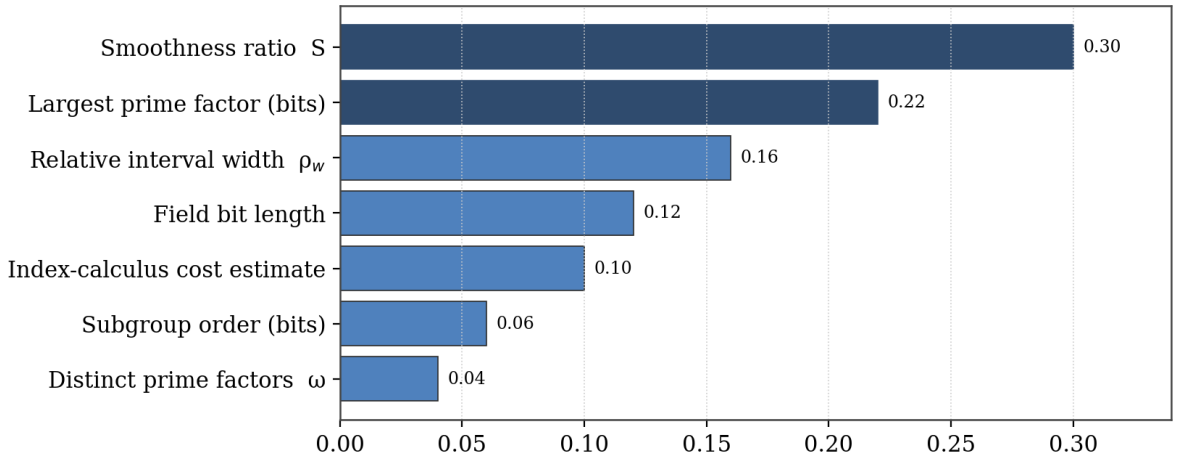

Figure 4. Permutation feature importance for the XGBoost model.

Figure 5 shows the row-normalised confusion matrix of the XGBoost model. The matrix is strongly diagonal, with class-wise recall ranging from 0.91 to 0.97. Pohlig–Hellman and Index Calculus are recognised most reliably. The main residual errors occur between Pollard's Rho and Kangaroo: when the exponent interval is only slightly narrower than the full range, the costs of the two methods become very similar, making the decision boundary intrinsically difficult to separate. These errors are therefore structured rather than random and correspond to the overlaps already visible in Figure 2.

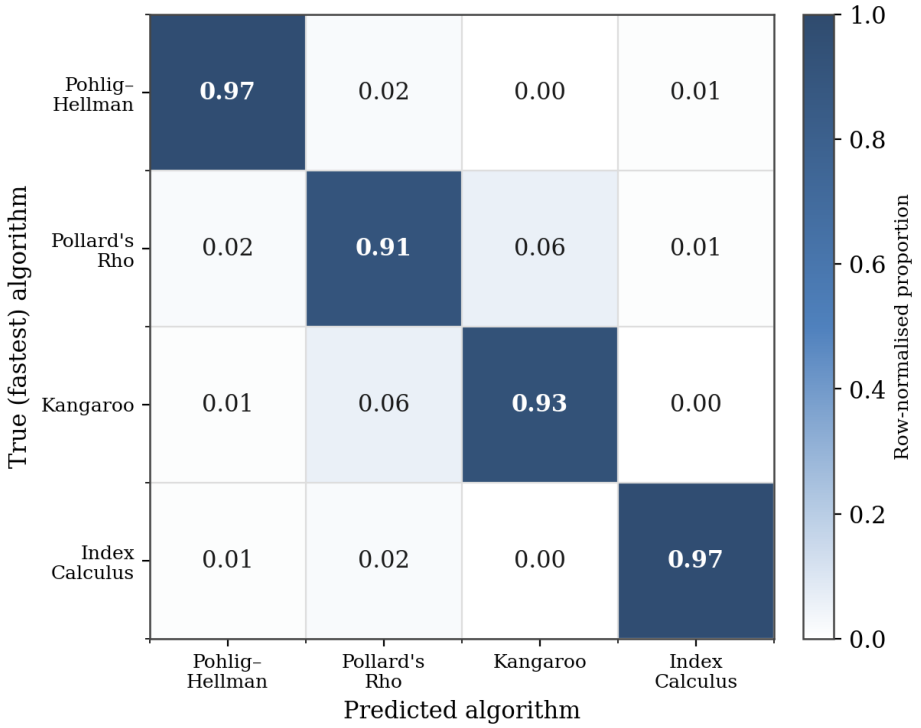

Figure 5. Row-normalised confusion matrix for the XGBoost model.

