Universiteit Leiden

LLaMEA for clustering problems

Anna Michelle van der Spek (s4077644)

Supervisors:
Niki van Stein & Thomas Bäck

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 1, 2026

Abstract

Benchmarks for clustering optimisers have only recently been standardised. The clustering benchmark suite introduced by Vermetten et al. (2025) [31] addresses the demand for evaluating clustering optimisers by formalising ten benchmark functions across four dimensionalities. This thesis investigates whether LLaMEA can generate effective optimisers for this suite. We evaluate LLaMEA under six configurations. The configurations cross domain knowledge provision ($\mathcal{D}$ or $\neg\mathcal{D}$) with baseline prompting, exemplar-injection, and warm start. $\mathcal{D}$ denotes provision of domain knowledge, while $\neg\mathcal{D}$ denotes no domain knowledge. Exemplar-injection provides the LLM with an additional example algorithm during each generation step, encouraging exploration across different metaheuristic families. Warm start instead seeds the initial population with example algorithms, giving LLaMEA a structured starting point from which later generations can evolve. We find that none of the LLaMEA generated algorithms outperformed KMeans++. However, prompting strategy affects the structural diversity of generated algorithms, with exemplar-injection producing the broadest range of algorithm families. At the prompting configuration level, $\mathcal{D}$ did not improve performance; instead $\neg\mathcal{D}$ achieved higher fitness. At the generated algorithm level, $\mathcal{D}$ configurations were the closest to the KMeans++ and CMA-ES baselines. These findings motivate future work using larger search budgets and more runs for the application of LLaMEA to clustering problems.

Contents

1 Introduction

The volume of data generated globally continues to grow at an unprecedented rate. As many real world datasets contain limited labelled examples, unsupervised learning methods are necessary to identify patterns through shared features [13]. Clustering is one of the most widely used unsupervised learning approaches, aiming to identify groups of similar observations without predefined target labels. However, designing an effective clustering algorithm for a given dataset requires careful tuning of its parameters. As the volume and dimensionality of data grow, so does the demand for metaheuristics to automate this process.

In spite of this demand, progress in clustering algorithm design has been difficult to measure. For other algorithms there are standardised benchmark suites such as BBOB [15] which provide a common basis for comparison. Whereas in clustering, the experimental setup can vary widely across papers, making cross-study comparison unreliable. This gap was recently addressed in 'A Standardised benchmark for clustering problems' [31], which introduced a structured clustering benchmark suite comprised of ten functions across four dimensions. This suite exposes two properties that make clustering benchmarks fundamentally distinct from continuous optimisation problems: *symmetry*, due to $k!$ equivalent labellings of any valid solution, and *regions of neutrality*, where large portions of the search space yield identical fitness values [12]. Critically, while this work defines the benchmarks and evaluates them using CMA-ES, it does not investigate the performance of other metaheuristic algorithms or detail the best CMA-ES configuration. Both gaps are addressed within this thesis.

To find optimal solutions within a search space, there are two main strategies: metaheuristics or gradient-based updates. Gradient updates require a continuous search space, which clustering problems do not provide. In contrast, the metaheuristic approach does not share this constraint and has shown better exploration capabilities than gradient updates [25]. Among these, CMA-ES [16] is widely used for continuous black-box optimisation [2] and serves as a performance baseline in this thesis. However, traditional metaheuristics share a fundamental limitation: their search space is fixed by design, constraining the structural diversity of solutions they can explore.

Large Language Models (LLMs) offer a compelling alternative. Recent work has demonstrated that LLMs are increasingly capable of generating, modifying, and reasoning about code [9]. LLaMEA (Large Language Model Evolutionary Algorithm) [30] leverages this capability within an evolutionary framework: LLaMEA prompts an LLM to generate and refine an algorithm's structure at each generation step. This allows for a fundamentally broader search over the space of possible algorithms, rather than solutions that only involve adjustment to algorithm parameters. LLaMEA has been shown to discover algorithms that outperform CMA-ES on the BBOB benchmark suite [30], motivating its application to distinct clustering problems.

This thesis applies LLaMEA to the clustering benchmark suite [31] under six main configurations. This represents the first systematic application of LLM-driven algorithm discovery to clustering benchmarks.

1.1 Research Questions

This thesis evaluates whether LLaMEA can generate effective clustering optimisers for the IOHclustering benchmark, and how the chosen prompting strategy influences the generated algorithms. The following research questions guide the analysis:

1. RQ1. How do prompting strategies such as the inclusion of domain knowledge ($\mathcal{D}$), exemplar-injection and warm start influence the fitness attained by LLaMEA on clustering problems in comparison to the KMeans++ and CMA-ES baselines?

2. RQ2. How do prompting strategy, elitism, and algorithmic structure affect the types of algorithms generated by LLaMEA and their resulting fitness?

3. RQ3. How does the provision of domain knowledge ($\mathcal{D}$) affect LLaMEA fitness, final candidate algorithm performance, and generalisability?

These research questions will allow us to determine how LLaMEA generated optimisers perform on clustering problems.

1.2 Thesis overview

Section 2 provides the necessary background on evolutionary algorithms, LLaMEA, and the clustering benchmark. Section 3 discusses related work on AAD and other analysis tools. Section 4 describes the experimental methodology, including the benchmark suite, the experimental configurations, and the evaluation metrics used throughout this thesis. Section 5 presents the experiments and their outcomes, covering comparisons between configurations, the role of domain knowledge and algorithm categories. Section 6 summarizes the work, detailing current limitations and future improvements.

2 Background

2.1 Clustering Problems

2.1.1 Definition

Clustering problems can be formalised as a continuous black box optimisation problem [12]. As the internal structure of the data is not provided to the algorithm, the MSE equation is used to calculate fitness. The goal of the optimiser is to minimise $f(C|X)$, where $C = \{c_1, ..., c_j\}$ is the set of clustering centres locations and $X = \{x_1, ..., x_i\}$ is the set of data points. n corresponds to the number of records in the dataset of the clustering problem.

$$f(C|X) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} b_{i,j} ||x_i - c_j||^2 \tag{1}$$

$||x_i - c_j||$ calculates the distance of the current data point to the current centre. If c_j is the closest centre to x_i, that is recorded by $b_{i,j}$. If it is not then a new x_{i+1} is selected as the centre and the equation is repeated.

Where $b_{i,j}$ is an indicator variable for the point closest to the centre.

$$b_{i,j} = \begin{cases} 1 & \text{if } \|x_i - c_j\| = \min_j \|x_i - c_j\| \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Equations 1 and 2 follow the formulation by Gallagher (2019) [12]. The dimensionality of this problem is dk, where d is the number of attributes in the dataset and k is the number of clusters. Since PCA has been applied to each dataset in the benchmark, d = 2 for all X. Therefore each problem instance is defined by the choice of dataset X and the number of clusters k.

2.1.2 Landscape Properties

Understanding the landscape properties is essential to explaining the complexity of clustering benchmark problems for optimisers. The fitness landscape of clustering problems is highly discontinuous. Each data point contributes to its own basin of attraction, and assignment to a centre point is binary, defined by $b_{i,j}$. This leads to abrupt boundaries between basins and creates a complex and non-linear problem space [12]. This produces a landscape with a large number of local optima, which poses a problem in reaching the global optimum. The optimiser makes step by step improvements and stops once no neighbouring move in the search space reduces the MSE, even if the global optimum has not yet been found.

Permutation invariance is another identified quality. Clustering centres in the benchmark are not assigned a label. Therefore, any permutation of the assigned centres leads to $k!$ equivalent solutions [31]. For example, with $k = 5$, there would be 120 representations of the same solution. This can cause redundant searching for optimisers, re-exploring the search space to find solutions with no tangible improvement.

Regions of neutrality are another challenge. Suppose a centroid c_j is positioned such that no data point is $b_{i,j} = 1$ for all i. This creates flat regions in the search space that do not provide a signal to the search algorithm [31]. The centre makes no contribution to the objective function, and moving it within this region produces no change in fitness. This is a direct challenge for optimisation, as the current evaluation provides no objective value until a new centre is chosen.

Finally, the underlying Euclidean sum of squares clustering problem is NP-hard [3]. No algorithm is guaranteed to find a global optimum solution in polynomial time. As a result, the generated optimisation algorithms in this thesis aim to find the best approximate solutions within our evaluation budget.

2.1.3 Replication Study

The original benchmark paper aimed to establish the validity of the clustering benchmark suite by demonstrating that it is sensitive to algorithmic design choices. The paper explicitly states that identifying the best performing algorithm was not its goal [31]. Consequently it does not report the MSE of the best CMA-ES configuration. For this thesis, the CMA-ES performance is required as a baseline. A replication study was therefore conducted.

The configuration $\lambda = 20$, $\mu = 10$, elitist = True, covariance = False, bound_correction = None achieved the best overall mean performance and was selected as the CMA-ES baseline for this thesis. This configuration was evaluated on the ten benchmark functions and across four dimensions. The evaluation budget was set to 5000 with 25 repetitions, identical to the original paper. Figure 1 shows the best mean MSE values achieved.

MSE decreases consistently with increasing dimension across all functions. Since each dataset was reduced to $d = 2$ via PCA, the problem dimension dk is determined entirely by k, so dimension $[4, 6, 10, 20]$ corresponds to $k = [2, 3, 5, 10]$ respectively. This decrease in MSE is an expected result

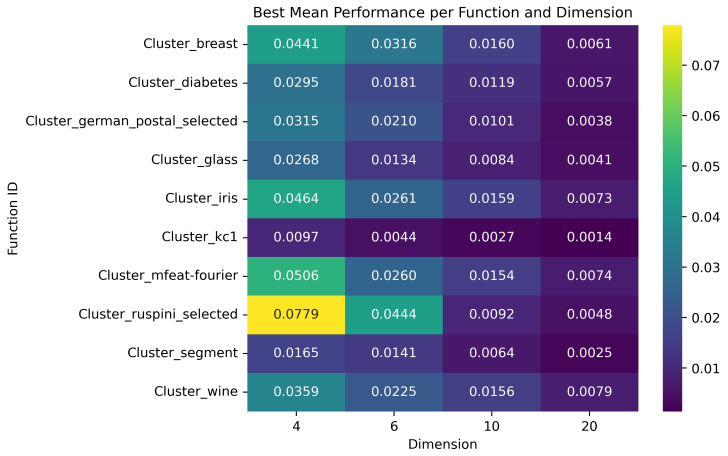


Figure 1: Best mean MSE achieved by the CMA-ES configuration across FID and dimensions

of the objective rather than a measure of CMA-ES performance: as k increases, more centres are available to capture the dataset's structure, reducing the distance between points and their assigned centre.

Overall, the trend captured in Figure 1 is similar across most functions and dimensions. However three function IDs deviate from the general pattern. For `Cluster_ruspini_selected` at $k = 2, 3$ (dimension 4, 6), the MSE values are higher than other functions at the same dimension. In contrast, `Cluster_kc1` and `Cluster_Segment` show comparatively lower MSE values across all dimensions.

This pattern is consistent with the KMeans++ baseline values reported in Appendix A.3, which show the same difference between datasets. These differences are also in line with the landscape properties discussed in Section 2.1.2.

2.2 Metaheuristics

Metaheuristics are optimisation strategies that guide the search for candidate solutions in a search space. This thesis focuses on three main types: evolutionary algorithms, swarm techniques and trajectory based.

Evolutionary algorithms (EA) are inspired by biological evolution for continuous parameter optimisation. In line with the 'survival of the fittest' metaphor, the algorithm searches for the solution that best fits the problem. An initial population of candidate solutions is generated and evaluated on an objective function that determines the fitness [20]. Higher fitness individuals are selected to 'survive' and are used as parents for further generations. EAs are particularly well-suited to problems where the fitness landscape has multiple local optima, as maintaining a population of diverse candidates prevents premature convergence.

The following algorithm families fall under evolutionary algorithms. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [14] is considered the state-of-the-art for continuous black-box optimisation. Particularly suited to non-convex landscapes, CMA-ES samples the search space to calculate the covariance matrix and mean search distribution [14]. The covariance adapts to the shape of the fitness landscape and the value of the mean search distribution determines where candidate solutions are sampled from. Since CMA-ES was used in the original evaluation of the clustering benchmark [31], it also serves as a key baseline in this thesis.

Differential Evolution (DE) [29] generates new candidate solutions by perturbing possible vector solutions using scaled differences between existing population members. This is effective because the search direction considers the distribution of the existing population.

The Univariate Marginal Distribution Algorithm (UMDA) is an estimation-of-distribution algorithm that samples new candidate solutions from probability distributions learned from the best-performing individuals in the current population [6]. Allowing the identification of high performing regions in the current landscape. Memetic algorithms [22] adapt evolutionary algorithms with local search mechanics. The evolutionary process ensures exploration and local search acts as a 'sanity check' for real improvement. In this thesis it appears as an emerged algorithm, instead of a provided example.

Figure 2 shows a partial tree of the metaheuristic families relevant to this thesis. The created figure was informed by the general classification of metaheuristics and genetic algorithms [28, 27]. The dichotomy between techniques is created through their search mechanism.

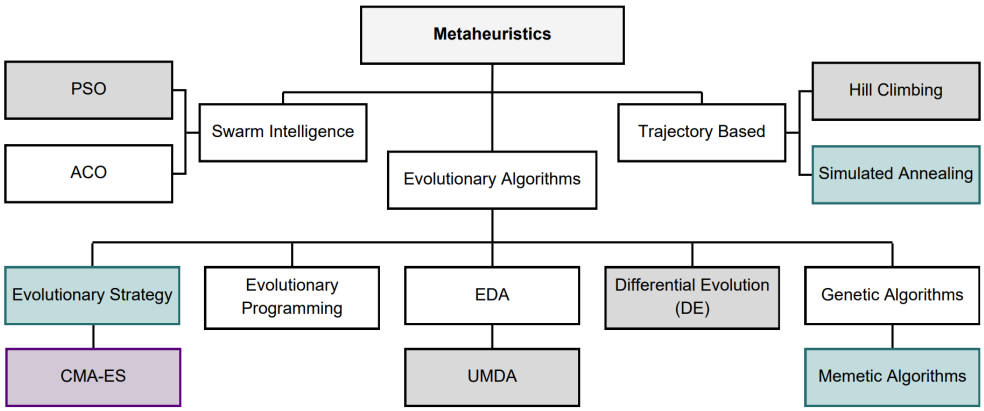


Figure 2: Families of metaheuristic algorithms, the gray highlighted categories (PSO, DE, UMDA, Hill Climbing) are the example algorithms provided to LLaMEA in the E3, E4, E5 and E6 configurations. The teal highlighted categories (Evolutionary Strategy, Simulated Annealing and Memetic Algorithms) emerged during the evaluation runs but were not provided as examples to LLaMEA, as seen in Section 5. CMA-ES is indicated with a purple highlight as it is used as a baseline.

