Universiteit Leiden

Beyond Scalar Fitness: Behavioural Feedback for LLM-Driven Metaheuristic Evolution

Max Harell (s3815129)

Supervisors:
Dr. Niki van Stein & Dr. Thomas Bäck

Bachelor Thesis– Data Science and Artificial Intelligence

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

June 30, 2026

0 Abstract

The LLaMEA framework embeds a large language model in an evolutionary loop to generate optimisation algorithms, with the Area Over the Convergence Curve (AOCC), a single scalar value, as its only feedback signal. Recent work has shown that augmenting feedback with structural features extracted from the algorithm's source code improves performance, but the analogous question for *behavioural* features (descriptors of the search trajectory the algorithm produces when it runs) has remained open. This thesis addresses that question through a four-stage pipeline. Stage one screens ten LLMs and identifies Gemini 3 Flash as the strongest combination of quality, reliability, and cost. Stage two reduces a 32-feature behavioural vocabulary to a tractable subset of ten via tri-criterion consensus. Stage three compares neutral, directional, and comparative framings of each behavioural feature on Gemini 3 Flash across 29 conditions. Stage four is a head-to-head full-benchmark comparison of behavioural, structural (LLaMEA-SAGE), and combined feedback against vanilla AOCC-only on a 20-function MA-BBOB set with 10 seeds per condition. We find that neutral framing of behavioural features, reporting the feature value without prescriptive advice, consistently outperforms the prescriptive variants, and that prescriptive feedback steers the median feature value in the advised direction in only 37% of cases despite empirically grounded advice. On the stage four benchmark, all three feedback-augmented conditions outperform vanilla LLaMEA with the decisive effect being variance reduction. Combined behavioural and structural feedback shrinks seed-to-seed standard deviation by roughly a factor of five (Levene $p = 0.030$, the only Stage 4 contrast to clear $\alpha = 0.05$), and the three feedback conditions split 19 of 20 leaderboard wins between them. Re-evaluated on the full 1,000-instance MA-BBOB pool, all four LLM-generated winning algorithms outperform a CMA-ES baseline across $d \in \{5, 10, 20\}$. Trajectory-based behavioural feedback does guide LLMs to design better optimisation algorithms, but the improvement manifests primarily as reduced failure-mode variance rather than as a large mean performance lift.

1 Introduction

Designing effective optimisation algorithms is hard. Decades of research have produced hundreds of metaheuristic strategies, each with its own strengths, parameters, and failure modes [1], and choosing or tuning among them for a given problem is itself a difficult task. Automated algorithm design has tackled this meta-problem through a spectrum of paradigms. Algorithm selection [2] picks the best candidate from a fixed portfolio, configuration [3] tunes parameters within a predefined space, modular frameworks [4] compose algorithms from interchangeable components, and programming by optimisation [5] exposes ever more design decisions to automated search. Across all of them, however, the set of reachable algorithms is fixed in advance by human designers. Automation enables navigation of a design space, but does not extend it. Large Language Models (LLMs) have introduced a qualitatively different approach. Rather than searching within such a fixed space, LLMs can generate complete optimisation algorithms as executable code by drawing on the algorithmic knowledge encoded in their training data. The LLaMEA framework [6] operationalises this by embedding an LLM within an Evolutionary Strategy (ES) that generates candidate algorithms, evaluates them on benchmark problems, and uses the performance feedback to guide subsequent generations. On standard benchmarks, LLaMEA-generated algorithms rival or surpass established metaheuristics such as the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [7].

In the standard LLaMEA loop, the sole feedback signal to the LLM is the Area Over the Convergence Curve (AOCC) [6], a scalar performance score. While AOCC captures anytime performance, it compresses the entire optimisation trajectory, spanning thousands of evaluations across multiple problem instances, into a single number. Two algorithms with identical AOCC scores may exhibit fundamentally different search behaviours. One might converge quickly through aggressive exploitation, while another might explore broadly before settling on a good solution. Richer descriptions of how algorithms search are therefore worth considering as feedback signals.

Research on metaheuristic optimisation distinguishes three kinds of features that can characterise a candidate algorithm and its problem. Landscape features describe the problem itself, computed from samples of the objective function [8, 9]. Structural features describe the algorithm's code, computed from the generated source [10]. Behavioural features describe how the algorithm and problem interact, computed from the trajectory of sampled points and fitness values that each evaluation produces [11, 12].

Of the three, only structural features have so far been used as in-loop feedback to guide LLM-driven algorithm design. LLaMEA-SAGE [10] showed that feeding structural descriptors back to the LLM as natural-language instructions improves the quality of generated algorithms. To our knowledge, landscape and behavioural features have not been tested in this role. Behavioural features in particular have been used only post-hoc. Prior work [12] has shown they can explain why certain LLM-generated algorithms outperform others, but that information was not fed back into the generation loop.

This thesis examines the usefulness of behavioural features as a feedback signal. The explanatory power observed post-hoc suggests that if behavioural features can describe quality differences between algorithms, they may also be able to guide the LLM toward better designs. But the steering task is harder than for structural features. Structural features are proximal. They are computed directly from the code, so the LLM's edits directly modify the feature values. Behavioural features are distal. They are not properties of the generated code but of the trajectory that unfolds when the code runs, emerging over thousands of evaluations on multiple problem instances. Whether such indirect signals can guide the

LLM the way proximal features have is an open question.

The following research question and sub-questions guide the exploration of this thesis,

RQ. *Can trajectory-based behavioural feedback guide LLMs to design better optimisation algorithms?*

SubQ1. *Which LLMs can reliably generate high-quality metaheuristics for continuous black-box optimisation?*

SubQ2. *Which trajectory-based behavioural features are most informative about algorithm quality?*

SubQ3. *Does the framing of behavioural feedback (neutral observation, directional guidance, or comparative anchoring) affect the performance of LLM-generated algorithms?*

This thesis makes four contributions to the literature on LLM-driven automated algorithm design. The first is an extended behavioural feature set that combines the 11 metrics previously introduced for LLM-generated algorithm analysis [12] with 21 additional features. These span basic trajectory statistics drawn from random-walk and movement-analysis traditions [13], an adaptation of population-dynamics features from DynamoRep [11], information-theoretic quantities that treat fitness as a longitudinal signal [14, 15, 16, 17], and metrics new to the optimisation literature [18]. The second is a multi-model screening of 10 LLMs, spanning 3B to 32B parameters across six model families plus two cloud API models, that identifies viable candidates for automated metaheuristic design and finds that medium-sized open-weight models (24–32B) can produce viable metaheuristics although they trail closed API models. The third is evidence that behavioural feedback functions best as informative context rather than prescriptive guidance. Across 29 experimental conditions comparing neutral, directional, and comparative formats, neutral reporting (simply stating the behavioural observation without interpretation) consistently outperforms prescriptive formats that tell the LLM which direction is better, and prescriptive feedback steers the median feature value in the advised direction in only 37% of cases despite directional advice being empirically grounded in the Stage 1 data. The fourth is a full-benchmark head-to-head comparison of behavioural, structural [10], and combined feedback against the vanilla AOCC-only baseline. All three feedback-augmented conditions raise the floor of LLaMEA performance rather than its ceiling, with combined feedback shrinking seed-to-seed standard deviation by roughly a factor of five (the only Stage 4 contrast to clear $\alpha = 0.05$). All four winning LLM-generated algorithms also outperform both a generic CMA-ES baseline and the 2025 MA-BBOB competition winner across $d \in \{5, 10, 20\}$ on the full 1,000-instance MA-BBOB pool.

2 Background and Related Work

2.1 Metaheuristic Optimisation

A metaheuristic is a high-level, problem-independent strategy for solving optimisation problems of the form

$$\min \mathcal{F}(x), \qquad x \in \mathcal{S} \subseteq \mathbb{R}^d,$$

where x is a d-dimensional candidate solution, $\mathcal{S}$ is the bounded feasible search space, and $\mathcal{F} : \mathcal{S} \to \mathbb{R}$ is a black-box objective function that can be evaluated but whose gradient or structure is unknown [19]. Metaheuristics work by iteratively generating, evaluating, and selecting candidate solutions according to heuristic rules that govern how the search space is explored. Unlike exact methods, they make no guarantees of optimality but can find good solutions efficiently across a wide range of problem types.

The field encompasses a broad family of algorithms. ES maintain a population of candidate solutions and apply Gaussian mutation with self-adaptive step sizes [20]. CMA-ES extends this by learning a full covariance matrix that captures the local landscape geometry, and is widely considered the state of the art for continuous black-box optimisation [7]. Differential Evolution (DE) generates trial solutions by combining vector differences sampled from the current population [21].

Despite their success, the design of metaheuristics remains largely a manual endeavour. A vast number of methods have been proposed, many with overlapping mechanisms and limited systematic benchmarking [22]. Choosing the right algorithm for a given problem, configuring its parameters, or designing a new variant requires deep expertise and extensive experimentation. This motivates the pursuit of automated, and increasingly explainable, approaches to algorithm design [23].

2.2 Automated Algorithm Design

The automated design of optimisation algorithms has evolved through several paradigms, each expanding the scope of what can be automated. The earliest framing is algorithm selection. Rice [2] formalises the task as learning a mapping from problem features to the best-performing algorithm, drawn from a fixed portfolio of candidates evaluated on a set of problem instances. Meta-learning approaches [24] build on this framework by using landscape features to predict algorithm performance.

Rather than selecting among fixed algorithms, algorithm configuration optimises the hyperparameters of a single algorithm. ParamILS [3] applies iterated local search in parameter space, treating algorithm performance on a benchmark as a black-box function of its parameters. These methods can find configurations that substantially outperform defaults, but they operate within a fixed algorithmic structure.

Modular frameworks bridge selection and configuration by decomposing algorithms into interchangeable components. Van Rijn et al. [4] evolve the structure of evolution strategies by treating algorithmic modules (selection, recombination, mutation operators) as parameters of a meta-optimisation problem, and Vermetten et al. [25] apply the same principle to DE. Such frameworks

can search over millions of algorithm configurations, but they remain limited to the modules their designers included. The design space is large but bounded.

A broader methodology is proposed by Hoos [5] under the banner of programming by optimisation. Rather than designing a single algorithm, developers should expose all design decisions as parameters and let automated methods search the resulting space. Genetic Improvement [26] takes this further by applying evolutionary operators directly to source code, modifying and recombining program fragments to improve non-functional properties. These ideas foreshadow the shift to code-generation based approaches, where algorithms are constructed from scratch rather than configured from templates.

2.3 LLMs for Algorithm Design

The emergence of LLMs capable of generating complex code has introduced a qualitatively different approach to algorithm design. Rather than searching within a predefined design space, LLMs can generate complete algorithms as executable programs, unconstrained by modular templates. Building on existing taxonomies of LLM–EC integration [6, 27], we group the relevant literature by the role the LLM plays in the evolutionary loop.

Prompt optimisation applies evolutionary algorithms to optimise the prompts given to LLMs. EvoPrompt [28] uses genetic algorithms to evolve prompts, outperforming human-engineered alternatives. The Automated Prompt Engineer (APE) [29] takes a similar approach using Monte Carlo search. These methods optimise how we communicate with LLMs but do not use LLMs to generate algorithms directly.

Code generation embeds the LLM itself inside the evolutionary loop. FunSearch [30] demonstrated that LLMs embedded in this way can discover novel mathematical results, generating Python programs for the cap-set problem that surpassed known bounds through a distributed island model with millions of LLM calls. Evolution of Heuristics (EoH) [31] applies a similar principle to combinatorial optimisation, treating candidate heuristics as individuals in an evolutionary population. Algorithm Evolution using Large Language Model (AEL) [32] is a related approach from the same group. The Self-Taught Optimiser (STOP) [33] demonstrates recursive self-improvement where the LLM-generated code is itself a prompt optimiser, creating a meta-learning loop.

Complete metaheuristic generation, the setting this thesis operates in, extends code generation from heuristic components to full algorithms. LLaMEA [6] generates full Python classes with initialisation, state management, and search logic for continuous black-box problems, unlike EoH or AEL which generate small heuristic components for combinatorial problems. The framework combines in-context learning (the LLM sees previous algorithms and their scores) with an evolutionary loop that provides selection pressure. A follow-up work [12] extends LLaMEA by logging and analysing the behavioural profiles of generated algorithms, showing that higher-performing algorithms consistently exhibit more intensive exploitation behaviour and faster convergence across six LLaMEA configurations. However, this behavioural information is used only for post-hoc analysis, not as feedback to the LLM.

LLaMEA-SAGE [10] goes a step further by feeding structural information about the generated code back into the prompt. SAGE extracts features from the Abstract Syntax Tree (AST) of each algorithm (graph-theoretic metrics, code complexity measures, and structural statistics) stores them in an archive alongside fitness scores, and uses SHAP analysis [34] on a surrogate model to identify which features most influence performance. The resulting guidance is translated into natural-language instructions (e.g., "try decreasing the cyclomatic complexity") that augment the mutation prompt, yielding faster convergence on MA-BBOB than vanilla LLaMEA.

Several recent approaches offer alternative search strategies over the same design space. MCTS-AHD [35] replaces the evolutionary loop with Monte Carlo Tree Search, organising generated heuristics in a tree structure that balances exploration and exploitation. LLM-driven Heuristic Neighbourhood Search (LHNS) [36] abandons population-based evolution in favour of single-solution neighbourhood search with ruin-and-recreate operators. Both achieve competitive results on combinatorial benchmarks, demonstrating that the search strategy over LLM-generated code is itself an active research question.

2.4 Benchmarking and Evaluation

Rigorous benchmarking is essential for comparing generated algorithms and providing meaningful feedback to the LLM. The Black-Box Optimisation Benchmark (BBOB) [37] defines 24 noiseless test functions spanning five function groups (separable, moderate conditioning, high conditioning, multimodal with adequate structure, multimodal with weak structure). Many-Affine BBOB (MA-BBOB) [38] extends this by constructing novel instances through affine combinations of BBOB functions, providing much greater instance diversity while preserving known landscape properties. MA-BBOB is the benchmark used in recent LLaMEA versions [10] and in this thesis.

The evaluation infrastructure underneath these benchmarks is IOHexperimenter [39], which runs algorithms on benchmark functions, enforces budget limits, and logs full trajectories;.lyzer [40] provides statistical analysis and visualisation of the logged data. BLADE [41] builds on IOHexperimenter to provide a standardised evaluation framework specifically for LLM-driven algorithm design, including prompt management, candidate logging, and behavioural metric computation.

The primary performance signal used in this thesis is AOCC [6], an anytime performance metric that captures not just the final solution quality but the entire convergence trajectory. Unlike fixed-target metrics that measure time-to-threshold, AOCC rewards algorithms that converge quickly *and* reach good final solutions. It is the standard performance metric in the BLADE

framework (see Equation (38) for the formal definition).

2.5 Trajectory-Based Analysis of Metaheuristics

A growing body of work analyses algorithm behaviour through the lens of its search trajectory. Search Trajectory Networks (STNs) [42] discretise the search space and represent trajectories as graphs, enabling visual comparison of algorithm dynamics. DynamoRep [11] uses trajectory-based population dynamics, specifically per-generation statistics of objective and position distributions, to classify optimisation problems. Kostovska et al. [43] combine landscape features with algorithm-internal state variables (step size, covariance matrix) computed along trajectories for warm-started algorithm selection. Cenikj et al. [44] survey the broader meta-feature landscape spanning problem, trajectory, and algorithm-internal signals. Across this body of work, however, trajectory-based features have been used for algorithm selection and problem classification, not as feedback signals within an algorithm design loop.

3 Behavioural Feature Design

This chapter defines the behavioural feature set used throughout the thesis. Section 3.1 states the criteria that guided feature design and selection. The remaining sections present the 32 resulting features, organised by origin and function: inherited metrics, step-size and movement dynamics, information-theoretic measures, adapted population dynamics, and features new to the optimisation literature.

Every feature is computed from a single trajectory, the ordered sequence of candidate solutions that an algorithm evaluates together with their fitness values. Formally, a trajectory is a sequence $\{(x_t, y_t)\}_{t=1}^{T}$ where $x_t \in \mathbb{R}^d$ is the t-th candidate solution with decision variables $x_t^{(0)}, \ldots, x_t^{(d-1)}$, and $y_t = \mathcal{F}(x_t)$ is its fitness. The best-so-far fitness at evaluation t is denoted $f_t^* = \min_{s \le t} y_s$. Throughout this thesis, trajectories are collected over $T = 2{,}000 \times d$ evaluations in $d = 5$ dimensions with bounds $[-5, 5]$, giving $T = 10{,}000$ evaluations per trajectory.

3.1 Design Principles

Four criteria guided the selection and design of behavioural features.

1. **Computational cheapness.** MA-BBOB evaluations already carry the bulk of experiment runtime. Behavioural features should add only a small fraction of additional expense.
2. **Distinctness.** Each feature should capture a dimension of behaviour not already covered by other features.
3. **Interpretability for LLM feedback.** Features are reported to the language model as part of the behavioural feedback string. They must therefore carry clear, concise semantics that an LLM can act on.
4. **Discrimination.** Features should separate high-performing from low-performing algorithms. We quantify this via Spearman correlation [45] with AOCC and Kolmogorov–Smirnov effect size [46] between the top-25% and bottom-25% candidates.

3.2 Existing BLADE Metrics

BLADE [41] ships with eleven behavioural metrics introduced by van Stein et al. [12]. They fall into four functional categories covering exploration and diversity, exploitation and intensification, convergence, and stagnation. We define each metric formally below, using the trajectory notation from the start of the chapter.

`avg_nearest_neighbor_distance`. For each evaluation x_k (sampled every tenth step for efficiency), the Euclidean distance to its nearest neighbour among the $H = 1{,}000$ most recent prior evaluations:

$$\mathrm{ANN} = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \min_{\max(0,\, k-H) \le j < k} \|x_k - x_j\|, \qquad (1)$$

where $\mathcal{K} = \{1, 11, 21, \ldots\}$. Large values indicate the search covers new territory at each step. Small values indicate repeated returns to previously visited regions.

`dispersion`. The radius of the largest empty hypersphere that fits inside $\mathcal{S}$ without containing any evaluated point:

$$\mathrm{disp}(X) = \sup_{y \in \mathcal{S}} \min_{1 \le t \le T} \|y - x_t\|_2. \qquad (2)$$

In practice the supremum is approximated by drawing $N = 10{,}000$ points z_i uniformly from $\mathcal{S}$ and computing

$$\widehat{\mathrm{disp}}(X) = \max_{1 \le i \le N} \min_{1 \le t \le T} \|z_i - x_t\|_2, \qquad (3)$$

with nearest-neighbour queries accelerated by a KD-tree. Large values indicate that large regions of $\mathcal{S}$ remain unvisited. Lower values signal better search-space coverage.

`avg_exploration_pct`. For any finite set $S \subset \mathbb{R}^d$, the average pairwise Euclidean distance is

$$D(S) = \frac{2}{|S|(|S|-1)} \sum_{\substack{p<q \\ x_p, x_q \in S}} \|x_p - x_q\|_2. \qquad (4)$$

With axis widths $w_k = u_k - l_k$ and mean width $\bar{w} = \frac{1}{d}\sum_{k=1}^{d} w_k$, the expected pairwise distance of two i.i.d. uniform samples in $\mathcal{S}$ is approximated by

$$D_{\mathrm{rand}} = \bar{w}\sqrt{d/6}. \qquad (5)$$

The trace X is partitioned into C consecutive chunks $S_0, \ldots, S_{C-1}$ of size $K = 500$, and each chunk is scored relative to random search:

$$E_c = \min\left\{100 \cdot \frac{D(S_c)}{D_{\mathrm{rand}}},\, 100\right\} \quad (0 \le E_c \le 100). \qquad (6)$$

The average exploration percentage is the mean across chunks:

$$\bar{E} = \frac{1}{C} \sum_{c=0}^{C-1} E_c. \qquad (7)$$

Values near 100 indicate chunk-level diversity on par with random search. Values near 0 indicate concentrated, exploitative search.

avg_distance_to_best. Let $b^*(k) = \arg\min_{t \le k} y_t$ denote the index of the best-so-far point at evaluation k. The mean distance from each evaluation to the best-so-far incumbent:

$$\bar{\Delta} = \frac{1}{T-1} \sum_{k=1}^{T-1} \|x_k - x_{b^*(k)}\|. \tag{8}$$

Low values indicate strong local exploitation around the incumbent.

intensification_ratio. The fraction of evaluations within a radius $r = 0.1 \cdot \bar{w}$ of the final best point $x^* = \arg\min_t y_t$:

$$I = \frac{1}{T} \sum_{t=1}^{T} \mathbf{1}[\|x_t - x^*\| < r]\,. \tag{9}$$

High values indicate strong exploitation near the eventual best solution.

avg_exploitation_pct. The complement of the exploration percentage:

$$\bar{X} = 100 - \bar{E}. \tag{10}$$

It captures the share of chunk-level variance attributable to local refinement.

average_convergence_rate. Let $e_t = f_t^* - f_T^*$ be the error of the best-so-far above the final best, and ε machine epsilon (added to avoid division by zero). The geometric mean of successive error ratios:

$$\mathrm{ACR} = \exp\left(\frac{1}{T-1} \sum_{t=1}^{T-1} \log \frac{e_{t+1} + \varepsilon}{e_t + \varepsilon} \right). \tag{11}$$

$\mathrm{ACR} < 1$ indicates convergence. Smaller values indicate faster convergence.

avg_improvement. Let $\mathcal{I} = \{t : y_t < f_{t-1}^*\}$ be the set of improving iterations. The mean improvement magnitude over those iterations:

$$\bar{I} = \frac{1}{|\mathcal{I}|} \sum_{t \in \mathcal{I}} (f_{t-1}^* - y_t). \tag{12}$$

Large values indicate the algorithm makes substantial gains when it improves. Small values indicate refinement in small increments.

success_rate. The fraction of iterations that improve the best-so-far:

$$S = \frac{|\mathcal{I}|}{T-1}. \tag{13}$$

High values indicate the algorithm frequently finds better solutions, which may reflect either an easy problem or effective search steps. Low values indicate the algorithm struggles to improve on the incumbent.

longest_no_improvement_streak. The length of the longest contiguous run of iterations that do not improve the best-so-far:

$$L = \max\{b - a + 1 : y_t \ge f_{t-1}^* \text{ for all } t \in [a, b]\}. \tag{14}$$

Streaks of several thousand out of $T = 10{,}000$ indicate the algorithm has effectively stopped progressing.

last_improvement_fraction. Let $t^* = \max \mathcal{I}$ be the index of the last improving iteration. The fraction of the budget that elapses after it:

$$\Phi = \frac{T - 1 - t^*}{T - 1}. \tag{15}$$

High values indicate the algorithm plateaued early and made no further progress. Low values indicate it continued improving until near the end.

