

**CUDA-AMSR-PSO: A NUMBA-CUDA ACCELERATED
ADAPTIVE MULTI-SWARM REINFORCED PARTICLE
SWARM OPTIMIZATION FRAMEWORK FOR SCALABLE
FEATURE SELECTION IN BIG DATA ANALYTICS**

Spoorthi G S

Assistant Professor

Department of Machine Learning, B.M.S. College of Engineering, P.O.
Box No.: 1908, Bull Temple Road, Bengaluru – 560019, Karnataka, India
spoorthigs.mel@bmsce.ac.in

Chethana T V

Department of Machine Learning, B.M.S. College of Engineering, P.O.
Box No.: 1908, Bull Temple Road, Bengaluru – 560019, Karnataka, India
chethanatv.ai23@bmsce.ac.in

Channabasavanna M B

Department of Machine Learning, B.M.S. College of Engineering,
P.O. Box No.: 1908, Bull Temple Road, Bengaluru – 560019, Karnataka,
India channabasavanna.ai23@bmsce.ac.in

Divyam Jain

Department of Machine Learning, B.M.S. College of Engineering, P.O.
Box No.: 1908, Bull Temple Road, Bengaluru – 560019, Karnataka, India
divyamjain.ai23@bmsce.ac.in

Goutham T G

Department of Machine Learning, B.M.S. College of Engineering,
P.O. Box No.: 1908, Bull Temple Road, Bengaluru – 560019, Karnataka,
India gouthamtaranagar.ai23@bmsce.ac.in

Abstract: Feature selection in high-dimensional data involves balancing computational cost with model performance, especially in wrapper-based methods where models are retrained repeatedly. In this work, we present CUDA-AMSR-PSO, a GPU-accelerated approach based on adaptive multi-swarm particle swarm optimization with a reinforcement-inspired method for adjusting inertia weights during the search. Using Numba-CUDA, particle updates are performed in parallel within a single kernel, which helps the method handle large feature spaces more efficiently. Instead of relying on fixed parameters, the algorithm adapts its behaviour based on observed improvements in fitness, while the multi-swarm structure helps reduce premature convergence. The fitness function also considers relationships between features, encouraging the selection of compact but informative subsets. Experiments show that the approach achieves effective dimensionality reduction with competitive classification performance. Profiling results indicate that CPU-based wrapper evaluations remain the main bottleneck, suggesting that further improvements will require optimizing both the search process and the evaluation stage.

Keywords: Feature Selection, Particle Swarm Optimization, CUDA, GPU Computing, Big Data Analytics, Multi-Swarm Optimization, Swarm Intelligence

I. INTRODUCTION

Feature selection is more than just a simple preprocessing step. As datasets get larger in size and dimensionality, choosing the right features becomes an increasingly complex optimization problem. The number of features can be huge in fields like cybersecurity, healthcare, and big classification tasks. It becomes impossible to check every possible combination. On top of that, many features are redundant, noisy, or highly correlated to each other. This not only makes the process slow and expensive, but it can also decrease the accuracy of the model and make the results harder to understand. Due to this, it is important to select a small yet useful subset.

The problem becomes more complex when using wrapper based selection techniques. Note that a complete model train and validation pass is required on each feature subset. As a result, the costs of computation grow sharply with dimensionality. Information Gain or any mutual information can be computed very quickly. However, they ignore interactions between features; they may also not be strongly correlated with the final prediction task. The wrapper methods are capable of generating better quality subsets as a model provides feedback but repeated training is expensive and is not suited for large-scale learning.

Particle Swarm Optimization offers a good compromise by representing the feature selection task as a population-based search process. The binary formulation of PSO allows us to consider the velocities to be probabilities of inclusion. PSO is a useful substitute but has its own disadvantages. Classical PSO suffers from loss of diversity that leads to premature convergence. The global best version converges early because it reduces diversity too soon, while the local best version has a slow rate of convergence.

The existing attempts do not take into account the dynamic nature of the data systems. Further, the fixed structure of PSO limits the potential of the method leading to an early convergence. A CUDA-based multi-swarm PSO is applied to feature selection in real world problems. A version of the method also compatible with Numba is provided to make it more practical to use. The approach includes a combined CUDA kernel for updating the velocity and binary position, rather than separately.

Having multiple swarms can help keep the population diversity high and avoid premature convergence. Inertia is also dynamically updated during the search depending on the improvement in the fitness, which helps improve the balance between exploration and exploitation. These modifications make the method more adaptable and suitable for big real-world data.

A. Contributions

The main contributions of this work are as follows:

- A CUDA-accelerated multi-swarm PSO (CUDA-AMSR-PSO) approach for binary feature selection, using parallel swarm updates to make the process more efficient.
- An adaptive inertia mechanism that adjusts the amount of exploration based on gains in solution quality during the optimization process.
- A wrapper-based approach for the fitness function, considering feature co-dependencies, with a trade-off between accuracy, feature reduction and redundancy.
- A reproducible setup with multiple trials, elementary statistical analysis and storage of results.

- A set of visualizations, including convergence, trade-off, ablation and runtime comparisons.