Swarm intelligence is a stochastic search method, influenced by the organisation of animals and insects [7]. Information is shared across the swarm and used to update solutions. The current and prior experience of the entire swarm is taken into consideration [11]. In comparison to EA methods, the candidate selection mechanic differs and convergence tends to be slower [7].

Particle Swarm Optimisation (PSO) [17] maintains a set of solutions influenced by their own best known position and the global best found by the swarm.

Trajectory based methods maintain and iteratively improve a single candidate solution rather than a population, navigating the search space through a sequence of local moves [4]. The following algorithm families fall under the Trajectory based category.

Simulated Annealing (SA) iteratively improves a single candidate solution. To prevent getting stuck in a local optimum, it uses probability to explore alternative solutions [1].

Hill Climbing is a trajectory-based method that iteratively moves from the current solution

to a neighbouring solution if it improves fitness, and terminates when no improving neighbour exists [26]. It is computationally inexpensive but highly susceptible to local optima.

Despite their diversity, all metaheuristics share a fundamental limitation in the context of algorithm discovery: their variations are fixed. Regardless of the problem structure, a DE algorithm always applies vector difference mutations, and a PSO algorithm always updates velocities according to the same rule. This constrains the diversity of solutions that can be explored and motivates the use of LLM algorithm generation, elaborated on in Section 2.3.

2.3 LLaMEA

The Large Language Model Evolutionary Algorithm (LLaMEA) utilises LLMs in an evolutionary loop to generate metaheuristic algorithms. Rather than mutating the parameters of a fixed algorithm, LLaMEA prompts the LLM to generate new algorithm implementations at each generation step [30]. Each generated algorithm is treated as an individual in the population.

This design has two key advantages over traditional EAs. First, the search space is no longer restricted to the parameter space of a specific algorithm. LLaMEA can generate structurally distinct algorithms at each step, including hybridization of metaheuristic types. Second, the LLM brings implicit prior knowledge from its training data: it has been exposed to a large corpus of algorithm implementations and can draw on this knowledge when generating new candidates.

The behaviour of LLaMEA is sensitive to how it is prompted. The structured prompt provided to the LLM determines the context available to it during generation. There are four main steps for the generation of the best fit algorithm for a problem. The initialisation step receives a structured prompt that consists of the role, task, example and format prompts. The role prompt is used as a prompt engineering strategy to enhance the performance of the LLM. Similarly the task and example prompt provide the LLM with a reference point for the type of algorithm to generate.

The algorithm generation step includes extracting the algorithm from the LLM in the required format as specified by the format prompt. A version of the algorithm is instantiated to catch syntax or logic errors made by the LLM. These errors are then assigned a negative fitness and included in the feedback loop.

The evaluation step scores each generated algorithm on a provided benchmark based on the problem. The performance or fitness is calculated by the evaluation function provided to LLaMEA. In our case, this evaluation function returns the MSE improvement over KMeans++, described further in Section 4.2.2.

The feedback and mutation step provides the original task prompt and the history of previously tried algorithms and their scores along with a parent algorithm (a previous generation) to improve upon.

The paper introducing LLaMEA [30] shows that LLaMEA, using GPT-4o as the underlying model, is capable of outperforming CMA-ES on the BBOB benchmark suite [15]. However LLaMEA has not previously been applied to clustering optimisation problems. Whether it can meaningfully tackle the challenges posed by clustering problems, such as symmetry and regions of neutrality [31] will be investigated in this thesis. The sensitivity of LLaMEA to prompt design also raises the question of whether domain knowledge about these properties affects the generated algorithms. These two questions motivate the experimental setup of this thesis, described in Section 4.

3 Related Work

3.1 Automatic Algorithm Design

The goal of Automated Algorithm Design (AAD) is to reduce the manual effort required to produce effective optimisation algorithms. Hyper-heuristics [5] moved beyond this by automating the construction of new heuristics from low-level operators, though the results remain bound to this predefined operator set. The emergence of LLMs capable of generating and reasoning about code [10] caused a paradigm shift. Rather than the recombination of operators seen in Hyper-heuristics, LLMs directly generate novel algorithmic structures. FunSearch [24] demonstrated this by pairing an LLM with a systematic evaluator in an evolutionary loop, discovering new solutions to combinatorial problems. Evolution of Heuristics (EoH) extended this by evolving both natural language descriptions of algorithmic ideas and their code implementations, outperforming hand-crafted heuristics on combinatorial optimisation benchmarks [19].

LLaMEA [30] was selected as the basis for candidate generation in this thesis because, unlike FunSearch and EoH which focus primarily on combinatorial heuristics, LLaMEA was developed specifically for continuous metaheuristic design and has been demonstrated to outperform CMA-ES on the BBOB suite [30]. This makes it the most directly applicable framework for this thesis.

3.2 Automatic Algorithm Analysis

Understanding how problem structure affects algorithm behaviour requires infrastructure for running and logging the resulting experiments.

Exploratory Landscape Analysis (ELA) [21] defines a set of computable features for continuous optimisation problems that describe properties such as convexity and local-search behaviour. These features have subsequently been used for automated algorithm selection on continuous black-box optimisation benchmarks [18]. However, as discussed in Section 2.1.2, clustering landscapes exhibit structural properties such as permutation invariance and regions of neutrality. These properties are central to the search space, but are not explicitly represented by standard ELA features originally developed for continuous BBOB-style benchmark functions. In Appendix A.3, an ELA random walk technique was applied to our benchmark problems to provide a simplified impression of variation in the benchmark functions.

On the infrastructure side, IOHprofiler [8] provides a benchmarking and profiling environment for iterative optimisation heuristics, supporting standardised logging and analysis of algorithm performance across problem instances. The clustering benchmark suite used in this thesis [31] is integrated with the IOHprofiler, enabling reproducible comparison between optimisers. Consistent use of this framework across experiments ensures that LLaMEA generated algorithms and the CMA-ES baseline are compared under the same benchmark conditions.

4 Methodology

4.1 Experimental Setup

To address the research questions, we conducted experiments applying the generated LLaMEA algorithms to clustering problem benchmarks. All experiments used a fixed budget of $T = 200$ LLM

calls per run, with 5 runs per configuration. A benchmark instance is defined by fid (function ID), iid (instance id) and k (number of clustering centres). The solution representation encodes cluster centres as a continuous decision vector. Over a single run, LLaMEA requires training instances and testing instances. Our split is fid= 1- 8 for training and fid = 9-10 for testing, with iid and k kept constant over both training and testing splits. This separation ensures that the test set remains unseen for the generalisability analysis in Section 5.4.1.

The fitness metric returned to LLaMEA is:

$$\text{Fitness} = \frac{1}{T}\sum_{t=1}^{T} MSE_{min(KMeans++)} - MSE_{min}(t) \tag{3}$$

$MSE_{min(KMeans++)}$ is the minimum MSE achieved by KMeans++ on the same instance, and $MSE_{min}(t)$ is the minimum MSE found by the generated algorithm up to evaluation t. This fitness value measures improvement over the KMeans++ baseline. LLaMEA maximises this fitness function over evaluations to find better candidate algorithms.

The LLM used for all configurations is `qwen3-coder:30b`. Performance is reported as the mean best so far fitness, as defined in Equation 4 and evaluated through the Clustering Adapter in Algorithm 2.

4.2 Clustering Adapter

The clustering adapter connects the IOHProfiler evaluation [8] with instances of the clustering problem. The adapter's purpose is to construct the prompt and to evaluate the performance of the generated algorithm on the instance of the benchmark.

4.2.1 Prompt Construction and algorithm generation

The prompt construction depends on the prompting strategy defined in Table 1. All configurations use a task_prompt, mutation_prompts and a feedback_prompt. As seen in Algorithm 1, at initialisation the feedback prompt is empty, since no candidate has been evaluated yet. During the evaluation loop the feedback prompt is updated with the performance of the prior algorithm.

If warm start is enabled, the initial population is seeded with n_parents example algorithms before the main generation loop begins. These examples provide a structured starting point and introduce variation in the types of algorithms that may be generated. Without warm start, the first algorithm is generated with the task_prompt and the selected mutation_prompt. Enabling exemplar-injection does not change the seeding process of LLaMEA. Instead, it provides the exemplar algorithm as an addendum to the current prompt. It does this by replacing the current mutation_prompt with the exemplar-injection algorithms.

During the budget loop $t < T$, the subsequent generations are constructed using the feedback_prompt of the prior algorithm. The algorithm generates a new candidate solution as a_{t+1}, which is evaluated using the `EVAL()` function in Section 4.2.2. If the resulting fitness improves upon the current best_y value, then both y_b (fitness) and a_b (current best algorithm) are updated.

Algorithm 1 LLaMEA generation

$S \leftarrow task\_prompt$, $M_0 \leftarrow [mutation\_prompts]$, $F_0 \leftarrow feedback\_prompt$
$t \leftarrow 0$
$m \leftarrow random.choice(M)$
$y_b \leftarrow \emptyset$, $a_b \leftarrow \emptyset$ ▷ Initialise best fitness and algorithm
if Warm_Start = True **then**
  for i = 0 to n_parents **do**
    $example\_prompt \leftarrow example\_prompts[i]$ ▷ Provided example algorithm till n_parents
    $a_i \leftarrow LLM(S + m + example\_prompt)$ ▷ Generate a candidate for seeded algorithm
  end for
else
  $a_0 \leftarrow \text{LLM}(S + m)$ ▷ Generate first candidate
end if
while $t < T$ **do** ▷ Budget not exhausted
  $F_t \leftarrow feedback(y_t, a_t)$ ▷ Construct feedback prompt from current candidate
  $a_{t+1} \leftarrow LLM(S + F_t + m)$ ▷ Generate next candidate
  $y_{t+1} \leftarrow EVAL(a_{t+1})$ ▷ Evaluate on training instances
  if $y_{t+1} \geq y_b$ **then**
    $y_b \leftarrow y_{t+1}$, $a_b \leftarrow a_{t+1}$ ▷ Update best if improved
  end if
end while

4.2.2 Evaluation Function

The fitness of each algorithm generated by `qwen3-coder:30b` is determined by Algorithm 2. The algorithm is executed in a temporary environment to check for errors or logic inconsistencies. A validation step is necessary because LLM generation is imperfect. Algorithms that raise errors are penalised with a fitness of -10^9. This choice was made during early experiments, as heavier penalties resulted in less exploration. However algorithms that produce no valid evaluations are penalised with a fitness of $-\infty$.

For valid algorithms, the adapter evaluates the performance across all training instances passed to the evaluation function. For each instance (fid, iid, k), the best MSE is recorded and compared to the KMeans++ baseline from the paper [31].

The fitness contribution is computed as the mean improvement over the KMeans++ baseline.

$$\text{mse\_imp} = \text{mean}\left(\text{baseline\_mse}[(fid, k)] - y_b\right) \tag{4}$$

A positive value indicates the generated algorithm outperforms KMeans++ on that instance; a negative value indicates it performs worse.

Algorithm 2 ClusteringProblemAdapter

Require: `training_instances`, `t`, `T`, `testing_instances`, `baseline_mse`[(`fid`,`k`)] ▷ KMeans++ baseline for fitness
  function EVAL(a)
    $Code \leftarrow exec(a.code)$ ▷ Code executed in temporary environment
    if $Exception == True$ **then**
      return $-1e9$ ▷ Penalise errors
    end if
    for each $(fid, iid, k) \in training\_instances$ **do**
      $p \leftarrow GetProblem(fid, iid, k)$ ▷ Load benchmark problem
      $y_b \leftarrow$ min-so-far MSE at each evaluation of $Code(p)$
      $baseline\_mse \leftarrow kmeans\_min.get(fid, k)$
      if $|y_b| = 0$ **then**
        return $-\infty$ ▷ Generated algorithm made no evaluations
      end if
      $y_b \leftarrow y_b[:T]$ **if** $|y_b| \geq T$ ▷ Ensure budget is upheld
      $mse\_imp \leftarrow$ mean$(baseline_mse[(fid, k)] - y_b)$ ▷ Mean improvement over (fid, k)
      $fitnesses$.append(mse_imp)
    end for
    return $\frac{1}{T} fitnesses$ ▷ Aggregated fitness returned to LLaMEA
  end function

4.3 Configuration of Prompting Strategy

Six experimental configurations were tested to evaluate the performance of LLaMEA on clustering problems. The prompting strategy factor determines how example algorithms are provided, to ensure diverse solutions are found.

Configuration	Domain Knowledge	Strategy	Elitism
E1	$\neg\mathcal{D}$	Baseline	Yes
E2	$\mathcal{D}$	Baseline	Yes
E3	$\neg\mathcal{D}$	Exemplar-injection	Yes (also w/o)
E4	$\neg\mathcal{D}$	Warm start	Yes (also w/o)
E5	$\mathcal{D}$	Exemplar-injection	Yes
E6	$\mathcal{D}$	Warm start	Yes

Table 1: Summary of the six experimental configurations.

Baseline (E1, E2). In the baseline strategy, LLaMEA receives only the task prompt and the previously generated parent candidate. E1 uses the task prompt $\neg\mathcal{D}$ provided in Appendix A.6.1, whereas E2 additionally includes descriptions of clustering-specific landscape properties $\mathcal{D}$ provided in Appendix A.6.2.