3.3 Step-Size and Movement Dynamics

The first group of new features characterises the basic mechanics of how a candidate algorithm moves through decision space. Step-size statistics (mean, standard deviation, OLS slope) and directional persistence are basic trajectory descriptors. Step-size statistics in particular are common in time-series feature libraries such as `tsfresh` [47]. De Nobel et al. [48] applied `tsfresh` to CMA-ES internal strategy parameters (step-size σ, evolution paths), though notably none of the simple statistics over σ that they considered were retained by their feature selector. We compute analogous statistics directly on Euclidean step magnitudes $\|x_{t+1} - x_t\|$ from the trajectory, which is algorithm-agnostic rather than CMA-ES-specific. Directional persistence, the mean cosine similarity between consecutive displacement vectors, is a standard descriptor in random-walk and movement-ecology analyses [13]. Let $s_t = \|x_{t+1} - x_t\|$ denote the Euclidean step size at evaluation t.

step_size_mean. The mean step size across the trajectory:

$$\bar{s} = \frac{1}{T-1} \sum_{t=1}^{T-1} s_t. \tag{16}$$

Large values of the mean step size $\bar{s}$ indicate broad exploration. Small values indicate local exploitation.

step_size_std. The standard deviation of step sizes:

$$\sigma_s = \sqrt{\frac{1}{T-2} \sum_{t=1}^{T-1} (s_t - \bar{s})^2}. \tag{17}$$

High step-size standard deviation σ_s signals alternating exploration and exploitation phases. Low variability signals a uniform step-size regime throughout the run.

step_size_trend. The ordinary least-squares slope of step size against evaluation index:

$$\beta_s = \frac{\sum_t (t - \bar{t})(s_t - \bar{s})}{\sum_t (t - \bar{t})^2}. \tag{18}$$

A negative step-size trend β_s indicates the classic convergence pattern of contracting step sizes. A positive slope suggests the algorithm diverges over time.

`directional_persistence`. The mean cosine similarity between consecutive displacement vectors $d_t = x_{t+1} - x_t$:

$$\phi = \frac{1}{T-2} \sum_{t=2}^{T-1} \frac{d_t \cdot d_{t-1}}{\|d_t\| \, \|d_{t-1}\|}. \tag{19}$$

Values of the directional persistence ϕ near $+1$ indicate that the search maintains a consistent heading (gradient-following behaviour). Values near 0 indicate a random walk. Negative values indicate anti-persistent zigzag motion.

3.4 Information-Theoretic Features

The four features in this category apply established time-series complexity measures to the raw fitness sequence $\{y_1, \ldots, y_T\}$. While each measure is well-known in signal processing and physics, their application to optimisation algorithm fitness trajectories on BBOB benchmarks is new.

`fitness_sample_entropy`. Sample Entropy (SampEn) quantifies the regularity of the fitness series [14]. Given embedding dimension m and tolerance $r = 0.2 \cdot \mathrm{std}(y)$, it counts the conditional probability that sequences matching for m consecutive values still match at length $m+1$:

$$\mathrm{SampEn}(m, r) = -\ln \frac{A^m(r)}{B^m(r)}, \tag{20}$$

where B^m is the number of template matches of length m and A^m the number that also match at length $m+1$ (excluding self-matches). Low SampEn indicates a repetitive, self-similar convergence pattern. High SampEn indicates complex, unpredictable dynamics. We use $m = 2$ and subsample every 10th evaluation ($N = 1{,}000$) to reduce the $O(N^2 m)$ cost from $\sim$30 s to $\sim$0.3 s per trajectory.

`fitness_permutation_entropy`. Normalised Permutation Entropy (PE) maps the fitness series to a sequence of ordinal patterns and measures their distributional uniformity [15]. Each window of D consecutive values is replaced by its rank permutation $\pi \in S_D$, and PE is the Shannon entropy of the permutation distribution normalised by $\log_2(D!)$:

$$\mathrm{PE}(D) = -\frac{1}{\log_2(D!)} \sum_{\pi \in S_D} p(\pi) \log_2 p(\pi). \tag{21}$$

We use $D = 5$, giving $D! = 120$ possible patterns against $T = 10{,}000$ data points. PE ≈ 0 corresponds to perfectly monotone convergence. PE ≈ 1 to a completely random sequence.

`fitness_autocorrelation`. The lag-1 autocorrelation of the raw fitness series, following Weinberger's framework for fitness landscape correlation structure [16]:

$$\rho_y(1) = \frac{\sum_{t=1}^{T-1} (y_t - \bar{y})(y_{t+1} - \bar{y})}{\sum_{t=1}^{T} (y_t - \bar{y})^2}. \tag{22}$$

Importantly, while Weinberger's autocorrelation has long been used to characterise landscapes via *separate* random walks, it has not previously been computed on the algorithm's *own* trajectory on BBOB. High positive $\rho_y(1)$ indicates smooth, locally exploitative search. Values near zero indicate the algorithm jumps between unrelated fitness regions.

`fitness_lempel_ziv_complexity`. Normalised Lempel–Ziv complexity (LZC) measures the rate at which new patterns appear in a binarised version of the fitness series [17]. The binarisation maps each step to $b_t = \mathbf{1}[y_{t+1} < y_t]$ (improvement or not), yielding a binary string. LZ76 parsing counts the number of distinct substrings $c(N)$, and the normalised complexity is:

$$\mathrm{LZC}_n = \frac{c(N)}{N / \log_2 N}. \tag{23}$$

$\mathrm{LZC}_n \approx 1$ indicates a random improvement/stagnation pattern. $\mathrm{LZC}_n \approx 0$ indicates highly repetitive structure (e.g., long unbroken stagnation followed by a burst of improvements).

3.5 Adapted Population Dynamics

Cenikj et al. [11] introduced DynamoRep, a set of trajectory-based features that characterise how a population's spatial and fitness distributions change across generations. Their features assume a population-based algorithm with multiple individuals per generation. In our (1+1)-ES setting the population size is 1, so per-generation statistics are undefined. We adapt the DynamoRep idea by comparing the first 25% of evaluations (*early*) with the last 25% (*late*), treating each window as a proxy for the population at that stage of the search.

Let $X_{\text{early}} = \{x_t : t \le T/4\}$ and $X_{\text{late}} = \{x_t : t > 3T/4\}$.

`x_spread_early` and `x_spread_late`. Mean per-dimension standard deviation of decision variables in the early and late windows, respectively:

$$S_{\mathrm{E}} = \frac{1}{d} \sum_{j=0}^{d-1} \mathrm{std}\big(\{x_t^{(j)} : t \le T/4\}\big). \tag{24}$$

The late spread S_{L} is defined identically over $t > 3T/4$. High early spread S_{E} indicates broad initial exploration. Low S_{L} indicates the search has converged to a small region.

`spread_ratio`. The ratio of late to early spread:

$$R_S = S_{\mathrm{L}} \,/\, S_{\mathrm{E}}. \tag{25}$$

Values of the spread ratio R_S below 1 indicate the algorithm narrows its search over time (the expected convergence pattern). Values above 1 indicate late-stage diversification.

`centroid_drift`. The Euclidean distance between the centroids of the early and late windows:

$$D_c = \left\| \bar{x}_{\text{early}} - \bar{x}_{\text{late}} \right\|. \tag{26}$$

Large centroid drift D_c means the algorithm relocated its search centre during the run it discovered a new promising region. Small drift means the algorithm stayed local throughout.

`f_range_early`, `f_range_late`, and `f_range_ratio`. The range (maximum minus minimum) of raw fitness values in each window, and their ratio:

$$F_{\text{E}} = \max_{t \le T/4} y_t - \min_{t \le T/4} y_t, \tag{27}$$

$$F_{\text{L}} = \max_{t > 3T/4} y_t - \min_{t > 3T/4} y_t, \tag{28}$$

$$R_F = F_{\text{L}} \,/\, F_{\text{E}}. \tag{29}$$

Here F_{E} and F_{L} are the early and late fitness ranges, and R_F is their ratio. A shrinking fitness range ($R_F < 1$) indicates the algorithm is converging to a narrow fitness band near the optimum. A persistent or growing range suggests the algorithm still samples points of widely varying quality.

3.6 Additional Trajectory Features

The six features in this section adapt established statistical and time-series concepts to the analysis of metaheuristic search trajectories. In each case the underlying mathematical tool (autocorrelation, plateau detection, coefficient of variation, Pearson correlation, per-dimension range analysis) is standard in adjacent fields, but its application as a behavioural feature for metaheuristic characterisation is, to our knowledge, new to the optimisation literature.

`step_size_autocorrelation`. Autocorrelation of fitness values is a well-studied landscape descriptor [16], and we include it above as `fitness_autocorrelation`. The closest precedent for autocorrelation as a behavioural feature in optimisation is de Nobel et al. [48], who applied lag-1 autocorrelation to the CMA-ES conjugate evolution path $\|p_\sigma\|$ as part of a `tsfresh`-extracted feature set. Our feature differs in two ways. First, it operates on Euclidean displacement step sizes from the trajectory, which exist for any algorithm, rather than on a CMA-ES internal control signal. Second, it is selected and reported as a single named, interpretable behavioural descriptor rather than embedded in an automatically extracted feature bank.

Let $s_t = \|x_{t+1} - x_t\|$ denote the step size at evaluation t. The lag-1 autocorrelation of the step-size series is:

$$\rho_s = \frac{\sum_{t=1}^{T-2} (s_t - \bar{s})(s_{t+1} - \bar{s})}{\sum_{t=1}^{T-1} (s_t - \bar{s})^2}. \tag{30}$$

The step-size autocorrelation ρ_s captures *search momentum*. High positive autocorrelation ($\sim$0.9) means the algorithm takes similarly-sized steps consecutively, characteristic of smooth step-size adaptation mechanisms such as the 1/5-th success rule or CMA-ES's path-length control. Low or negative autocorrelation indicates erratic step-size variation, typical of random restarts or poorly tuned mutation operators.

`fitness_plateau_fraction`. Existing stagnation metrics in BLADE (`longest_no_improvement_streak`, `last_improvement_fraction`) measure plateaus in the *best-so-far* curve. This conflates two distinct phenomena, the algorithm may be sampling diverse points that happen to be worse (active exploration), or it may be stuck in a genuinely flat region of the landscape. We disentangle these by measuring plateaus in the *raw* fitness sequence.

Let $\epsilon = 10^{-8} \cdot (\max_t y_t - \min_t y_t)$ be a relative tolerance. The plateau fraction is:

$$P_{\text{flat}} = \frac{1}{T-1} \sum_{t=1}^{T-1} \mathbf{1}\big[|y_{t+1} - y_t| < \epsilon\big]. \tag{31}$$

High plateau fraction P_{flat} means the algorithm repeatedly evaluates points with nearly identical fitness, indicating it is trapped in or deliberately exploiting a flat region. This differs fundamentally from best-so-far stagnation, an algorithm can have a long no-improvement streak while still sampling diverse fitness values (because all are worse than the incumbent). Conversely, a high plateau fraction with a short no-improvement streak indicates the algorithm has found a flat basin and is refining within it.

`half_convergence_time`. While `average_convergence_rate` captures the geometric mean of per-step convergence ratios, it says nothing about when during the budget the bulk of improvement occurs. Two algorithms can have the same convergence rate but very different temporal profiles. One may improve rapidly in the first 10% of evaluations and then plateau, while another may start slowly but achieve a late breakthrough.

The half-convergence time measures when 50% of the total best-so-far improvement is reached:

$$t_{1/2} = \frac{1}{T-1} \min\big\{ t : f_t^* \le f_1^* - \tfrac{1}{2}(f_1^* - f_T^*) \big\}. \tag{32}$$

Low values of the half-convergence time $t_{1/2}$ ($\sim$0.001–0.01) indicate rapid early convergence followed by slow refinement. High values ($\sim$0.5–1.0) indicate a slow start with late breakthroughs. The feature returns 1.0 when no improvement occurs.

`improvement_spatial_correlation`. We are not aware of a prior feature that relates the *distance* an algorithm travels to the *magnitude* of the improvement it achieves. We define the improvement spatial correlation

as the Pearson correlation between step size and improvement magnitude, restricted to improving steps:

$$\mathcal{I} = \{t : y_t < f^*_{t-1}\},$$
$$r_{\text{IS}} = r_{\text{Pearson}}\big(\{s_{t-1}\}_{t\in\mathcal{I}},\ \{f^*_{t-1} - y_t\}_{t\in\mathcal{I}}\big). \tag{33}$$

Positive values of the improvement spatial correlation r_{IS} indicate that large jumps yield large improvements. Negative values indicate that small, local steps produce the biggest gains. Values near zero mean improvement magnitude is independent of the distance travelled. This feature separates algorithms that improve through directed long-range moves from those that improve through local refinement.

`improvement_burstiness`. Barabási [18] showed that many natural processes exhibit bursty dynamics. Events tend to cluster in time rather than arriving at a steady rate. We apply this idea to the temporal pattern of fitness improvements.

Let $\mathcal{I} = \{t_1, t_2, \ldots, t_k\}$ be the sorted indices of improving evaluations, and $\tau_i = t_{i+1} - t_i$ the inter-improvement intervals. The improvement burstiness is the coefficient of variation of these intervals:

$$B_{\text{imp}} = \frac{\text{std}(\tau)}{\text{mean}(\tau)}. \tag{34}$$

The improvement burstiness B_{imp} is a coefficient of variation (CV). Values near 1 correspond to Poisson-like random improvements. CV $\gg 1$ indicates bursty dynamics where improvements arrive in clusters separated by long stagnation phases. CV $\ll 1$ indicates metronomic, evenly-spaced improvements. This feature characterises the temporal *texture* of the search process. Does the algorithm make steady incremental progress, or does it alternate between productive bursts and idle periods?

`dimension_convergence_heterogeneity`. Standard convergence metrics (spread ratio, step-size trend) aggregate across all dimensions. They cannot detect whether the algorithm converges uniformly in all dimensions or locks in on some dimensions while continuing to explore others, a pattern that reflects sensitivity to problem separability.

For each dimension j, we compute the range shrinkage ratio between the early and late windows:

$$\sigma_j = \frac{\max_{t>3T/4} x_t^{(j)} - \min_{t>3T/4} x_t^{(j)}}{\max_{t\le T/4} x_t^{(j)} - \min_{t\le T/4} x_t^{(j)}}. \tag{35}$$

The convergence heterogeneity is the standard deviation of these ratios across dimensions:

$$H_{\text{dim}} = \text{std}\big(\sigma_0, \sigma_1, \ldots, \sigma_{d-1}\big). \tag{36}$$

Low dimension convergence heterogeneity H_{dim} means all dimensions converge at a similar rate, so the algorithm treats the problem as isotropic. High heterogeneity means some dimensions converge tightly while others remain spread, suggesting the algorithm identifies and exploits coordinate-wise structure.

Table 1: Computational cost of behavioural feature extraction. Timings are the mean of 50 runs on a single CPU core for $T = 10{,}000$ evaluations in $d = 5$ dimensions. SampEn uses subsampled trajectories ($N = 1{,}000$).

Category	#	Per traj. (ms)	Per cand.
Existing (BLADE)	11	743.7 ± 5.3	∼37 s
Step-size dynamics	4	1.6 ± 0.1	∼80 ms
Information-theoretic	4	27.5 ± 0.5	∼1.4 s
Adapted pop. dynamics	7	1.0 ± 0.1	∼48 ms
Newly introduced	6	1.5 ± 0.1	∼75 ms
Total	**32**	$\mathbf{775.2 \pm 5.4}$	**∼39 s**

3.7 Computational Cost Analysis

Table 1 summarises the computational cost of each feature category, measured over 50 repeated runs on a synthetic trajectory matching the experimental setup ($T = 10{,}000$ evaluations, $d = 5$). Each candidate algorithm is evaluated on 50 MA-BBOB trajectories (10 functions × 5 instances), so per-candidate cost is 50× the per-trajectory cost. The dominant cost is the existing BLADE metrics, which build a KD-tree for dispersion and iterate over all evaluations for nearest-neighbour distance. The 17 features from categories 1, 4, and 5 add under 250 ms per candidate in total.

Relative to the MA-BBOB evaluation itself (30–120 s per candidate to run the algorithm 50 times), the total feature extraction overhead of ∼39 s per candidate is modest. Over a full experimental run of 100 candidates, this amounts to roughly 65 minutes of feature extraction compared to 1–3 hours of algorithm evaluation.

4 Methodology

This chapter describes the experimental pipeline used to answer the research questions posed in Section 1. The pipeline runs in four stages. Stage 1 screens ten LLMs and identifies the strongest model for algorithm generation. Stage 2 analyses the resulting trajectories and selects an informative subset of behavioural features. Stage 3 tests three framings of those features as feedback to the LLM. Stage 4 brings the chosen model, features, and framing together in a full-benchmark head-to-head comparison against vanilla and SAGE feedback baselines. We first describe the LLaMEA framework and benchmark configuration that underpin all four stages.

4.1 LLaMEA Framework

Vanilla LLaMEA operates a (1+1)-ES loop, summarised in Algorithm 1. Every run starts from a shared initial algorithm, `RandomSearch`, which performs uniform random sampling across the search space and returns the best sample found. Pre-seeding the population this way ensures that all models and conditions begin from identical starting conditions, so observed differences are attributable to the LLM and the feedback rather than to initialisation luck.

At each generation the LLM receives a prompt assembled from a role description, a task specification, an example algorithm, a feedback string reporting that algorithm's performance, an optional mutation prompt, and an output-format constraint. The role, task, and output-

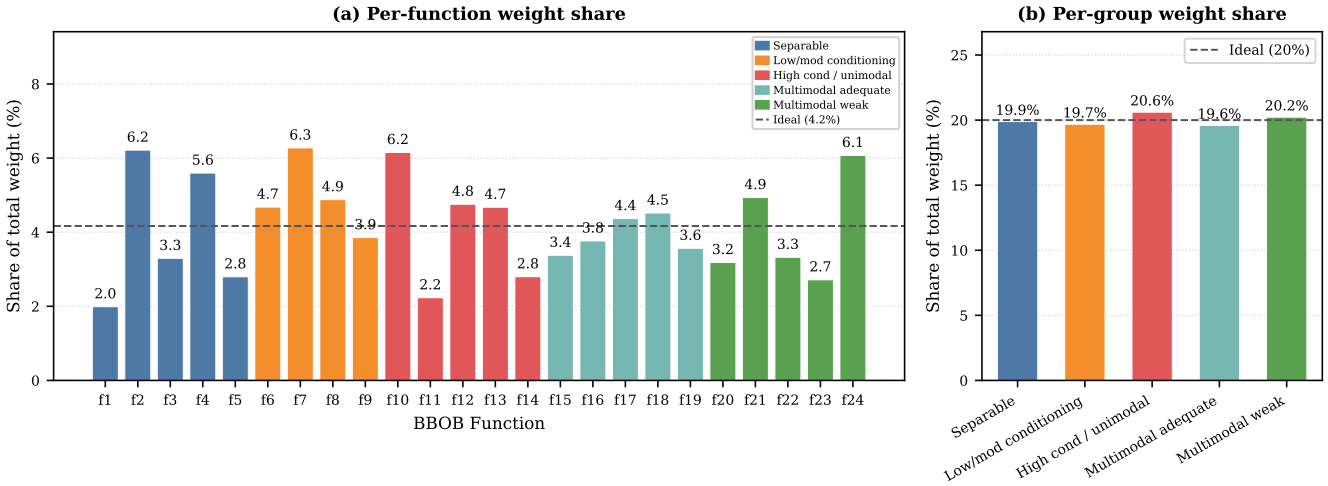

Figure 1: BBOB function weight distribution of the selected 10 MA-BBOB functions, coloured by BBOB function group. All 24 BBOB functions are covered with near-uniform balance.

Algorithm 1: (1+1)-LLaMEA evolutionary loop, adapted from [6].

