Metaheuristic Hyperparmameter Optimisation for LoRA Fine-Tuning of Foundation Models

Zayaan K. Khan
Computer Science
University of Surrey
Guildford, UK
6779724
zk00332@surrey.ac.uk

Rita San
Computer Science
University of Surrey
Guildford, UK
6779405
rs02004@surrey.ac.uk

Steven Thomas
Computer Science
University of Surrey
Guildford, UK
6779252
st01634@surrey.ac.uk

Joel D'Souza
Computer Science
University of Surrey
Guildford, UK
6776444
jd01606@surrey.ac.uk

***Abstract*—Currently, there are many approaches to optimisation, where most approaches are gradient based, limiting the function to the concept of differentiability, risking a solution being stuck in local optima. In our study, we propose various approaches to optimisation via evolutionary and meta-heuristic algorithms in hyperparameter optimisation. Specifically, we conduct various experiments on a type of foundation model called DistilBERT which is a large language model (focused on text generation, translation and semantic analysis) where we make use of LoRA (Low-Rank Adaptation) [?] to adapt the initial weight matrix in the model and optimise and adapt LoRA's parameters. We have concluded that the approaches suggested in this paper via evolutionary and metaheuristic algorithms exceed performance of baseline algorithms currently being used such as Random Search and therefore, we believe this is a good approach in tackling parameter optimisation.**

1 Introduction & Literature Review

Foundation Models (FMs) are a subset of deep learning models trained on vast amounts of data to be applied to various downstream tasks such as text classification, text generation, image segmentation and image generation. As FMs are deep learning models at a larger scale, they must have an underlying architecture which supports how the model learns and hence this is where the Transformer [?] is utilised. The Transformer is composed of two components; the Encoder and Decoder, each consisting of a stack of layers. The key layers are the Multi-Headed Attention (MHA) mechanism and the position-wise Feed-Forward Network (FFN). The challenge with FMs arises during the fine-tuning phase. Due to the nature of how all parameters in each layer would require an update through backpropagation, gradient update calculations for millions, if not billions, of parameters will need to be performed. This results in long training and inference times. Hence, we justify the usage of PEFT (Parameter-Efficient Fine-Tuning). Initially, Adapter Tuning [?] was an early solution to PEFT where models with a bottleneck architecture had 'adapter modules' inserted in between transformer layers, and only these modules get updated after freezing the original weights during fine-tuning. Similarly, BitFit [?] updates only the biases in foundation models while freezing the rest of the other modules. These methods, whilst effective, constrain where and how the adaptation occurs during the model while adding inference overhead.

However, Low-Rank Adaptation (LoRA) [?], offers greater flexibility by allowing selective adaptation of any weight matrix in the model through low-rank decomposition. This addresses the previous limitations of updating the gradients of all the parameters by utilising trainable rank decomposition matrices into the model layers. These are later merged with the original weights post training to eliminate inference overhead while maintaining competitive performance. LoRA freezes pretrained weights W_0 and injects trainable low-rank decompositions such that $W = W_0 + BA$, where $B \in \mathbb{R}^{dr}$ and $A \in \mathbb{R}^{rk}$ such that $W \in \mathbb{R}^{dk}$ with r ¡¡ min(d,k), scaled by α/r. This decomposition results in a significant reduction in the overall number of trainable parameters compared to the full fine-tuning, yielding key advantages such as enabling the fine-tuning of FMs on consumer-grade GPUs by lowering the requirements of computational resources.

Although LoRA improves the model's efficiency and the problem of overfitting through reducing the number of trainable parameters, performance is highly sensitive to the configuration of its own hyperparameters.

Hyperparameter optimisation addresses this challenge of searching through the configuration space to identify settings that maximise model performance. Approaches such as grid search work by exhaustively evaluating combinations over a predefined grid. This scales poorly as the number of hyperparameters increase due to the curse of dimensionality. Random search offers improved efficiency by sampling configurations stochastically, usually outperforming Grid Search in high dimensional search spaces. Methods such as Bayesian Optimisation rely on probabilistic surrogate models of the objective function to guide the search, however they suffer the same problem of computational expense as dimensionality increases.

1.1 Literature Review

Nature-inspired metaheuristics have been studied for as long as five decades now. Sörensen and Glover [?] define metaheuristic algorithms as strategies which guide search processes to locate optimal or near-optimal solutions for complex optimisation problems. We see in the real-world metaheuristic algorithms being applied to a multitude of tasks, including test case generation for software engineering [?], cognitive computing within healthcare [?], etc. However, a critical limitation in applying these search strategies to deep learning is the computational infeasibility of Neural Architecture Search (NAS) on foundation models like DistilBERT. Since modifying the underlying architecture requires expensive retraining, we identify that the strength of metaheuristics is better applied to Hyperparameter Optimisation (HPO) of the LoRA adapters, rather than searching for a new foundational architecture.