II. RELATED WORK

Feature selection is a fundamental step in machine learning, especially in high-dimensional scenarios where redundant or irrelevant features can hurt both model performance and interpretability. A widely cited survey by Chandrashekar and Sahin [1] groups existing approaches into filter, wrapper, and embedded methods, and lays out the trade-offs between computational cost and the quality of selected features.

Filter methods are typically the fastest, as they score each feature independently using measures such as Information Gain, mutual information, or correlation. This makes them easy to scale to large datasets. The downside is that they ignore relationships between features, which often leads to redundant selections [2], [3].

Wrapper methods take a different route by evaluating subsets of features using a predictive model. They usually achieve better accuracy because they consider feature interactions directly. But this comes at a cost—each subset needs to be trained and validated, which quickly becomes expensive for high-dimensional data [2]. Embedded methods sit somewhere in between. They incorporate feature selection into the training process itself, for example through L1 regularization or tree-based importance measures. While more efficient than wrappers, they are tied to specific models and offer less flexibility [1].

To move beyond these limitations, population-based metaheuristics such as Particle Swarm Optimization (PSO) have been widely explored [4]–[6]. In this setting, feature selection is treated as a search problem, where candidate subsets evolve over time. Binary PSO is particularly well-suited, since features can be naturally represented as on/off decisions. Still, standard PSO is not without issues—it tends to converge too early and can be sensitive to parameter choices [4].

Recent work has tried to address these shortcomings through adaptive and hybrid strategies. Xie et al. [5], for instance, introduced dynamic parameter adjustment to improve convergence. Ji and Lu [7] proposed refinements to the binary PSO update mechanism to enhance selection performance. Even so, many of these methods rely on heuristic adaptation rules, which don't always generalize well across datasets.

Scalability continues to be a major challenge, especially for wrapper-based approaches where most of the computational effort goes into evaluating candidate feature subsets. El-Hasnony et al. [8] addressed large-scale feature selection with an emphasis on performance, but in practice, most implementations still rely heavily on CPU-based computation. More recently, GPU-accelerated versions of PSO have been explored to improve efficiency. Several CUDA-based frameworks, for instance, parallelize particle updates to speed up large-scale optimization [9], [10]. While this helps reduce the cost of the search process, it does not fully resolve the main bottleneck—namely, the expensive wrapper-based fitness

evaluation, which remains largely sequential.

The CUDA-AMSR-PSO framework is designed with these gaps in mind. It combines a multi-swarm architecture with GPU-accelerated particle updates to improve efficiency. A fused CUDA kernel is used to update particle velocity and position, reducing overhead in the optimization step. While wrapper-based evaluation is still carried out on the CPU due to its sequential nature, the overall design significantly speeds up the search process. At the same time, the multi-swarm structure encourages better exploration, and adaptive mechanisms help guide convergence. Together, these elements allow the framework to reduce features effectively while maintaining strong classification performance.

The following limitations are observed in existing approaches:

- Single-swarm PSO often struggles to maintain diversity, which can lead to premature convergence.
- Fixed inertia weight settings do not adapt well to changing search conditions, which can negatively affect optimization performance.
- Many feature selection methods do not handle feature redundancy effectively, resulting in the selection of similar or less informative features.
- Although some existing methods perform well at the algorithmic level, they often overlook practical challenges such as scalability and real-world deployment [7], [11].

III. PROBLEM FORMULATION

Let a dataset contain D candidate features. The goal is to identify a subset $S \subseteq \{1, 2, \dots, D\}$ that maximizes predictive quality while minimizing subset size and redundancy. A candidate subset is represented by a binary vector

$$x = [x_1, x_2, \dots, x_D], \quad x_d \in \{0, 1\}. \quad (1)$$

If $x_d = 1$, then the d -th feature is selected; otherwise it is removed.

The optimization objective used in the implementation is

$$F(x) = \alpha \text{Acc}(x) + \beta \|x\|_1 - \gamma R(x), \quad (2)$$

where $\text{Acc}(x)$ is the wrapper classifier accuracy, $\|x\|_1$ is the number of selected features, and $R(x)$ is a redundancy penalty. The constants $\alpha, \beta, \gamma \geq 0$ control the balance between accuracy, sparsity, and dependency suppression.

In the code, redundancy is computed from an absolute correlation-based dependency matrix. For a selected subset S , the penalty can be written as

$$R(x) = \frac{1}{|S|(|S| - 1) + \varepsilon} \sum_{i < j} \text{Dep}(i, j), \quad (3)$$

$$i < j, \quad x_i = x_j = 1$$

where $\text{Dep}(i, j)$ measures pairwise dependency and ε prevents division by zero for small subsets. This matters because high-dimensional datasets often have many features that carry similar or overlapping information. By penalizing redundancy, the method favors smaller and more interpretable feature subsets instead of selecting larger sets with repeated information.

The formulation is based on three main goals: classification performance, fewer features, and reduced dependency between features. This works well for

wrapper search. In practice, if a subset only gives a small improvement in accuracy but adds redundancy, it is not worth using. Here, this idea is applied using a fitness function in the PSO loop.