Input : Task prompt S, feedback prompt template F_0, budget T
Output : Best algorithm a_b and its quality y_b

$a_0 \leftarrow \text{LLM}(S)$;
$(y_0, \sigma_0, e_0) \leftarrow f(a_0)$;
$a_b \leftarrow a_0$; $y_b \leftarrow y_0$;
for $t \leftarrow 1$ **to** T **do**
  $F \leftarrow (S,\ \{(\text{name}(a_i), y_i)\}_{i=0}^{t},\ (a_b, y_b, \sigma_b),\ F_0)$;
  $a_t \leftarrow \text{LLM}(F)$;
  $(y_t, \sigma_t, e_t) \leftarrow f(a_t)$;
  if $e_t \neq \varnothing$ **then**
    $y_t \leftarrow 0$
  if $y_t \geq y_b$ **then**
    $a_b \leftarrow a_t$; $y_b \leftarrow y_t$; $\sigma_b \leftarrow \sigma_t$
return a_b, y_b

format portions are taken verbatim from LLaMEA [6]. The full prompt template is shown below.

```
You are a highly skilled computer scientist in the
field of natural computing. Your task is to design
novel metaheuristic algorithms to solve black box
optimization problems.

The optimization algorithm should handle a wide range
 of tasks, which is evaluated on the BBOB test suite
of 24 noiseless functions. Your task is to write the
optimization algorithm in Python code to minimize the
 function value. The code should contain an `__init__
(self, budget, dim)` function and the function `def
__call__(self, func)`, which should optimize the
black box function `func` using `self.budget`
function evaluations.

The func() can only be called as many times as the
budget allows, not more. Each of the optimization
functions has a search space between -5.0 (lower
bound) and 5.0 (upper bound). The dimensionality can
be varied.

Give an excellent and novel heuristic algorithm to
solve this task.

An example of such code is as follows:
<example algorithm>

Feedback:

The algorithm <algorithm name> got an average Area
over the convergence curve (AOCC, 1.0 is the best)
score of <score> with standard deviation <std>.

<mutation prompt>

Provide the Python code and a one-line description
with the main idea (without enters). Give the
response in the format:
# Description: <short-description>
# Code:
```python
<code>
```
```

On the first generation the example is `RandomSearch` (reproduced in Section A) and no mutation prompt is added. From the second generation onwards, the previous best algorithm replaces `RandomSearch` as the example, and the mutation prompt is drawn from the dual-prompt configuration identified as effective by van Stein et al. [12]. With 90% probability the LLM receives a *refine* prompt ("*Refine and simplify the selected algorithm to improve it*"), and with 10% probability an *explore* prompt ("*Generate a new algorithm that is different from those tried before*"). This split balances exploitation of promising algorithm structures with exploration of novel designs. The 90/10 ratio is our own choice and to our knowledge no systematic study of its optimality exists.

The offspring then competes against its parent under elitist $(1 + 1)$ selection. It replaces the parent only if it achieves equal or better fitness, and the surviving algorithm becomes the parent for the next generation. Recent work has explored larger population sizes. Van Stein et al. [12] compare six LLaMEA configurations including population-based variants with $\mu/\lambda = 4/12$, and the SAGE experiments [10] use 4 parents and 16 offspring. Among the six configurations in the Behaviour Space study [12], the elitist $(1 + 1)$ variant (LLaMEA-4) outperformed the population-based alternatives, and we retain it for that reason.

After each evaluation, the LLM receives a feedback

Table 2: BBOB function weights for the 10 selected MA-BBOB functions. Only non-zero contributions are shown (as percentage of total weight). BBOB functions are grouped into separable (f1–f5), low/moderate conditioning (f6–f9), high conditioning/unimodal (f10–f14), multimodal adequate (f15–f19), multimodal weak (f20–f24).

	Sep.					**Low/mod**				**High/uni**					**MM adeq.**					**MM weak**				
Func.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
191						28													25	21	1		25	
277	15		14				10			13			13	6				20				6		4
300			19				20					18						12			29		2	
412		10			12			13		14							11		11					29
455		13						17		14	14							14						28
635				23	4		13					9		16		23						14		
648	5			9			20		18				18		17					11	2			
744		17				19				10				7	17	15	15							
760		1		9	12			4	20		7		15								17	14		
843		21		14				14		10	2	20					18							

Values are percentages of total weight. Empty cells indicate zero weight.

string reporting the algorithm's performance. Standard (vanilla) feedback reports only the AOCC score and its standard deviation across evaluation runs. Our experiments augment this string with additional behavioural feature information detailed in in Section 4.5.

4.2 Benchmark Configuration

Candidate algorithms are evaluated on MA-BBOB [38]. Each MA-BBOB function is parameterised by a weight vector over the 24 BBOB functions. All functions used in this thesis are 5-dimensional with search bounds $[-5, 5]^5$.

This thesis uses two function sets drawn from the same 1,000-function MA-BBOB pool. The first three stages use a ten-function set, sized to discriminate between models, features, and feedback formats while keeping screening compute tractable across the many models, conditions, and seed replicates that those stages require. Stage 4 uses an extended twenty-function set, disjoint from the screening set, both to give the head-to-head comparison tighter AOCC variance bands and to test that the conclusions of Stages 1–3 generalise to functions the pipeline was not implicitly tuned on.

Both function sets are selected by the greedy forward procedure in Algorithm 2. At each step every remaining function is scored as a potential addition to the current set, with the score favouring candidates that cover BBOB functions not yet represented, that bring the five group shares closer to the ideal 20% each, and that distribute weight evenly across BBOB functions. Missing BBOB function coverage dominates the score, ensuring all 24 BBOB functions are included before the procedure optimises for balance among individual BBOB function representation. For the ten-function screening set the resulting selection covers all 24 BBOB functions with group shares within ±0.6% of the ideal 20% (Figure 1). Table 2 shows the per-function weight breakdown, where each MA-BBOB function blends between 5 and 9 BBOB functions.

The twenty-function Stage 4 set is produced by the same procedure with the ten-function screening set excluded from the candidate pool, ensuring disjointness. The resulting subset covers all 24 BBOB functions with a coefficient of variation of 0.080 across BBOB function weight totals, and gives the five BBOB function groups shares within ±1.3% of the ideal 20%. The BBOB function weight distribution and per-function breakdown are reported in Section B.

Algorithm 2: Greedy MA-BBOB function selection. At each step the candidate whose addition most reduces missing BBOB function count, group imbalance, and per-BBOB-function weight variability is selected.

Input : $\mathbf{W} \in \mathbb{R}^{1000\times 24}$ (weight matrix), $k = 10$
Output : Selected function set S

$S \leftarrow \varnothing$;
for $i \leftarrow 1$ **to** k **do**
  foreach $c \notin S$ **do**
    $\mathbf{w} \leftarrow$ column sums of $\mathbf{W}[S \cup \{c\}]$;
    $m \leftarrow |\{j : w_j = 0\}|$;
    $\sigma_g \leftarrow \mathrm{std}(\{w_g / \sum w : g \in \text{groups}\})$;
    $\mathrm{CV} \leftarrow \mathrm{std}(\mathbf{w}_{>0}) / \mathrm{mean}(\mathbf{w}_{>0})$;
    $\mathrm{score}(c) \leftarrow -100\, m - 10\, \sigma_g - \mathrm{CV}$;
  $c^* \leftarrow \arg\max_c \ \mathrm{score}(c)$;
  $S \leftarrow S \cup \{c^*\}$;
return S

This selection procedure was used to mitigate design bias inherited from the underlying BBOB function set. MA-BBOB is itself a technique for reducing such bias by allowing arbitrary affine combinations of the 24 base functions, but because our function sets are sampled from a finite MA-BBOB pool, residual structure from BBOB persists. The base functions share landscape characteristics within their group as exposed by exploratory landscape analysis [49], and the five groups are unevenly populated (4 functions in low/moderate conditioning, 5 in each of the others). Selecting for uniform BBOB-function coverage and balanced group shares at the contribution-weight level minimises the influence of any single function or group on aggregated performance, though some residual bias from the fixed BBOB pool is unavoidable.

Each candidate algorithm is evaluated on every function in the active set over 5 configuration instances. For Stages 1–3 this yields 50 independent runs per candidate (10 functions × 5 instances) and for Stage 4 it yields

100 (20 functions × 5 instances). The evaluation budget per run is $T = 2{,}000 \times d = 10{,}000$ function evaluations (FEs), following the BBOB convention [37].

Algorithm quality is measured by the Area Over the Convergence Curve (AOCC). For a single run of algorithm a on function f and instance s, with evaluation budget T, the normalised per-run area is:

$$A(\mathbf{y}_{a,f,s}) = \frac{1}{T}\sum_{t=1}^{T}\left(1 - \frac{\min(\max(y_t, lb), ub) - lb}{ub - lb}\right) \tag{37}$$

where $\mathbf{y}_{a,f,s} = (y_1, \ldots, y_T)$ is the sequence of log-scaled best-so-far fitness values, and $ub = 10^2$, $lb = 10^{-8}$ are the clipping bounds. The AOCC for a candidate algorithm is the mean of A over all functions f and instances s:

$$\mathrm{AOCC}(a) = \frac{1}{N}\sum_{f=1}^{F}\sum_{s=1}^{S} A(\mathbf{y}_{a,f,s}) \tag{38}$$

In Stages 1–3 we have $F = 10$ and $S = 5$, giving $N = 50$ independent runs per candidate. Stage 4 uses $F = 20$, raising N to 100. This single scalar is reported to the LLM as feedback.

For every successful candidate evaluation, the full trajectory ($T = 10{,}000$ evaluations of decision variables and fitness values) is logged via IOHexperimenter, and all 32 behavioural features described in Section 3 are computed post-hoc. Features are averaged across the 50 runs per candidate to produce a single feature vector per algorithm. This dataset, comprising feature vectors for every valid candidate across all models and seeds, forms the basis for the Stage 2 feature analysis.

4.3 LLM Screening

Stage 1 screens LLMs for their ability to reliably generate high-quality metaheuristics for continuous black-box optimisation. Small language models vary substantially in their ability to generate valid optimisation code, and inference cost matters when each evolutionary run requires hundreds of LLM calls. The screening identifies the model that is later carried forward into Stages 3 and 4, and supplies the dataset of trajectories on which Stage 2 is built.

Ten models are evaluated, spanning locally served open-weight models and cloud API models. Table 3 lists all models with their parameter counts and providers. Each model runs the LLaMEA loop for 100 candidates across 5 independent seeds, all using vanilla AOCC-only feedback, for a total of 10 models × 5 seeds × 100 candidates = 5,000 LLaMEA iterations. All 32 behavioural features are computed for every successful candidate, producing a dataset of several thousand algorithm–feature observations.

Models are compared on three factors. The failure rate is the fraction of candidates that fail to produce valid fitness, due to syntax errors, interface mismatches, or runtime exceptions. The best AOCC is the highest AOCC achieved across all 5 seeds, measuring peak algorithm quality. The inference efficiency is the rate of tokens generated per second, which determines wall-clock time per evolutionary run. The model that performs best across these factors is carried forward into later stages.

Table 3: LLMs evaluated in Stage 1. Local models are served via Ollama [50] or vLLM [51]. API models use Google's Gemini API. Parameter counts reflect total (not active) parameters.

Model	Provider	Params	Backend
Qwen3.5 4B	Qwen/Alibaba [52]	4B	Ollama
Qwen3.5 9B	Qwen/Alibaba [52]	9B	vLLM
Qwen3.5 27B	Qwen/Alibaba [52]	27B	Ollama
RNJ-1 8B	Essential AI [53]	8B	Ollama
Devstral Small 2	Mistral AI [54]	24B	Ollama
OLMo 3 7B	Allen AI [55]	7B	Ollama
OLMo 3 32B Think	Allen AI [55]	32B	Ollama
Granite 4 Micro	IBM [56]	3B	vLLM
Gemini 3 Pro	Google [57]	—	API
Gemini 3 Flash	Google [58]	—	API

4.4 Feature Analysis and Selection

Stage 2 analyses the Stage 1 dataset of behavioural feature vectors for all valid candidates across 10 models and 5 seeds to determine which of the 32 features in Section 3 are most predictive of algorithm quality, and to select a tractable subset for the feedback experiments. The target subset size is fixed at ten features. The cap is set by the available compute budget and API costs rather than statistical considerations. Stage 3 evaluates each retained feature in three framings across 5 seeds of 100 candidates with the model identified in Stage 1, and the per-feature cost of running these evaluations makes a larger subset infeasible. Selecting on predictive power keeps the ten most informative features in the loop, while pruning redundancy ensures they capture distinct aspects of behaviour rather than re-stating the same signal.

Three complementary approaches are applied to rank the 32 features, each capturing a different aspect of feature quality. Spearman rank correlation [45] captures monotonic associations between each feature and AOCC across all valid candidates, regardless of functional form. Random Forest permutation importance [59] captures non-linear and interaction effects. A regression model is trained to predict AOCC from all 32 features, and each feature's importance is measured by the decrease in out-of-bag R^2 when its values are permuted. Kolmogorov–Smirnov distributional effect size [46] captures distributional separation between high- and low-quality algorithms. It computes the KS statistic between feature distributions of the top-25% and bottom-25% candidates by AOCC, after winsorising each feature at the 1st and 99th percentiles to limit the influence of outliers.

Each method produces a ranked list of 32 features. The three rankings are aggregated via Borda count [60]. Each feature receives $32 - r_i$ points from each method (where r_i is its rank in method i). Summed scores determine the consensus ranking, with lower values indicating a more informative feature.

The consensus list is then de-duplicated by greedy redundancy pruning. Features are processed in Borda-rank order. If a candidate feature has Spearman $|\rho| > \tau$ with

any already-selected feature, it is discarded as redundant. To determine τ we sweep across the range $[0.5, 1.0]$ in steps of 0.05 and record, at each value, the surviving subset and its mean Borda rank, then pick the value that produces the largest single-step improvement in mean Borda rank. This corresponds to the elbow in the quality-vs-threshold curve, beyond which further tightening prunes already-distinct features without comparable gain. The top ten survivors of pruning at this threshold form the final feature subset taken into Stage 3.

4.5 Feedback Format Screening

Stage 3 tests whether the framing of behavioural feedback affects the performance of LLM-generated algorithms. Each feature selected in Stage 2 is embedded into the feedback string under three contrasting framings (neutral, directional, comparative), with one feature added at a time so that effects can be attributed to the framing rather than to the joint presence of multiple features. The comparative format is excluded for `longest_no_improvement_streak` because that feature has a U-shaped relationship with AOCC for which a single reference target would be misleading, leaving 29 active conditions.

The neutral format reports the feature value and its definition, without any indication of whether high or low values are desirable,

> *The algorithm X got an average AOCC score of 0.7234 with standard deviation 0.0891. It achieved a* `step_size_autocorrelation` *of 0.8421 with standard deviation 0.0532, which measures whether large steps tend to follow large steps, capturing momentum in the search.*

The directional format extends the neutral format with a sentence indicating which direction is associated with better performance. The direction is derived from the prior Spearman correlation analysis,

> *... capturing momentum in the search. Higher values are associated with better performance, indicating consistent, smooth step-size adaptation rather than erratic jumps.*

The comparative format replaces the directional guidance with a distance-aware comparison against a concrete reference value. The reference is the median feature value of top-10% algorithms by AOCC from the LLM screening stage. The normalised distance from this reference is $d = (v_{\text{ref}} - v)/(v_{\text{ref}} - v_{\text{bot25}})$, where v is the candidate's feature value, v_{ref} is the top-10% median, and v_{bot25} is the bottom-25% median. This denominator provides a meaningful scale representing the range between good and bad algorithms for each feature. The numeric reference value is reported in every tier so that the LLM has a concrete target rather than only a qualitative direction. The wording of the comparison itself adapts across four tiers, strengthening progressively as d grows:

1. **Already meets reference** ($d \leq 0$): "*This already meets or exceeds the top-performing reference of 0.9070. Maintain this.*"
2. **Close to reference** ($0 < d \leq 0.15$): "*This is close to the top-performing reference of 0.9070. A small increase could help.*"
3. **Moderate gap** ($0.15 < d \leq 0.50$): "*This is moderately below the top-performing reference of 0.9070. Consider increasing this value.*"
4. **Far from reference** ($d > 0.50$): "*This is far from the reference of 0.9070. Increasing this value is a priority.*"

Each condition runs for 100 candidates across 5 independent seeds, for a total of 29 conditions $\times$ 5 seeds $\times$ 100 candidates = 14,500 LLaMEA iterations, and the vanilla AOCC-only results from Stage 1 serve as the baseline for measuring performance improvements.

4.6 Full-Benchmark Comparison

Stage 4 is a head-to-head comparison of four feedback conditions, run on the twenty-function MA-BBOB set with longer evolutionary runs and more independent seeds than the screening stages so condition differences can be detected at the resolution required to draw substantive conclusions.

The vanilla condition uses AOCC-only feedback, identical to the baseline used in Stage 1, repeated here on the new function set as the reference point against which all other conditions are measured. The neutral condition extends AOCC with trajectory-derived behavioural features, isolating the contribution of distal (behavioural) feedback. The sage condition pairs AOCC with structural code-level guidance from LLaMEA-SAGE [10], used unmodified, isolating the contribution of proximal (structural) feedback. The combined condition appends both behavioural and structural feedback in the same prompt, testing whether the two channels are complementary or redundant.

The behavioural feedback uses the five features ranked highest by mean best AOCC across the Stage 3 neutral conditions. These features are `intensification_ratio`, `dimension_convergence_heterogeneity`, `fitness_plateau_fraction`, `avg_improvement`, and `improvement_spatial_correlation`. All five are appended to the AOCC feedback string in the neutral framing. The five-feature cap keeps the prompt compact while still spanning the four functional categories of Section 3 (intensification, convergence, stagnation, and exploration).

Benchmark parameters from Section 4.2 are inherited unchanged, with the Stage 4 function set detailed in Section B. Each condition runs for 500 candidates per seed, five times the screening budget, giving feedback signals that act gradually a fair chance to express their effect. Ten independent seeds per condition (double the previous stages) tighten the variance bands when comparing condition means and make condition-vs-condition differences detectable at smaller effect sizes. The total compute is 4 conditions $\times$ 10 seeds $\times$ 500 candidates = 20,000 candidate algorithms, each evaluated over 20 functions $\times$ 5 instances = 100 MA-BBOB runs hence totalling 2,000,000 MA-BBOB function evaluations.

Finally, to assess how the strongest generated algo-

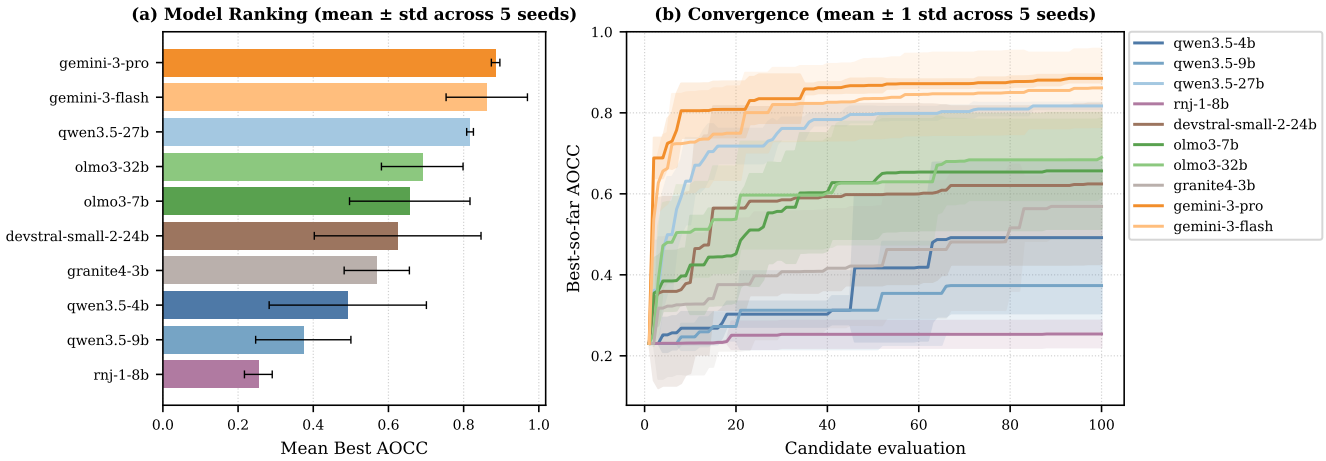

Figure 2: LLM screening results. (a) Mean best AOCC by model (± std across 5 seeds). (b) Best-so-far AOCC over 100 evaluations, averaged across 5 seeds. Shaded regions show ±1 std.

rithms generalise beyond the twenty-function set and the 5-dimensional regime in which they were developed, the best-found algorithm from each of the conditions is re-evaluated on the full 1,000-instance MA-BBOB pool. A generic CMA-ES [7] and `NeighborhoodAdaptiveDE` (NADE) [61], the winner of the 2025 MA-BBOB competition, are used as basline performance references. Each of the six algorithms runs on all 1,000 MA-BBOB functions across 5 instance seeds and three dimensionalities $d \in \{5, 10, 20\}$, under a $2{,}000d$ FE budget per run, totalling approximately 2.1×10^9 function evaluations.