Genetic and metaheuristic algorithms: Genetic algorithms are an approach inspired by a population of candidate solutions represented as chromosomes. Initial random populations are typically selected and undergo recombination based on fitness to obtain successive generations of solutions. We discuss PSO, a suitable candidate for hyperparameter optimisation [?]. PSO is a metaheuristic (evolutionary) algorithm which simulates swarm intelligence in nature (e.g. fish schools, bird flocks etc.) where each system contains particles representing a candidate solution. Each particle keeps track of its own localised best solution so far (known as p^{best}) and the aim of PSO is to accelerate each particle towards it's p^{best} and g^{best} (global best) with randomly weighted acceleration.

RCGA BLX-α on the other hand, was used in this paper because it combines tournament selection, blend crossover (sampling within $/alpha$-extended parent intervals), and Gaussian mutation on real-valued genes. This enables gradient exploitation and broader exploration within the mixed discrete-continuous search space.

Advanced Variants of Metaheuristic algorithms: DE (Differential Evolution) [?] is an evolutionary algorithm that was initially designed to solve continuous/real-valued optimisation problems. However, DE still suffers from manual selection of control hyperparameters specific to the algorithm (e.g. Crossover Rate [CR] and Scaling Factor [F]) which can lead to an undesired rate of convergence. We use a variant of DE called SHADE (Success-History based Adaptive Differential Evolution) [?]. SHADE improves on DE by using self-adaptation on CR and F by storing a history of success, which helps in efficiently being able to optimise within the search space without manual selection.

Multi-objective optimisation: This is an approach used to find solutions that balance multiple objectives. Rather than producing a single best solution, it produces a set of non-dominated solutions, also known as the “Pareto Front”. We chose NSGA-II [?] as it identifies Pareto-optimal trade-offs between competing objectives well. Our case is, model accuracy versus no. trainable parameters. The algorithm ranks solutions by non-dominated sorting (solutions characterised by having no further possibility of improving an objective without harming the other objectives) and uses crowding distance to maintain diversity along the Pareto front. This allows for discovery of solutions each representing unique accuracy-efficiency trade-offs, enabling us to select configurations based on our constraints.

Contribution: Our contribution constrains the search space to specific ranges, such as Rank $\{2, \ldots, 24\}$ and Learning Rate $[10^{-5}, 2 \times 10^{-4}]$, to make the optimisation computationally feasible. We apply these metaheuristic algorithms for hyperparameter optimisation to the underlying architecture of DistilBERT, adapting the foundation model via LoRA to maximise performance.

2 Optimisation Algorithms

2.1 *Problem Representation*

A chromosome (candidate solution) is represented as a vector $G = [g_1, g_2, \ldots, g_6]$. The learning rate ($g_1$) is encoded directly as a continuous float. The remaining five discrete parameters are encoded as integer indices pointing to a pre-defined list of valid options. For example, if the valid Ranks are $\{2, 4, 8, 16, 24\}$, a gene value of 2.0 corresponds to Rank 8. During fitness evaluation, the vector is decoded with a repair mechanism; continuous genes are used as-is, while index-based genes are rounded to the nearest integer to select the corresponding hyperparameter.

The search space is defined as follows:

- **Learning Rate:** $[1 \times 10^{-5}, 2 \times 10^{-4}]$
- **Rank** r: $\{2, 4, 8, 16, 24\}$
- **Alpha** α: $\{8, 16, 32, 64, 96\}$
- **Warm-up Ratio:** $\{0.0, 0.06, 0.1\}$
- **Dropout:** $\{0.0, 0.05, 0.1, 0.2\}$
- **Target Modules:** $\{\text{Attention}, \text{Attention+FFN}\}$

We use index-based encoding to ensure uniform exploration across discrete options. For example, a mutation step size of 5 with value-based encoding would have different effects at $rank = 4$ (significant change), as opposed to $rank = 24$ (minimal change). Index-based encoding ensures consistency between irregularly spaced out hyperparameter options.