Exemplar-injection (E3, E5). In the exemplar-injection strategy, each LLM call additionally injects an example algorithm from a metaheuristic family, as shown in Appendix A.6.3. As a result, the LLM receives not only the current parent candidate, but another example that may come from

a different algorithm family. This strategy is intended to encourage broader exploration of the algorithmic design space and reduce early convergence to a single type of generated optimiser.

Warm start (E4, E6). In the warm start strategy, the initial population is seeded with $n_$parents example algorithms, giving the search a structured starting point, as shown in Appendix A.6.4. Unlike exemplar-injection prompts, these examples are only used during initialisation and are not repeatedly injected into later LLM calls. This strategy ensures generations evolve from the seeded population and is intended to stabilise convergence to an algorithm type.

Elitism ablation. All six main configurations use elitism by default, with 4 parents and 8 children per generation. To further investigate the impact of selection pressure on algorithm diversity, the Exemplar-injection and Warm start strategies were evaluated without elitism. In the without elitism approach the high performing candidates from the parent population can be lost through subsequent generations. Therefore, increasing the number of children (4 parents and 16 children) provides each generation a larger pool of offspring from which promising candidates could be selected.

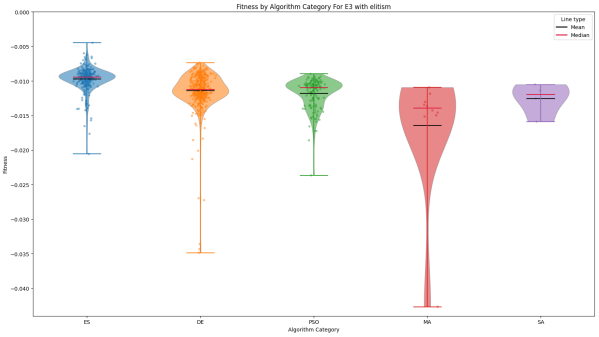
5 Experiments

5.1 Algorithm Categories

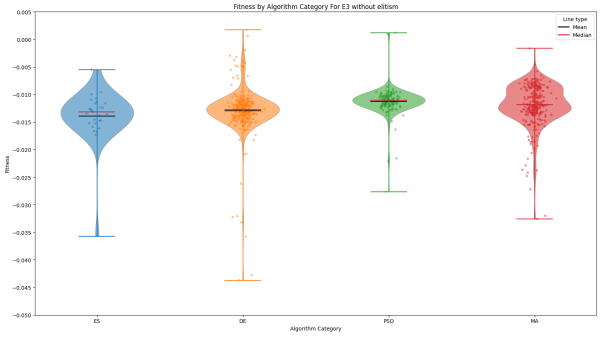
Variation in generated candidate solutions was encouraged through E3 and E4. As mentioned in Section 2, there are different types of metaheuristics that can be generated. The algorithms generated by LLaMEA are split into 5 categories: Evolutionary Strategy, Differential Evolution, Particle Swarm Optimisation, Memetic Algorithm and Simulated Annealing. The distinction between algorithms was created through the name or description of an algorithm generated by LLaMEA. Two algorithms emerged that were not present in the initialisation prompt for either E3 or E4, these were ES, MA and SA.

The results of each experimental setup were processed by filtering out algorithms that had fitness values of -10^9, as these algorithms returned errors. Figure 3 compares the fitness distribution per algorithm category, both with elitism and without. The distribution per category changes depending on whether elitism is enabled. With elitism, there were 496 algorithms generated for DE, while without elitism this was significantly more balanced. Elitism seemed to have more of an impact on E3, for the diversity of algorithms produced. This stems from the fact that a random algorithm is provided in the exemplar-injection prompt. Therefore the selection mechanism of elitism, seems to have guided the choice of algorithm category (more than the random example algorithm provided). For exemplar-injection prompts with elitism after filtering 868 valid points remained, whereas without elitism, there were 787 valid calls. The fitness distributions are broadly similar in mean, suggesting that the diversity of algorithms did not cost performance. However the whisker plot showed a wide range for E3, so the runs were inconsistent.

This can be attributed to the random example algorithm provided in the exemplar-injection prompt: under elitism, the selection mechanism can reinforce or discard this initial choice consistently across a run, producing the DE distribution visible in Figure 3a, whereas without elitism the larger offspring pool dilutes the influence of any single example.



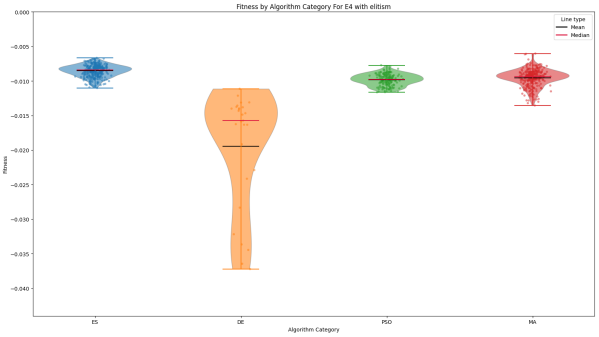
(a) Exemplar-injection Prompt E3: With Elitism



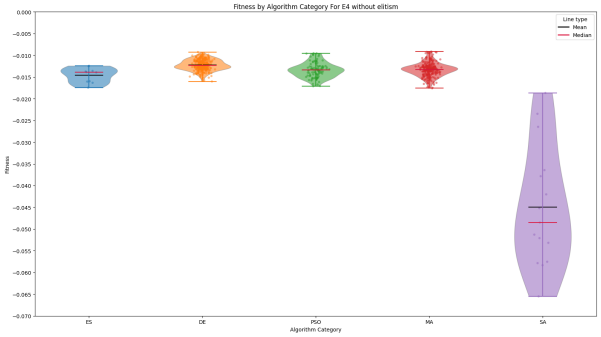
(b) Exemplar-injection Prompt E3: Without Elitism

Figure 3: Fitness distribution per algorithm category under Warm start, comparing both with and without Elitism. The whisker plot indicates the outliers in the plot, each scatter point represents a completed candidate of that family.

Warm start was the second strategy used to implement the diversity of generated candidates in Figure 4. In direct contrast to E3, this resulted in a more consistent spread of algorithms regardless of whether elitism was enabled or not. The with elitism configuration again generated more valid algorithms 899 in comparison to 781. With elitism the candidates contained mostly ES, PSO and MA algorithms. Without elitism most of the algorithms were DE and MA, however SA was included as well. The fitness distributions achieved across ES, PSO and MA were similar. Warm start seemed to increase the diversity of algorithms more than exemplar-injection prompts. In addition the fitness had fewer outliers, especially for algorithm categories that had at least 100 candidate solutions. By contrast, exemplar-injection prompts regardless of the quantity of generated solutions there was a higher variance in performance. Indicating that exemplar-injection prompts drive broader exploration of the algorithm space at the cost of consistency within any single category.



(a) Warm start E4: With Elitism



(b) Warm start E4: Without Elitism

Figure 4: Fitness distribution per algorithm category for E4 (Warm start), comparing with and without elitism. The whisker plot indicating the outliers in the plot, each scatter point represents a completed candidate of that family.

Figure 5 demonstrates how the proportions of algorithm categories change over the LLM calls without elitism. This indicates how the diversity of algorithms change over the course of the run.

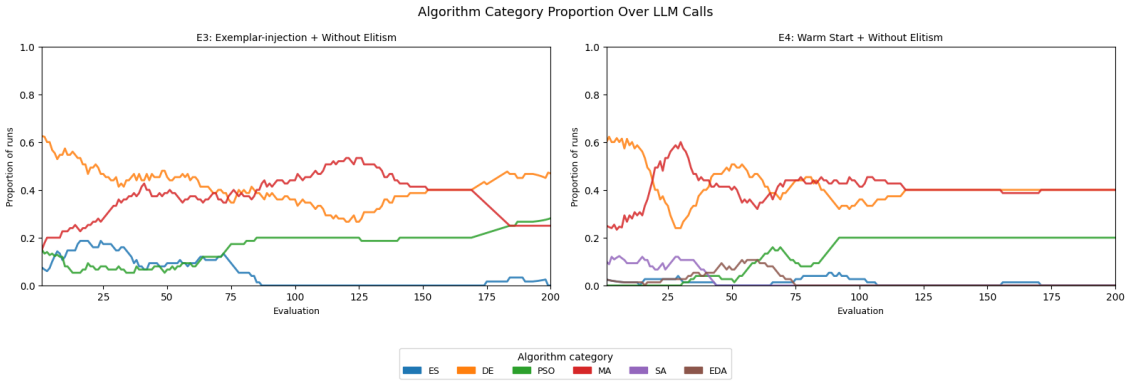


Figure 5: Proportion of algorithm categories generated over 200 LLM calls, without elitism, for E3 (Exemplar-injection prompts) and E4 (Warm start).

Exemplar-injection prompts show high volatility, which indicates sustained exploration. In the Warm start configuration without elitism, the proportions are still dynamic. However, they indicate stronger exploration in the early calls. This difference is still conflated with the larger offspring pool, as increased exploration may stem from the offspring size rather than the absence of elitism alone. After the 120th call the proportion of algorithm categories stabilise. While Exemplar-injection prompts do not clearly converge to a specific algorithm type, Warm start does.

Removing elitism increases the exploration of the different algorithm categories. However this does not directly lead to an improvement of the fitness. As noted above, there were fewer stable candidates generated and more inconsistency within the fitness.

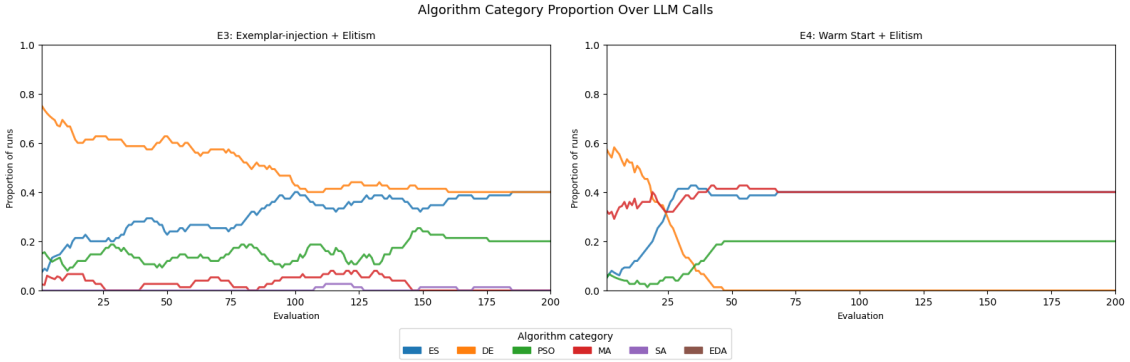


Figure 6: Proportion of algorithm categories generated over 200 LLM calls, with elitism, for E3 (Exemplar-injection prompts) and E4 (Warm start).

Figure 6 shows how the proportions of algorithm categories shift with elitism. In comparison with Figure 5, both E3 and E4 are more stable. This is specifically visible in warm start where the search quickly converges to a subset of different algorithms by evaluation 50. This shows how the selection pressure from elitism narrows the exploration and types of algorithms generated. There is a clear stability-exploration trade-off, most evident in the exemplar-injection configuration as sustained algorithm diversity came at a cost to fitness consistency.

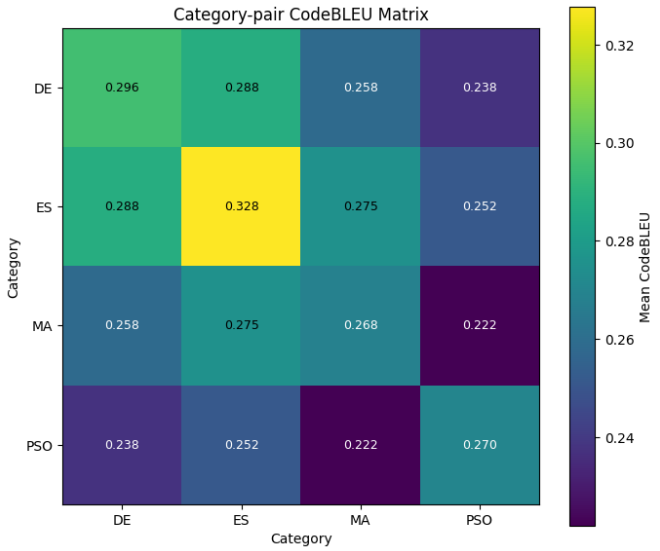


Figure 7: Mean CodeBLEU similarity between pairs of algorithm categories, computed across all generated algorithms.

CodeBLEU is an evaluation metric for generated code that combines n-gram match, abstract syntax tree (AST) match, and data-flow match [23]. CodeBLEU is used to quantify the structural similarity between pairs of LLaMEA-generated algorithms, allowing the relationship between code similarity and fitness difference to be examined. The metric returns a similarity score for algorithm pairs, between 0 and 1, where values closer to 0 indicate low similarity and values closer to 1 indicate a close match [23]. Additionally, this analysis provides greater context into the algorithm families that are generated by LLaMEA. Specifically, it indicates whether candidate solutions under the same family are structurally different. Figure 7 shows the mean CodeBLEU similarity between each pair of algorithm categories: algorithms within the same category (e.g. ES vs. ES, on the diagonal) generally show a higher similarity (0.328). While structurally distinct categories such as MA and PSO show the lowest (0.222). However, because this is an average the difference is less pronounced, leading to Figure 8.

Figure 8 examines whether structural similarity between two algorithms predicts how different their fitness is. The within-category all panel (top left) provides the clearest signal: median fitness difference decreases monotonically from approximately 7×10^{-3} at low similarity to below 10^{-3} at high similarity. The y-axis log scale was used to make the differences more visible. The unscaled MSE values are provided in Appendix A.2. The trend is clear higher structural similarity corresponds to substantially smaller fitness gaps. This relationship holds consistently for DE vs ES, DE vs MA, and ES vs MA, all of which have sufficient sample sizes across the similarity range to support the trend. The PSO pairs (DE vs PSO, ES vs PSO and MA vs PSO) show a broadly similar pattern, though data becomes sparse above similarity 0.5, 0.45, 0.5 respectively. Indicating that almost all generated algorithms within these categories are structurally different and the differences between algorithm families are pronounced.