4.7 Infrastructure

All experiments are executed on the LIACS SLURM cluster node **saronite**, equipped with 4× NVIDIA L40S GPUs (48 GB each), 2× 32-core Intel Xeon Gold 6438M CPUs, and 512 GB RAM. Most local models are served via Ollama [50], which processes requests sequentially in a queue. For Qwen3.5 9B and Granite 4 Micro, we use vLLM [51], which supports continuous batching and manages KV caches efficiently via PagedAttention [51]. This allows multiple seeds to issue concurrent inference requests rather than waiting in a serial queue, reducing wall-clock time per evolutionary run. Gemini 3 models [57, 58] are accessed through Google's Gemini API. All code, configuration, and analysis notebooks are available in the thesis repository.[1] The BLADE [41] and LLaMEA [6] repositories are included as Git submodules, with minor modifications to support custom feedback strings and extended behavioural metric logging.

5 Results

This chapter reports the experimental outcomes of the four-stage pipeline described in Section 4. Section 5.1 ranks the ten candidate LLMs and identifies the model carried into later stages. Section 5.2 reduces the 32 behavioural features to a tractable, predictive subset. Section 5.3 compares the three feedback framings of those features against the vanilla baseline. Section 5.4 runs the head-to-head comparison of vanilla, neutral-behavioural, SAGE, and combined feedback on the twenty-function MA-BBOB set with longer evolutionary runs and more independent seeds.

[1] https://github.com/Maxwe11h/thesis-experiments

Table 4: Stage 1 model ranking (100 candidates × 5 seeds, vanilla feedback). Fail% and AOCC averaged across seeds (± std).

#	Model	Fail%	AOCC	h/run
1	Gem. 3 Pro	10.2	.885±.011	5.1
2	Gem. 3 Flash	**2.0**	.862±.108	3.9
3	Qwen3.5 27B	41.0	.817±.009	38.9
4	OLMo 3 32B	13.2	.690±.109	39.3
5	OLMo 3 7B	27.6	.657±.160	18.4
6	Devstral 2	32.6	.625±.222	5.0
7	Granite 4 3B	53.8	.569±.087	**0.6**
8	Qwen3.5 4B	76.2	.492±.209	7.9
9	Qwen3.5 9B	91.6	.373±.127	3.0
10	RNJ-1 8B	34.6	.254±.037	14.5

5.1 LLM Screening

Table 4 presents the performance ranking of all 10 models, sorted by mean best AOCC across 5 seeds. The API models lead. Gemini 3 Pro achieves the highest mean best AOCC of 0.885 ± 0.011, while Gemini 3 Flash follows at 0.862 with the lowest failure rate of any model at 2.0%. The convergence curves in Figure 2b show that Gemini models improve steadily throughout the full 100-evaluation budget, whereas most open-weight models plateau early or stagnate after initial gains.

Among open-weight models, performance does not scale monotonically with parameter count. Qwen3.5 27B achieves 0.817, outperforming the larger OLMo 3 32B at 0.690. Granite 4, the smallest model at 3B parameters, reaches 0.569 and outperforms both Qwen3.5 9B at 0.373 and RNJ-1 8B at 0.254. Failure rates span 2.0% for Gemini 3 Flash to 91.6% for Qwen3.5 9B. Wall-clock time also varies substantially. The two models served via vLLM, Granite 4 at 0.6 h/run and Qwen3.5 9B at 3.0 h/run, benefit from continuous batching that allows concurrent seed evaluation. The largest open-weight models, Qwen3.5 27B and OLMo 3 32B, require approximately 39 hours per run under Ollama due to both model size and serial request queuing.

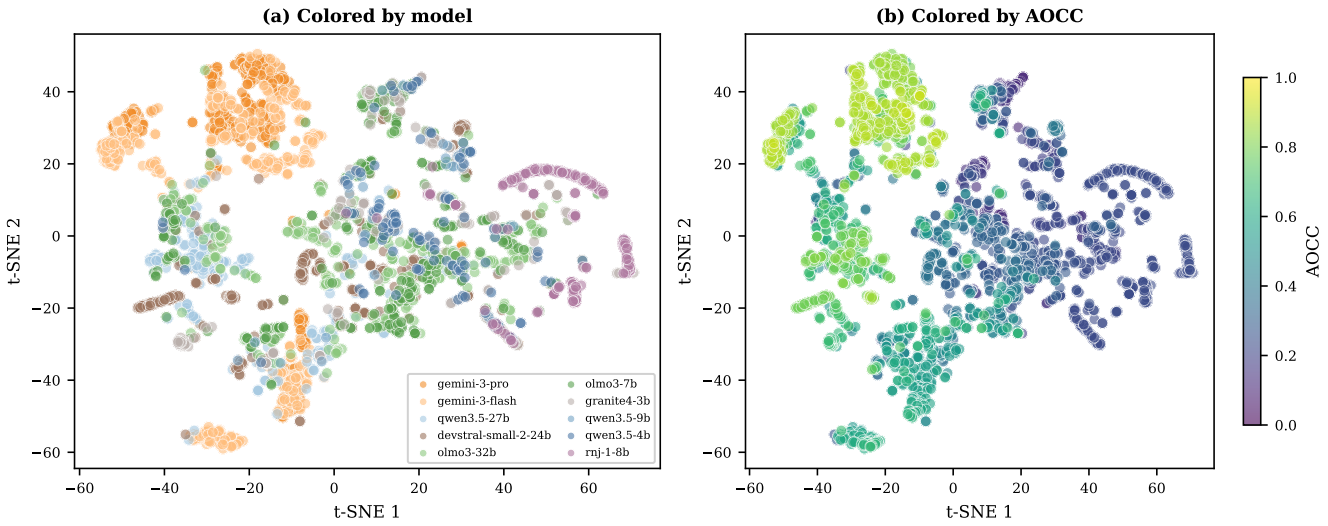

Figure 3: t-SNE projection of the behavioural feature space for all 3,071 valid algorithms. (a) Points coloured by model reveal distinct clustering. Each LLM occupies a characteristic region, with some models forming elongated trajectories indicating incremental refinement. (b) Points coloured by AOCC show that high-performing algorithms (yellow) concentrate in specific regions, while low-performing ones (purple) spread diffusely.

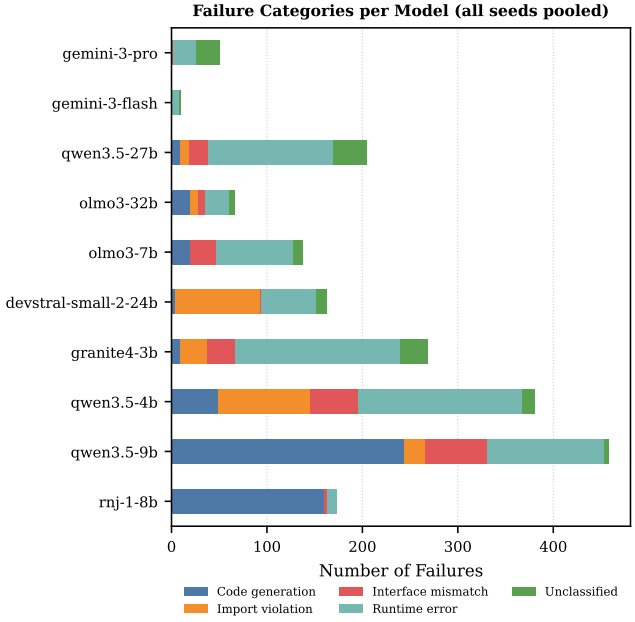

Figure 4: Failure mode distribution by model, classified by replaying each failed candidate through the BLADE compile, instantiate, and trial-call pipeline.

5.1.1 Failure Mode Analysis

Across all 10 models and 5 seeds, 5,000 candidates were generated in total, of which 3,071 (61.4%) produced valid fitness scores. To classify the remaining failures we replay every failed candidate's source through the same gate that BLADE applies at runtime. First we parse the file, check that imports stay within the allowed `numpy`-only environment, instantiate the algorithm class with `(budget=100, dim=2)`, and invoke it once on BBOB function 11 (Discus, $d = 2$), with a 10-second per-candidate budget. The result lands in one of five categories. These are code generation (no class is produced, or the file does not parse), import violation (the code imports a disallowed package), interface mismatch (`__init__(budget, dim)` or `__call__(func)` signature mismatch), runtime error (compiles and instantiates, but raises during the trial call), and unclassified.

The five categories are distributed unevenly across models. Runtime errors are the most common failure mode at 42.1% of the 1,914 classified events, and they dominate every model that successfully clears the structural gates (Granite-3B, Qwen3.5-4B, Qwen3.5-27B, OLMo-7B, OLMo-32B, and Gemini-3 Pro). Code-generation failures (26.9%) concentrate on two models. RNJ-1 fails this way in 160 of its 173 failures (92.5%), never producing a Python class at all, and Qwen3.5-9B contributes 244 further code-generation events. Over three quarters of all code generation failures come from a this pair of models. Import violations (13.3%) are concentrated in Qwen3.5-4B (96 events) and Devstral-24B (89). Interface mismatches (10.7%) spread across the smaller Qwen and Granite models, where `__init__` or `__call__` signatures drift from the framework contract. Unclassified failures (7.1%) concentrate on Gemini-3 Pro (25 of its 51 failures) and Qwen3.5-27B (35 of 205). These candidates fail on specific MA-BBOB instances or dimensions rather than along any of the four structural axes. The two Gemini models retain the lowest overall failure rates, with their failure breakdowns differing in shape. Pro is dominated by runtime and instance-specific failures, whereas Flash fails only ten times in 500 candidates and never on a structural axis.

5.1.2 Behavioural Space

Figure 3 projects all valid candidates into a two-dimensional behavioural space via t-SNE on the 32 standardised features. Algorithms from the same model cluster together, with the Gemini models occupying the upper-left quadrant, which panel (b) reveals as the high-AOCC region (predominantly yellow-green). Several models produce elongated, chain-like trajectories

rather than round clusters. For example, on the right side RNJ-1 forms distinct arcs that trace the $(1+1)$-ES refinement path through behavioural space. Some models fragment into multiple sub-clusters, corresponding to different seeds exploring distinct algorithmic strategies. The highest-performing algorithms from open-weight models (Qwen3.5 27B, OLMo 32B) partially overlap with the Gemini cluster. Weaker models (RNJ-1, Qwen3.5 4B) are confined to peripheral, low-AOCC regions with no overlap.

5.1.3 Model Selection

Gemini 3 Flash is selected for later stages based on its combination of competitive AOCC (0.862), lowest failure rate (2.0%), and fast inference (3.9 h/run). Its vanilla performance across 5 seeds (0.862 ± 0.108) serves as the baseline for all Stage 3 comparisons.

5.2 Feature Analysis and Selection

The Stage 1 dataset contains 3,071 valid candidates. Each is accompanied by a 32-dimensional behavioural feature vector computed from its evaluation trajectory. This section determines which features are most predictive of algorithm quality and selects a tractable subset for the feedback experiments.

5.2.1 Feature–AOCC Correlations

Figure 5 shows the Spearman rank correlation between each of the 32 behavioural features and AOCC. Given the sample size, all 32 correlations are statistically significant at $p < 0.001$ after Bonferroni correction, so significance is not a useful discriminator. We report the magnitude and direction of ρ.

The five strongest associations are `avg_improvement` (-0.782), `success_rate` ($+0.766$), `step_size_autocorrelation` ($+0.766$), `improvement_spatial_correlation` ($+0.757$), and `intensification_ratio` ($+0.725$). Three of these five are introduced to LLaMEA-generated metaheuristic analysis in this thesis.

Examining correlations by feature category, the exploitation features show the strongest positive associations. High `intensification_ratio` ($+0.725$) and low `avg_distance_to_best` (-0.691) both indicate search focused near the best-known solution. The convergence features reinforce this picture. Low `avg_improvement` (-0.782) and high `fitness_plateau_fraction` ($+0.685$). Among the step-size and movement features, `step_size_autocorrelation` ($+0.766$) leads the category. The early/late dynamics features show that `x_spread_early` (-0.599) correlates negatively with AOCC. The exploration features show moderate negative correlations ($|\rho| \approx 0.4$–0.6). The weakest associations appear among `directional_persistence` ($+0.139$) and `fitness_lempel_ziv_complexity` ($+0.152$).

5.2.2 Predictive Power

A Random Forest regressor with 200 trees is trained on all 32 features to predict AOCC. The model achieves an out-of-bag R^2 of 0.962 and a mean absolute error of 0.024 AOCC units. The features are not independent pre-

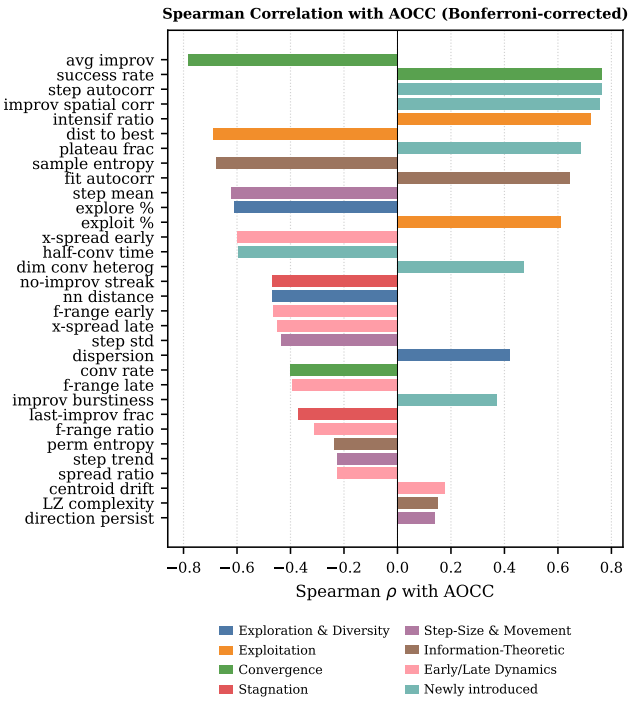

Figure 5: Spearman rank correlation ρ between each behavioural feature and AOCC across all 3,071 valid candidates, sorted by $|\rho|$ and coloured by feature category. All features significant at $p < 0.001$ after Bonferroni correction.

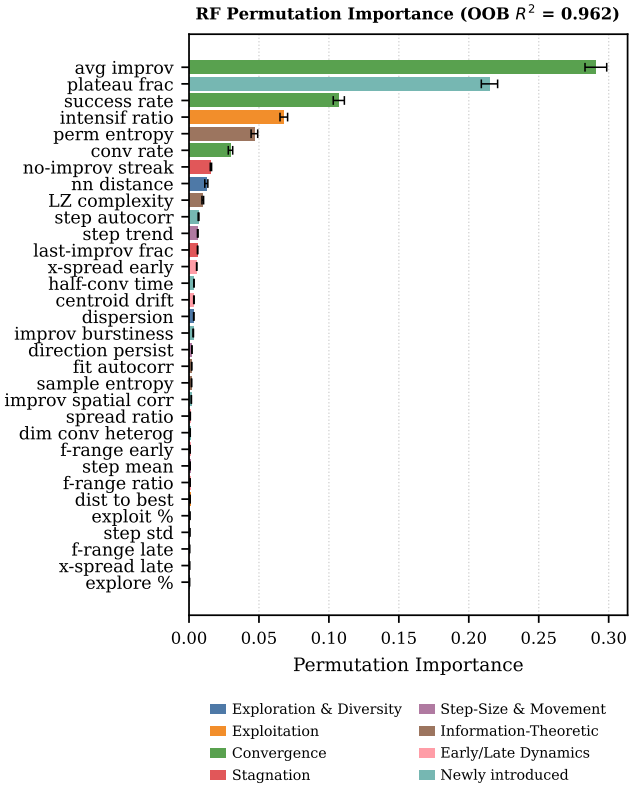

Figure 6: Random Forest permutation importance for all 32 features, sorted by importance. Error bars show standard deviation across 10 permutation repeats.

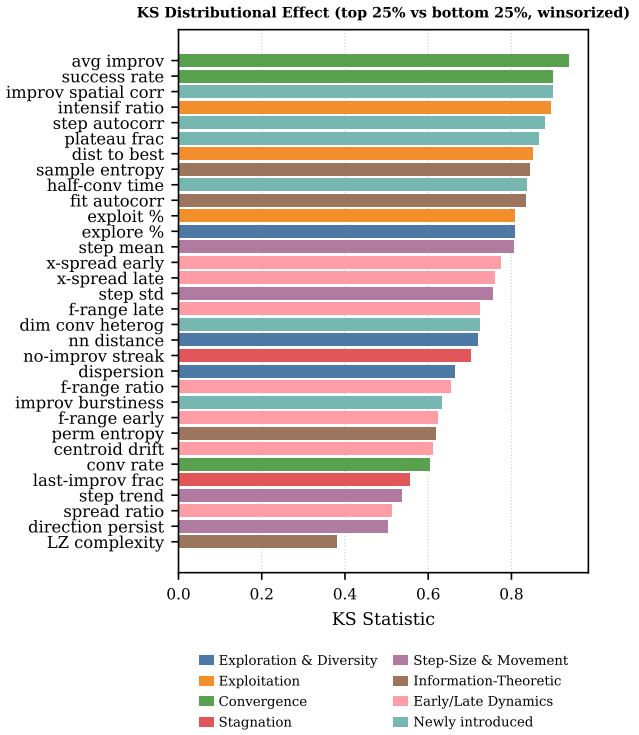

Figure 7: KS distributional effect size between top-25% and bottom-25% algorithms by AOCC, sorted by KS statistic. Feature values are winsorized at the 1st/99th percentile before testing.

dictors but complementary summaries of the underlying search dynamics that determine AOCC, which accounts for the tight fit.

Figure 6 shows the permutation importance ranking. `avg_improvement` (0.291) and `fitness_plateau_fraction` (0.215) account for the majority of predictive power, far exceeding the third-ranked `success_rate` (0.107). `step_size_autocorrelation`, which ranks third by Spearman $|\rho|$, drops to rank 10 by importance. The RF ranking elevates `average_convergence_rate` (rank 6) and `fitness_permutation_entropy` (rank 5), features with moderate Spearman correlations but stronger non-linear predictive power.

5.2.3 Distributional Separation

Figure 7 reports the KS statistic for each feature across the 3,071 valid candidates. The KS ranking largely agrees with Spearman on which features matter most. Several features that ranked moderately by Spearman show very high KS values. `avg_exploitation_pct` and `step_size_mean` both achieve KS statistics above 0.8 despite more moderate Spearman $|\rho|$ values. At the bottom of the KS ranking, `fitness_lempel_ziv_complexity` and `spread_ratio` show weak distributional separation.

5.2.4 Consensus Ranking and Selection

Figure 8 shows the effect of the redundancy-pruning threshold τ on the selected subset across the swept range $\tau \in [0.50, 1.00]$ in 0.05 steps. Top-ranked features remain stable across the full range, while features lower in the Borda order swap in and out with small threshold

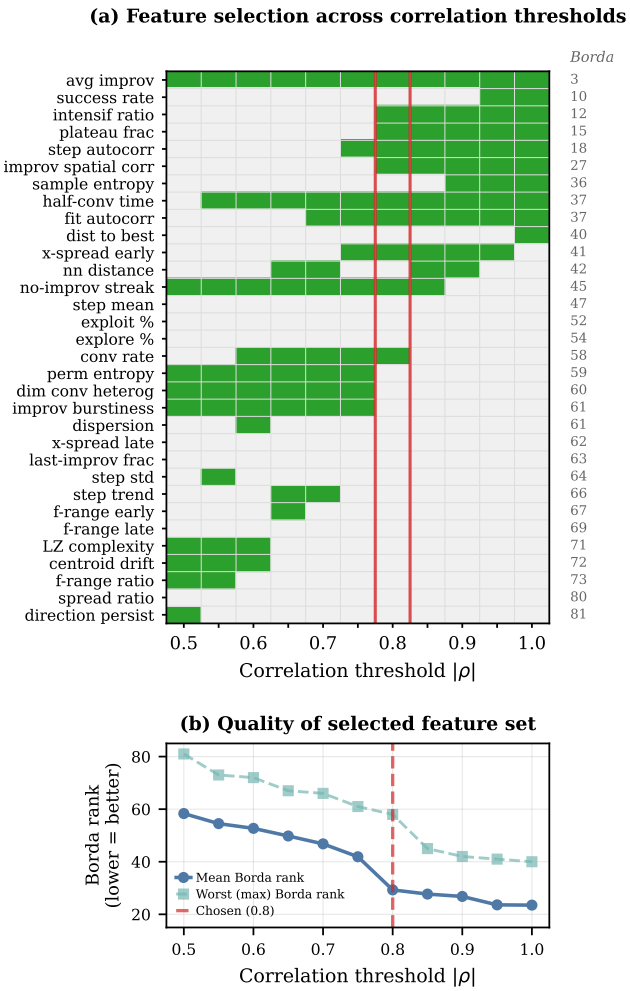

Figure 8: Sensitivity of the 10-feature subset to the redundancy-pruning threshold τ. (a) Presence heatmap: which of the 32 candidates is selected at each $\tau \in [0.50, 1.00]$ in 0.05 steps, sorted top-to-bottom by Borda rank. The chosen $\tau = 0.80$ column is highlighted in red. (b) Mean and worst Borda rank of the selected subset as τ varies; the dashed line marks the chosen value, which sits at the elbow of the mean curve.

changes. The mean Borda rank shows its largest single-step improvement at the 0.75 to 0.80 transition and plateaus thereafter, motivating $\tau = 0.80$ as the elbow of the quality–threshold curve.

Table 5 presents all 32 features with their per-method values and ranks, sorted by Borda consensus score. The final set comprises three BLADE features and seven newly introduced features from this thesis. `intensification_ratio` alone makes four other metrics redundant, including two BLADE exploration/exploitation percentages and `avg_distance_to_best`. Several features with strong individual method rankings fail the consensus. For example, `fitness_permutation_entropy` ranks 5th by RF importance but only 27th by Spearman.