2.2 *Algorithm 1: RCGA BLX-α*

We propose a Real-Coded Genetic Algorithm (RCGA) for optimising LoRA hyperparameters. While standard binary GAs often struggle with Hamming cliffs (where small mutations in the binary representation cause drastic jumps in the parameter value), RCGA operates directly on the parameter vector. This is particularly advantageous for LoRA, where key parameters like Rank (r) and Alpha (α) possess an inherent ordinal relationship ($r = 4 < r = 8 < r = 16$). By treating these as continuous variables during evolution, RCGA allows the optimiser to exploit local gradients and infer performance trends between discrete steps, leading to more efficient convergence in the mixed-integer search space.

2.2.1 Crossover (BLX-α): To balance exploration and exploitation, we utilise the Blend Crossover (BLX-α) operator. This operator does not merely swap genes between parents but generates offspring in the interval extended by α beyond the parents' values. For two parent genes x_1 and x_2 (assuming $x_1 < x_2$), the offspring y is sampled uniformly:

$$y \sim U(x_1 - I \cdot \alpha, x_2 + I \cdot \alpha), \quad \text{where } I = x_2 - x_1 \tag{1}$$

We select $\alpha = 0.5$, allowing the search to extend slightly beyond the parents, boundaries to preserve population diversity. This interval extension is critical for escaping local optima in the continuous learning rate dimension.

2.2.2 Mutation and Repair: A random reset mutation is applied with a probability of 0.1 per gene to prevent premature convergence. A repair mechanism is enforced after every operation: continuous genes (Learning Rate) are clipped to range $[1e-5, 2e-4]$, and discrete genes are rounded to the nearest integer index to map back to valid LoRA configurations. This allows the algorithm to explore a continuous landscape while respecting the discrete nature of the LoRA hyperparameters.

2.2.3 Elitism: To ensure monotonic performance improvement, we implement strict elitism. The single best individual from generation g is copied unchanged to generation $g+1$. This guarantees that the highest validation accuracy found so far is never lost due to stochastic crossover or mutation, a crucial feature given the high cost of each fitness evaluation.

Algorithm 1 RCGA with BLX-α Crossover

```
1: Input: Population Size $N$, Generations $G$, $\alpha = 0.5$
2: Output: Best Hyperparameter Vector $x_{best}$
3: Initialise population $P$ with random real-valued vectors
4: Evaluate fitness $f(x)$ for all $x \in P$
5: for $gen = 1$ to $G$ do
6:     $P_{new} \leftarrow \emptyset$
7:     $x_{best} \leftarrow \text{argmax}_{x \in P} f(x)$
8:     Add $x_{best}$ to $P_{new}$                // Elitism
9:     while $|P_{new}| < N$ do
10:        Select parents $p_1, p_2$ via Tournament Selection
11:        $child \leftarrow$ empty vector
12:        for each gene $i$ do
13:            $d = |p_{1,i} - p_{2,i}|$
14:            $min = \min(p_{1,i}, p_{2,i}) - \alpha \cdot d$
15:            $max = \max(p_{1,i}, p_{2,i}) + \alpha \cdot d$
16:            $child_i \leftarrow \text{Uniform}(min, max)$
17:            $child_i \leftarrow \text{Clip}(child_i, \text{lower\_bound}_i, \text{upper\_bound}_i)$
18:        end for
19:        Apply Random Reset Mutation to $child$ ($prob = 0.1$)
20:        Add $child$ to $P_{new}$
21:    end while
22:    $P \leftarrow P_{new}$
23:    Evaluate fitness for all $x \in P$
24: end for
25: return $x_{best}$
```

2.3 Algorithm 2: SHADE

The comparative study by Dhar et al. [?] mentioned in the introduction suggests that Evolutionary Algorithms, particularly Differential Evolution (DE), are an ideal choice for navigating gradient-free search spaces. A notable drawback in standard DE is its sensitivity to the manual selection of its control parameters (Crossover Rate CR and Scaling Factor F), such that initialising suboptimal values for these parameters can lead to undesired convergence rates. We instead adopt an enhanced variant of DE: Success-History based Adaptive Differential Evolution (SHADE). SHADE improves on standard DE by automatically adapting its control parameters, driven by historical success information. This is critical for hyperparameter optimisation, where the optimal exploration-exploitation balance may shift as the search progresses through the fitness landscape.

Thoughtful considerations regarding the computational expense of training transformer models (approximately 3–5 minutes per evaluation) were carried out, leaving us with a severely constrained optimisation budget (100 evaluations). However, SHADE's adaptive mechanism allows it to learn from successful parameter combinations, and focus computational resources more effectively compared to random or grid search approaches.