MA vs PSO presents the flattest relationship of all pairs, with median fitness difference remaining relatively stable across the full similarity range, suggesting that for this specific combination, structural similarity is less predictive of fitness similarity than for other pairs. ES vs MA panel, from $0.18 \leq$ similarity ≤ 0.32, indicates a shift in trend in comparison to the other figures. The

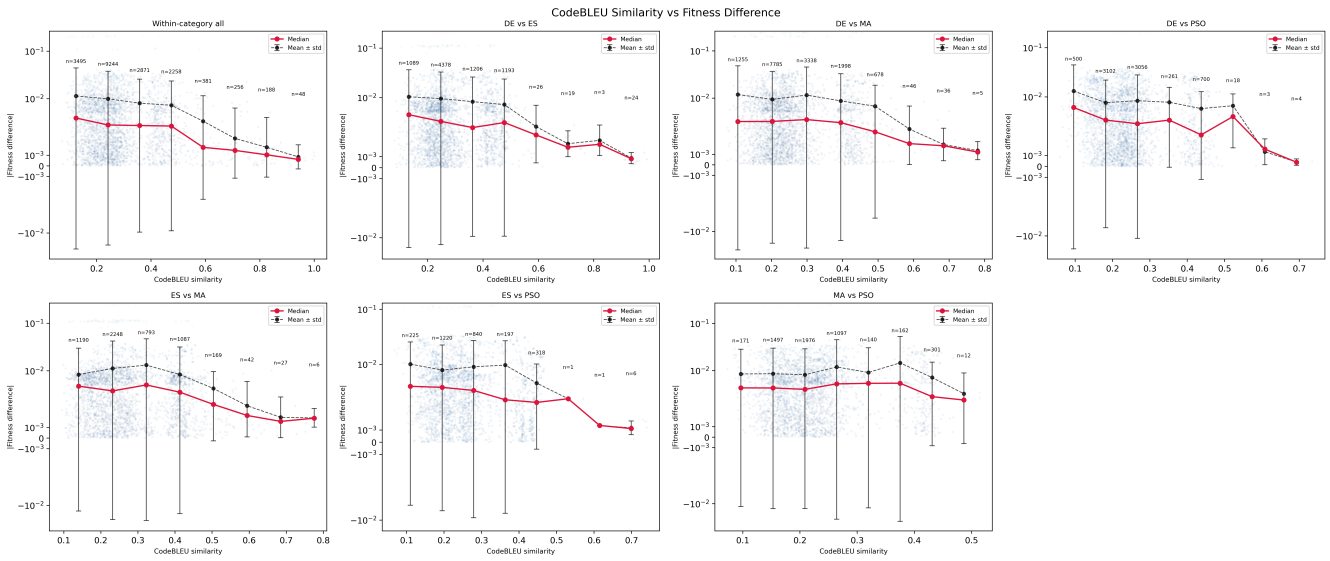


Figure 8: CodeBLEU similarity versus absolute fitness difference for each pair of algorithm categories. Points show individual pairwise comparisons; the red line shows the median, with error bars showing mean ± one standard deviation. n indicates the number of pairs in each similarity bin.

fitness difference increases gradually plateauing at similarity = 0.32 and then resuming the expected downward trend. This suggests that in this structural similarity range, ES and MA algorithm pairs share surface-level code features, that do not result in similar fitness values. DE vs ES and DE vs MA show a directly comparable monotonic trend as observed in the Within-category all panel.

The CodeBLEU results complement the above Figures 4, 3, as we can compare fitness distributions per algorithm category and the structural differences observed. The fitness distributions for ES, PSO, and MA are broadly similar and have comparable median values. CodeBLEU analysis confirms that even distinct algorithms within those categories do not consistently achieve different fitness outcomes. Although the generated algorithms have clear structural differences, these differences do not correspond to significant fitness variations between these algorithm families.

5.2 Comparison of Prompting Configurations

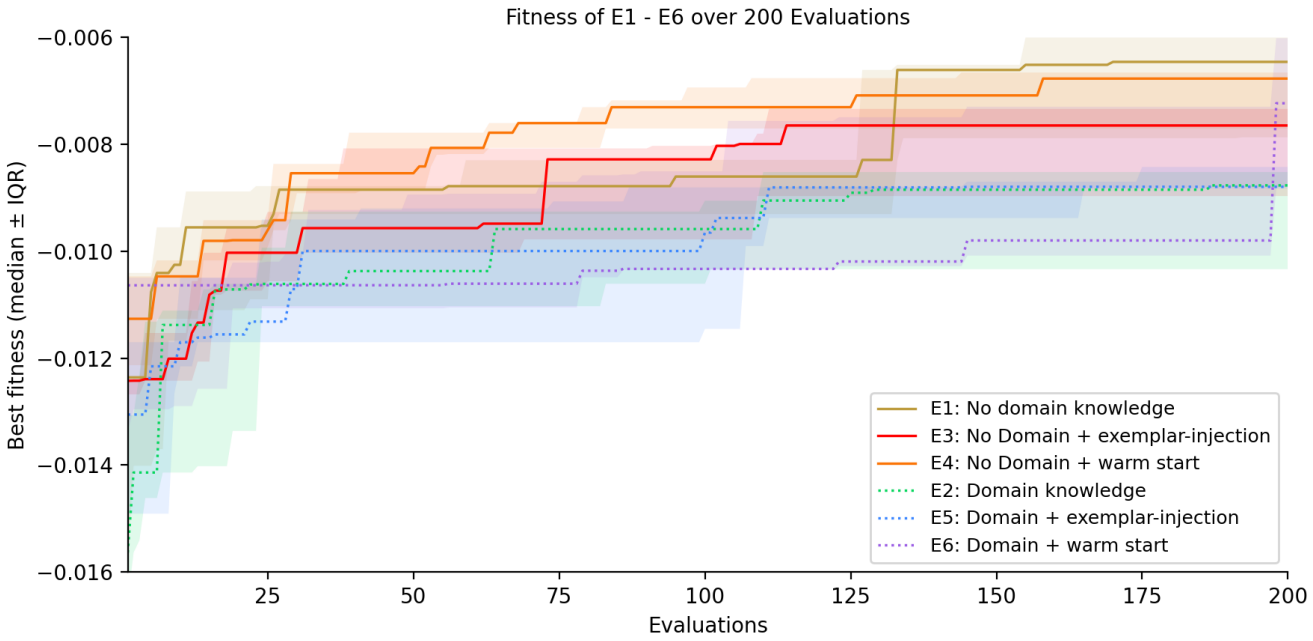
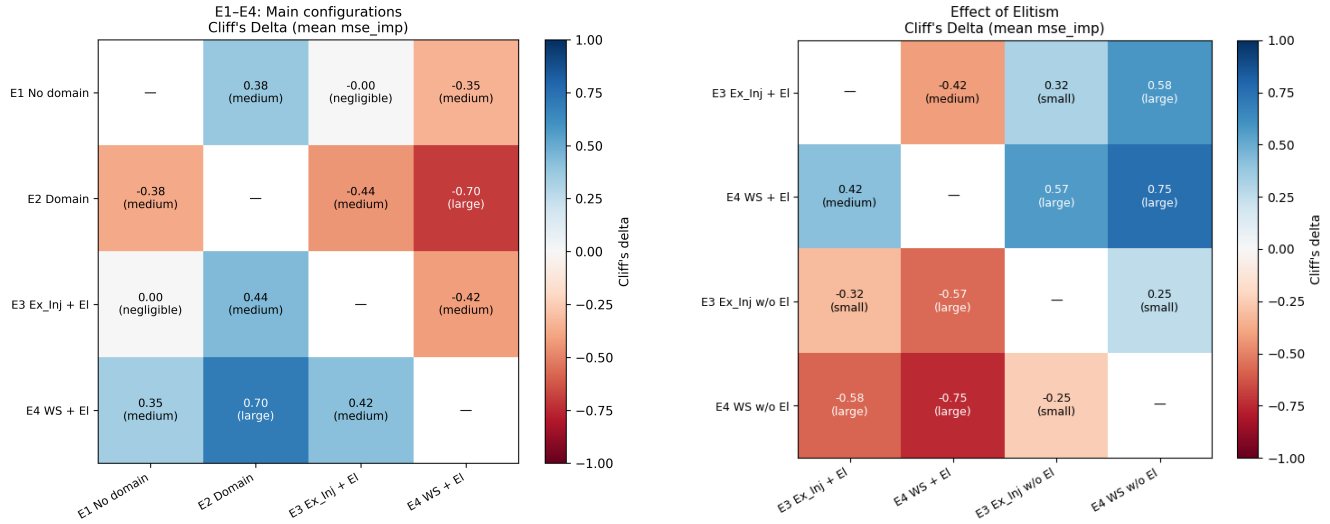


Figure 9: Fitness over time (improvement over the KMeans++ baseline) for each prompt configuration, with elitism, over 200 evaluations.

Figure 9 compares the fitness over time of each prompting configuration. It indicates how the fitness changes for each of these experimental setups. As seen in Section 4.2.2, the fitness is calculated as improvement over the KMeans++ baseline. Therefore values closer to 0 indicate better performance. All $\neg\mathcal{D}$ configurations perform better than their counterparts. It is a counterintuitive result, as we expected that providing the LLM descriptions of clustering landscape features would improve the result. However, including $\mathcal{D}$ seems to have constrained the generations to prioritise those specific properties. The IQR values of the domain lines (E2, E5 and E6) are wider, indicating wider variance. Hence the LLM is attempting to find more diverse solutions, but due to the specificity of the task prompt provided. It is unable to achieve meaningful improvement over the $\neg\mathcal{D}$ configurations.

The advantage of Warm start is evident with E4 consistently outperforming the other configurations from evaluation 30 onwards. The IQR range is lower, so after reaching a proportion of high performing algorithms it limits its exploration. The IQR range of exemplar-injection prompts E3 is significantly higher compared to E4. This was also evident from the IQR ranges in 4, where exemplar-injection prompts demonstrated higher exploration. The overall median of E3 is around -0.008 while E4 achieves a fitness of around -0.006. It should be noted that there is very minimal difference between configurations. Although, some performance comparisons can be made. More than 5 runs are necessary to identify significant patterns.



(a) E1–E4: Main configurations

(b) Effect of elitism in E3 & E4

Figure 10: Pairwise Cliff's delta effect sizes comparing main configurations and elitism. Positive values indicate that the row configuration outperforms the column configuration.

Figure 10 provides a comparison between the different configurations using Cliff's delta. Beyond the convergence curve that demonstrates performance over time, Cliff's delta quantifies the magnitude of pairwise differences. It measures the probability that one configuration outperforms another. The left heatmap compares E1-E4, which were shown in Figure 9. The reported effect sizes demonstrated that the different prompting strategies affected performance. Specifically warm start and exemplar-injection prompts produced meaningful differences. The performance of E4 measurably outperformed E2, with a large effect size 0.70. By contrast, the comparison between E3 and E1 yielded a negligible effect, indicating that, despite the visible separation in the convergence plot, the exemplar-injection prompt alone was not sufficient to produce a discernible difference from the $\neg\mathcal{D}$ baseline once aggregated across all problems. E4 also outperformed E1 (0.35, medium effect), though this margin was smaller than the effect of E4 over E3 (0.42, medium effect). This suggests that the warm start configuration's advantage was more pronounced relative to the exemplar-injection configuration than relative to the $\neg\mathcal{D}$ baseline strategy.

The right heatmap compares the elitism configurations and the impact of preserving the best-found solutions across generations. While direct comparison between elitism and without elitism configurations is limited, the results can be used as an indication of how performance changes without elitism alongside an increase in offspring. The without elitism configurations consistently delivered worse performance than their with elitism counterparts. Following the trend, 'E3 + El' performed worse than 'E4 + El', which is also visible in the fitness graph. One notable exception to this overall pattern was 'E3 w/o El', which showed better performance than 'E4 w/o El'. This reversal can be explained by the differing roles elitism plays for each prompting strategy. The advantage of warm start depends on that initial seeded solution being preserved across generations. Without elitism, the favourable seeded candidates can be lost, causing E4 to suffer disproportionately. The exemplar-injection configuration (E3), by contrast, does not solely rely on the parent solutions, so removing elitism has a comparatively smaller cost and may even support the exploratory diversity that the exemplar-injection prompt is designed to encourage.

The results indicate that elitism is beneficial, particularly for warm start. However the without elitism comparison should not be interpreted as isolating the effect of elitism alone. As the observed differences reflect the combined effect of selection strategy and offspring-size adjustment.

5.3 Comparison of candidates with Baselines

Figure 11 shows the convergence of the best algorithm against the CMA-ES and KMeans++ baselines over 5000 function evaluations. The choice of best algorithm was based on the algorithm with the highest recorded fitness from all 5 runs of a prompting configuration. This comparison differs from Section 5.2 in two important ways. First, where Section 5.2 reported the aggregated LLaMEA fitness metric: the mean improvement over KMeans++ across the evaluation trajectory, Figure 11 shows the median MSE improvement of a single best discovered algorithm evaluated over 5000 function evaluations. Second, rather than averaging across all 200 generated candidates, we isolate the ceiling performance of each configuration by selecting the single best algorithm it produced.