5.3 Behavioural Feedback

The Stage 3 dataset contains 14,127 valid candidates produced by Gemini 3 Flash across 29 feedback conditions (10 features × 3 framings, minus the comparative version

Table 5: All 32 behavioural features ranked by Borda consensus. For each method, both the value and rank are shown. The Status column indicates whether a feature was selected for feedback experiments, pruned for redundancy (with the correlated selected feature noted), or not selected.

			Spearman		RF Importance		KS		
Borda	**Feature**	**Abbrev.**	ρ	**R**	**Imp**	**R**	**Stat**	**R**	**Status**
3	avg_improvement	avg improv	−.782	1	.291	1	.938	1	**Selected**
7	success_rate	success rate	+.766	2	.107	3	.899	2	Pruned ($\|r\|$=.92, avg_impr)
13	intensification_ratio	intensif ratio	+.725	5	.068	4	.894	4	**Selected**
15	fitness_plateau_frac	plateau frac	+.685	7	.215	2	.867	6	**Selected**
18	step_size_autocorr	step autocorr	+.766	3	.007	10	.881	5	**Selected**
28	impr_spatial_corr	improv spatial corr	+.757	4	.002	21	.899	3	**Selected**
36	fitness_sample_entropy	sample entropy	−.680	8	.002	20	.845	8	Pruned ($\|r\|$=.87, plateau_frac)
37	half_convergence_time	half-conv time	−.597	14	.004	14	.840	9	**Selected**
38	fitness_autocorrelation	fit autocorr	+.645	9	.002	19	.835	10	**Selected**
40	avg_distance_to_best	dist to best	−.691	6	.001	27	.852	7	Pruned ($\|r\|$=.96, intens_ratio)
40	x_spread_early	x-spread early	−.599	13	.006	13	.775	14	**Selected**
43	longest_no_improv_streak	no-improv streak	−.469	16	.016	7	.703	20	**Selected**
44	avg_nn_distance	nn distance	−.469	17	.012	8	.720	19	Pruned ($\|r\|$=.82, x_spread)
48	step_size_mean	step mean	−.622	10	.001	25	.806	13	Pruned ($\|r\|$=.83, intens_ratio)
52	avg_exploitation_pct	exploit %	+.612	12	.001	28	.808	12	Pruned ($\|r\|$=.87, intens_ratio)
55	avg_exploration_pct	explore %	−.612	11	.000	32	.808	12	Pruned ($\|r\|$=.87, intens_ratio)
55	dim_conv_heterogeneity	dim conv heterog	+.471	15	.001	23	.727	17	**Selected**
55	average_convergence_rate	conv rate	−.400	22	.030	6	.604	27	—
57	fitness_perm_entropy	perm entropy	−.238	27	.047	5	.622	25	—
58	dispersion	dispersion	+.421	21	.003	16	.664	21	—
64	improvement_burstiness	improv burstiness	+.370	24	.003	17	.635	23	—
65	last_improvement_frac	last-improv frac	−.370	25	.006	12	.555	28	—
65	step_size_std	step std	−.437	20	.001	29	.755	16	—
65	x_spread_late	x-spread late	−.450	19	.000	31	.759	15	—
66	f_range_early	f-range early	−.466	18	.001	24	.624	24	—
68	step_size_trend	step trend	−.228	28	.006	11	.538	29	—
71	f_range_late	f-range late	−.394	23	.000	30	.724	18	—
71	centroid_drift	centroid drift	+.177	30	.004	15	.612	26	—
72	fitness_lempel_ziv	LZ complexity	+.152	31	.010	9	.380	32	—
74	f_range_ratio	f-range ratio	−.313	26	.001	26	.654	22	—
81	spread_ratio	spread ratio	−.224	29	.001	22	.511	30	—
81	directional_persistence	direction persist	+.139	32	.002	18	.502	31	—

Borda = sum of ranks (lower is better). R = rank within method. Values are rounded to 3 decimal places. Where features appear tied, ranks reflect the full-precision ordering. Features above the mid-rule were evaluated by the greedy selection and those below were not considered (10 slots already filled). Pruned features show $|r|$ with the correlated selected feature.

Table 6: Behavioural steering results per feature. For each format: mean best AOCC (± std) and median behavioural feature value (± std). The final columns show whether directional/comparative shifted the feature toward the advised direction relative to neutral.

Feature	Adv.	Neutral AOCC	Neutral Feat.	Directional AOCC	Directional Feat.	Comparative AOCC	Comparative Feat.	Steered D	Steered C
avg_impr	↓	**.883±.054**	.094±.198	.866±.060	.105±.206	.860±.076	.104±.237	×	×
intens_ratio	↑	**.907±.021**	.813±.213	.806±.044	.783±.125	.721±.046	.852±.189	×	✓
plateau_frac	↑	**.891±.005**	.448±.210	.874±.055	.364±.177	.811±.077	.553±.210	×	✓
step_autocorr	↑	**.810±.111**	.897±.111	.797±.075	.863±.123	.761±.103	.877±.135	×	×
impr_spatial	↑	.853±.089	.649±.091	**.857±.044**	.630±.107	.786±.100	.629±.092	×	×
half_conv	↓	.847±.069	.003±.007	**.862±.083**	.001±.004	.758±.060	.003±.022	✓	×
fit_autocorr	↑	**.840±.074**	.718±.092	.814±.068	.753±.128	.749±.121	.790±.140	✓	✓
x_spread	↓	.844±.067	1.55±.562	.793±.108	0.55±.451	**.852±.092**	0.74±.443	✓	✓
longest_str	↓	.789±.109	849±1762	**.860±.059**	1028±2351	—		×	—
dim_conv_het	↑	**.905±.006**	.050±.052	.788±.068	.007±.056	.779±.071	.034±.168	×	×
Steering accuracy								3/10	4/9

Adv. = direction associated with higher AOCC from Spearman analysis (↑/↓). AOCC = mean best across 5 seeds. Feat. = median behavioural feature value across all valid candidates. ✓/× = shifted toward/away from advised direction relative to neutral.

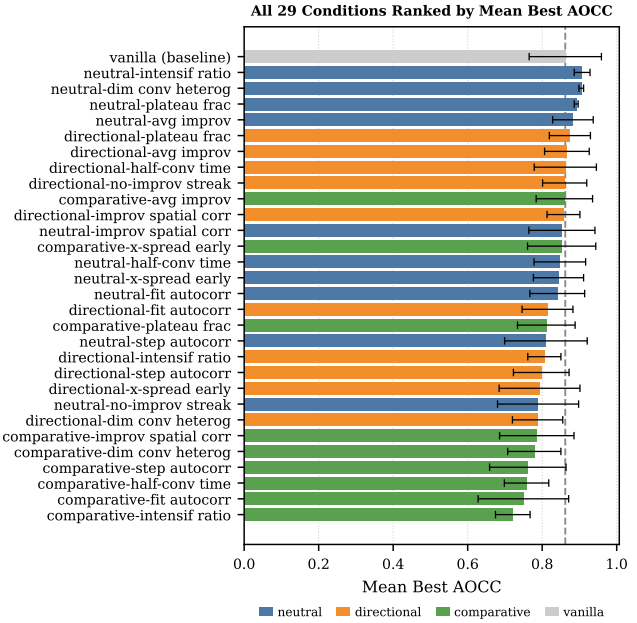

Figure 9: All 29 conditions ranked by mean best AOCC (± std, 5 seeds). Grey bar and dashed line show the vanilla baseline (0.862).

of `longest_no_improvement_streak` excluded for non-monotonicity) with 5 independent seeds per condition. This section compares the three feedback formats against each other and against the vanilla AOCC-only baseline, and reports how each format shifts feature values, convergence dynamics, and the structural complexity of generated code.

5.3.1 Format Ranking

Figure 9 ranks all 29 conditions by mean best AOCC alongside the vanilla baseline. The top four conditions are all neutral format. To test whether the format ordering reflects a difference rather than sampling noise, we apply the Kruskal–Wallis test [62], a non-parametric alternative to one-way ANOVA that compares the rank distributions of independent groups without assuming normality. The test treats the per-(format, feature) mean AOCCs as independent samples within each format. The test returns $H = 9.25$, $p = 0.010$ for differences among formats, with neutral achieving a mean AOCC of 0.855, directional 0.827, and comparative 0.784. The AOCC columns in Table 6 show this pattern per feature, with the best format highlighted in bold. Neutral produces the best AOCC for 6 of 10 features, directional for 3, and comparative for 1. Neutral conditions also have the highest failure rate at 3.5% compared to 2.6% for directional and 1.5% for comparative.

Six conditions exceed the vanilla baseline AOCC of 0.862. Four are neutral (`intensification_ratio`, `dim_conv_heterogeneity`, `plateau_fraction`, `avg_improvement`) and two are directional (`plateau_fraction`, `avg_improvement`), with `neutral-intensification_ratio` at 0.907 ± 0.021 the best. None of these reach statistical significance against vanilla under Mann–Whitney U at the 5-seed budget.

When comparing the three feedback formats against each other for a single behavioural feature, two features reach within-feature statistical significance. For `intensification_ratio`, neutral (0.907) outperforms directional (0.806) and comparative (0.721) with complete pairwise separation across seeds (Cliff's $d = 1.0$ for every pairwise comparison, Kruskal–Wallis $p = 0.002$). The same holds for `dim_conv_heterogeneity`, where neutral (0.905) outperforms directional (0.788) and comparative (0.779) with no overlap between neutral and either prescriptive format ($d = 1.0$ pairwise, $p = 0.009$).

5.3.2 Behavioural Drift Between Stages

The Stage 3 dataset is produced exclusively by Gemini 3 Flash, while the Stage 1 dataset spans all ten screened models. Before interpreting the format effects further, we check whether the directional advice carried into Stage 3 (calibrated on Stage 1) still describes the AOCC relationship inside the Gemini 3 Flash sub-population. We recompute Spearman correlations between each feature and AOCC on the Stage 3 dataset and re-derive the bottom-25% and top-10% reference values used by comparative feedback. Table 7 reports both stages side by side. The Stage 3 mean AOCC is higher (0.646 vs 0.484 in Stage 1) and the performance range is narrower.

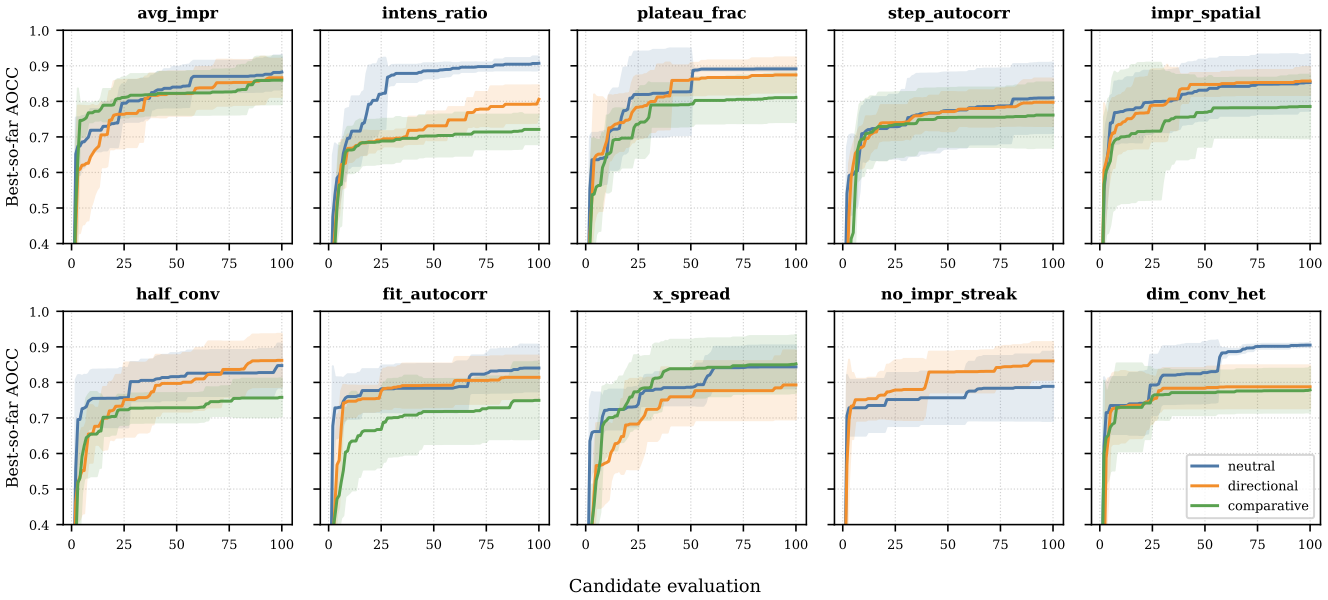

Figure 10: Best-so-far AOCC over 100 evaluations for each feature, split by feedback format (mean ± 1 std, 5 seeds). Neutral (blue) typically converges fastest and highest.

Table 7: Feature–AOCC relationship shift from Stage 1 (all models, n=3,071) to Stage 3 (Gemini Flash only, n=14,127). Tier columns show median feature values for bottom-25% and top-10% algorithms by AOCC in each stage.

	Stage 1 (all models)				Stage 3 (Gemini Flash)				
Feature	ρ	**Bot25**	**Top10**	**Gap**	ρ	**Bot25**	**Top10**	**Gap**	**Dir.**
avg_impr	−.782	1.752	0.093	1.659	−.184	0.136	0.078	0.059	same
intens_ratio	+.725	0.000	0.879	0.879	+.475	0.714	0.873	0.160	same
plateau_frac	+.689	0.000	0.597	0.597	+.578	0.183	0.597	0.414	same
step_autocorr	+.764	0.130	0.910	0.780	+.468	0.822	0.903	0.082	same
impr_spatial	+.757	0.087	0.682	0.595	+.446	0.615	0.689	0.073	same
half_conv	−.611	0.010	0.001	0.009	−.640	0.003	0.001	0.002	same
fit_autocorr	+.650	0.003	0.766	0.764	+.138	0.720	0.738	0.018	same
x_spread	−.570	2.889	0.621	2.268	−.410	1.608	0.578	1.030	same
longest_str	−.427	6282	5543	739	+.495	1499	6018	4520	**flip**
dim_conv_het	+.424	0.001	0.081	0.080	+.167	0.049	0.089	0.040	same

Bot25/Top10 = median feature value among bottom-25%/top-10% algorithms by AOCC. Gap = |Top10 − Bot25|. Dir. = whether correlation direction is preserved.

9 of 10 features maintain the same correlation direction.

The correlation magnitudes weaken substantially. For example, `avg_improvement` drops from $\rho = -0.782$ in Stage 1 to -0.184 in Stage 3. Only one feature, `half_convergence_time`, strengthens its correlation in the same direction (-0.611 to -0.640). `longest_no_improvement_streak` (-0.427 to $+0.495$) increases in magnitude but reverses sign, consistent with the non-monotonic relationship that motivated its exclusion from comparative feedback.

Table 7 also shows that the top-10% reference values stay relatively stable between stages, while the bottom-25% values shift dramatically. The top-10% in Stage 1 was already populated mostly by Gemini Flash and Pro candidates, so narrowing to Gemini Flash alone in Stage 3 changes its values very little. The bottom-25% in Stage 1 was populated mostly by the weaker open-weight models, whereas Stage 3's bottom-25% is still produced by Gemini Flash. The bottom-25% of `step_size_autocorrelation` rises from 0.130 to 0.822, `fitness_autocorrelation` from 0.003 to 0.720, and `intensification_ratio` from 0.000 to 0.714, while their top-10% values move by less than 0.01. The gap between the two tiers, which determines the normalisation scale for comparative feedback, collapses by an order of magnitude for several features (`fitness_autocorrelation` from 0.764 to 0.018, `step_size_autocorrelation` from 0.780 to 0.082, and `avg_improvement` from 1.659 to 0.059).

This collapse falls disproportionately on the comparative format. Neutral and directional feedback do not use the tier values, so neither is affected by the narrowing. Comparative feedback is built entirely on them. Its distance signal $d = (v_{\text{ref}} - v)/(v_{\text{ref}} - v_{\text{bot25}})$ is normalised by exactly the top-to-bottom gap that shrinks by an order of magnitude here. When that denominator collapses, the same absolute difference is rescaled into a far larger normalised distance, so the format's tier-based wording is driven by a scale calibrated on a model pool the deployment model no longer resembles.

Table 8: Structural metrics for the 14,127 valid Stage 3 algorithms, parsed via Python's `ast` module. Columns are grouped into per-format mean ± std (N = neutral, D = directional, C = comparative; per-format $n \approx 4{,}700$), between-format tests (Kruskal–Wallis p across the three formats and three pairwise Cliff's d effect sizes), and the Spearman rank correlation between the metric and AOCC together with its p-value. Rows are sorted by $d_{N\text{–}C}$ descending.

	Mean by format			Between-format tests				AOCC correlation	
Metric	**N**	**D**	**C**	p_{KW}	$d_{N\text{–}C}$	$d_{N\text{–}D}$	$d_{D\text{–}C}$	p_ρ	ρ_{AOCC}
Total AST nodes	1404±401	1262±383	1145±409	2.4×10^{-229}	+0.38	+0.19	+0.21	$<10^{-300}$	+0.48
NumPy calls	26.6±11.2	23.4±10.0	19.5±9.5	2.3×10^{-223}	+0.38	+0.15	+0.24	$<10^{-300}$	+0.60
Assignments	67.5±15.9	63.3±16.2	58.0±15.6	4.9×10^{-192}	+0.35	+0.15	+0.21	$<10^{-300}$	+0.40
`try/except` blocks	0.5±0.6	0.3±0.6	0.2±0.4	5.1×10^{-200}	+0.28	+0.20	+0.08	$<10^{-300}$	+0.48
`return` stmts	2.3±1.2	2.0±1.2	1.9±1.3	2.2×10^{-123}	+0.26	+0.16	+0.12	7.0×10^{-4}	+0.03
`while` loops	3.0±1.4	2.8±1.3	2.4±1.2	2.5×10^{-90}	+0.22	+0.06	+0.17	2.8×10^{-73}	−0.15
`for` loops	2.4±1.3	2.2±1.4	2.0±1.5	3.8×10^{-63}	+0.20	+0.08	+0.11	1.3×10^{-12}	+0.06
Max loop depth	3.0±0.6	2.9±0.7	2.7±0.7	2.5×10^{-81}	+0.19	+0.03	+0.15	2.8×10^{-86}	+0.16
Numeric constants	70.2±32.7	63.1±26.6	58.6±24.1	1.3×10^{-36}	+0.15	+0.08	+0.09	$<10^{-300}$	+0.66
Function definitions	2.6±0.8	2.4±0.8	2.4±0.8	8.0×10^{-35}	+0.12	+0.09	+0.04	7.2×10^{-172}	+0.23
`if` branches	13.6±4.3	13.8±4.1	13.5±4.6	5.0×10^{-14}	+0.05	−0.05	+0.09	3.8×10^{-81}	+0.16
Augmented assigns	5.9±3.8	6.4±3.3	6.6±3.1	1.7×10^{-29}	−0.13	−0.09	−0.05	$<10^{-300}$	+0.47

p_{KW} is the Kruskal–Wallis omnibus test that the three format distributions differ at all. Cliff's $d \in [-1, +1]$ measures pairwise rank dominance. $|d| < 0.147$ negligible, 0.147–0.33 small, 0.33–0.474 medium, ≥ 0.474 large. ρ_{AOCC} is the Spearman rank correlation between the metric and AOCC across all 14,127 algorithms. p_ρ is its two-sided significance. Values reported as $<10^{-300}$ underflow scipy's floating-point precision. NumPy calls = `ast.Call` nodes whose target is a method on the `np` namespace. numeric constants count integer and float literals.

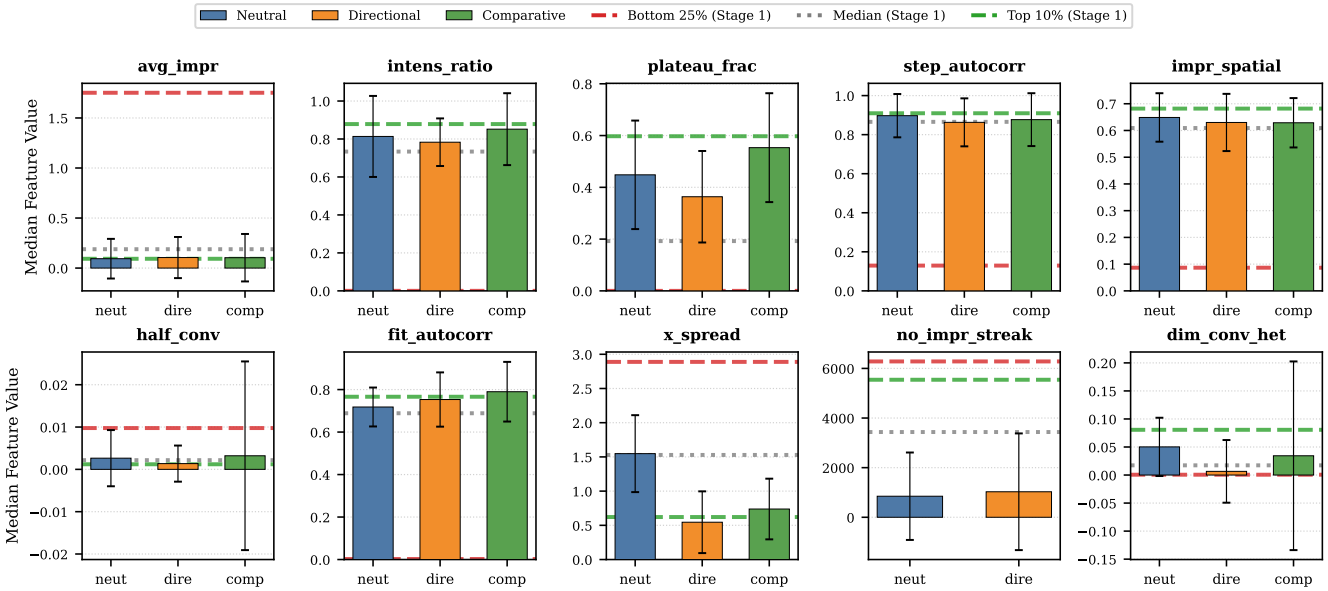

Figure 11: Median behavioural feature value by feedback format (bars) compared against Stage 1 AOCC performance tiers (dashed lines). Bottom-25% (red), median (grey), and top-10% (green). Reference values are computed from all Stage 1 candidates, independent of the feedback experiments.