The LoRA hyperparameter optimisation problem presents itself as a mixed continuous-discrete search space. We employ the repair mechanism to map the outputs of SHADE's continuous mutation operators to discrete values. The continuous candidate vectors produced by the mutation operator can be defined as:

$$v_i = x_i + F_i \cdot (x_{pbest} - x_i) + F_i \cdot (x_{r1} - x_{r2}) \quad (2)$$

This approach allows for smooth exploration while respecting discrete constraints.

Furthermore, SHADE maintains an external archive of recently replaced solutions, providing additional diversity for the mutation operator. This is especially valuable in hyperparameter optimisation where similar configurations may perform differently due to local interactions between parameters.

SHADE has demonstrated superior performance on the CEC2014 benchmark suite and various real-world optimisation problems, particularly excelling in problems with limited evaluation budgets; precisely our scenario [?]. Although JADE (Joint Adaptive Differential Evolution) similarly addresses the limitations of manual parameter tuning found in standard DE, it does not guarantee diversity within the population due to its reliance on a single average. SHADE improves upon this foundation by saving successful parameters to a history list and randomly selects parameter pairs from the list, preventing premature convergence. [?]

3 Experimental Results

3.1 Experimental Setup & Parameter Ranges

This experiment evaluates the performance of four distinct algorithms. A 20-trial random search serves as the baseline against three metaheuristics: PSO, RCGA BLX-α, and SHADE, all of which were discussed in the previous section. We first initialise a population of candidates with random configurations of indices mapped to a specific set of LoRA hyperparameters we decide on. The fitnesses of these individuals are then calculated by obtaining the validation accuracy after training DistilBERT for 3 epochs. The following process outlines the method of evaluation:

Algorithm 2 Success History Adaptive Differential Evolution

```
Input: $N = 20$ (pop size), $H = 20$ (memory size), $G_{max} = 5$, $arc\_rate = 1.0$, $p = 0.4$
Initialise:
$M_{CR}, M_F = [0.5] \times H$ // historical parameter memories
$Archive = \emptyset, k = 0$
$P = \{$random individuals$\}$, evaluate all, track $x_{best}$
// Main Loop:
for $generation = 1$ to $G_{max}$ do
    sort population by fitness (descending)
    $p_{num} = \max(2, \lfloor p \times N \rfloor)$
    // generate offspring
    for each individual $i$ do
        // sample parameters from memory
        $r = $ rand_int$(0, H-1)$
        $CR_i = 0$ if $M_{CR}[r] == -1$ else clip$(\mathcal{N}(M_{CR}[r], 0.1), 0, 1)$
        $F_i = \min($Cauchy$(M_F[r], 0.1), 1.0)$ where $F_i > 0$
        // mutation: current-to-pbest/1
        Select: $p_{best}$ (from top $p_{num}$), $r1 \neq i$ (from pop), $r2$ (from pop $\cup$ Archive)
        $v_i = x_i + F_i \times (x_{pbest} - x_i) + F_i \times (x_{r1} - x_{r2})$
        // binomial crossover
        $u_i = $ crossover$(x_i, v_i, CR_i)$
        $u_i = $ repair$(u_i)$
        $f_{ui} = $ evaluate$(u_i)$
    end for
    // selection and parameter recording
    $S_{CR}, S_F, \Delta f = []$
    for each individual $i$ do
        if $f_{ui} \geq f_i$ then
            if $f_{ui} > f_i$ then
                $Archive \leftarrow x_i$ (randomly replace if full)
                Record: $S_{CR} \leftarrow CR_i, S_F \leftarrow F_i, \Delta f \leftarrow |f_{ui} - f_i|$
            end if
            $x_i \leftarrow u_i, f_i \leftarrow f_{ui}$
            Update $x_{best}$ if improved
        end if
    end for
    // update memory (weighted Lehmer mean)
    if $|S_{CR}| > 0$ then
        $w = \Delta f / $sum$(\Delta f)$    // weights from improvement
        $M_F[k] = \Sigma(w \times S_F^2) / \Sigma(w \times S_F)$
        $M_{CR}[k] = -1$ if all $S_{CR} = 0$ else $\Sigma(w \times S_{CR}^2) / \Sigma(w \times S_{CR})$
        $k = (k+1) \mod H$
    end if
end for
Output: $x_{best}, f_{best}$
```

1) Decode the index vector x into LoRA hyperparameters via a repair function
2) Load a fresh DistilBERT-base-uncased model
3) Apply LoRA adaptation with decoded hyperparameters and freeze the original weights of DistilBERT
4) Train the LoRA matrices and the final, fully connected layer for 3 epochs on 3,000 samples from the Emotion datasets train set using the AdamW optimiser
5) Evaluate on the full validation set (2,000 samples)
6) Return validation accuracy as fitness (maximisation objective)

All algorithms are initialised with a population of 20, with individuals being evaluated at the start. Offsprings and updates are generated from the starting population and undergo four generational updates, totalling 100 evaluations. We have chosen these small values (relative to literature) for the population and generations, as the search space itself has been constrained due to the computational demand of the task at hand. Three epochs are used during training for all algorithms to reinforce fairness, and limit evaluation time. P100 GPUs are used in training for their speed and availability. Finally, the top five configurations are taken from each algorithm and are trained on three unique seeds. The results shown in Table **??** and Table **??** are from the best performing configuration of each algorithm.