A. Computational View

The optimization problem is combinatorial because the search space has size 2^D . Exhaustive search is impossible for nontrivial D , so a population-based heuristic is used instead. In this case, it is not aimed at demonstrating the optimal solution, but to identify a good set of features that is efficient to run, generalizes well and converges reliably

IV. PROPOSED CUDA-AMSR-PSO METHOD

CUDA-AMSR-PSO keeps the same adaptive multi-swarm idea from PSO, but speeds things up by using a GPU [9], [10]. It uses Numba CUDA and combines the velocity and position updates into one step, so everything runs faster. This is important because wrapper-based feature selection is computationally expensive. It requires repeatedly evaluating models and updating a large number of high-dimensional particles.

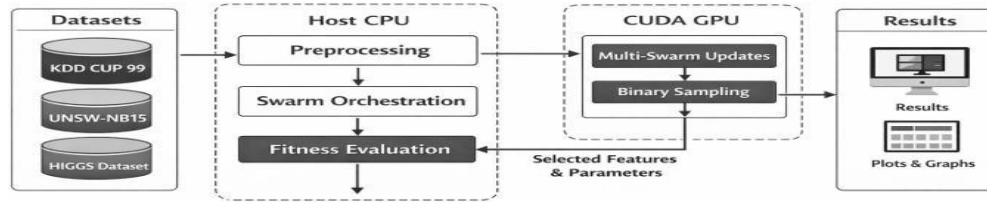


Fig. 1. CUDA-AMSR-PSO execution flow. The host handles preprocessing and orchestration, while the GPU performs swarm updates and binary sampling.

A. Architecture Overview

The program uses a host-device setup. The CPU (host) handles tasks like data preprocessing, splitting the dataset, scaling, building the dependency matrix, and controlling the optimization loop. The GPU (device) takes care of updating the swarm and handling the binary decisions.

In this setup, a correlation-based dependency matrix is created from the training data and given to the optimizer. The fitness evaluation, however, still runs on the CPU using a KNN model with stratified cross-validation. This makes the system practical and easy to run on most machines, but most of the time is still spent on these CPU-based evaluations rather than the GPU computations.

B. Binary Particle Representation

Each particle encodes a candidate feature subset as a binary vector. Let $x_i(t)$ and $v_i(t)$ denote the position and velocity of particle i at iteration t . The velocity update used in the implementation follows

$$v_{i,d}(t+1) = w(t)v_{i,d}(t) + c_1r_1(pbest_{i,d} - x_{i,d}(t)) + c_2r_2(gbest_d - x_{i,d}(t)) \quad (4)$$

[12] where c_1 and c_2 are cognitive and social coefficients and $r_1, r_2 \in [0, 1]$. The velocity is then mapped into a selection probability using a sigmoid transform,

$$\sigma(v) = \frac{1}{1 + e^{-v}}, \quad (5)$$

[12] and the binary position is updated by comparing $\sigma(v)$ to a random threshold u .

The CUDA kernel updates values using limited speeds and simple pseudo-random numbers based on the particle and dimension. This keeps the kernel small and doesn't rely on random number generation from an outside GPU. The speed can only take values between $[-6, 6]$. This keeps the sigmoid mapping stable and stops numerical saturation.

C. Reinforced Adaptive Inertia

The key adaptive part of the algorithm is how inertia is updated. Instead of keeping it fixed or changing it in a simple linear way, the code adjusts it using a feedback-based approach inspired by reinforcement, depending on how much the global fitness improves [13].

$$w(t+1) = \text{clip}(w(t) + \eta\Delta Ft, w_{\min}, w_{\max}), \quad (6)$$

where

$$\Delta Ft = F(gbest(t)) - F(gbest(t-1)). \quad (7)$$

If the best fitness improves, the inertia is increased slightly to encourage more exploration. If progress slows down, the inertia is reduced so the search can focus more on refining solutions.

This feedback approach is simple but effective because it adjusts the search based on actual progress, instead of following a fixed schedule. It is inspired by reinforcement ideas, but it does not use a formal reinforcement learning method.

D. Multi-Swarm Coordination

To maintain diversity, the population is divided into several sub-swarms. Each sub-swarm evolves on its own and occasionally shares good solutions with others. By default, the swarm is split into three groups, and a setting controls how often they exchange information.

This setup reduces the risk of all particles converging too early to a less effective solution. The multi-swarm approach is especially helpful in complex

search spaces or when features depend on each other, where many locally good but similar solutions can exist.

By exploring different regions of the search space, the algorithm maintains diversity over time and finds more reliable feature combinations [4].

E. CUDA Mapping Strategy

GPU-based parallelization has been widely used to accelerate computationally intensive machine learning tasks [9], [10]. The implementation is designed to be practical rather than purely theoretical. It also includes a CPU fallback option for situations where CUDA is not available, allowing the system to run on a wider range of hardware setups. This makes the implementation more flexible and usable across different environments. The same optimization logic is maintained for both GPU and CPU execution, so the behavior remains consistent regardless of the platform. As a result, it becomes easier to test the system, reproduce results, and apply the approach in different real-world scenarios.