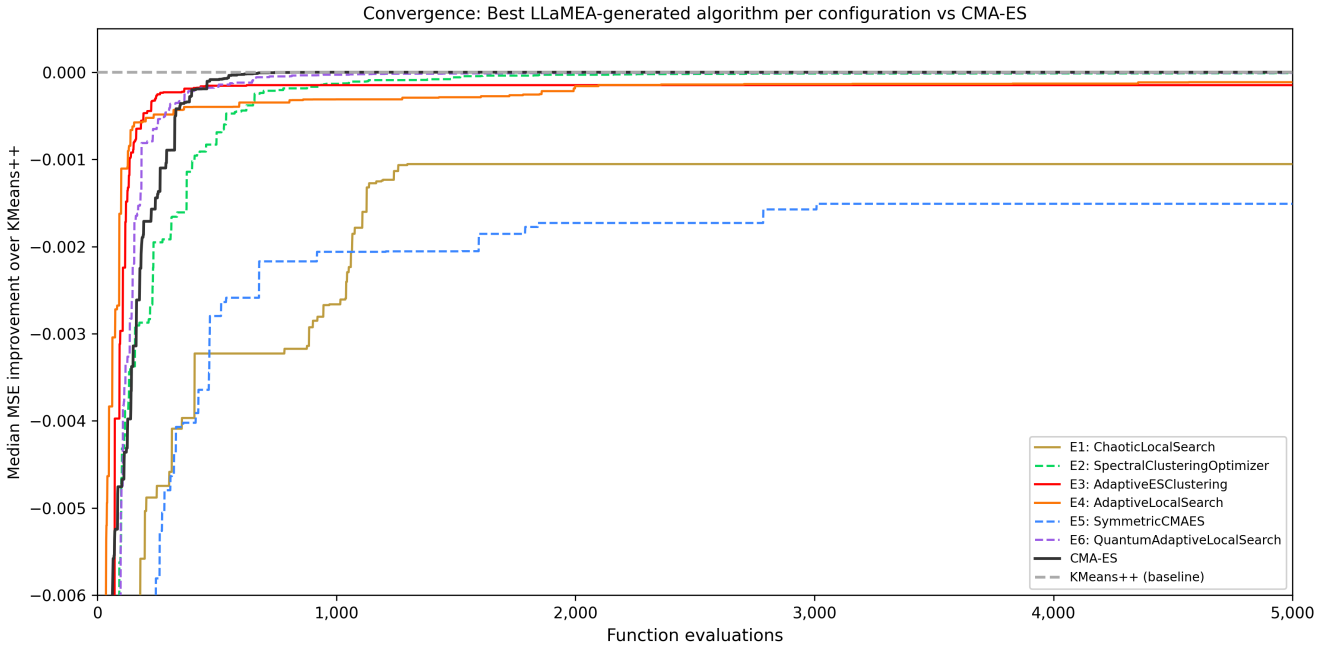


Figure 11: Convergence of the best LLaMEA generated algorithm per configuration compared to CMA-ES and the KMeans++ baseline. Measured across all function IDs and dimensionalities with 5000 evaluations.

Notably, the two algorithms domain knowledge algorithms (E2, E6) converge to a higher median fitness than compared to E3 and E4. This inverts the observed pattern in Figure 9 to a degree. The two closest algorithms to the KMeans++ baseline are E6: `QuantumAdaptiveLocalSearch` and E2: `SpectralClusteringOptimiser`, both converge to -0.00001. E6 converges quicker than E2, however the convergence rate of both $\neg\mathcal{D}$ configurations are slower than the best two algorithms from $\mathcal{D}$ configuration. The best $\neg\mathcal{D}$ candidate algorithms are E3: `AdaptiveESclustering` and E4: `AdaptiveLocalSearch`, converge to -0.00015 and -0.00011 respectively. E1: `ChaoticLocalSearch`

and E5: `SymmetricCMAES` are the weakest individual candidates, with their Median MSE convergence well below the other configurations. CMA-ES is the only algorithm that reaches the KMeans++ baseline exactly.

E3 has the most rapid convergence among all configurations, by approximately evaluation 400, after which it shows no further improvement. E4 has the steepest initial improvement and similar to E3 appears to plateau around evaluation 400, until it starts making incremental stepwise improvements throughout the remaining budget. Leading to a lower final MSE than E3. Figure 9 demonstrates this as well, E4 achieves a higher median fitness than E3, confirming that warm start outperforms exemplar injection at both the aggregate and individual algorithm level.

The difference between LLaMEA configurations and the individual algorithms is meaningful. $\mathcal{D}$ configurations produced higher aggregate variance in Section 5.2, so the majority of generated algorithms performed poorly. The $\neg\mathcal{D}$ configurations were more consistent overall. Domain knowledge therefore acts as a high variance strategy, constraining most generations but enabling a small set of candidates to be better suited to clustering problems.

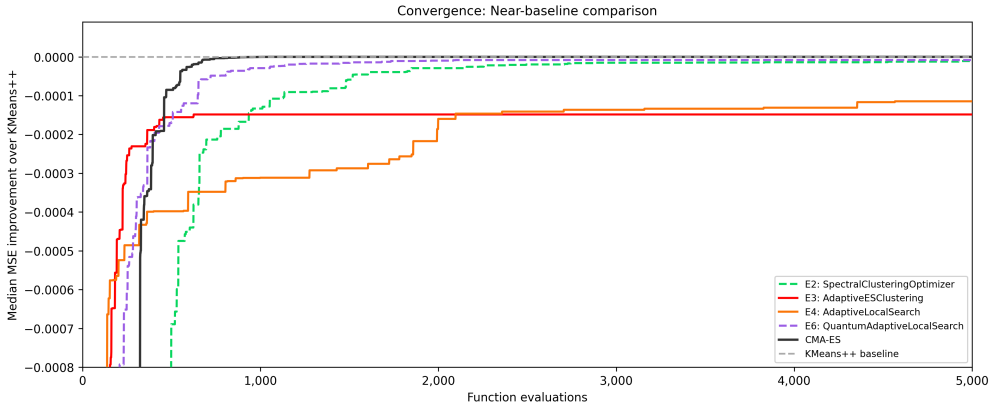


Figure 12: Zoomed convergence near the KMeans++ baseline, showing the four closest algorithms (E2, E3, E4, and E6) along with CMA-ES.

Figure 12 shows the same comparison but zoomed in near the baseline. It confirms that the gap between the best LLaMEA generated algorithm and the KMeans++ baseline is substantially smaller than the aggregate fitness reported by Figure 9. The four best algorithms all converge within the 10^{-4} range of the baseline, with E6 and E2 reaching the range of a 10^{-5} difference.

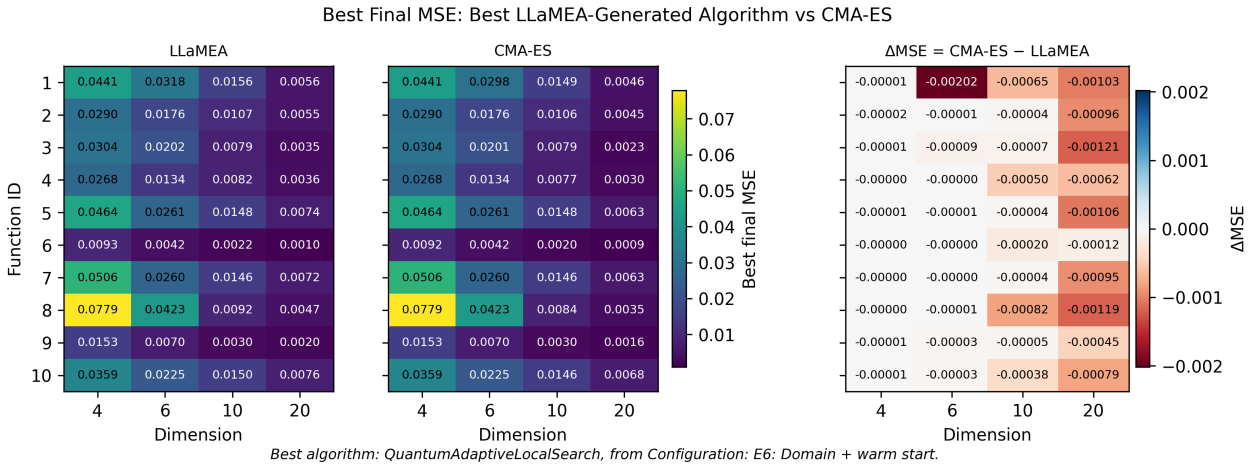


Figure 13: Best final MSE achieved by the best overall LLaMEA-generated algorithm (*QuantumAdaptiveLocalSearch*, E6: Domain + Warm start) and CMA-ES. Showing the difference between the two, across FIDs (1–10) and all dimensions.

Figure 13 provides a per-function and per-dimension breakdown comparing the best LLaMEA algorithm overall. While both E2 and E6 had the same final median MSE improvement, E6 was selected due to the quicker convergence. All ΔMSE values are negative, indicating that E6 did not improve upon the CMA-ES value in any of the benchmark functions. However, the distribution of the gap reveals important structure. At dimensions 4 and 6, the ΔMSE values are negligible for almost all functions, with several cells rounding to −0.00000, indicating near identical performance between E6 and CMA-ES at lower dimensions. There is an exception in F1 at dimension 6, which shows a notably larger gap of −0.00202, the largest single difference in the panel and a clear outlier relative to all surrounding cells.

This outlier suggests that F1 at $k = 3$ presents a landscape where CMA-ES's adaptive covariance mechanism provides a meaningful advantage that E6 does not replicate. At dimensions 10 and 20, differences become larger and more consistent across functions, with ΔMSE values in the range −0.00045 to −0.00121 at dimension 20. These differences indicate that as the size of the search space increases with dimension, CMA-ES's ability to adapt its search distribution to the landscape geometry provides an advantage over the generated candidate.

Configuration and algorithm	**Best run**	**Median**	**Spread**	**Best/Median**
E1: `ChaoticLocalSearch`	−0.002368	−0.003379	0.001709	0.70×
E2: `SpectralClusteringOptimizer`	−0.000044	−0.000066	0.000085	0.67×
E3: `AdaptiveESClustering`	−0.000585	−0.000810	0.000606	0.72×
E4: `AdaptiveLocalSearch`	−0.000788	−0.001114	0.000508	0.71×
E5: `SymmetricCMAES`	−0.003415	−0.003992	0.001282	0.86×
E6: `QuantumAdaptiveLocalSearch`	−0.000012	−0.000033	0.000088	0.35×

Table 2: Variability of the best algorithm per configuration, evaluated across 25 runs across FID and dimensions.

Table 2 provides additional context on run-level variability. E6 achieves the best single run (−0.000012) and lowest median (−0.000033) of all configurations. Its spread is also very small

(0.000088), comparable to E2 (0.000085), indicating that the $\mathcal{D}$ configurations are relatively stable across seeds.

Additionally this table provides a possible explanation for why the median fitness recorded by LLaMEA in Figure 9 was so low for these configurations. The fitnesses recorded by LLaMEA for each algorithm are provided in Appendix A.5.2. LLaMEA evaluates fitness as improvement over KMeans++ across the evaluation budget, rather than only by the final best MSE. Therefore, an algorithm that starts close to its final value, or improves only marginally during the evaluation trajectory, can receive a low fitness even if its final performance is relatively strong. The low spread observed for the best candidates of E2 and E6 may indicate that these algorithms have limited room for improvement over the budget. This could help explain why configurations with strong final MSE can still receive low LLaMEA fitness values. However, this conclusion is based only on the selected best algorithm from each configuration. A broader analysis of the algorithms generated within E2 and E6 would be required to determine if a lower spread is a general property of these configurations rather than a feature of the current selected candidate. A further possibility is that the selected best algorithms are outliers relative to the broader set of generated candidates, so these results should be interpreted as a diagnostic rather than as a definitive explanation. In Appendix A.5.1, the median and worst algorithms from the four best configurations (E2, E3, E4, E6) were plotted alongside CMA-ES and the KMeans++ baseline, providing a broader comparison.

5.4 The Impact of Domain Knowledge

5.4.1 Generalisability

The algorithm with the best overall fitness was selected and then evaluated on new instances of the training set, and the test set. Figures 14 and 15 report the median MSE achieved by configurations with ($\mathcal{D}$) and without domain knowledge ($\neg\mathcal{D}$). For the training function IDs (Figure 14), the Δ MSE values are predominantly negative. Indicating that the $\neg\mathcal{D}$ configurations (E1, E3, E4) tend to perform better. The largest differences were concentrated at dimension 4.

For the test FIDs (Figure 15), the pattern is less consistent: the sign of Δ MSE varies across (FID, k) with smaller differences overall. There is a shift here, as the majority of the $\mathcal{D}$ perform better than $\neg\mathcal{D}$. Suggesting that the identification of landscape properties may improve generalisability on unseen test instances. Section 5.4.2 will assess whether the differences between configurations are statistically reliable.

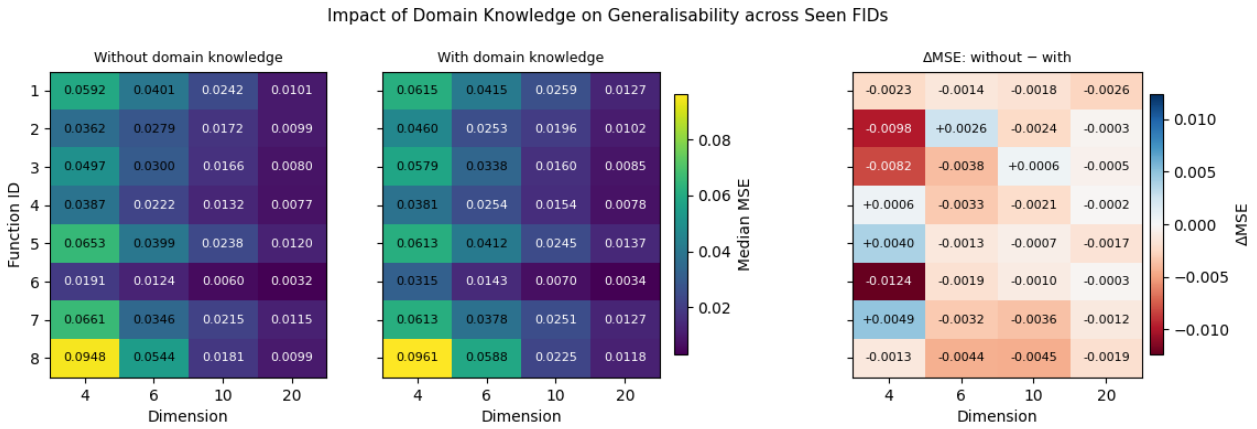


Figure 14: Median MSE achieved by the best generated algorithm. Combined runs from $\neg\mathcal{D}$ (E1, E3, E4) and $\mathcal{D}$ (E2, E5, E6), and their difference, across training FIDs (1–8) and all dimensions.