TABLE I
ALGORITHM PARAMETER RANGES

Algorithm	Parameters
PSO	$w = 0.7$, $c_1 = c_2 = 1.5$
RCGA BLX-α	$\alpha = 0.5$, $p_{mutation} < 10\%$
SHADE	$H = 20$, A $= 1.0$, $p_{\text{best}} = 0.4$

For PSO, w controls velocity momentum while c_1 and c_2 are used for cognitive and social learning, respectively. In RCGA BLX-α, α defines the exploration range beyond parent bounds, with mutation applied to individual hyperparameters. SHADE adapts its parameters using a history of size H, maintains an archive of inferior solutions scaled by the archive rate, and an individual from the top p_{best} fraction of the population is used to help create a mutant vector.

Memory Size: $H = 20$

The original SHADE implementation used $H = N = 100$ [**?**]. Tanabe and Fukunaga found that $H \in [5, 50]$ performed comparably to $H = 100$, while $H > 100$ degraded performance. We set $H = N = 20$ based on our limited evaluation budget: with only 5 generations and approximately 4–5 successful updates per generation, a larger memory (e.g. $H = 100$) would leave most positions at default values, reducing SHADE's adaptive benefit.

Archive Rate: $arc\_rate = 1.0$

Archive size = $\lfloor arc\_rate \times N \rfloor = 20$ individuals. This doubles the selection pool from x_{r2} to 40 maximum candidates, preserving diversity as the population converges. While L-SHADE implementations commonly use $arc\_rate \in [1.4, 2.6]$ [**?**], we use a smaller archive.

P-Best Rate: $p = 0.4$
In standard SHADE, each individual samples $p_i \sim \text{rand}[2/N, 0.2]$ per generation. Our implementation uses a fixed $p = 0.4$, selecting $x_{p\text{best}}$ uniformly from the top 8 individuals. This higher value increases exploration compared to the standard maximum of $p = 0.2$, which is beneficial due to our smaller population where greedy selection ($p \approx 0.1$) risked premature convergence.
Justification:
Our constrained discrete search space (4–5 values per LoRA hyperparameter) reduces the effective problem dimensionality compared to continuous optimisation benchmarks. This allows SHADE to converge faster, enabling effective search with fewer generations and a smaller population than the original implementation.

3.2 Results and Key Findings

This section discusses the experimental results obtained from the four metaheuristic optimisation algorithms. We evaluate the efficacy of these algorithms in maximising the validation accuracy of the DistilBERT-LoRA model under a fixed computational budget.

TABLE II
THREE SEED PERFORMANCE OF TOP LoRA CONFIGURATIONS

Top Configuration Three Seed Results					
Algorithm	Best Acc.	Mean Acc.	±Std.	Evaluations	Time (min)
Random Search	0.878	0.877	0.003	20	21.04
PSO	0.905	0.893	0.002	100	108.44
RCGA BLX-α	0.906	0.896	0.004	100	108.55
SHADE	0.900	0.897	0.003	100	72.90

TABLE III
BEST LoRA HYPERPARAMETER CONFIGURATIONS CHOSEN BY ALGORITHMS

Best Performing LoRA Configurations						
Algorithm	Learning Rate	Warm Up	Rank	Alpha	Dropout	Target Modules
Random Search	1.7×10^{-4}	0.06	4	64	0.1	Attn+FFN
PSO	2.0×10^{-4}	0.1	4	96	0.0	Attn+FFN
RCGA BLX-α	2.0×10^{-4}	0.0	16	96	0.0	Attn+FFN
SHADE	2.0×10^{-4}	0.0	24	96	0.0	Attn+FFN