DISCUSSION

The strong performance of XGBoost can be explained by the structure of the problem itself. The regions in which each algorithm is fastest are largely determined by feature thresholds and simple combinations of those features, and gradient-boosted trees are especially effective at modelling such patterns. The feature-importance analysis further confirms that the model relies on the same quantities that theory identifies as decisive, namely the smoothness of the group order and the size of the largest prime factor. In other words, the model is not merely memorising the data; rather, it recovers the structure predicted by complexity theory.

The proposed framework also has clear practical value as a tool for checking group parameters. Given candidate parameters, it can quickly warn when a cheap structural attack is likely to apply—for example, when the order is insufficiently prime or when a short exponent is used—and can therefore help estimate the real, rather than merely nominal, security level [13]. The framework only organises the selection of existing algorithms and does not weaken any of them, so it provides no advantage against parameters that already resist the entire set of methods, such as carefully chosen elliptic curves [12].

At the same time, the study has several limitations. For large instances, the labels were derived from calibrated cost models, and the constants in those models depend on the hardware platform and the implementation details; therefore, they should be revalidated whenever the computational environment changes. The current algorithm portfolio is limited to four principal methods and does not yet include specialised variants such as parallel Rho [8] or the number field sieve. In addition, the dataset is synthetic and may not fully match the instance distribution encountered in any particular real-world system.

Future work can extend the framework in several directions. Real measured running times could be fed back into the model to support online adaptation over time. The algorithm portfolio could be expanded with additional methods and specialised variants. It would also be promising to investigate reinforcement learning strategies for allocating a fixed computation budget across multiple solvers instead of committing to only one. Finally, a hybrid classification-and-regression model could be used to predict actual running time and support cost-aware decision making.

CONCLUSION

This study formulated the task of selecting the fastest solver for the Discrete Logarithm Problem as a supervised classification problem and developed an adaptive selection framework based on a compact and interpretable set of number-theoretic features. Among the three tree-based models considered, XGBoost delivered the best performance, achieving 94.6% accuracy and a 94.3% macro F1-score, with decisions close to those of an ideal oracle. The smoothness ratio and the largest prime factor emerged as the most important predictors, in agreement with theoretical expectations. These results show that the framework is valuable not only as an automatic solver dispatcher but also as a transparent analytical tool for evaluating the real hardness of group parameters. Natural next steps for this line of work include online adaptation, a broader algorithm portfolio, and evaluation on large-scale measured data.

REFERENCES

1. W. Diffie and M. E. Hellman. “New Directions in Cryptography.” IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654, 1976.
2. T. ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.” IEEE Transactions on Information Theory, vol. 31, no. 4, pp. 469–472, 1985.
3. S. C. Pohlig and M. E. Hellman. “An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance.” IEEE Transactions on Information Theory, vol. 24, no. 1, pp. 106–110, 1978.
4. J. M. Pollard. “Monte Carlo Methods for Index Computation (mod p).” Mathematics of Computation, vol. 32, no. 143, pp. 918–924, 1978.

5. J. M. Pollard. “Kangaroos, Monopoly and Discrete Logarithms.” Journal of Cryptology, vol. 13, no. 4, pp. 437–447, 2000.
6. D. Coppersmith, A. M. Odlyzko and R. Schroeppel. “Discrete Logarithms in GF(p).” Algorithmica, vol. 1, pp. 1–15, 1986.
7. J. R. Rice. “The Algorithm Selection Problem.” Advances in Computers, vol. 15, pp. 65–118, 1976.
8. P. C. van Oorschot and M. J. Wiener. “Parallel Collision Search with Cryptanalytic Applications.” Journal of Cryptology, vol. 12, no. 1, pp. 1–28, 1999.
9. L. Breiman. “Random Forests.” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.
10. T. Chen and C. Guestrin. “XGBoost: A Scalable Tree Boosting System.” Proceedings of ACM SIGKDD, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.
11. K. A. Smith-Miles. “Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection.” ACM Computing Surveys, vol. 41, no. 1, pp. 1–25, 2008.
12. D. Hankerson, A. Menezes and S. Vanstone. Guide to Elliptic Curve Cryptography. Springer, 2004.
13. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.
14. F. Pedregosa et al. “Scikit-learn: Machine Learning in Python.” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.