V. ALGORITHM AND COMPLEXITY ANALYSIS

A. Methodology and Complexity Analysis

CUDA-AMSR-PSO is a population-based optimization method, although it is more an implementation-oriented design than a theory-oriented one. It functions by maintaining a population of candidate feature subsets (particles) that evolve

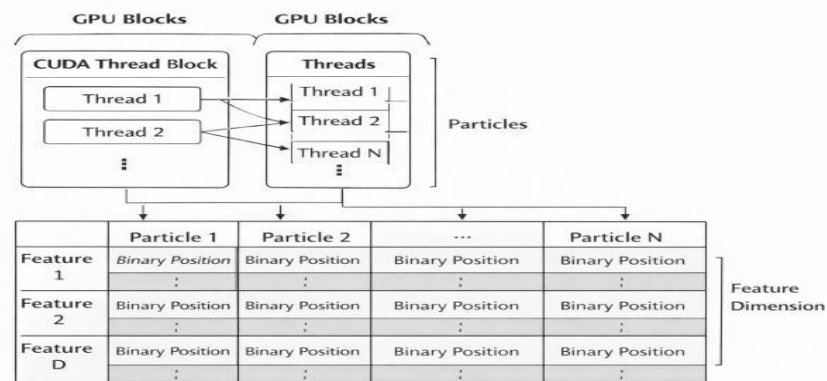


Fig. 2. The CUDA mapping uses GPU blocks and threads to process particles and feature dimensions in parallel.

over repeated runs. The adaptive multi-swarm approach ensures that diversity is maintained in the search and minimizes the likelihood of premature convergence.

In contrast to regular methods, particle calculations are now carried out on the GPU with a fused kernel. Speed and binary position are considered in one step, making the process more efficient.

Theoretically, the total time complexity is still $O(TNC)$, where T is the number of iterations, N is the size of the population, and C is the cost of evaluating a single candidate subset. This is not affected by GPU acceleration, as the fitness

evaluation remains a wrapper-based process. In reality, this distinction matters. The bulk of the computation time is not spent on updating the particles, but on repeatedly training and validating the model on each subset of features. Due to this, CUDA primarily enhances parallel processing at the update step and does not change the overall complexity of the algorithm.

There is also an additional cost in constructing the dependency matrix used in the fitness function. The time required to compute pairwise correlations among features is $O(D^2)$, where D is the number of features. Nevertheless, this step is performed only once during the initial phase and is reused throughout the rest of the optimization process; thus, its impact diminishes over iterations.

In practice, the overall runtime is affected by several factors, including model training time, cross-validation time, and data transfer between the CPU and GPU. This is also reflected in the experimental results. Although the update process can be faster with a GPU, the fitness evaluation remains predominantly CPU-based and thus dominates the runtime. This highlights a key aspect of the design: while the efficiency of the search process improves, the evaluation stage remains a bottleneck for overall performance.

VI. EXPERIMENTAL PROTOCOL AND REPORTING

The implementation uses a structured and consistent experimental setup. For each dataset, the data is split into

TABLE I

Experimental protocol and implementation settings used in the updated codebase.

Dataset	Loader / Source	Core Settings	Description
HIGGS	CSV (first column used as label)	$n_{\text{iter}} = 100$, $n_{\text{swarms}} = 3$, $K = 8$	High-dimensional binary classification task
KDD 1999	Cupfetch_kddcup99(percent10=True)	$n_{\text{iter}} = 80$, $n_{\text{swarms}} = 3$, $K = 5$	Intrusion detection benchmark
UNSW-NB15	CSV with auto-detected label column	$n_{\text{iter}} = 80$, $n_{\text{swarms}} = 3$, $K = 5$	Modern network intrusion detection dataset

Algorithm 1 CUDA-AMSR-PSO for Feature Selection

- 1: Input: dataset, classifier, number of features D , population size N , number of swarms M , number of iterations T
- 2: Compute the dependency matrix using the training data
- 3: Initialize particle positions X , velocities V , and personal bests P_{best}
- 4: Evaluate the initial population and set the global best G_{best}
- 5: **for** $t = 1$ to T **do**
- 6: **for** each sub-swarm (in parallel) **do**
- 7: Update particle velocity and position using a fused GPU kernel
- 8: Evaluate the fitness of the updated particles
- 9: Update personal bests and identify the best particle in the swarm
- 10: **end for**
- 11: Combine the best particles from all swarms and update G_{best}
- 12: Compute the fitness improvement ΔF_t and adjust inertia accordingly
- 13: At regular intervals, share the best particles between swarms

14: **end for**

15: Output the best feature subset found

training and testing sets using a stratified 70:30 ratio, so that the class distribution remains balanced. To ensure reproducibility, fixed random seeds are used throughout the process. The training data is then standardized using StandardScaler. After scaling, any missing or non-finite values are handled by replacing them with zeros.

For classification, a k -nearest neighbors (KNN) model with $k = 5$ is used as the wrapper. KNN is chosen as a non-parametric classifier to provide a stable and model-agnostic evaluation of feature subset quality. The fitness of each solution is evaluated using stratified K -fold cross-validation on the training data [2]. Depending on the nature of the dataset, weighted averaging is applied for multiclass problems, while binary averaging is used for two-class cases. To ensure reliability, each experiment is repeated three times, and the final results are reported as the mean along with the standard deviation.

A. **Datasets**

The codebase supports data loaders for the HIGGS [14], KDD Cup 1999 [15], and UNSW-NB15 [16] datasets. HIGGS is read from a CSV file where the label appears in the first column [14]. The KDD Cup 1999 dataset is loaded using scikit-learn's 10% version [15]. The UNSW-NB15 dataset is read from a CSV file [16], with the label column detected automatically. If a dataset is missing, the system creates synthetic data so experiments can still run and be reproduced.