Convergence Analysis:
All four algorithms converged to values for learning rate (2.0×10^{-4}) and target modules (attention + feedforward), suggesting that these represent ideal optima within the search space. Similarly, the metaheuristic methods identified $alpha = 96$ and $dropout = 0.0$ as optimal, while Random Search resolved at suboptimal values ($alpha = 64$, and $dropout = 0.1$). This is likely due to its limited smaller evaluation budget of 20 evaluations, compared to 100 for the metaheuristic algorithms. The most notable difference occurred in the rank parameter. SHADE and RCGA BLX-α discovered higher rank values of 24 and 16 respectively, which could correlate with their marginally better performance. PSO converged to $rank = 4$, suggesting that its velocity-based updates were less effective at escaping the local region compared to the mutation-based exploration of SHADE and RCGA BLX-α. All metaheuristic methods rejected dropout regularisation. This could happen

convergence_plot.png

Fig. 1. Best Accuracy of Algorithm vs. Objective Evaluations

because the 3-epoch training loop for evaluations is too short for overfitting to occur, leading to the benefit of dropout regularisation not being able to manifest.

The clearest connection from these results was that the more trainable parameters available, the better the model performs. This is shown in the two best metaheuristic algorithms converging to a higher rank, and importantly, all top configurations of the four algorithms selecting both attention heads and feed-forward layers as the target modules for LoRA's application, greatly increasing the number of trainable parameters. Another interesting find is the scaling factor alpha being the highest possible value for the three metaheuristic algorithms. This shows that the LoRA adapted weights having a very high influence compared to DistilBERTs pre-trained knowledge (weights), is conducive to a greater accuracy on newly learned data.
Convergence Speed and Search Space Analysis:
Figure **??** shows all the population achieving accuracy > 0.85 within the 20 initial evaluations. This rapid convergence can be attributed to 2 factors:

1) The constrained discrete search space increases the probability of sampling good configurations randomly.
2) AdamW's robustness as the training optimiser pushing the accuracy ceiling higher, allowing many configurations to perform relatively better than if a worse optimiser like SGD (Stochastic Gradient Descent) was used for training.

The level of convergence suggests that for similarly constrained LoRA fine-tuning problems, a smaller evaluation budget, with simply more random restarts may be a more cost-effective approach than an extended metaheuristic search with several generational updates.

Computational Efficiency:
SHADE achieved comparable results to RCGA BLX-α; 0.897 vs. 0.896 in 33% less time; 72.90 vs. 108.55 minutes. This makes our implementation of SHADE the most efficient metaheuristic method in this experiment. However, the 3–5$\times$ computational cost of any metaheuristic algorithm over Random Search yielded only a $\sim$2% accuracy improvement. This raises questions about the cost-effectiveness of these algorithms. To scrutinise further, if we go back to Figure **??**, for both SHADE and RCGA BLX-α, the optimal parameters are found by sheer luck in the initial random population generation phase, further reinforcing the point that if we must use metaheuristic methods, random restarts with checkpoints of the best solutions or simple Random Search with more trials may just be the better approach for problems with constrained search spaces like ours.

4 Multi-objective Extension

In this section, we explore the use of multi-objective evolutionary algorithms to determine the optimal trade-off boundary between validation accuracy and model complexity (measured by trainable parameters) for the LoRA adaptation of DistilBERT. We opted for the Non-dominated Sorted Genetic Algorithm II (NSGA-II), proposed by Deb et al. [**?**], as it provides computational efficiency in bi-objective problems through its fast non-dominated sorting mechanism. Furthermore, its elitist strategy means that optimal hyperparameter configurations are not lost through stochastic selection processes, but are rather passed onto the next generation. This ensures monotonic improvement in the Pareto front.

We use the same LoRA hyperparameter configurations as those in the other algorithms discussed in section 3, with the evolutionary parameters also unchanged (population size $P = 20$ and generations $G = 5$), matching the exact search space and evaluations to ensure fairness. While the single-objective algorithms discussed in this paper focus on maximising validation accuracy, the complexity of the resultant neural architectures does not contribute to selection pressure. In other words, these algorithms favour models with higher accuracy, regardless of over-parametrisation. For example, one LoRA configuration could produce an accuracy of 85.0% with $rank = 4$ and $\#trainable\_params = 600,000$, resulting in low training cost. Another configuration could produce an accuracy of 85.1% with $rank = 24$ and $\#trainable\_params = 2,000,000$, in which the training cost has significantly increased, but this cost is insignificant to single-objective algorithms, and it will only select the model configuration with higher accuracy (85.1% > 85.0%). Therefore, we can justify the need to introduce parameter efficiency as a primary optimisation goal alongside accuracy.

We present the Pareto front procured from the NSGA-II optimisation process, plotting validation accuracy against the number of trainable parameters.