5.3.3 Convergence Dynamics

Figure 10 shows the best-so-far convergence curves for each feature, split by format. The two features with the strongest format effects, `intensification_ratio` and `dim_conv_heterogeneity` (the same pair that reach within-feature statistical significance above), show a clear separation early in the run that widens throughout. The features where directional performs well (`half_convergence_time`, `longest_streak`) show the formats tracking closely. For features such as `avg_improvement`, all three formats converge to similar levels with high variance.

5.3.4 Behaviour Steering

In addition to per-feature AOCC values, Table 6 shows the median behavioural feature value produced by each format and whether prescriptive feedback steered the feature in the advised direction relative to neutral. Figure 11 visualises these values against the Stage 1 AOCC performance tiers. Across all features, directional feedback steers correctly for 3 of 10 cases and comparative for 4 of 9, giving an overall steering accuracy of 37%. Two features (`fitness_autocorrelation` and `x_spread_early`) are steered in the correct direction by both guidance formats but achieve lower AOCC than neutral. For `x_spread_early`, directional feedback reduces the median spread from 1.55 to 0.55, well past

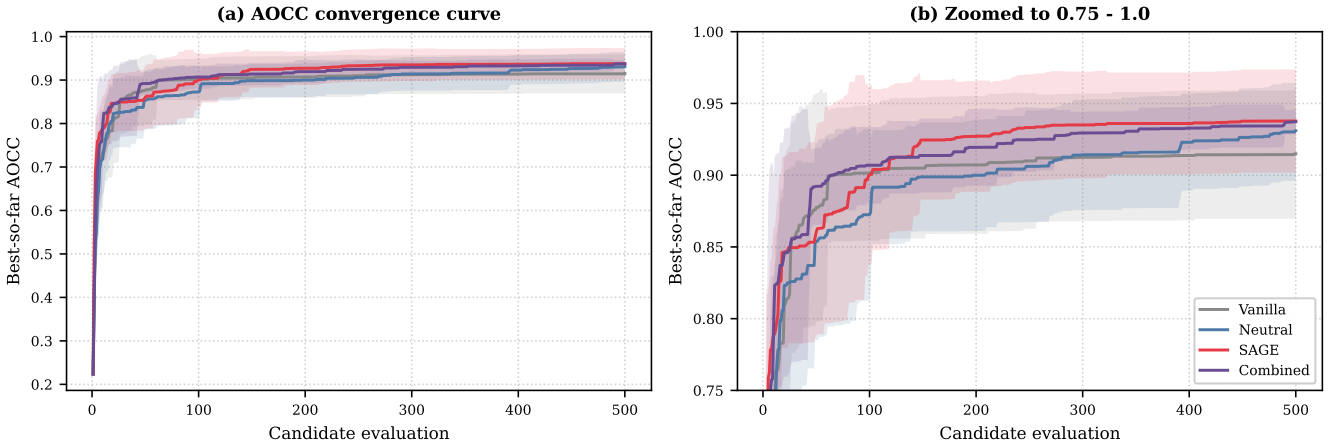

Figure 12: Best-so-far AOCC across the 500-candidate budget, mean ± 1 std across 10 seeds. (a) Full AOCC range. (b) Zoomed to $[0.75, 1.0]$ to resolve between-condition differences.

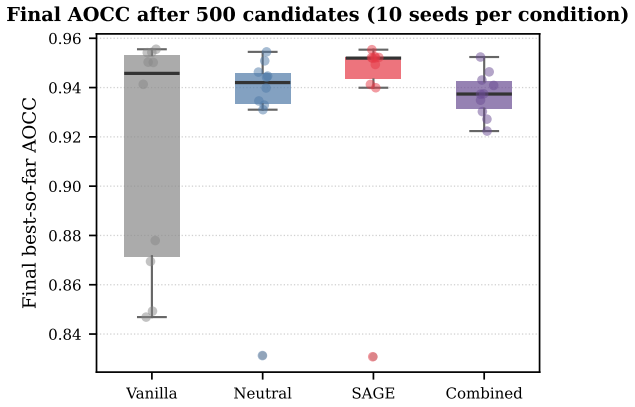

Figure 13: Final best-so-far AOCC after 500 candidates, 10 seeds per condition. Box shows quartiles, whisker shows 1.5×IQR, individual seeds plotted as black dots.

the top-10% reference of 0.621, yet AOCC drops from neutral's 0.844 to 0.793. The same pattern appears for `fitness_autocorrelation`, where both prescriptive formats increase the value as advised but achieve lower AOCC than neutral.

5.3.5 Code Complexity

We measure structural complexity by parsing every valid Stage 3 algorithm with Python's `ast` module and extracting twelve metrics covering raw size, arithmetic content, and control-flow structure (Table 8). Neutral feedback produces more complex code than the prescriptive formats. We test for between-format differences using Kruskal–Wallis on each metric. At ∼ 4,700 algorithms per format, the test rejects the null at $p < 10^{-13}$ on every row, so effect sizes carry the comparison. The largest pairwise gap on each metric is between neutral and comparative. Cliff's d (d_{N-C}) is at least +0.20 (small effect or larger) on six of twelve metrics: total AST nodes, NumPy calls, assignments, `try/except` blocks, `return` statements, and `while` loops. Directional sits between the other two formats, with d_{N-D} and d_{D-C} both smaller and of the same sign as d_{N-C} except on the `if`-branches metric.

We test the metric–AOCC association with Spearman rank correlation across the full set. All twelve correlations are statistically significant. Total AST nodes ($\rho_{\text{AOCC}} = +0.48$), NumPy calls (+0.60), and numeric constants (+0.66) show the strongest correlations with AOCC.

Table 9: Final best-so-far AOCC summary statistics across 10 seeds per condition, and per-condition failure rate over all 5,000 candidates per condition.

Condition	Mean	Median	Std	Fail%
Vanilla	0.9149	0.9502	0.0475	16.8
Neutral	0.9310	0.9442	0.0359	13.9
SAGE	0.9378	0.9520	0.0379	21.0
Combined	0.9372	0.9375	0.0090	17.6

5.4 Full-Benchmark Comparison

The Stage 4 dataset contains 16,538 valid candidates produced by Gemini 3 Flash across four feedback conditions (vanilla, neutral-behavioural, SAGE, combined) at 500 candidates per seed and 10 seeds per condition, evaluated on 20 disjoint MA-BBOB functions with five instance seeds each. This section reports the head-to-head comparison across aggregate condition performance, failure-rate dynamics, and the identity of the best-found algorithm in each condition.

5.4.1 Aggregate Condition Performance

Figure 13 and Table 9 show the final best-so-far AOCC distribution across 10 seeds per condition. Sorted by mean, vanilla finishes lowest at 0.915 ± 0.048, followed by neutral at 0.931 ± 0.036, combined at 0.937 ± 0.009, and SAGE at 0.938 ± 0.038. The seed-to-seed standard deviation under combined is roughly five times smaller than under vanilla. At the per-seed level, vanilla has four seeds finishing below 0.88, neutral one, SAGE one, and combined none.

Figure 12 traces the best-so-far AOCC across the 500-candidate budget. Within the first 100 generations, vanilla, SAGE, and combined all reach a mean of roughly 0.90, with neutral lagging at 0.87. Vanilla's gen-100 mean of 0.901 already exceeds the 0.862 that Gemini 3 Flash reached after 100 candidates on the Stage 1 instance

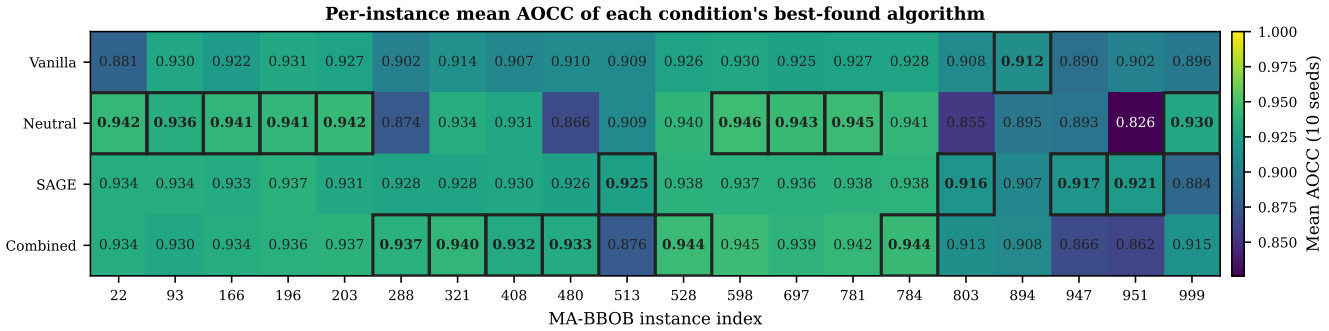

Figure 14: Per-instance mean AOCC of each condition's best-found algorithm across the 20 MA-BBOB functions in the Stage 4 set. Each cell averages 10 seeds; the leading condition for each instance is outlined and bolded.

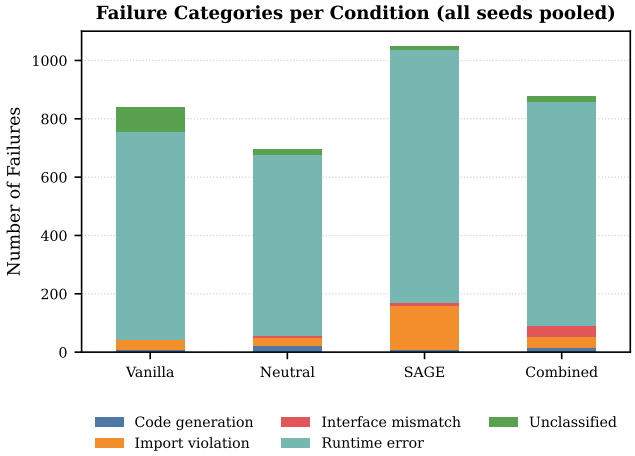

Figure 15: Stage 4 failure-mode breakdown by condition, classified through the same replay procedure used in Figure 4.

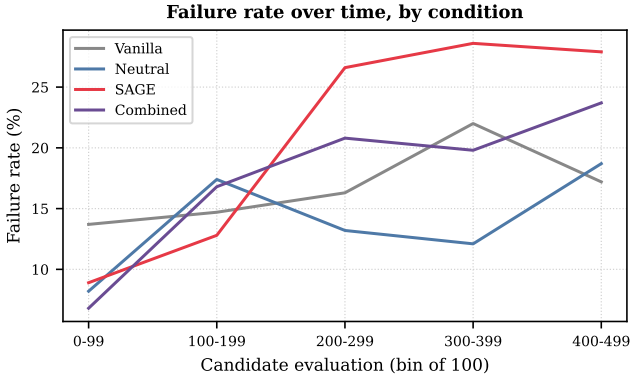

Figure 16: Per-condition failure rate as a function of candidate generation index, binned in 100-candidate windows ($[0, 100)$, $[100, 200)$, ...). The x-axis is the generation-index bin; the y-axis is the fraction of candidates in that bin which fail the BLADE validation pipeline (Figure 15).

subset. From generation 100 onwards the four trajectories diverge. Vanilla plateaus near 0.91, while neutral, SAGE, and combined continue rising, with neutral nearly matching SAGE and combined by generation 500.

Figure 14 reports the same comparison at the per-instance level. It shows the mean AOCC of each condition's best-found algorithm on each of the 20 MA-BBOB functions. Neutral takes the leading mean on 9 instances, combined on 6, SAGE on 4, and vanilla on 1.

A Welch one-sided t-test of each feedback condition against vanilla returns $p = 0.088$ for combined, $p = 0.126$ for SAGE, and $p = 0.203$ for neutral. Under Holm correction across the three pairs no test reaches $\alpha = 0.05$, so the mean lift over vanilla, while consistent in direction across all three feedback conditions, does not reach significance at this seed budget. The variance reduction noted above is the more decisive finding. To test whether combined's roughly fivefold tightening of the seed-to-seed spread relative to vanilla is real rather than a sampling artefact, we apply Levene's test for equal variance between the two conditions. It returns $T = 5.57$, $p = 0.030$, the only Stage 4 contrast to clear $\alpha = 0.05$.

5.4.2 Failure-Rate Analysis

The failure rates in Stage 4 are higher than in earlier stages despite no changes to the candidate-extraction code or the BLADE evaluation pipeline. Vanilla fails on 16.8% of candidates, neutral on 13.9%, SAGE on 21.0%, and combined on 17.6%, against the 2.0% rate reported for Gemini 3 Flash in Section 5.1.1 and the 1.5–3.5% range across formats in Section 5.3.1.

Figure 15 shows the per-condition failure breakdown across the five categories. Runtime errors dominate every condition (between 82.7% and 89.1% of failures). The conditions differ on the lower-frequency modes. SAGE produces 14.4% import-violation failures, more than three times any other condition. The failed candidates repeatedly import libraries beyond the allowed `numpy`-only environment (`scipy.optimize`, `sklearn`). Combined drives interface mismatches to 4.3%, four times any other condition. The failed candidates alter the prescribed `__init__(budget, dim)` or `__call__(func)` signature in ways that break the framework contract. Vanilla has the highest unclassified rate at 10.1%, with most of these failures tied to specific MA-BBOB instances or dimensions rather than to any structural defect.

Figure 16 bins the same failures by generation. Over the first 100 candidates the failure rate is 6.8% for combined, 8.2% for neutral, 8.9% for SAGE, and 13.7% for vanilla. Beyond the 100-candidate mark every condition fails substantially more often. Vanilla settles between 14.7% and 22.0%, neutral between 12.1% and 18.7%,

Table 10: Best-found algorithm in each Stage 4 condition (highest final best-so-far AOCC across the 10 seeds). Static code metrics computed on the generated source: lines of code (excluding blanks and comments) and maximum block-nesting depth (counted from indentation).

Cond.	Algorithm name	Seed	AOCC	LoC	Nest
vanilla	`IGA_CMA`	3	0.9555	106	6
neutral	`BIPOP_MA_CMA_ES`	0	0.9545	115	6
sage	`CMA_EES`	5	0.9554	95	6
combined	`OM_CMA_EIS`	7	0.9524	95	6

Table 11: Statement and branch coverage of the four Stage-4 winners under MA-BBOB evaluation (90 runs each: $d \in \{5, 10, 20\}$, 15 instances, 2 seeds, full $2{,}000d$ budget). Rows, top to bottom: vanilla (`IGA_CMA`), neutral (`BIPOP_MA_CMA_ES`), SAGE (`CMA_EES`), combined (`OM_CMA_EIS`). "dead" is the count of unreached statement lines; "except" is how many of the algorithm's eigendecomposition `except` handlers executed at least once, out of its total.

Algorithm	stmts %	branch %	dead	except
`IGA_CMA`	99.0	92.9	1	0/1
`BIPOP_MA_CMA_ES`	97.2	94.7	3	0/1
`CMA_EES`	97.8	95.8	2	0/1
`OM_CMA_EIS`	94.6	82.1	5	0/1

SAGE climbs from 12.8% in 100–199 to 27.9% in 400–499, and combined from 16.8% to 23.7%.

5.4.3 Best-Found Algorithm Identity

Table 10 reports the best-found algorithm per condition, taken as the candidate with the highest final best-so-far AOCC across the 10 seeds. The four winners (`IGA_CMA`, `BIPOP_MA_CMA_ES`, `CMA_EES`, and `OM_CMA_EIS`) span only a 0.0031 AOCC range, are each between 95 and 115 lines of code, and have a maximum nesting depth of 6. The full source for all four is available in the project repository.[2]

All four winners are CMA-ES [7] variants and contain the canonical CMA-ES mechanisms in full. These are log-decreasing recombination weights, the standard learning-rate constants c_c, c_s, c_1, c_μ, damps, and χ_n verbatim, both evolution paths p_c and p_s, the Heaviside h_σ indicator, the rank-1 plus rank-μ covariance update, and CSA step-size control. None of the four replaces, simplifies, or fundamentally rederives this core. Each instead decorates it with two or three heuristic add-ons.

Three add-ons recur. Mirrored sampling [63] (antithetic $\pm z$ pairs to halve sampling noise on the mean shift) appears in all four winners. QR-decomposed orthogonal sampling also appears in all four, applied to a `randn` matrix and rescaled by $\sqrt{n}$ (low-dim-only in `BIPOP_MA_CMA_ES`). Active CMA with negative rank-μ weights [64] appears in two of the four. `IGA_CMA` uses a flat -0.8 negative coefficient, while `BIPOP_MA_CMA_ES` caps it at $0.5\,c_\mu$. The conditions diverge on the remaining heuristic interventions.

- **`IGA_CMA` (vanilla)** wraps the inner CMA in an IPOP-style [65] restart loop that doubles the population size on collapse. A DE-style mean-injection step overrides x_{mean} on stagnation. Biased re-initialisation draws the new mean near the running best with probability 0.3.
- **`BIPOP_MA_CMA_ES` (neutral)** scaffolds BIPOP [66], alternating between large- and small-population regimes on each restart. The budget accounting that normally selects the next regime is omitted. An unused `success_history` list and the absence of any local-search procedure leave the "MA" (memetic) prefix decorative.
- **`CMA_EES` (SAGE)** adds edge-biased radial scaling on the orthogonal samples (a Beta$(0.5, 0.5)$ mixed into the χ-scale). Modulo-reflective boundary handling bounces out-of-bounds points back into the interior. Restart fires only on near-total stagnation and is suppressed past 80% of the FE budget.
- **`OM_CMA_EIS` (combined)** injects a momentum term, formed as an exponential moving average of recent mean-shift directions, into each sample. There is no proper restart, only a $1.1\times$ sigma kick when the within-generation fitness range collapses.

Beyond the deliberate add-ons, all four winners carry passages that contribute nothing to the search. To separate code that never runs from code that runs without effect, each winner was traced under branch-level coverage on a sample of MA-BBOB instances at the full $2{,}000d$ FE budget. Table 11 reports the fraction of statements and branches reached. Between 94.6% and 99.0% of statements execute, and the unreached remainder is concentrated in defensive guards that never fire.

The clearest case is the eigendecomposition guard common to all four. Each wraps the covariance eigendecomposition in a `try/except` (a bare `except` in `IGA_CMA` and `CMA_EES`, `LinAlgError/ValueError` in `BIPOP_MA_CMA_ES` and `OM_CMA_EIS`) and in every winner that handler executes zero times. The fallback each guard protects (identity covariance in `CMA_EES` and `OM_CMA_EIS`, a parameter reset in `BIPOP_MA_CMA_ES`, a loop `break` in `IGA_CMA`) is therefore dead under normal evaluation rather than a live safeguard. The remaining unreached lines are guards of the same kind. A non-positive-eigenvalue clamp in `OM_CMA_EIS`, whose 82.1% branch coverage is the lowest of the four, and an odd-population-size correction whose branch the chosen dimensions never reach, since the population formula yields an even size at every $d \in \{5, 10, 20\}$.