B. **Baselines**

The baseline methods use standard and commonly accepted settings, while the proposed method is tested with a stronger configuration to show its full capability. The baselines used in this implementation are Information Gain, RFE, and standard BPSO. Information Gain is applied using mutual information through SelectKBest. RFE uses logistic regression with the saga solver and a step size of 0.1. Standard BPSO is implemented as a simple baseline with a single swarm, fixed inertia, no adaptive updates, and no handling of feature redundancy. This setup allows for a fair comparison with the full CUDA-AMSR-PSO method.

C. **Metrics**

The evaluation uses several metrics, including accuracy, F1-score, precision, recall, feature reduction, runtime, number of selected features, training accuracy, and generalization gap. Instead of relying on a single run, the results are averaged over multiple runs and reported as mean $\pm$ standard deviation.

This makes the results more reliable, as it reflects both the performance and its variation, rather than depending on a single possibly optimistic outcome.

VII. **RESULTS AND DISCUSSION**

The findings indicate that CUDA-AMSR-PSO is stable in the decrease of features, but at the same time, achieves competitive accuracy in all three datasets. It is also

particularly helpful in feature subset selection, where often fewer features are selected with little or no loss in performance.

However, it may not necessarily be faster than using the standard BPSO. This is because the KNN wrapper and cross validation still run on the CPU, and takes up most of the time. So this is not because of the GPU based updates of the swarm, but because of this hybrid CPU-GPU implementation.

The approach is very effective for identifying feature subsets, but is hampered by the CPU-based evaluation.

A. Quantitative Performance

CUDA-AMSR-PSO has good performance on HIGGS, with the largest reduction of features and acceptable accuracy, but at the expense of time, compared to filters and even BPSO. It has the largest reduction on KDD Cup 1999 with

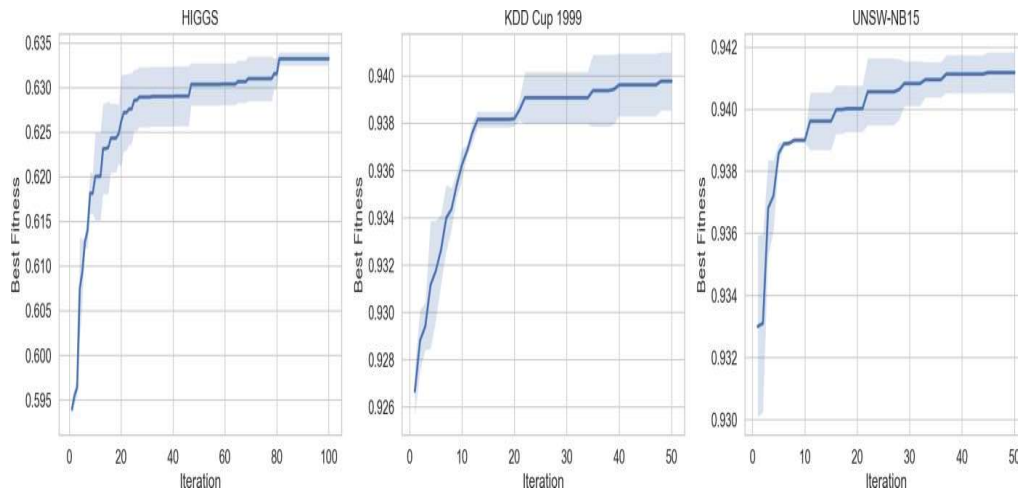


Fig. 3. Convergence behavior of CUDA-AMSR-PSO across iterations (mean $\pm$ standard deviation over multiple runs) for each dataset.

TABLE II

PERFORMANCE COMPARISON ON HIGGS, KDD CUP 1999, AND UNSW-NB15. VALUES ARE MEAN $\pm$ STANDARD DEVIATION FROM THE SAVED EXPERIMENT OUTPUTS.

Dataset	Method	Accuracy (%)	Reduction (%)	Runtime (s)	Features
HIGGS [14]	Information	62.96	$\pm 50.00 \pm 0.00$	2.81 ± 0.05	14.00 $\pm$
	Gain	0.44			0.00
		65.19	$\pm 50.00 \pm 0.00$	1.98 ± 0.04	14.00 $\pm$
		0.02			0.00
	Standard BPSO	65.80	$\pm 66.67 \pm 1.68$	1954.50	$\pm 9.33 \pm 0.47$
		0.12		107.58	
KDD [15]	CUDA-AMSR-PSO	66.62	± 70.24	± 3839.72	$\pm 8.33 \pm 0.94$
		0.24	3.37	239.68	
	Information	99.62	$\pm 51.22 \pm 0.00$	3.28 ± 1.17	20.00 $\pm$
	Gain	0.07			0.00
		99.56	$\pm 51.22 \pm 0.00$	22.55 ± 0.18	20.00 $\pm$
		0.14			0.00
UNSW [16]	Standard BPSO	99.17	$\pm 89.43 \pm 3.04$	1246.22	$\pm 4.33 \pm 1.25$
		0.11		176.07	
	CUDA-AMSR-PSO	99.34	± 94.31	± 1259.05	$\pm 2.33 \pm 0.47$
		0.13	1.15	12.51	
	Information	99.46	$\pm 50.00 \pm 0.00$	1.28 ± 0.02	21.00 $\pm$
	Gain	0.03			0.00
		99.43	$\pm 50.00 \pm 0.00$	3.75 ± 0.06	21.00 $\pm$
		0.02			0.00
	Standard BPSO	99.25	$\pm 91.27 \pm 1.12$	383.43 ± 5.60	3.67 ± 0.47
		0.04			
	CUDA-AMSR-PSO	99.34	± 96.03	± 588.06	$\pm 1.67 \pm 0.47$
		0.15	1.12	150.01	