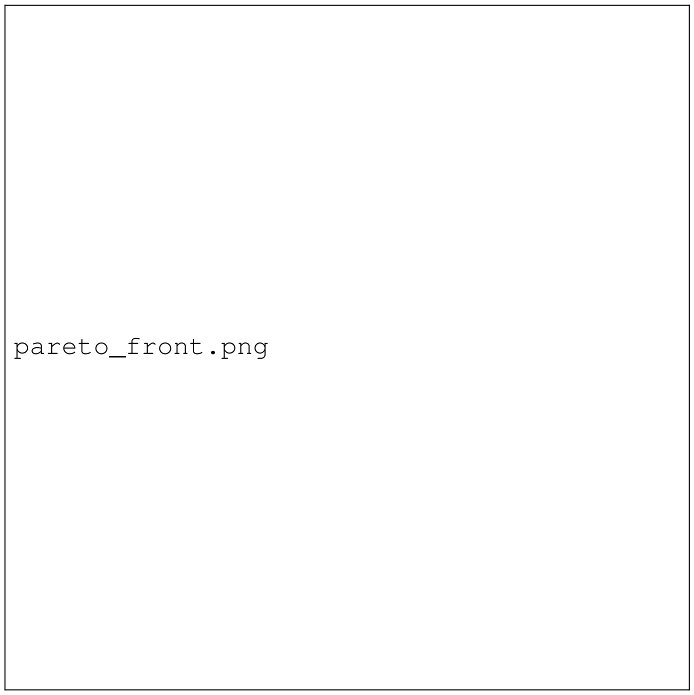

Fig. 2. NSGA-II: Pareto Front Plot

TABLE IV
Pareto optimal LoRA configurations from NSGA-II

NSGA-II Pareto Front Solutions				
Solution	Rank	Trainable Params	Accuracy	Target Modules
1	2	0.63M	82.05%	Attn
2	4	0.67M	82.15%	Attn
3	2	0.72M	87.75%	Attn+FFN
4	4	0.85M	89.55%	Attn+FFN
5	8	1.11M	90.20%	Attn+FFN

The red dots in figure **??** represent Pareto optimal solutions, also known as non-dominated solutions, which are characterised by having no further possibility of improving an objective without harming the other objectives [**?**]. Our Pareto optimal solutions in this case, are the optimal trade-off points between validation accuracy and number of trainable parameters. The grey dots, on the other hand, represent the Pareto dominated solutions discovered during the optimisation process. These solutions are considered suboptimal as they are overcome by non-dominated (Pareto front) solutions which achieve equal or higher accuracies with the same or lower amount of trainable parameters.

The front path displays a steep vertical ascent as the number of trainable parameters increases, until the path halts at a validation accuracy of 90.2% with 1.11M trainable parameters. The steepest ascent occurs when the parameters are below 0.72M, suggesting that the model is constrained by capacity. Evidently from the results, an increase of model size from 0.67M to 0.72M (7.5% increase) in training parameters provided a gain in accuracy of 82.15% to 87.75% (5.6% increase). This "High-Gain, Low-Cost" behaviour demonstrates that

LoRA configurations targeting only attention layers (shown in Table **??** [solution 1 & 2]) underfit the emotion dataset regardless of rank. The jump from 82.15% to 87.75% when adding FFN modules, $solution_2 \rightarrow solution_3$, suggests that adapting the feedforward layers is critical for this task.

The knee point becomes visible at approximately 0.72M parameters, representing the most efficient trade-off in the search space as it approaches the peak accuracy. At this point, the model size is roughly a third smaller than the largest Pareto solution.

As the front path exceeds the knee point, the curve plateaus, converging to maximum accuracy. During this stage, it can be observed that achieving a final 0.73% gain in accuracy requires a costly increase of approximately 30.6% in trainable parameters.

This analysis suggests that the LoRA hyperparameter optimisation problem benefits significantly from multi-objective optimisation algorithms. In contrast to single-objective optimisation algorithms, which have no mechanism to penalise model complexity and tend to select high-capacity configurations for marginal accuracy gains, multi-objective optimisation algorithms can penalise high model complexity. This strategy ensures smooth convergence towards the 'knee point' of the Pareto front, eventually achieving optimal efficiency by avoiding the selection of over-parametrised LoRA configurations. Ultimately, treating the number of training parameters and validation accuracy as conflicting objectives introduces the aspect of efficiency to the model selection process that would otherwise be ignored by single-objective algorithms.