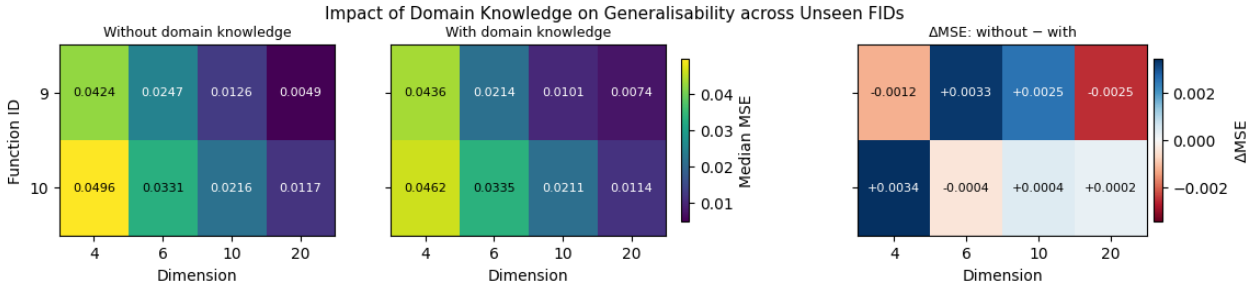


Figure 15: Median MSE achieved by the best generated algorithm. Combined runs from $\neg\mathcal{D}$ (E1, E3, E4) and $\mathcal{D}$ (E2, E5, E6), and their difference, across test FIDs (9–10) and all dimensions.

5.4.2 Significance of Domain Knowledge

The effect of domain knowledge was assessed at two levels: the prompting configuration level and the candidate algorithm level. The prompting configuration level, discussed in Section 5.2, concerns the fitness value assigned by LLaMEA during the evaluation loop. The candidate algorithm level, as seen in Section 5.3, uses the best generated algorithm per configuration and evaluates them on the benchmark (recorded as MSE improvement).

Level	n	W	p	r_{rb}
Prompting configuration	15 pairs	18.0	0.0151*	+0.70
Candidate algorithm	120 pairs	2948.0	0.0741	−0.19

Table 3: Wilcoxon signed-rank tests for the effect of domain knowledge at two levels of analysis Prompting configuration level and Candidate algorithm level. Significance levels: $^*p < 0.05$.

Across both analyses the paired differences are defined as $\Delta = f_{\text{without}} - f_{\text{with}}$, so positive values indicate that the configuration $\neg\mathcal{D}$ achieved higher fitness.

Prompting Configuration Level. The best fitness attained over 15 matched runs of $\mathcal{D}$ and $\neg\mathcal{D}$ was compared using a Wilcoxon signed-rank test. The test showed a statistically significant

difference between the matched conditions, with $W = 18.0$, $p = 0.0151$, and a rank-biserial correlation of $r_{rb} = +0.70$. The r_{rb} value shows a large effect size indicating that $\neg\mathcal{D}$ configurations consistently had higher fitness than $\mathcal{D}$ configurations.

Pair	n	**Median** Δ	**95% CI**	$\neg\mathcal{D}$ **Wins**	$\mathcal{D}$ **Wins**
E1-E2	5	+0.0025	[-0.0012, +0.0041]	3	2
E3-E5	5	+0.0012	[-0.0072, +0.0043]	4	1
E4-E6	5	+0.1598	[-0.0010, +0.1690]	4	1
Overall	15	+0.0025	[-0.0001, +0.0043]	11	4

Table 4: Prompting Configuration: Paired run-level differences in best fitness between matched configurations with domain knowledge ($\mathcal{D}$) and without domain knowledge ($\neg\mathcal{D}$). '$\neg\mathcal{D}$ Wins' records the instances of no domain knowledge being better than domain knowledge, and vice versa for '$\mathcal{D}$ Wins'. Differences are computed as $\Delta = f_{\text{without}} - f_{\text{with}}$, so positive values indicate higher best fitness without domain knowledge.

As shown in Table 4, 11 of the 15 paired runs favoured the configurations $\neg\mathcal{D}$, while 4 favoured those with $\mathcal{D}$. The overall median paired difference was small but positive, $\Delta = +0.0025$. However, this result should not be interpreted as a uniform effect across configurations.

The median difference for the E1-E2 and E3-E5 comparisons (favouring $\neg\mathcal{D}$) are smaller. The strongest contribution to the Wilcoxon result comes from the E4-E6 pair (Median $\Delta = +0.1598$).

These findings indicate that domain knowledge did not reliably improve fitness over 15 LLaMEA runs. The significant Wilcoxon result reflects a genuine difference in the warm start pairing specifically, rather than a consistent advantage of $\neg\mathcal{D}$ across all strategies. This is supported in the aggregated analysis of different configurations in Section 5.2, where $\neg\mathcal{D}$ configurations produced higher consistent median fitness.

Candidate Algorithm Level. Taking the best algorithms per configuration and evaluating them on the clustering benchmarks. A paired Wilcoxon signed-rank test was applied to the median MSE improvement across all 120 matched pairs (3 configuration $\times$ 10 FID $\times$ 4 dimensions). This test yielded $W = 2948.0$, $p = 0.0741$, $r_{\text{rb}} = -0.19$ (small effect), indicating no statistically significant effect of domain knowledge at an algorithm level. The negative sign of r_{rb} indicates a weak directional tendency favouring $\mathcal{D}$, in contrast to the prompting configuration result which significantly favoured $\neg\mathcal{D}$.

Pair	n	**Median** Δ	$\neg\mathcal{D}$ **Wins**	$\mathcal{D}$ **Wins**	**Direction**
E1-E2	40	−0.003230	0	40	Domain wins all
E3-E5	40	+0.002917	40	0	No-domain wins all
E4-E6	40	−0.000954	1	39	Domain wins mostly
Overall	120	−0.000689	41	79	Domain wins overall

Table 5: Candidate Algorithm: Paired differences in median MSE improvement. The complete reversal of direction for the exemplar-injection pair E3-E5, explaining the non-significant aggregate Wilcoxon result ($p = 0.0741$). Differences are computed as $\Delta = f_{\text{without}} - f_{\text{with}}$, so positive values indicate higher best fitness for $\neg\mathcal{D}$.

As shown in Table 5, 79 of 120 pairs favoured the $\mathcal{D}$ configuration. In this case the Wilcoxon

test is influenced by the E3-E5 comparison. The overall median paired difference was small but negative, $\Delta = -0.000689$. The E1-E2 and E4-E6 pairings strongly favoured the $\mathcal{D}$ configuration, while the E3-E5 pair showed the exact opposite trend (favouring $\neg\mathcal{D}$).

The results do not support a robust performance benefit from $\mathcal{D}$ at the prompting configuration level. Since the $\neg\mathcal{D}$ configuration significantly attained higher LLaMEA fitness. At the candidate algorithm level this is less clear. The algorithms generated by $\mathcal{D}$ configurations won 79 of 120 comparisons. However, this trend did not reach statistical significance ($p = 0.0741$) and should be treated as inconclusive instead of evidence of a $\mathcal{D}$ benefit. These results suggest that the provision of domain knowledge does not improve fitness at the prompting configuration level, but more candidate level comparisons are required before drawing a firm conclusion.

5.5 Performance of Qwen3.6:35b

As a final exploratory check, the best four experimental configurations (E1-E4) were additionally run once using `qwen3.6:35b`, a larger and more recent model than `qwen3-coder:30b`. The motivation for this test was to determine whether a larger newer model would produce improved fitness relative to the KMeans++ baseline.

As shown in Figure 16, the results did not support this assumption. Across all configurations and (FID, k) combinations, the results closely follow the patterns already observed (Section 2.1.3). There is no evidence in these results of an improvement attributable to the larger model.

Given that this test was conducted with a single run per configuration, these results should be treated as a preliminary finding. Nonetheless, the absence of any visible improvement suggests that the performance ceiling observed across configurations in this thesis is unlikely to be attributed to model size. Investigating this with a larger language model was nonetheless a useful check. It indicates that the gap to CMA-ES is more likely attributable to the prompts used or the limited budget and runs.

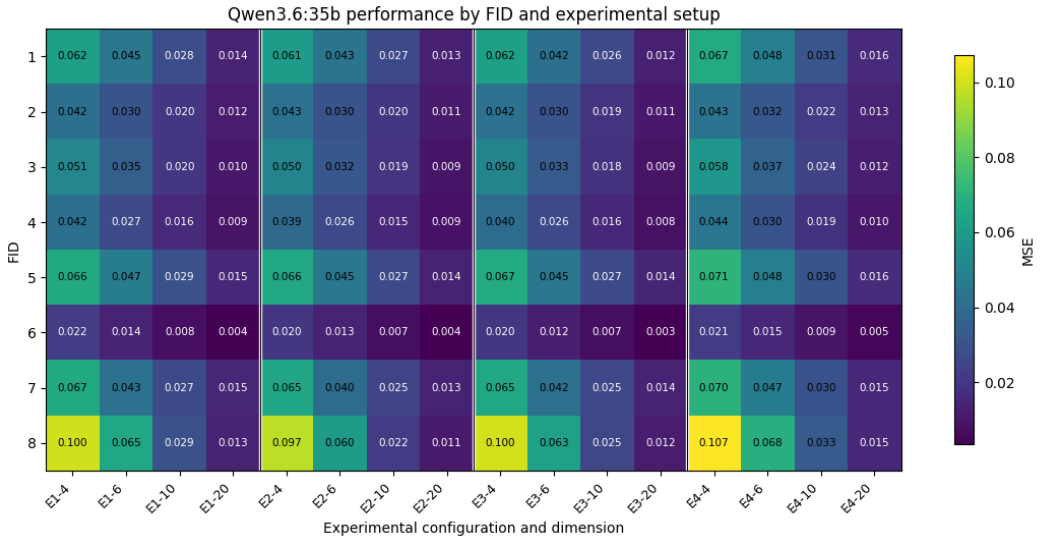


Figure 16: Recorded performance of Qwen3.6:35b across benchmark functions and experimental configurations. Rows correspond to FIDs and columns combine the experimental configuration (E1-E4).

6 Conclusions and Further Research

This thesis investigated the performance of metaheuristics generated by the LLaMEA framework on clustering problem benchmarks [31]. Six prompting configurations (E1-E6) were evaluated, varying the provision of $\mathcal{D}$, exemplar-injection and warm start. The best candidate algorithms generated by each configuration were selected and compared against the CMA-ES and KMeans++ baselines. While LLaMEA was able to generate optimisers that approach the KMeans++ baseline, none of them improved upon the baselines. The CMA-ES baseline remained the strongest method especially when the dimensionality of the problem increased.

At the prompting configuration level, none of the LLaMEA configurations reached the KMeans++ baseline. Since LLaMEA fitness was computed as improvement over KMeans++ over the LLaMEA budget, values closer to zero indicate better performance, and no configuration attained a median value of zero. Warm start without domain knowledge ($\neg\mathcal{D}$) was the strongest performing configuration, attaining the highest median fitness. This was also supported by the Cliff Delta analysis, where E4 showed the largest positive effect size. This demonstrates that seeding the initial population with example algorithms led to faster convergence than the other two strategies. However, the individual candidate level analysis diverged from this result. When the best algorithm of each configuration was evaluated, the convergence was within a 10^{-4} range of KMeans++ and CMA-ES. The best overall candidate was from the E6 configuration, achieved near identical performance to CMA-ES in low dimensionality cases. Nevertheless, CMA-ES remained better across the full benchmark, with the gap becoming more consistent at higher dimensions.

This answers RQ1 1.1 by showing that prompting strategy affects the fitness attained by LLaMEA, but that none of the generated candidates improved upon the KMeans++ or CMA-ES baselines. Warm start achieved the strongest aggregate configuration-level performance, while the best individual candidates approached the KMeans++ baseline more closely than the aggregate fitness values alone suggested.

The analysis of algorithm categories showed that the warm start and exemplar-injection successfully influenced the types of algorithms generated. Exemplar-injection sustained high diversity in the algorithm categories generated throughout the run and did not converge to a dominant type. Whereas warm start converged to a stable subset of categories by approximately evaluation 120–150. The greater diversity of exemplar-injection did not translate into higher fitness: warm start consistently achieved higher median fitness at both the configuration and candidate level. Narrowing down to the distribution per category, there were limited differences between the median fitness of each category in E3 and E4. The main difference was that the outliers in E4 were mitigated with elitism, which seems to have improved performance. The without elitism ablation (with differing offspring population size) enabled E3 to perform better. The CodeBLEU analysis revealed that structural similarity between generated algorithms was a reliable predictor of fitness similarity for most category pairs. The fitness difference decreased monotonically as the similarity of the code increased. Notably, several algorithm categories: MA, SA, and ES, emerged during evaluation without being explicitly provided as examples in the prompting strategy. LLaMEA was able to generate algorithm families beyond those provided directly.

This answers RQ2 1.1 by showing that prompting strategy, elitism, and algorithmic structure all affected the types of algorithms generated by LLaMEA. Exemplar-injection encouraged the most exploration, at a cost to overall fitness. The CodeBLEU and fitness-distribution analyses showed that differing categories could still achieve similar fitness values.

The role of domain knowledge proved to be more limited than initially expected. At the prompting configuration level, a paired Wilcoxon test found that $\neg\mathcal{D}$ configurations significantly attained higher LLaMEA fitness ($W = 18.0$, $p = 0.0151$, $r_{rb} = +0.70$), driven primarily by the warm start pairing (E4, E6). One plausible explanation is that providing explicit descriptions of clustering landscape properties constrained the LLM's exploration toward those specific properties, reducing the diversity of generated candidates without improving their quality. At the candidate-algorithm level, however, this advantage reversed: $\mathcal{D}$ algorithms won 79 of 120 paired comparisons, but this tendency was not statistically significant ($p = 0.0741$). This was partially supported by the generalisability analysis, where $\mathcal{D}$ configurations showed a more favourable pattern on the test set than on training instances. However, this distinction was neither large nor consistent enough to draw a firm conclusion. This answers RQ3 1.1 by showing that domain knowledge had a mixed effect under the current experimental design. At the prompting-configuration level and candidate-algorithm level, the performance of $\neg\mathcal{D}$ and $\mathcal{D}$ differed. Domain knowledge influences the type of algorithms LLaMEA selects without reliably improving benchmark performance; further research is needed to prove the impact on performance.