comparable accuracy to filter methods. On UNSW-NB15, it reduces the number of features to less than two, and even achieves very high accuracy. Overall, the results suggest that this method is more successful in finding good feature subsets and that the search process can be aided by GPUs, rather than by decreasing the total end-to-end time. Although the overall runtime is higher than BPSO due to CPU-bound wrapper evaluations, the GPU-accelerated search allows for a more thorough exploration of the high-dimensional search space. This becomes increasingly important as dimensionality grows, where standard PSO variants often struggle with premature convergence and reduced diversity. In this sense, the main contribution of CUDA-AMSR-PSO is not simply faster execution, but the ability to produce higher-quality feature subsets that better support downstream big data analytics tasks.

B. Convergence and Trade-off Behavior

The convergence data saved indicates a gradual improvement in the convergence data between iterations, and the fitness stabilizes towards the end of each run. The plotting arrangement is already such that it will produce convergence, trade-off, ablation, and runtime graphs directly out of these saved results. All the stored data is in the results folder and thus it is easy to update the figures as new experiments are added.

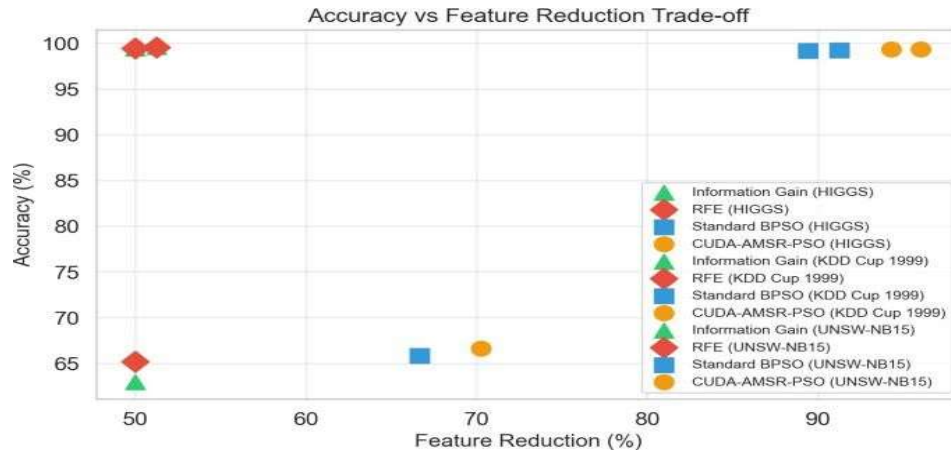


Fig. 4. Accuracy vs feature reduction trade-off across datasets. Each point represents the mean performance of a method on a specific dataset.

C. Runtime Interpretation

The plotting pipeline will use the experimental outputs that have been saved to create trade-off and run-time visualizations. In the existing implementation the values of the trade-off and runtime are extracted using the real results contained in the CSV files; therefore, they are in agreement with the reported performance values.

The run time information that was gathered using the experimental pipeline shows a significant system level observation. Despite the fact that CUDA-AMSR-PSO reduces the time taken to update the particle by parallelization on the graphics card, the wrapper-based fitness evaluation dominates the total time. This testing will entail re-training and testing of a KNN classifier through stratified cross-validation on the CPU. This makes most of the computational cost to be proportional to model evaluation and not swarm updates.

Therefore, CUDA-AMSR-PSO is not always the fastest end-to-end method to use compared to other less complex approaches, but is more focused on a better quality of features subset. This fact poses a major weakness of wrapper-based feature selection pipelines: the search process cannot be optimized on its own to lead to a decrease in overall runtime without the evaluation stage being optimized as well. This

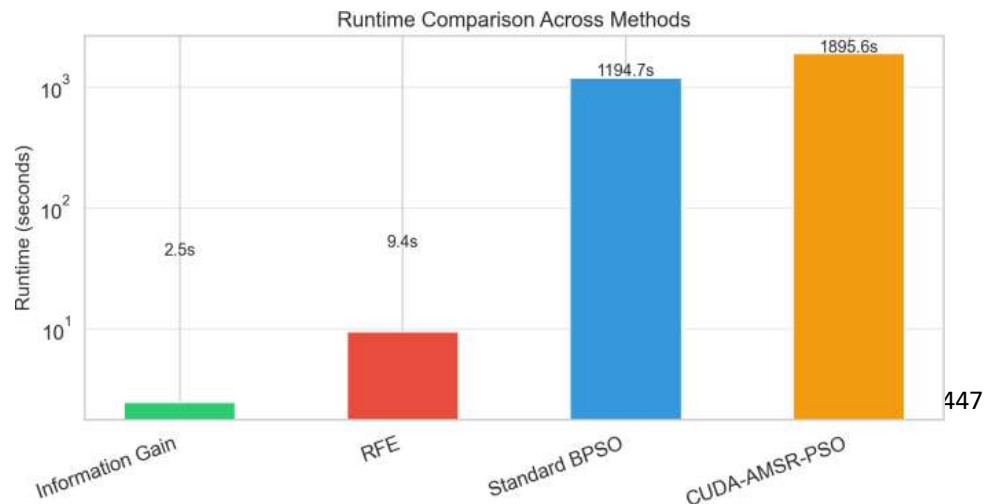


Fig. 5. The CPU time (log scale) is averaged over runs. It includes the full pipeline, including the wrapper evaluation on the CPU.