5 Conclusion

For our experiment we had the baseline Random Search run for a total of 20 evaluations, whereas the meta-heuristic methods were initialised with a random population of 20 and thereafter ran for four generations totalling 100 evaluations. After running the experiment, we took the top 5 configurations for each algorithm and ran them on three unique seeds to find the most optimal configurations of LoRA hyperparameters attributable to each. The best configurations, were used for comparison. The final best validation accuracies of the three meta-heuristic methods were virtually identical, with RCGA BLX-A having a marginal lead on SHADE and PSO. The same can be stated for the mean accuracies with the SHADE obtaining the highest mean accuracy, still remaining marginal. Comparatively, Random Search achieved fairly similar results only being 1-2% lower than the meta-heuristic algorithms.

In terms of actual successful hyperparameter configuration, all four algorithms converged to values for learning rate (0.0002) and target modules (attention + feedforward), suggesting that these represent ideal optima within the search space. The most notable difference occurred in the rank parameter. SHADE and RCGA BLX-α discovered higher rank values of 24 and 16 respectively, which could correlate with their marginally better performance. PSO converged to rank=4, possibly suggesting that its velocity based updates were less effective at escaping the local region compared to the mutation based exploration of SHADE and RCGA.

A key finding from these results was that the more trainable parameters that are available, the better DistilBERT performs on validation set accuracy. This is shown by the two best meta-heuristic algorithms converging to a higher rank, and all top configurations of the four algorithms having selected both attention heads and feed forward layers as the target modules for LoRA's application. This greatly increases the number of trainable parameters and makes sense that it would allow greater generalisation of the dataset. Another insight is the scaling factor alpha being the highest possible value for all the meta-heuristic algorithms. This shows that the LoRA adapted weights having a very high influence compared to DistilBERTs pre-trained knowledge (weights), is crucial for greater performance.

A critical point to note is the rapid convergence to an accuracy of > 0.85 for both RCGA-BLX-a and SHADE within the random initialisation of the population. We believe that this can be attributed 2 factors; 1) Due to the search space being so small (for discrete hyperparameter choices), the probability of sampling good configurations randomly increases. 2) AdamW as an optimiser allows many different configurations to perform relatively better than if a worse optimiser like SGD was used for training.

For these reasons we conclude that by using random restarts on the metaheuristic methods with checkpoints to save the best configuration, would be more suitable than the baseline 20-trial Random Search and the normal runs of the algorithms. The primary reason for this verdict comes down to the search space. The hyperparameter search space was selected as a trade-off between complex and simple (higher vs lower amount of possible hyperparameter combinations). We decided on this due to fine-tuning of DistilBERT being computationally intensive even when utilising LoRA. The decision allows for a higher probability of choosing better configurations by relying on a smaller search space.

We discovered through figure **??** and table **??** that the "High-Gain, Low-Cost" phase of multi-objective optimisation indicates that LoRA configurations targeting only attention layers under-fit the emotion dataset. Furthermore, achieving a marginal gain in accuracy required a substantial increase in training parameters towards the last Pareto point. The single-objective vs. multi-objective optimisation analysis provided valuable justification as to why the LoRA hyperparameter optimisation problem benefits from multi-objective strategies. It allows for consideration of training expenses required for optimal solutions, which single-objective optimisation algorithms do not scrutinise.

Limitations:

Our work faces limitations, such as a budget of 100 trials per algorithm ($N = 20$, $G = 5$) and 3-epochs per trial. This is due to the high computational cost of fine-tuning transformer models like DistilBERT. We recognise that this budget restricts wider sampling across the discrete search space, and as a result, the optimal results obtained may only represent local optima as opposed to true global maximum. Moreover, only 3000 subsamples from the emotion datasets test set were used for training, which is likely to lower the maximum achievable accuracy.

Future Work:
Additional experiments to be conducted in the future may include increasing the maximum bounds for the learning rate. This is because all assessed algorithms presented with their highest accuracy when the learning rate was at the maximum learning rate bound *i.e.*, 2×10^{-4}, evident in table **??**. As observed within the computational efficiency analysis in section 3, the continued use of metaheuristic methods for LoRA hyperparameter optimisation could benefit from random restarts to escape worse solutions. While we opted for AdamW as the standard optimiser throughout the training process, future work may also benefit from the use of alternative optimisers to assess their impact on LoRA hyperparameter optimisation performance. To introduce more diversity and accuracy in the search process, future works could entail widening the search space to allow for more value coverage instead of adhering to a discrete set of values.

Appendix

`final_performance.png`

Fig. 3. Comparison of Final Performances on Three Seeds

`training_cost.png`

Fig. 4. Training Time of All Algorithms

Contributions
Due to the nature of the components of this coursework being inseparable, all the group members have agreed to contribute equally in all components. Therefore, each group member has had a significant impact on all graded sections of this work.