The exploratory comparison with Qwen3.6:35b indicated that the factor limiting performance (in comparison to CMA-ES or KMeans++) is unlikely to be due to the model's size. This is because substituting a larger, newer model produced no visible improvement.

Limitations. The most significant limitation of this work is the number of runs available per configuration. With five runs per configuration for the main experiments, and a single run for the Qwen3.6:35b comparison, the statistical power available to detect genuine differences between configurations was limited throughout this thesis. This is reflected in several observed effects that were too small or too inconsistent to be treated as reliable. The five run design used here should be understood as a preliminary evaluation rather than a definitive assessment of performance. Future work should prioritise increasing the number of runs per configuration.

A second limitation concerns the evaluation budget allocated to each LLaMEA run. The comparison with CMA-ES showed that LLaMEA's improvement trajectory plateaus well before CMA-ES's. A possibility is that our budget was too low for LLaMEA to fully exploit its search process. Future work should investigate whether increasing the evaluation budget allows LLaMEA to improve upon the two baselines.

A third limitation concerns the elitism ablation. The comparison between elitism and without-elitism configurations does not isolate the effect of elitism alone, because the without-elitism condition also increased the offspring pool. As a result, the observed differences reflect a combined change in selection pressure and offspring size.These factors can be tested independently by varying elitism and offspring count independently.

Future research. Beyond increasing the number of runs and the evaluation budget, the most promising direction suggested by this work would involve ablations of the prompting strategies themselves. Since the domain knowledge prompts appeared to constrain rather than productively guide the LLM's exploration, future work should isolate which specific elements of the domain knowledge prompt are responsible for this effect. Changing the current LLaMEA fitness calculation, to test how algorithms with an MSE improvement perform in comparison to evaluating the MSE improvement over time. A similar approach could be applied to the elitism and selection mechanisms. As the results showed a clear interaction between configuration and selection mechanism used, understanding which selection pressures are most compatible with each prompting strategy would be beneficial.

References

[1] E.H.L. Aarts and P.J.M. van Laarhoven. Simulated annealing: An introduction. *Statistica Neerlandica*, 43(1):31–52, March 1989.

[2] Nahum Aguirre, Erik Cuevas, Alberto Luque-Chang, Oscar Barba-Toscano, and Mario Vasquez-Franco. Covariance matrix adaptation evolution strategy (CMA-ES): A comprehensive survey of variants and hybridizations. *Arch. Comput. Methods Eng.*, April 2026.

[3] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, May 2009.

[4] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[5] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[6] Tianshi Chen, Ke Tang, Guoliang Chen, and Xin Yao. Rigorous time complexity analysis of univariate marginal distribution algorithm with margins. In *2009 IEEE Congress on Evolutionary Computation*, pages 2157–2164, 2009.

[7] Malihe Danesh and Hossein Shirgahi. A novel hybrid knowledge of firefly and pso swarm intelligence algorithms for efficient data clustering. *J. Intell. Fuzzy Syst.*, 33(6):3529–3538, December 2017.

[8] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv preprint arXiv:1810.05281*, 2018.

[9] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. A survey on code generation with llm-based agents. *ArXiv*, abs/2508.00083, 2025.

[10] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083*, 2025.

[11] Russell Eberhart and Yuhui Shi. Comparison between genetic algorithms and particle swarm optimization. volume 1447, pages 611–616, 03 1998.

[12] Marcus Gallagher. Fitness landscape analysis in data-driven optimization: An investigation of clustering problems. pages 2308–2314, 06 2019.

[13] Derek Greene, Padraig Cunningham, and Rudolf Mayer. *Unsupervised Learning and Clustering*, pages 51–90. 01 2008.

[14] Nikolaus Hansen. The cma evolution strategy: A tutorial. *ArXiv*, abs/1604.00772, 2016.

[15] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. Real-parameter black-box optimization benchmarking 2009: Experimental setup. 01 2009.

[16] Wathsala Karunarathne, Indu Bala, Dikshit Chauhan, Matthew Roughan, and Lewis Mitchell. Modified cma-es algorithm for multi-modal optimization: Incorporating niching strategies and dynamic adaptation mechanism, 06 2024.

[17] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[18] Pascal Kerschke and Heike Trautmann. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, 27(1):99–127, 2019.

[19] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the International Conference on Machine Learning*, pages 32201–32223, 2024.

[20] Saman M. Almufti, Awaz Ahmad Shaban, Zeravan Arif Ali, Rasan Ismael Ali, Jayson A. Dela Fuente, and Renas Rajab Asaad. Overview of metaheuristic algorithms. *Polaris Global Journal of Scholarly Research and Trends*, 2(2):10–32, Apr. 2023.

[21] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 829–836, New York, NY, USA, 2011. Association for Computing Machinery.

[22] Phan Trung Hai Nguyen and Dirk Sudholt. Memetic algorithms beat evolutionary algorithms on the class of hurdle problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, page 1071–1078. ACM, 2018.

[23] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis, 2020.

[24] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[25] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. 03 2017.

[26] Krutika Sarode and Shashidhar Reddy Javaji. Hybrid genetic algorithm and hill climbing optimization for the neural network, 2023.

[27] ScienceDirect Topics. Metaheuristic optimization algorithm, n.d. Online topic overview.

[28] Nuwan I. Senaratna. Genetic algorithms: The crossover-mutation debate, 2005. Literature survey indexed by Semantic Scholar.

[29] Rainer Storn and Kenneth V. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[30] Niki van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics, 2025.

[31] Diederick Vermetten, Catalin-Viorel Dinu, and Marcus Gallagher. A standardized benchmark set of clustering problem instances for comparing black-box optimizers. In *Proceedings of the 18th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, FOGA '25, page 297–307, New York, NY, USA, 2025. Association for Computing Machinery.

A Appendix

A.1 Supplementary Algorithm Category Distributions

Exemplar-injection Prompt with Elitism	
Algorithm Category	Number of valid algorithms
MA	28
ES	311
PSO	161
DE	496
SA	4

Table 6: Algorithm categories for exemplar-injection prompt with elitism.

Exemplar-injection Prompt without Elitism	
Algorithm Category	Number of valid algorithms
MA	301
ES	29
PSO	143
DE	314

Table 7: Algorithm categories for exemplar-injection prompt without elitism.

Warm start with Elitism	
Algorithm Category	Number of valid algorithms
MA	394
ES	367
PSO	170
DE	69

Table 8: Algorithm categories for warm start with elitism.

Warm start without Elitism	
Algorithm Category	Number of valid algorithms
MA	407
ES	28
PSO	134
DE	411
SA	20

Table 9: Algorithm categories for warm start without elitism.

A.2 Supplementary Fitness Scaling Analysis

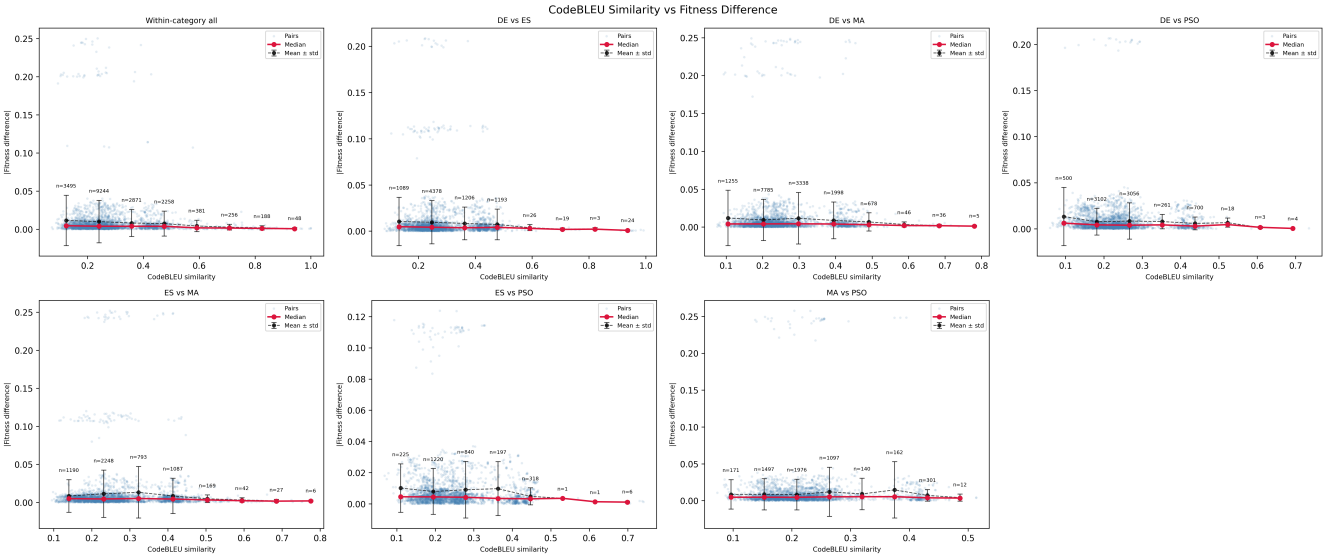
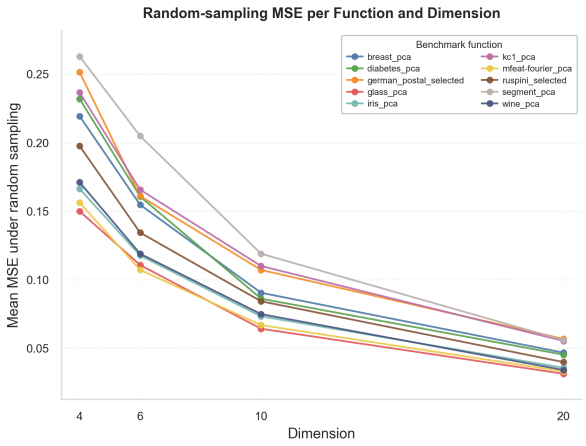


Figure 17: Unscaled log-fitness distribution used to inspect the effect of scaling on the observed performance differences between generated algorithms.

A.3 KMeans++ Baselines and Random Walk exploration of landscape

	FID	Dataset	N	mse k - 2	mse k - 3	mse k - 5	mse k - 10
0	1	breast_pca	106	0.04407	0.02982	0.01490	0.00457
1	2	diabetes_pca	500	0.02903	0.01761	0.01063	0.00451
2	3	german_postal_selected	89	0.03036	0.02014	0.00786	0.00220
3	4	glass_pca	214	0.02685	0.01337	0.00770	0.00275
4	5	iris_pca	150	0.04636	0.02610	0.01480	0.00631
5	6	kc1_pca	500	0.00925	0.00421	0.00204	0.00074
6	7	mfeat-fourier_pca	500	0.05061	0.02597	0.01458	0.00627
7	8	ruspini_selected	75	0.07786	0.04233	0.00841	0.00344
8	9	segment_pca	500	0.01529	0.00695	0.00296	0.00133
9	10	wine_pca	178	0.03590	0.02249	0.01464	0.00682

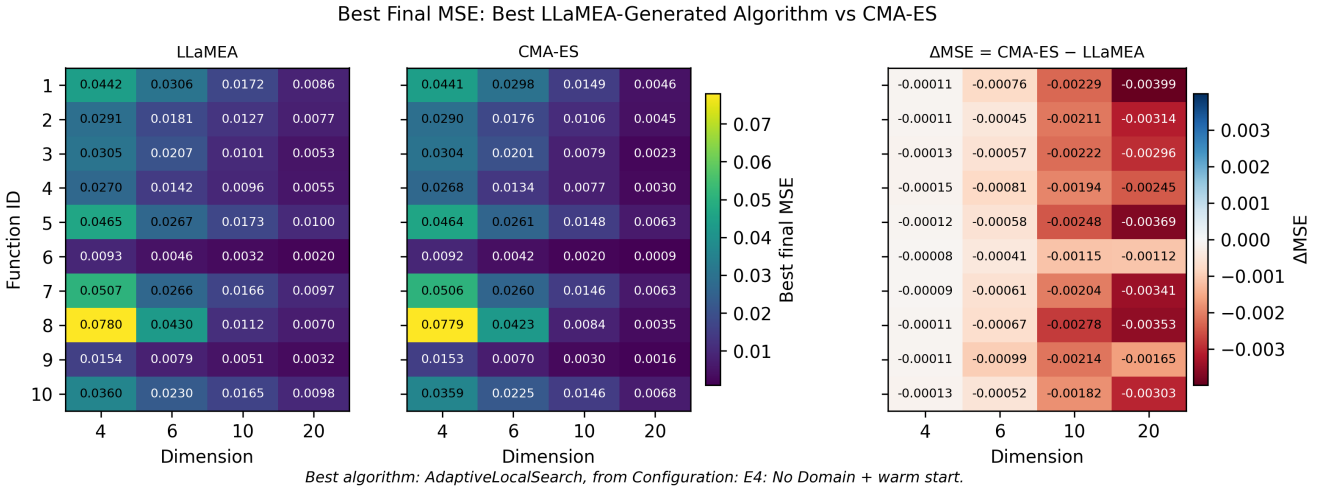
(a) Best MSE value from the KMeans++ Baseline that was used in the clustering benchmarks [31] across FIDs and dimensions



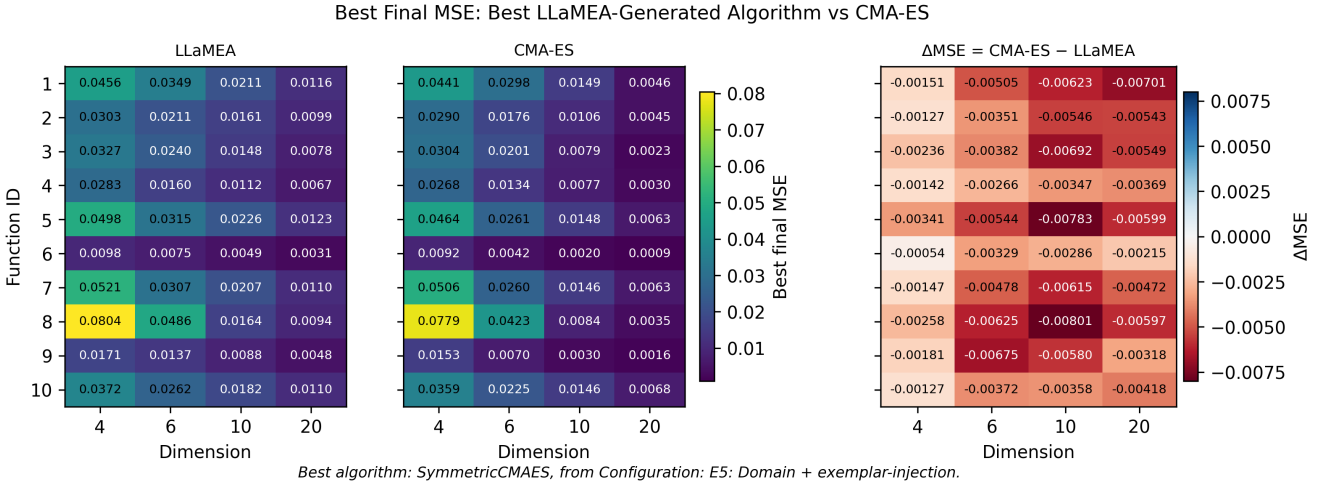
(b) Random-walk landscape analysis over FIDs and dimensions. As a simple ELA inspired exploratory technique, the random walk provides a qualitative indication of landscape characteristics.