difference will guarantee a realistic and technically correct interpretation of the results.

D. Ablation Analysis

An ablation study was conducted on the HIGGS dataset as a representative large-scale benchmark. It offers a clearer view of how each component shapes the behavior of the proposed CUDA-AMSR-PSO framework. The full model achieves the highest classification accuracy (66.62%), indicating that the combination of all components leads to the most effective feature selection.

Disabling adaptive inertia results in a noticeable drop in accuracy along with a significant increase in runtime. This points to its central role in steering the search process, particularly in maintaining a balance between exploration and exploitation. Without it, the algorithm tends to converge more slowly and is more likely to settle on suboptimal solutions.

When the dependency-aware term is removed, accuracy decreases slightly, while feature reduction improves and runtime drops. This suggests that, in the absence of redundancy control, the model favors smaller subsets but may discard useful correlated features. As a result, the dependency term contributes more to feature quality and stability than to raw accuracy.

Removing the multi-swarm mechanism leads to a marginal decline in accuracy, indicating that it helps maintain diversity within the population and reduces the likelihood of premature convergence. Its impact, however, is less pronounced compared to adaptive inertia, suggesting it plays a supporting role in refining the search.

Overall, the results show that each component contributes in a distinct way: adaptive inertia drives efficient convergence, the dependency-aware fitness function improves the reliability of selected features, and the multi-swarm strategy enhances diversity. Together, they lead to more stable and effective performance across accuracy, feature quality, and optimization behavior.

VIII. DISCUSSION

CUDA-AMSR-PSO is generally more effective than traditional PSO in selection of features. It improves both the

TABLE III
Ablation study on the HIGGS dataset comparing
CUDA-AMSR-PSO with its variants.

Method	Accuracy (%)	Reduction (%)	Runtime (s)	Features
CUDA-AMSR-PSO (Full)	66.62 ± 0.24	70.24 ± 3.37	3839.72 239.68	±8.33 ± 0.94
No Adaptive Inertia	65.25 ± 0.52	72.62 ± 1.68	5314.11 272.02	±7.67 ± 0.47
No Redundancy Penalty	66.20 ± 0.44	73.81 ± 1.68	3543.59 150.84	±7.33 ± 0.47

No Multi-Swarm	66.47 $\pm$ 0.05	71.43 $\pm$ 2.91	3561.39 452.52	$\pm$ 8.00 $\pm$ 0.82
----------------	------------------	------------------	-------------------	-----------------------

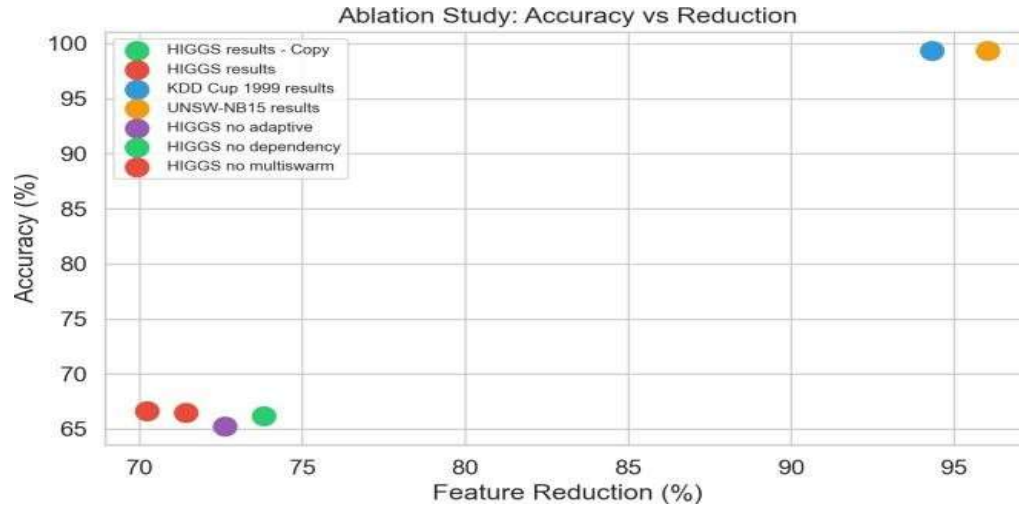


Fig. 6. Performance comparison between CUDA-AMSR-PSO and its ablated variants on the HIGGS dataset.

search process and computational efficiency [7], [13]. The multi-swarm architecture allows the diversity in the search and minimizing the chances of a premature convergence to the best solutions. The adaptive inertia mechanism is dynamically adjusted, so that exploration will be encouraged when positive gains are registered. More emphasis is given to exploitation when the gains are slowed down. Moreover, the objective functional takes into account feature redundancy, which prevents the presence of highly correlated features and increases the model robustness beyond mere accuracy.