A second class of inert content executes but changes nothing downstream, and so lies outside what coverage can measure. `BIPOP_MA_CMA_ES` carries an unused `success_history` list and, lacking any local-search procedure, leaves its "MA" (memetic) prefix decorative, while the budget accounting that normally drives BIPOP's regime selection is omitted. Several comments advertise techniques the source never implements. A named "noise-resilient gradient estimation" that is never coded, a "PSA" that is just standard CSA, and a "Constraint-based scaling" applied to a flat heuristic cap. The LLM-coined prefixes "EES", "EIS", and "IGA" are

[2] https://github.com/Maxwe11h/thesis-experiments/tree/main/docs/stage4_winners

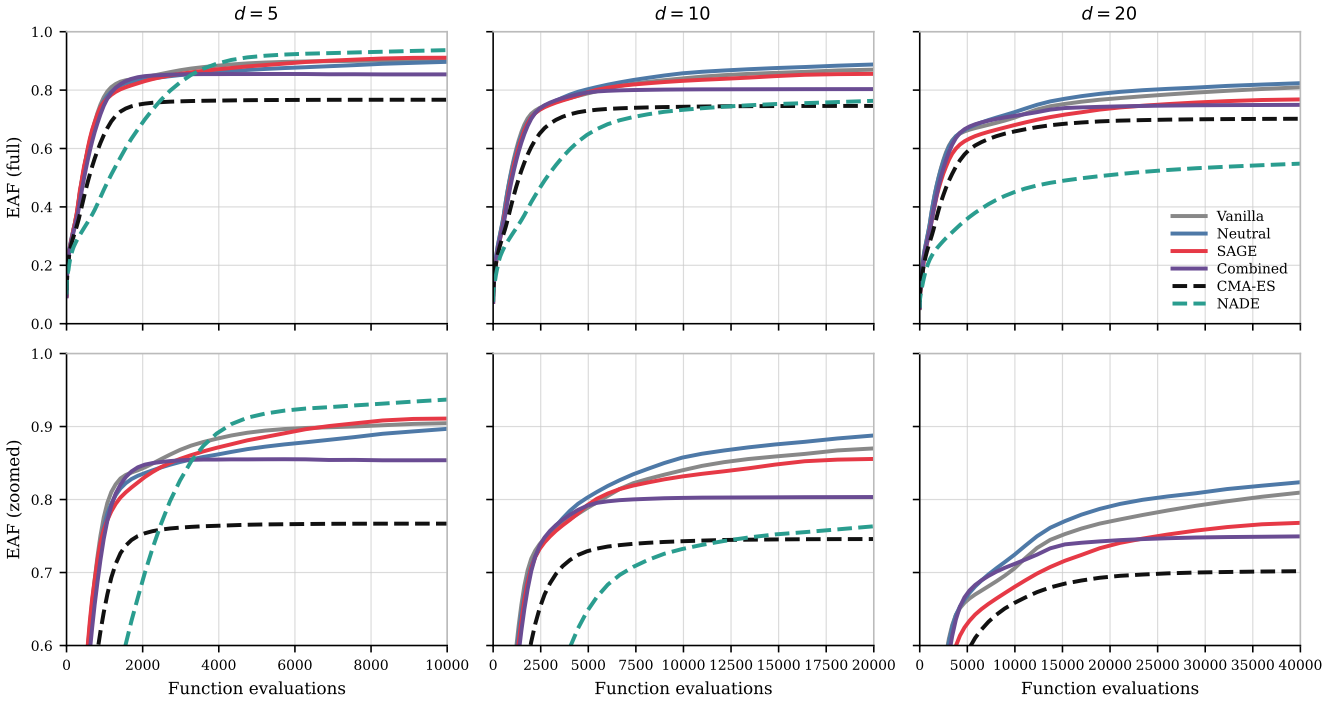

Figure 17: Empirical attainment function on the 1,000-instance MA-BBOB pool at $d \in \{5, 10, 20\}$, the fraction of (run, target) pairs attained by a given budget, over 5,000 runs and 51 log-spaced targets in $[10^{-8}, 10^{2}]$. Top row full range, bottom row zoomed to $[0.6, 1]$. Dashed lines mark the two external baselines (CMA-ES, NADE).

defined nowhere in the source. Because these passages are executed-but-inconsequential rather than unreached, they remain static observations that the coverage trace neither confirms nor refutes. The same holds for a correctness inconsistency in the same vein. The mirrored-and-orthogonalised sampling in `BIPOP_MA_CMA_ES`, `CMA_EES`, and `OM_CMA_EIS` breaks the Gaussian assumption that CSA's expected step-norm χ_n relies on, an inconsistency the code runs past without raising.

Taken together, the trace shows that the winners' defensive scaffolding is largely dead, and the misnamed or unused passages, though executed, do no work. In both cases the algorithms' performance rests on the canonical CMA-ES core and the two or three live add-ons, not on the surface elaboration.

5.4.4 Full-Suite Generalisation

To position the four winners against a broader benchmark, the best-found algorithm from each condition (Table 10) is re-evaluated against two external baselines: a generic CMA-ES [7] and `NeighborhoodAdaptiveDE` (NADE) [61], the winner of the 2025 MA-BBOB competition. NADE is itself an LLaMEA-generated algorithm, produced with Gemini 2.0 Flash, and, like every algorithm in this thesis, was designed and tuned only on 5-dimensional MA-BBOB (Section 4.2). The higher dimensionalities therefore measure how each algorithm generalises beyond its design regime. Each of the six algorithms runs on all 1,000 MA-BBOB functions across 5 instance seeds and three dimensionalities $d \in \{5, 10, 20\}$, under a $2{,}000d$ FE budget per run. The total compute is approximately 2.1×10^9 FEs across the six algorithms.

Table 12 reports both metrics per (algorithm, dimensionality). All four LLM-generated winners beat the generic CMA-ES baseline on both AOCC and final attainment at every dimensionality, and all degrade gracefully as dimension grows. At $d = 5$ the four winners are closely grouped on AOCC. `IGA_CMA`, generated by vanilla feedback, leads at 0.848. NADE has the lowest AOCC of the four-plus-NADE set at this dimension on the anytime metric (0.810, mid-pack) but the highest final attainment of any algorithm in the table (0.937), a gap of 0.13 between its two metrics. At $d = 10$ the order changes. `BIPOP_MA_CMA_ES`, generated by neutral behavioural feedback, leads the full table on both metrics (0.810 AOCC, 0.887 final attainment). NADE's AOCC falls to 0.663, below the generic CMA-ES (0.701), and its final attainment (0.763) drops below all four winners. At $d = 20$ `BIPOP_MA_CMA_ES` again leads the table on

Table 12: Full-suite generalisation on the 1,000-instance MA-BBOB pool (5,000 runs per cell). AOCC is the anytime score; final attainment is the terminal normalised coverage (the right edge of Figure 17). Best per column in bold. The lower block is the two external baselines (no LLM in the loop): a generic CMA-ES and NADE, the 2025 MA-BBOB competition winner.

	AOCC			Final attainment		
Algorithm	d=5	d=10	d=20	d=5	d=10	d=20
`IGA_CMA`	**0.848**	0.798	0.730	0.904	0.870	0.809
`BIPOP_MA_CMA_ES`	0.833	**0.810**	**0.745**	0.897	**0.887**	**0.823**
`CMA_EES`	0.843	0.789	0.696	0.911	0.855	0.767
`OM_CMA_EIS`	0.815	0.763	0.704	0.853	0.803	0.748
CMA-ES [7]	0.728	0.701	0.650	0.766	0.745	0.701
NADE [61]	0.810	0.663	0.471	**0.937**	0.763	0.546

both metrics (0.745 / 0.823), with `IGA_CMA` second on AOCC (0.730); `OM_CMA_EIS` and `CMA_EES` fall to 0.704 and 0.696. NADE is last on both metrics (0.471 AOCC, 0.546 final attainment).

6 Discussion

This chapter interprets the four-stage results in the light of the research questions posed in Section 1, identifies the limitations that bound the conclusions, and outlines directions for future work. Section 6.1 collects the main findings on whether trajectory-based behavioural feedback guides LLMs to design better metaheuristics. Section 6.2 addresses the statistical power that the seed budget afforded the head-to-head comparison. Section 6.3 examines the CMA-ES bias exhibited by Gemini 3 Flash and the consequences of that bias for behavioural feedback specifically. Section 6.4 contrasts proximal and distal feedback channels and explains why prescriptive behavioural feedback steered the LLM in only a minority of cases. Section 6.5 examines why successful steering, when it occurred, often failed to translate into higher AOCC. Section 6.6 interprets the AOCC-complexity correlation as partly an artefact of how the loop accumulates code over generations. Section 6.7 considers why every Stage 4 winner is a CMA-ES variant and what would be needed to push the loop beyond canonical recombination.

6.1 Main Findings

The headline result of Stage 4 is that feedback raises the floor of LLaMEA performance rather than its ceiling. Combined feedback's seed-to-seed standard deviation is roughly five times smaller than vanilla's, and a Levene test against vanilla returns $p = 0.030$, the only Stage 4 contrast to clear $\alpha = 0.05$. The mean-AOCC lift over vanilla is directionally consistent across all three feedback conditions but does not reach significance under Holm-corrected one-sided Welch tests at the same 10-seed budget. Two further patterns reinforce the variance interpretation. At the per-instance level, the three feedback conditions split 19 of 20 leaderboard wins between them while vanilla wins only one. At the per-seed level, vanilla produces four runs that finish below 0.88 against two such runs across the 30 feedback-augmented seeds combined. Both describe a tighter performance distribution under feedback rather than a higher mean.

Together these signals suggest that additional feedback channels do not so much expand what a strong LLM can produce as suppress its worst failures. Vanilla AOCC-only feedback occasionally lets the LLaMEA loop settle into a poor-performing region of code space from which it does not recover within the 500-candidate budget. Behavioural and structural feedback both appear to help the loop avoid those attractors. The fact that combining the two channels does not produce additive gains over either alone, despite both individually outperforming vanilla, is consistent with their carrying overlapping rather than complementary information about what makes an algorithm good. Behavioural features describe the search dynamics that the code produces, structural features describe the code itself, and on this benchmark both appear to push the LLM toward a similar, more elaborate, CMA-like region of design space.

The overall assessment is therefore qualified but positive. Behavioural feedback does guide LLMs to design better optimisation algorithms, but the improvement manifests primarily as reduced failure-mode variance and reliable per-instance wins rather than as a large mean shift. The full-suite re-evaluation in Section 5.4.4 reinforces this. All four LLM-generated winners outperform a CMA-ES baseline on the full 1,000-instance MA-BBOB pool across $d \in \{5, 10, 20\}$, with `BIPOP_MA_CMA_ES` from the neutral condition leading at $d = 10$ and $d = 20$ and remaining within 0.015 AOCC of the leader at $d = 5$.

A notable limitation on these conclusions is the rule used to select the five behavioural features carried into Stage 4. We took the top five Stage 3 conditions by mean AOCC, which guaranteed that the chosen features had performed well under at least one feedback framing but inherited the calibration mismatch documented in Section 5.3.2. An alternative selection rule, such as the top-k by Stage 3 Spearman correlation recomputed on Gemini 3 Flash output, or by a joint AOCC-and-Spearman rank, would test the sensitivity of Stage 4 to the feature-selection criterion. We did not run this ablation. The four-condition × ten-seed Stage 4 design already exhausted our Gemini API budget, and the comparison would have required at least one further full Stage 4 run. We flag this as the most defensible single-experiment extension of the present work.

6.2 Statistical Power Analysis

The single largest limitation of this thesis is the resolution at which the seed budget allowed condition-vs-condition differences to be detected. Stage 3 ran 5 seeds per condition across 29 conditions. Stage 4 ran 10 seeds per condition across 4 conditions.

With only 5 seeds per group, the Mann–Whitney U test has $\binom{10}{5} = 252$ possible rank interleavings, so the smallest two-tailed p-value it can ever produce is $2/252 \approx 0.008$. That minimum requires complete separation between the two groups, meaning every value in one group exceeds every value in the other. The expected min and max of $n = 5$ samples from a normal distribution sit about $\pm 1.16\,\sigma$ from the mean[3]. Complete separation thus requires the higher group's expected minimum $(\mu_A - 1.16\sigma_A)$ to exceed the lower group's expected maximum $(\mu_B + 1.16\sigma_B)$, equivalent to a mean gap of at least $1.16(\sigma_A + \sigma_B)$. For vanilla (seed-to-seed AOCC standard deviation $\sigma_v = 0.108$) versus the best-performing Stage 3 condition `neutral-intensification_ratio` ($\sigma_b = 0.021$) that threshold is about 0.15, well above the observed +0.045 gap. The test therefore cannot rule out chance even though the direction is consistent.

Doubling to 10 seeds per condition, as in the Stage 4

[3]For n i.i.d. samples from $\mathcal{N}(0, \sigma^2)$, the expected largest order statistic is $\sigma\, E[Z_{(n)}]$, where $Z_{(n)} = \max(Z_1, \ldots, Z_n)$ for standard normal Z_i. With $E[Z_{(n)}] = \int_{-\infty}^{\infty} x \cdot n\,\Phi(x)^{n-1}\phi(x)\,\mathrm{d}x$ and no closed form, numerical integration at $n = 5$ gives $E[Z_{(5)}] \approx 1.163$. The expected smallest is -1.163 by symmetry.

setup, raises the rank interleavings to $\binom{20}{10} = 184{,}756$. The expected min and max of $n = 10$ normal samples sit about $\pm 1.54\,\sigma$ from the mean (the $n = 10$ analogue of the 1.16 value above), so the complete-separation threshold becomes $1.54(\sigma_A + \sigma_B)$. Even for the tightest Stage 4 pair, vanilla ($\sigma_v = 0.048$) versus combined ($\sigma_c = 0.009$) (Table 9), this works out to a threshold of roughly 0.087, and the comparisons against neutral and SAGE land closer to 0.13. The observed Stage 4 mean lift of feedback over vanilla is only ~ 0.02 AOCC, still well below all three thresholds, which is why the Holm-corrected Welch tests of Section 5.4.1 fall short of significance even though the direction is consistent across all three feedback conditions.

The seed budget itself was constrained almost entirely by Gemini API cost. The 29-condition feature-by-format sweep at 100 candidates per seed already amounted to 14,500 LLaMEA iterations, and doubling to 10 seeds would have cost 29,000 iterations, larger than the entire Stage 4 budget reserved for the head-to-head comparison. Stage 4 itself consumed 20,000 candidate algorithms across four conditions and a sizeable Gemini API spend. Within the available budget the trade-off was either many feature-format conditions at low resolution (Stage 3) or fewer conditions at higher resolution (Stage 4), and we chose to spend our seeds where the design comparison was most critical.

A natural alternative is to step away from closed-API models entirely. Stage 1 showed that medium-sized open-weight models in the 24–32B parameter range, particularly Qwen3.5 27B and Devstral Small 2 (24B), can generate high-quality metaheuristics, although they trail the Gemini family in both AOCC and reliability. The next obvious step is to evaluate larger open-weight models in the 70–200B parameter range. Such models can be served on university hardware at marginal per-token cost, which would lift the seed budget by an order of magnitude or more and bring the kind of mean differences observed in Stage 4 within statistical reach. They would also permit the full 10×5 seed-by-condition factorial design that the Gemini budget could not support.

6.3 Model Bias

A qualiative finding of Stage 4 is that Gemini 3 Flash exhibits a strong CMA-ES family bias regardless of the feedback string it receives. Every best-found algorithm across all four conditions, vanilla, neutral, SAGE, and combined, is a CMA-ES variant, and Section 5.4.3 catalogues both the canonical mechanisms each winner inherits verbatim and the surface decorations layered on top. A reasonable interpretation is that Gemini 3 Flash, has a strong prior over which optimisation-algorithm code patterns are "correct", and that prior is anchored on the CMA-ES implementation. CMA-ES is a well-established strong algorithm on this class of benchmark, so the bias is empirically defensible. Converging on a CMA-ES variant is not a failure mode of the LLM but a constraint on what current feedback can induce it to produce.

One specific consequence of this bias is that it blunted Stage 3 comparative feedback by collapsing the reference values that format relies on. The Stage 1 dataset, drawn from ten heterogeneous models, populated a wide region of behavioural space and produced reference values for top-10% and bottom-25% performers that spanned several decades for some features. When Stage 3 deployed only Gemini 3 Flash, the behavioural distribution narrowed sharply. As Section 5.3.2 documents, the bottom-25% values for several features rose by an order of magnitude or more between stages while the top-10% values barely moved, because the Stage 1 bottom was populated mostly by weaker open-weight models whereas the Stage 3 bottom is still produced by Gemini Flash. The gap between the two tiers, which determines the normalisation scale for comparative feedback, therefore collapsed by an order of magnitude for several features, and the distance signals the comparative format sends became small relative to the actual feature dispersion in the Stage 3 generation. The implication for future work is that behavioural-feedback calibration must be model-specific. A target-model calibration pass, screening the chosen LLM on its own to derive feature ranks and tier boundaries before the feedback experiment, would tighten the signal that prescriptive feedback attempts to convey.

6.4 Proximal versus Distal Feedback Channels

Section 5.3.4 reported that prescriptive feedback moved the median feature value in the advised direction relative to neutral in only 7 of 19 cases, a steering accuracy of 37%. This is low enough to demand explanation, particularly because LLaMEA-SAGE [10] demonstrates that current-generation LLMs reliably comply with prescriptive natural-language advice when the target is something they can directly modify. SAGE issues prescriptive advice on structural features such as cyclomatic complexity and AST shape, and the resulting algorithms do shift on those axes with measurable AOCC gains. The contrast with our 37% steering rate on behavioural features therefore points at a structural difference rather than a model limitation.

The relevant difference is the causal channel. SAGE's structural advice is proximal. The LLM edits the abstract syntax tree, and the next AST is a function of those edits, so compliance is mechanical. Behavioural advice is distal. The LLM still edits code, but the behavioural feature emerges from running that code on a benchmark, and the mapping from a code change to a behavioural delta is uncertain even to a human author. An LLM that genuinely tries to comply with prescriptive behavioural advice must therefore guess at this mapping, and on a non-trivial mutation that guess is unreliable. The 37% rate is what we should expect from a steering channel that is causally indirect.

This framing has a concrete implication for future feedback designs. Behavioural features are not in themselves unsteerable, but they cannot be steered by direct prescription the way structural features can. They must instead be steered through their structural antecedents. A prescription like "increase the intensification ratio"

asks the LLM to control a measurement it cannot directly write. Guidance like "narrow the sampling distribution around the incumbent during the second half of the budget" asks it to control structural code that, if executed, would produce the desired behavioural shift. Translating behavioural targets into structural prescriptions before issuing them is the natural next step, and would test whether the steering deficit observed here is a property of behavioural features per se or only of how they were prescribed.

6.5 Why Successful Steering Did Not Translate to AOCC

Even when prescriptive feedback successfully steered a behavioural feature, it often failed to lift AOCC. `x_spread_early` and `fitness_autocorrelation` are both steered correctly by both prescriptive formats yet achieve lower AOCC than neutral. The previous section explains why prescriptive formats often fail to act on the LLM. It does not explain why, when they do act, the result is no better and sometimes worse. Two explanations are plausible, and we suspect both contribute.

The first concerns the multivariate structure of the high-performance region. Good algorithms appear to be characterised by a combination of feature values rather than by any single feature in isolation. Pushing one feature toward its top-tier reference while leaving the others unconstrained can place a candidate in a region of behavioural space that no actual top-tier algorithm occupies, so single-feature optimisation can take the LLM off the high-performance manifold rather than along it. Future feedback designs should target behavioural profiles rather than individual features. One concrete instantiation would be to identify clusters of high-performing algorithms in behavioural space (the upper-left quadrant of Figure 3 is a candidate), report the candidate's nearest-cluster centroid as the target, and frame the advice as "move toward this configuration of features" rather than "increase this one feature."

The second concerns the granularity at which behavioural features are computed. Each feature reported in the feedback prompt is averaged across 50 MA-BBOB runs (10 functions $\times$ 5 instances) in Stage 3 and 100 runs in Stage 4. This compresses behaviour across landscapes that differ substantially in conditioning, modality, and separability into a single scalar. An algorithm may need a high intensification ratio on unimodal high-conditioning functions and a low one on multimodal weak-structure functions, and reporting the cross-instance average obscures that conditional structure. A natural refinement, again left to future work, is to report behavioural features per MA BBOB function and let the LLM see that its algorithm exploits well on separable functions but explores too aggressively on multimodal ones. Furthermore, behavioural feedback may need to be studied at the instance or group level before it can be aggregated effectively.

A third candidate cause, which we examined and reject, is calibration mismatch. The prescriptive reference values were derived from the heterogeneous Stage 1 model pool but deployed against Gemini 3 Flash alone, whose behavioural distribution is much narrower. The natural worry is that even successful steering pushes a candidate to a reference value that sits at the wrong scale for this deployment model, so the shift overshoots or lands in a region where the original AOCC-feature relationship no longer holds. Inspection of the recomputed Stage 3 tier values, however, shows that for most features the prescriptive targets sit close to where Gemini's own distribution places its top-tier candidates, so successful steering does not place candidates in a fundamentally different region of behavioural space. Calibration mismatch is a real issue of the prescriptive design and plausibly degrades the ability of comparative feedback to correctly steer an algorithm to a specific behavioural value. Once steering succeeds, the mismatch ceases to act. The candidate has been pulled to a reference value that sits close to where Gemini's own top-tier algorithms already sit, so the original calibration error has no remaining channel through which to depress AOCC. We initially suspected this explanation but reject it on closer examination.

The fact that neutral framing, which sidesteps the two contributing problems by simply reporting values without imposing an objective, consistently outperforms the prescriptive variants is consistent with this view. Neutral feedback adds informative context to the prompt without asking the LLM to optimise something it should not be optimising in isolation. The takeaway is that using the methodology of this thesis, describing the candidate's behaviour to the LLM produces better algorithms than instructing it how to change that behaviour, but we should not entirely throw away using target behavioural profiles as a way to guide an LLM to better performance.

6.6 Code Complexity

Section 5.3.5 showed that neutral feedback produces structurally more complex code than directional or comparative feedback, and that across the full Stage 3 population code complexity correlates positively with AOCC. Total AST nodes ($\rho = +0.48$), NumPy calls (+0.60), and numeric constants (+0.66) are among the strongest individual predictors of algorithm quality. This reproduces the central finding of LLaMEA-SAGE [10], that complexity-promoting structural feedback improves performance, and offers a candidate explanation for why neutral behavioural feedback works. It shifts the LLM toward writing more elaborate code on axes that empirically correlate with stronger optimisation behaviour.