Figure 18: Benchmark information used to contextualise the performance of the LLaMEA-generated clustering optimisers.

A.4 Supplementary heatmaps comparing candidate to CMA-ES



(a) Best final MSE achieved by the LLaMEA-generated algorithm with middle final performance (AdaptiveLocalSearch, E4: Domain + Warm start) and CMA-ES. Showing the difference between the two, across FIDs (1–10) and all dimensions.

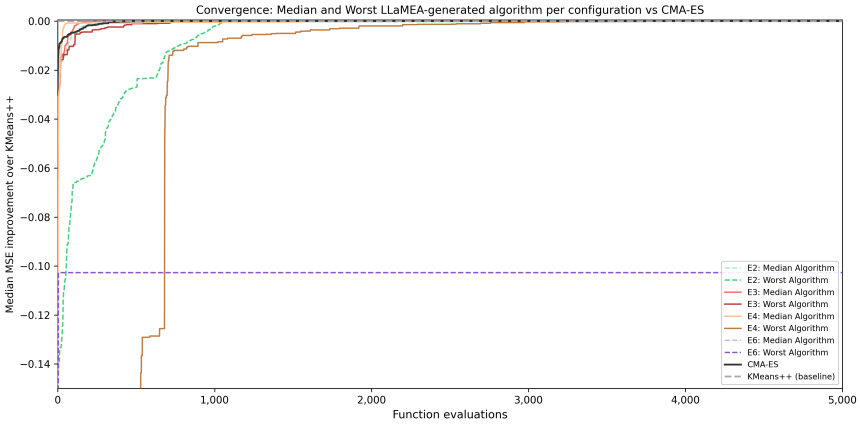


(b) Best final MSE achieved by the LLaMEA-generated algorithm with worst final performance (SymmetricCMAES, E5: Domain + Exemplar-injection) and CMA-ES. Showing the difference between the two, across FIDs (1–10) and all dimensions.

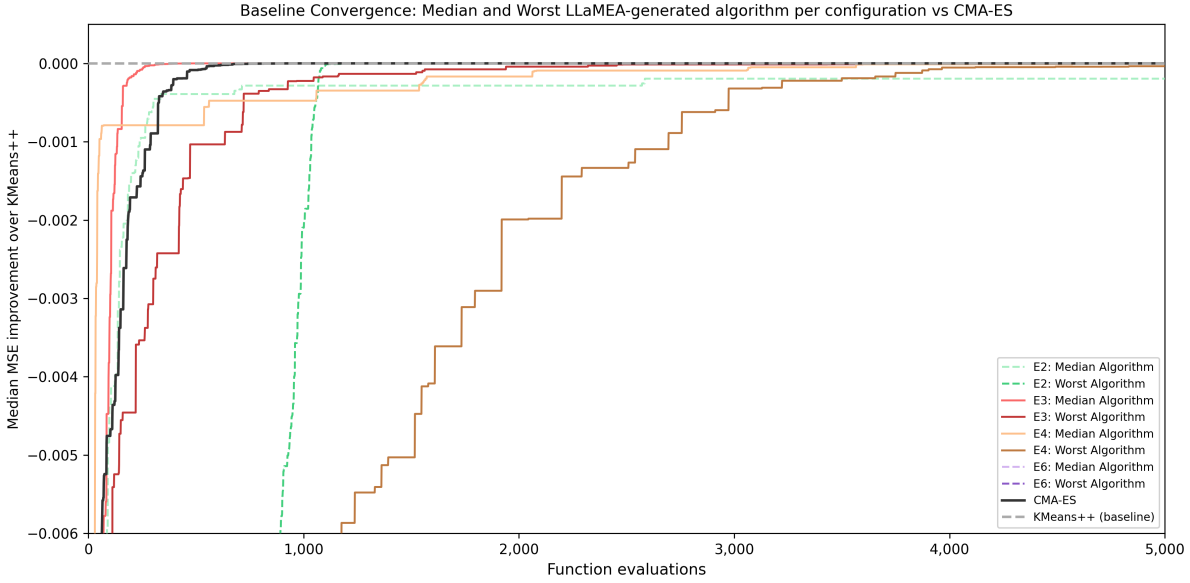
Figure 19: Comparison between CMA-ES and selected the best-fitness LLaMEA algorithms with configurations of intermediate (E4) and worst (E5) final performance measured as MSE. These heatmaps illustrate how the performance gap between CMA-ES and LLaMEA changes across configurations and across benchmark functions and dimensions.

A.5 Supplementary Analysis of Best, Median, and Worst Candidates

A.5.1 Convergence of median and worst candidates



(a) Median and worst algorithm (based on fitness) per configuration (E2, E3, E4, E6) compared against CMA-ES and the KMeans++ baseline over 5000 function evaluations. Only the four closest configurations to the baseline are shown.



(b) Zoomed view near the KMeans++ baseline, isolating the median algorithms of E2, E3, E4, and E6 alongside CMA-ES. The worst and median algorithms of E6 are excluded from the zoomed view.

Figure 20: Median and worst algorithm convergence per configuration, included to assess whether the strong best-algorithm performance of domain knowledge configurations (E2, E6) observed in Figure 11 reflects a consistent property of those configurations. The large spread between the median and worst algorithms for E2 and E6 seem to indicate the latter. Both $\mathcal{D}$ configurations have medians that perform worse than the $\neg\mathcal{D}$ configurations. The only exception being 'E2:Median Algorithm' that performs equally, but has a rather slow convergence rate initially.

A.5.2 Table of Fitness-selected candidates

Config	Fitness Position	Algorithm	Fitness
E1 ($\neg\mathcal{D}$)	**Best**	`ChaoticLocalSearch`	**-0.004464**
E2 ($\mathcal{D}$)	Worst	`DiversifyingLocalSearch`	-0.055183
	Median	`SymmetryAwareHybridClustering`	-0.012321
	Best	`SpectralClusteringOptimizer`	**-0.004379**
E3 ($\neg\mathcal{D}$)	Worst	`ES_DC_LS`	-0.042677
	Median	`AdaptiveMultiSwarmClustering`	-0.010581
	Best	`AdaptiveESClustering`	**-0.004475**
E4 ($\neg\mathcal{D}$)	Worst	`QuantumClusteringOptimizer`	-0.256416
	Median	`AdaptiveClusteringWithProgressiveExpansion`	-0.009235
	Best	`AdaptiveLocalSearch`	**-0.002583**
E5 ($\mathcal{D}$)	**Best**	`SymmetricCMAES`	**-0.002939**
E6 ($\mathcal{D}$)	Worst	`AdaptiveHybridUMDA`	-0.204114
	Median	`UMDAWithAdaptiveConvergenceMechanism`	-0.177526
	Best	`QuantumAdaptiveLocalSearch`	**-0.005679**

Table 10: Supplementary fitness-selected candidates used to contextualise the convergence behaviour of the generated algorithms. The best candidates, highlighted in bold, correspond to the algorithms analysed in Section 5.3. The middle and worst candidates are reported for E2, E3, E4, and E6 because these are the four configurations shown in the supplementary convergence analysis in Figure 20.

A.6 Experimental Configuration Prompts

A.6.1 E1: No Domain Knowledge

```
self.task_prompt = "You are designing a black-box optimization algorithm for an
    IOH clustering problem (minimizing MSE).

    INTERFACE:
    - Implement a class with __init__(self, budget, dim=None, seed=None) and
   __call__(self, func)
    - func(x) evaluates a 1D NumPy float64 array and returns a Python float MSE
    (lower is better)
    - Infer dim from len(func.bounds.lb). Enforce bounds with np.clip(x, func.
   bounds.lb, func.bounds.ub)
    - Do NOT instantiate the problem yourself. Never call func(x) after budget
   is exhausted
    - Use // or int(...) for any integer. Before reshape, verify compatibility
   with x.size

    return (f_opt, x_opt)"
```

Listing 1: Task prompt provided in the IOHClusteringAdapter. E1 the without domain knowledge configuration.

A.6.2 E2: Domain Knowledge

```
self.task_prompt = "You are designing a black-box optimization algorithm for an
    IOH clustering problem (minimizing MSE).

    INTERFACE:
    - Implement a class with __init__(self, budget, dim=None, seed=None) and
   __call__(self, func)
    - func(x) evaluates a 1D NumPy float64 array and returns a Python float MSE
    (lower is better)
    - Infer dim from len(func.bounds.lb). Enforce bounds with np.clip(x, func.
   bounds.lb, func.bounds.ub)
    - Do NOT instantiate the problem yourself. Never call func(x) after budget
   is exhausted
    - Use // or int(...) for any integer. Before reshape, verify compatibility
   with x.size

    PROBLEM CHARACTERISTICS:
    - The search space is a flat vector of concatenated cluster centroids
    - The landscape contains many equivalent optima due to centroid permutation
    symmetry
    - Large neutral plateaus make naive local search ineffective escaping them
   requires active diversity or perturbation
    - Good solutions tend to cluster in narrow, curved valleys that reward both
    exploration and precision

    return (f_opt, x_opt)"
```

Listing 2: Task prompt provided in the IOHClusteringAdapter. E2 the With Domain Knowledge Configuration.

A.6.3 E3 and E5: Exemplar-injection Prompt Configurations

```
self.task_prompt = "You are designing an optimization algorithm for an IOH
   clustering problem (minimizing MSE).

    INTERFACE:
    - __call__(self, func): optimize by calling func(x) to evaluate solutions.
    - func(x) returns MSE as a Python float (lower is better).
    - Do NOT create your own problem instance or use a proxy objective.
    - Infer dim from len(func.bounds.lb) inside __call__ rather than __init__.
    - Accept dim as an optional __init__ argument for compatibility but do not
   require it.

    PROBLEM STRUCTURE:
    - x is a flattened array of k centroids. Permutation symmetry means
   relabelling clusters gives identical MSE. Small perturbations frequently
   produce no MSE change (neutral regions).

    ALGORITHM DESIGN:
    - Draw from population-based (EA, PSO, DE), distribution-based (EDA, UMDA),
    or hybrid approaches.
```

```
    - Include at least 3 of: adaptive parameters, restart/multistart, archive/elitism, diversity preservation, budget-phase scheduling, symmetry-aware or neutral-escape operators."

self.mutation_prompts = [Example algorithms provided consist of - RandomSearch, NeighborhoodAdaptiveDE, ParticleSwarmOptimiser, UMDA, GreedyHillClimber]
```

Listing 3: Task prompt and exemplar-injection prompt examples used for the exemplar-injection prompt configurations. E3 excludes domain knowledge, whereas E5 includes domain knowledge.

A.6.4 E4 and E6: Warm-Start Configurations

```

example_prompts = [```The same examples are initialised in self.mutation_prompt in E3, the only difference is that GreedyHillClimber was excluded from our set of n_parents = 4```]


method = LLaMEA(
    llm,
    budget=100,
    name="LLaMEA",
    n_parents=4,
    example_prompts = example_prompts,
    n_offspring=8,
    elitism=True,
    minimization=False,
)

problem = IOHClusteringProblemAdapter(
    training_instances=train_instances,
    test_instances=test_instances,
    name="CLUSTERING",
    base_budget=25,
    budget_scale_with_dim=True,
    eval_timeout=120,
    full_ioh_log=False,
    ioh_dir="",  # or "ioh_logs"
)

experiment = Experiment(
    methods=method,
    problems=problem,
    runs=1,
    show_stdout=True,      # note: LLaMEA redirects stdout internally; your debug_eval.log is the truth
    exp_logger=logger,
    budget=140,            # IMPORTANT: must be >= LLaMEA_method.budget, ideally a bit higher
)
```

Listing 4: LLaMEA setup used for the warm-start configurations. E4 excludes domain knowledge, whereas E6 includes domain knowledge.

```
self.task_prompt = "You are designing a black-box optimization algorithm for an
    IOH clustering problem (minimizing MSE).

    INTERFACE:
    - Implement a class with __init__(self, budget, dim=None, seed=None) and
   __call__(self, func)
    - func(x) evaluates a 1D NumPy float64 array and returns a Python float MSE
    (lower is better)
    - Infer dim from len(func.bounds.lb). Enforce bounds with np.clip(x, func.
   bounds.lb, func.bounds.ub)
    - Do NOT instantiate the problem yourself. Never call func(x) after budget
   is exhausted
    - Use // or int(...) for any integer. Before reshape, verify compatibility
   with x.size

    PROBLEM CHARACTERISTICS:
    - The search space is a flat vector of concatenated cluster centroids
    - The landscape contains many equivalent optima due to centroid permutation
    symmetry
    - Large neutral plateaus make naive local search ineffective escaping them
   requires active diversity or perturbation
    - Good solutions tend to cluster in narrow, curved valleys that reward both
    exploration and precision

    return (f_opt, x_opt)"
```

Listing 5: Task prompt provided in the IOHClusteringAdapter. 4 excludes domain knowledge