Since the method is CUDA-based, the system-level implementation has the benefit of reducing computational overhead on the particle update step. It provides the ability to access swarm data concurrently. The next enhancement in kernel fusion is that it merges velocity updates, sigmoid transformations, and binary sampling into one operation. A CPU fallback mode is also included in the design, making it more convenient to run and test the system on non-GPU-supported machines.

At the same time, there are limitations. The wrapper-based assessment still runs on CPU execution, as model training and validation are carried out on the host. This reduces the overall performance due to the CPU workload and data transfer between the host and device. One way to improve this would be to move the classifier to the GPU or use a lighter approximation for fitness evaluation. The second issue shows up in large problems with many particles or features where the memory of the GPUs can get exhausted. To work around this challenge, tiled processing or multiple GPU configurations could be adopted.

A. Ethical and Practical Considerations

The process of feature selection has impacts on not only the performance of the model but also downstream implications and fairness. In places like cybersecurity, healthcare and user analytics, datasets can include sensitive data or hints at

sensitive data. Due to this, preprocessing should be done carefully, paying attention to potential biases and adequate supervision. While GPU acceleration can improve the efficiency and reproducibility of a workflow, it must not be a tool of uncontrolled automation or surveillance, particularly in large-scale systems where the stakes are greater.

IX. CONCLUSION

This work presents CUDA-AMSR-PSO, a Numba-CUDA accelerated framework based on adaptive multi-swarm particle swarm optimization, designed to make feature selection in high-dimensional data more practical and scalable. It uses a multiple-swarm search strategy and an adaptive inertia that is responsive to changes in fitness, as well as the ability to use CUDA to accelerate the search. The approach is adaptive to the search space, and uses an objective function that avoids redundant features, rather than pre-determined parameters.

The method can find small sets of features with comparable performance on the benchmark datasets. On the other hand, one thing is for sure: speeding up the search process might not lead to significant reductions in runtimes if model evaluation is done using a CPU. This suggests feature selection should be considered a pipeline, rather than a standalone task [8].

This approach can be further improved in the future by moving the costs of evaluation to the GPU, or by using surrogate models to estimate solution quality. Other options include using multiple GPUs, hyperparameter tuning and hybrid evaluation. In general, progress in feature selection depends not only on better algorithms, but also on how well they handle real computational constraints.

REFERENCES

- [1]. G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.
- [2]. M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis*, vol. 1, no. 1-4, pp. 131–156, 1997.
- [3]. H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.
- [4]. T. M. Shami, A. A. El-Saleh, M. Alswaiti, Q. Al-Tashi, M. A. Summakieh, and S. Mirjalili, "Particle swarm optimization: A comprehensive survey," *IEEE Access*, vol. 10, pp. 10 031–10 061, 2022.
- [5]. H. Xie, L. Zhang, C. P. Lim, and Y. Yu, "Feature selection using enhanced particle swarm optimisation for classification models," *Sensors*, vol. 21, no. 5, p. 1816, 2021.
- [6]. J. Wang, X. Wang, X. Li, and J. Yi, "A hybrid particle swarm optimization algorithm with dynamic adjustment of inertia weight based on a new feature selection method to optimize svm parameters," *Entropy*, vol. 25, no. 3, p. 531, 2023.
- [7]. B. Ji and X. Lu, "Bio-inspired feature selection: An improved binary particle swarm optimization approach," *IEEE Access*, vol. 8, pp. 85 989–86 002, 2020.
- [8]. I. M. El-Hasnony, S. I. Barakat, M. Elhoseny, and R. R. Mostafa, "Improved feature selection model for big data analytics," *IEEE Access*, vol. 8, pp. 66 989–67 004, 2020.

- [9]. D. L. Souza, G. D. Monteiro *et al.*, "PSO-GPU: Accelerating particle swarm optimization in cuda-based graphics processing units," in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '11)*, 2011, pp. 837–838.
- [10]. Y. Zhuo, T. Zhang, F. Du, and R. Liu, "A parallel particle swarm optimization algorithm based on gpu/cuda," *Applied Soft Computing*, vol. 144, p. 110499, 2023.
- [11]. K. Robindro, S. S. Devi, U. B. Clinton *et al.*, "Hybrid distributed feature selection using particle swarm optimization–mutual information," *Data Science and Management*, vol. 7, no. 1, pp. 64–73, 2024.
- [12]. J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.
- [13]. W.-m. W. Hwu, D. B. Kirk, and I. El Hajj, *Programming Massively Parallel Processors: A Hands-on Approach*, 4th ed. Morgan Kaufmann, 2022.
- [14]. P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature Communications*, vol. 5, p. 4308, 2014.
- [15]. S. Stolfo *et al.*, "Kdd cup 1999 data," <https://archive.ics.uci.edu/dataset/130/kdd+cup+1999+data>, 1999, uCI Machine Learning Repository.
- [16]. N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems," in *Military Communications and Information Systems Conference (MilCIS)*, 2015.