The interpretation of this complexity deserves scrutiny. Two observations from Stage 4 suggest that the AOCC-complexity correlation is partly an artefact of how the LLaMEA loop accumulates code rather than a sign that complexity itself drives quality. The first is the catalogue of dead code in the Stage 4 winners reported in Section 5.4.3. The second is the failure-rate climb in Figure 16. SAGE's failure rate rises from 12.8% in candidates 100–199 to 27.9% in candidates 400–499, and combined climbs from 16.8% to 23.7% over the same window, even though the prompt template does not change across generations.

A single mechanism plausibly produces both. LLaMEA's mutation prompt instructs the LLM to "refine the strategy of the selected solution to improve it", with no mention of simplification, pruning, or trimming. Conditional on the LLM treating "refine" as an instruction to add rather than to rewrite, the body of code under mutation grows monotonically across generations. The dead-code accumulation is the visible content of that growth. The failure-rate climb is its statistical consequence. Each marginal mutation lands on a larger surface where any single error fails the whole candidate, suggesting that LLMs struggle to one-shot working improvements to complex code. Under this reading, the AOCC-complexity correlation does not show that more elaborate code is better. It shows that the loop produces more elaborate code over time, and that AOCC also rises over time, with the underlying cause being whatever the LLM is doing well rather than the elaboration itself.

Two future-work directions address how the loop accumulates code. The first is diagnostic. Whether each marginal LLaMEA mutation adds executed code or only decorative tokens can be answered by AST-diffing consecutive candidates and tracing which added subtrees are actually executed during evaluation. This would discriminate between meaningful and inert complexity at the per-mutation level and turn the dead-code observations into a quantitative signal. The second is corrective. Mutation acceptance could be gated on a measured behavioural delta from the parent rather than on AOCC alone, so that mutations which only relabel or restructure code without changing what it does are filtered before they accumulate. The mutation prompt itself could also be modified to explicitly reward trimming as well as additions, replacing "refine" with language that admits both directions of edit.

6.7 Algorithmic Novelty

The novelty of the generated metaheuristics is also in question. Every Stage 4 winner is a CMA-ES variant with two or three surface decorations layered over a textbook implementation, and the decorations themselves are standard variants from the CMA-ES literature: mirrored sampling [63], active CMA with negative rank-μ weights [64], and IPOP and BIPOP restart schemes [65, 66]. The momentum-style mean-shift accumulation in `OM_CMA_EIS` is not a named published variant but is conceptually close to the cumulation already present in the canonical evolution path p_c. The LLM is recombining well-known mechanisms rather than discovering new ones, and the LLaMEA loop in its current form functions more as a sophisticated hyperparameter search across known mechanisms than as a design-space explorer in the sense Hoos [5] originally envisaged.

Some of this concentration reflects how hard genuine algorithmic novelty is. Decades of expert research have produced relatively few mechanisms that materially advance the state of the art on continuous black-box optimisation and many published metaheuristics rediscover algorithmic components under new names [22]. The bar for novelty in this domain is not "produce code that has not been written before" but "produce a mechanism that improves on a strong, mature baseline", and that bar is high enough that current LLMs may not be able to clear it regardless of how the loop is configured. The recombination behaviour observed here is consistent with what one would expect from a system that knows the canonical mechanisms well and lacks the capacity to invent new ones.

With that said, two structural features of the LLaMEA loop may reinforce a model's bias, though we did not run the ablations that would be needed to establish either as a cause, and we offer them as conjectures rather than findings. The first is the selection rule. The (1+1) elitist replacement keeps an offspring only if it matches or beats the parent on AOCC, which could in principle penalise a novel design that is not already competitive with the incumbent on the generation it appears, since the loop offers no protected niche in which such a design might mature. This intuition should be read with care. Unlike a conventional hillclimber, the LLaMEA mutation operator is not confined to local moves. Under the 10% explore prompt the LLM can propose a wholesale new design rather than a refinement of the incumbent. So the loop does have a channel through which a genuinely different algorithm can enter, and whether elitist selection nonetheless suppresses novelty on net is an empirical question we leave open. The second is the in-context example. Each mutation prompt shows the LLM the current best algorithm as the example to work from, which after the first few generations is already a CMA-ES variant. Whether this anchors subsequent generations to the CMA-ES family, or merely co-occurs with a prior the LLM would hold regardless, likewise cannot be settled without an ablation that varies the example independently of the incumbent.

We also note that the (1+1) configuration studied here is not the only option. More recent LLaMEA variants adopt larger populations with comma-selection (μ, λ) strategies and explicit diversity-preservation mechanisms [12], which relax precisely the selection pressure conjectured above and would be the natural setting in which to test whether it suppresses novelty.

We identify two complementary directions could push the loop beyond canonical recombination. The first is to expand the LLM's effective context with a corpus of past LLaMEA-generated algorithms via retrieval-augmented generation [67, 68]. The 16,538 valid Stage 4 algorithm-feature pairs produced by this thesis, together with the tens of thousands more available on public LLaMEA Zenodo repositories, form a candidate corpus. Each mutation prompt would include a handful of high-AOCC exemplars deliberately drawn from non-CMA regions of design space, biasing the LLM away from its default basin without forbidding it outright. The second is more direct. The mutation prompt could forbid canonical mechanisms by name and require the LLM to justify novelty against a list of known variants. This trades the implicit nudge of retrieval-augmented exemplars for an explicit constraint, at the cost of more brittle prompting.

7 Conclusion

This thesis has investigated whether trajectory-based behavioural features can serve as a feedback signal for LLM-driven metaheuristic design, advancing four contributions toward an answer. An extended behavioural feature vocabulary of 32 features was defined (Section 3) and reduced via a tri-criterion consensus procedure to a tractable subset of ten predictive, non-redundant descriptors (Section 5.2). A multi-model screening of ten LLMs identified Gemini 3 Flash as the strongest combination of generation quality, reliability, and inference cost (Section 5.1), with the secondary finding that medium-sized open-weight models in the 24–32B parameter range can produce viable metaheuristics. A three-way comparison of neutral, directional, and comparative feedback formats found that neutral framing, simply reporting the behavioural observation without interpretation, consistently outperformed the prescriptive variants across 29 feature-by-format conditions (Section 5.3). And a head-to-head comparison on a 20-function MA-BBOB set demonstrated that all three feedback-augmented conditions, neutral-behavioural, SAGE, and combined, outperform the vanilla AOCC-only baseline (Section 5.4). The decisive effect is a roughly fivefold reduction in seed-to-seed variance under combined feedback, the only Stage 4 contrast to clear $\alpha = 0.05$. The mean-AOCC lift over vanilla is directionally consistent across all three conditions but does not reach significance under Holm-corrected Welch tests at the 10-seed budget.

The answer to the central research question is therefore qualified but positive. Trajectory-based behavioural feedback does guide LLMs to design better optimisation algorithms, but the improvement manifests primarily as reduced failure-mode variance rather than as a large mean lift, and the framing of the feedback matters more than the feature content. Reporting behavioural observations to the LLM in a neutral tone helps. Telling the LLM which direction to push a feature in, even when the directional advice is empirically correct, often does not.

The first sub-question asked which LLMs can reliably generate high-quality metaheuristics for continuous black-box optimisation. The Stage 1 screening of ten models found that the closed API models lead. Gemini 3 Flash combined competitive quality (0.862 AOCC) with the lowest failure rate of any model (2.0%) and the fastest inference, and was carried forward for that reason. The secondary finding is that medium-sized open-weight models in the 24–32B range, notably Qwen3.5 27B and Devstral Small 2, can produce viable metaheuristics, though they trail the Gemini family in both quality and reliability.

The second sub-question asked which trajectory-based behavioural features are most informative about algorithm quality. The Stage 2 tri-criterion consensus reduced the 32-feature vocabulary to ten predictive, non-redundant descriptors, with exploitation- and convergence-related features emerging as the strongest correlates of AOCC. Seven of the ten selected features were introduced to LLaMEA-generated algorithm analysis in this thesis.

The third sub-question asked whether the framing of behavioural feedback affects the performance of generated algorithms. The Stage 3 sweep across 29 feature-by-format conditions found that framing matters more than feature content. Neutral observation consistently outperformed directional and comparative prescription. Prescriptive feedback steered the median feature value in the advised direction in only 37% of cases, despite that advice being empirically grounded in the Stage 1 data.

The most promising directions for future work follow directly from these caveats. Open-weight models in the 70–200B parameter range can be served on owned hardware and would lift the seed budget by an order of magnitude, bringing observed effect sizes within statistical reach. Target-model calibration of feature selection and prescriptive reference values would tighten the prescriptive signal that Stage 3 found to be blunted by behavioural-distribution collapse. Behavioural feedback should be studied at the instance or BBOB-group level before being aggregated, and should target multivariate behavioural profiles rather than individual features. AST-aware mutation guidance and retrieval-augmented prompting from a corpus of past LLaMEA-generated algorithms together offer a path toward generating algorithms that escape the gravitational pull of canonical mechanisms. None of these is a small undertaking, but each addresses a specific limitation surfaced by the experiments reported here, and together they outline a programme for moving LLM-driven algorithm design from the recombination of known mechanisms toward genuinely novel designs.

References

[1] Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2 edition, 2015.

[2] John R. Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976.

[3] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[4] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. Evolving the structure of evolution strategies. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.

[5] Holger H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.

[6] Niki van Stein and Thomas Bäck. LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29(2):331–345, 2025. Preprint: arXiv:2405.20132.

[7] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[8] 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)*, pages 829–836. ACM, 2011.

[9] Katherine M. Malan and Andries P. Engelbrecht. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163, 2013.

[10] Niki van Stein, Anna V. Kononova, Lars Kotthoff, and Thomas Bäck. LLaMEA-SAGE: Guiding automated algorithm design with structural feedback from explainable AI. *arXiv preprint arXiv:2601.21511*, 2026.

[11] Gjorgjina Cenikj, Gašper Petelin, Carola Doerr, Peter Korosec, and Tome Eftimov. DynamoRep: Trajectory-based population dynamics for classification of black-box optimization problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023. arXiv:2306.05438.

[12] Niki van Stein, Haoran Yin, Anna V. Kononova, Thomas Bäck, and Gabriela Ochoa. Behaviour space analysis of LLM-driven meta-heuristic discovery. In *Parallel Problem Solving from Nature – PPSN XVIII*. Springer, 2025. arXiv:2507.03605.

[13] Edward A. Codling, Michael J. Plank, and Simon Benhamou. Random walk models in biology. *Journal of the Royal Society Interface*, 5(25):813–834, 2008.

[14] Joshua S. Richman and J. Randall Moorman. Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology – Heart and Circulatory Physiology*, 278(6):H2039–H2049, 2000.

[15] Christoph Bandt and Bernd Pompe. Permutation entropy: A natural complexity measure for time series. *Physical Review Letters*, 88(17):174102, 2002.

[16] Edward Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.

[17] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

[18] Albert-László Barabási. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.

[19] Kenneth Sörensen and Fred W. Glover. Metaheuristics. In Saul I. Gass and Michael C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 960–970. Springer, Boston, MA, 3rd edition, 2013.

[20] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

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

[22] Claus Aranha, Christian L. Camacho Villalón, Felipe Campelo, Marco Dorigo, Rubén Ruiz, Marc Sevaux, Kenneth Sörensen, and Thomas Stützle. Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intelligence*, 16(1):1–6, 2022.

[23] Niki van Stein, Anna V. Kononova, and Thomas Bäck. From performance to understanding: A vision for explainable automated algorithm design. *arXiv preprint arXiv:2511.16201*, 2025.

[24] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):1–25, 2009.

[25] Diederick Vermetten, Fabio Caraffini, Anna V. Kononova, and Thomas Bäck. Modular differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 864–872, 2023.

[26] Justyna Petke, Saemundur O. Haraldsson, Mark Harman, William B. Langdon, David R. White, and John R. Woodward. Genetic improvement of software: A comprehensive survey. *IEEE Transactions on Evolutionary Computation*, 22(3):415–432, 2018.

[27] Dikshit Chauhan, Bapi Dutta, Indu Bala, Niki van Stein, Thomas Bäck, and Anupam Yadav. Evolutionary computation and large language models: A survey of methods, synergies, and applications. *arXiv preprint arXiv:2505.15741*, 2025.

[28] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.08532.

[29] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. arXiv:2211.01910.

[30] 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.

[31] 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 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32201–32223. PMLR, 2024.

[32] Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249*, 2023.

[33] Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024. arXiv:2310.02304.

[34] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, pages 4766–4777. Curran Associates, Inc., 2017.

[35] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. arXiv:2501.08603.

[36] Zhuoliang Xie, Fei Liu, Zhenkun Wang, and Qingfu Zhang. LLM-driven neighborhood search for efficient heuristic design. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 2025.

[37] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[38] Diederick Vermetten, Furong Ye, Thomas Bäck, and Carola Doerr. MA-BBOB: Many-affine combinations of BBOB functions for evaluating AutoML approaches in noiseless numerical black-box optimization contexts. In *Proceedings of the Second International Conference on Automated Machine Learning*, volume 224 of *Proceedings of Machine Learning Research*. PMLR, 2023.

[39] Jacob de Nobel, Furong Ye, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. IOHexperimenter: Benchmarking platform for iterative optimization heuristics. *Evolutionary Computation*, 32(3):205–210, 2024.

[40] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. IOHanalyzer: Detailed performance analyses for iterative optimization heuristics. *ACM Transactions on Evolutionary Learning and Optimization*, 2(1):1–29, 2022.

[41] Niki van Stein, Anna V. Kononova, Haoran Yin, and Thomas Bäck. BLADE: Benchmark suite for LLM-driven automated design and evolution of iterative optimisation heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2025. arXiv:2504.20183.

[42] Gabriela Ochoa, Katherine M. Malan, and Christian Blum. Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics. *Applied Soft Computing*, 109:107492, 2021.

[43] Ana Kostovska, Anja Jankovic, Diederick Vermetten, Jacob de Nobel, Hao Wang, Tome Eftimov, and Carola Doerr. Per-run algorithm selection with warm-starting using trajectory-based features. In *Parallel Problem Solving from Nature – PPSN XVII*, pages 46–60. Springer, 2022. arXiv:2204.09483.

[44] Gjorgjina Cenikj, Ana Nikolikj, Gašper Petelin, Niki van Stein, Carola Doerr, and Tome Eftimov. A survey of meta-features used for automated selection of algorithms for black-box single-objective continuous optimization. *Swarm and Evolutionary Computation*, 101, 2026. arXiv:2406.06629.

[45] Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[46] Andrey N. Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giornale dell'Istituto Italiano degli Attuari*, 4:83–91, 1933.

[47] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a Python package). *Neurocomputing*, 307:72–77, 2018.

[48] Jacob de Nobel, Hao Wang, and Thomas Bäck. Explorative data analysis of time series based algorithm features of CMA-ES variants. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 510–518. ACM, 2021.

[49] Fu Xing Long, Diederick Vermetten, Bas van Stein, and Anna V. Kononova. BBOB instance analysis: Landscape properties and algorithm performance across problem instances. In *Applications of Evolutionary Computation (EvoApplications 2023)*, volume 13989 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2023.

[50] Ollama. Ollama, 2024.

[51] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

[52] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[53] Essential AI. Rnj-1: Building instruments of intelligence. https://essential.ai/research/rnj-1, 2025. Accessed: 2026-03-17.

[54] Mistral AI. Devstral 2: Next-generation coding models. https://mistral.ai/news/devstral-2-vibe-cli, 2025. Accessed: 2026-03-17.

[55] OLMo Team, Allen Institute for AI. OLMo 3. *arXiv preprint arXiv:2512.13961*, 2025.

[56] Granite Team, IBM. Granite 4.0 language models. *arXiv preprint arXiv:2505.08699*, 2025.

[57] Google. Gemini 3: Our most intelligent model. https://blog.google/products/gemini/gemini-3/, 2025. Accessed: 2026-03-17.

[58] Google. Introducing Gemini 3 Flash. https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/, 2025. Accessed: 2026-03-17.

[59] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[60] Duncan Black. *The Theory of Committees and Elections*. Cambridge University Press, 1958.

[61] Niki van Stein. Neighborhood adaptive differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '25 Companion)*, pages 1–2. Association for Computing Machinery, 2025.

[62] William H. Kruskal and W. Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[63] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V. Arnold, and Tim Hohm. Mirrored sampling and sequential selection for evolution strategies. In *Parallel Problem Solving from Nature – PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 11–21. Springer, 2010.

[64] Grahame A. Jastrebski and Dirk V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2814–2821, 2006.

[65] Anne Auger and Nikolaus Hansen. A restart CMA evolution strategy with increasing population size. In *IEEE Congress on Evolutionary Computation (CEC)*, volume 2, pages 1769–1776, 2005.

[66] Nikolaus Hansen. Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO)*, pages 2389–2396. ACM, 2009.

[67] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020. arXiv:2005.11401.

[68] Md Rizwan Parvez, Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Retrieval augmented code generation and summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2719–2734. Association for Computational Linguistics, 2021.

A RandomSearch Reference Algorithm

The example portion of the vanilla LLaMEA initial prompt (Section 4.1) reproduces the `RandomSearch` class verbatim. The same class is also used to seed the initial population, ensuring that all runs begin from a behaviour-neutral starting point.

```
import numpy as np

class RandomSearch:
    def __init__(self, budget=10000, dim=10):
        self.budget = budget
        self.dim = dim
        self.f_opt = np.inf
        self.x_opt = None

    def __call__(self, func):
        for i in range(self.budget):
            x = np.random.uniform(func.bounds.
                lb, func.bounds.ub)
            f = func(x)
            if f < self.f_opt:
                self.f_opt = f
                self.x_opt = x
        return self.f_opt, self.x_opt
```

B Stage 4 Function Selection

The twenty-function set used in Stage 4 (Section 4.6) is disjoint from the ten-function screening set described in Section 4.2. The selection procedure (greedy forward pass) prioritises per-BBOB-function weight uniformity over per-group balance. Figure 18 reports the resulting per-BBOB-function and per-group weight distribution. Table 13 lists the BBOB function weights for all twenty selected MA-BBOB functions.

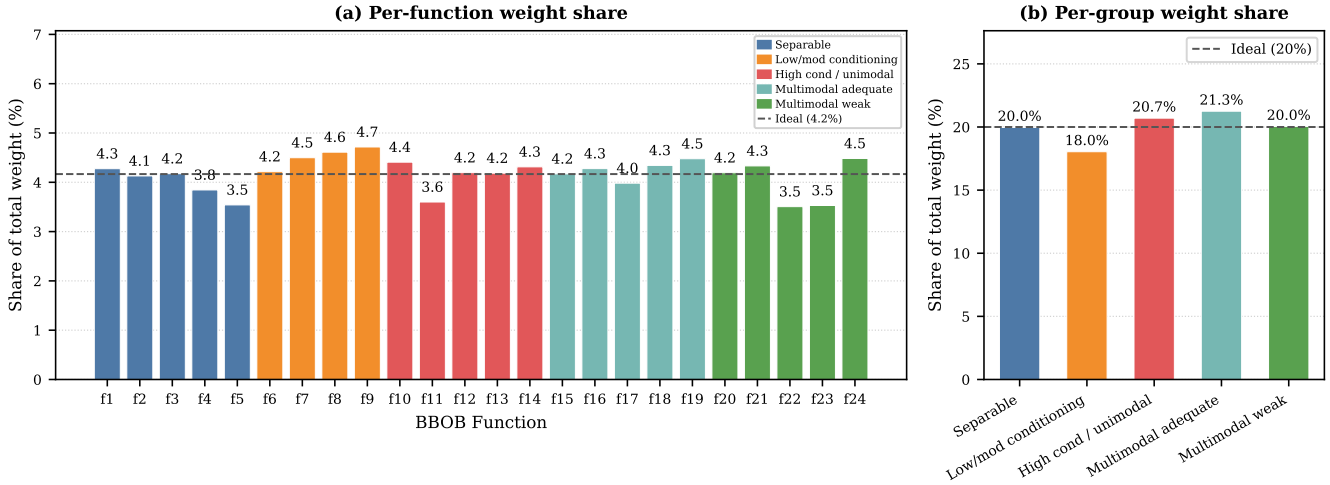

Figure 18: Per-BBOB-function (left) and per-group (right) weight distribution of the 20 MA-BBOB functions used in Stage 4. Dashed lines mark the ideal uniform shares (4.2% per BBOB function, 20% per group). All 24 BBOB functions are covered, with a coefficient of variation of 0.080 across per-BBOB-function totals and group shares within ±1.3% of the ideal.

Table 13: BBOB function weights for the 20 MA-BBOB functions used in Stage 4. Values are percentages of total weight. Empty cells indicate zero weight.

Func.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
22		18			7	13			18			15	3						12			13		
93									19		16		23					18	20				4	
166														15	17			23		30				15
196	5	19			11	10									15	11		4			10			16
203		30			30	18					22													
288				23			7			27	7										29		7	
321				9		5	16		17	10		18	8				5	12						
408	24			14														19	20		22			
480	19		22	3			6								8	13			10		18			
513	15		27													18				14				26
528			22		18		11			3		12	22			10					1			
598								29						30			27			2			12	
697								3		32		4		27		7					6		21	
781		15							14		12					13	21	10		15				
784			12		5		12				15	24			19							12		
803							21	19					21		15	13			7	5				
894	14					14			16	16			6		10				21					3
947	3					12	17	18	10								5			17				18
951	5			3		12		11									22					20	27	
999				24				13				11		15				1				25		12