Universiteit Leiden
The Netherlands

Opleiding Informatica

Prompting strategies for mutation and crossover operators in LLM based evolutionary algorithms

Koen Simonides

Supervisors:
Dr. N. van Stein & Dr. A.V. Kononova

BACHELOR THESIS

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

15/6/2026

Abstract

Large language models (LLMs) have recently enabled automated algorithm discovery through evolutionary frameworks such as LLaMEA, where new candidate algorithms are generated using prompts as operators. Despite their central role in the search process, little is known about the behaviour of these operators or how they should be selected during evolution.

This thesis contributes a methodology for evaluating prompt operators through isolated parent-child comparisons, enabling their behaviour to be characterized independently of complete evolutionary runs. Using this methodology, mutation operators and newly proposed crossover operators are evaluated within LLaMEA. In addition, both offline and adaptive online operator selection strategies are evaluated using complete evolutionary runs.

The results show that prompt operators exhibit distinct search characteristics and suggest no single operator performs best throughout the search process. Furthermore, crossover prompting integrates naturally into LLaMEA and becomes particularly effective during later stages of optimization. Finally, some informed operator selection strategies outperform uniform random sampling on average, suggesting that operator characteristics can be exploited to improve automated algorithm discovery.

Contents

1 Introduction

Many real-world tasks can be expressed as the search for a solution that optimizes a given objective, such as minimizing cost or maximizing performance. There exists a wide range of metaheuristic and evolutionary algorithms designed to solve such optimization problems. However, no single algorithm performs best across all problem instances. This observation is formalized by the No Free Lunch theorem [WM97], which states that averaged over all possible optimization problems, all optimization algorithms perform equally well. Therefore, effective optimization will require methods designed specifically for their domain.

To address this need for problem-specific algorithms, methods for Automated Algorithm Discovery (AAD) have been proposed [Koz92, vRWvLB16]. These approaches aim to automatically construct algorithms for a given task. Earlier forms of AAD relied on combining predefined code modules, which limited the range of possible solutions. The introduction of large language models (LLMs) significantly expands this space, as they are capable of generating and modifying code directly. This allows for the creation of arbitrary algorithms without relying on predefined structures.

LLaMEA [vSB25], the main focus of this work, is a recent method that applies LLMs within an evolutionary framework for automated algorithm discovery. It maintains a population of candidate algorithms, which are iteratively improved over generations. In each iteration, new solutions are generated using LLM prompts as mutation operators.

Though LLaMEA shows promising results on benchmark and real-world problems [YKBvS25], several aspects remain largely unexplored. In its original form, LLaMEA relies solely on mutation operators, which generate new solutions from a single parent.

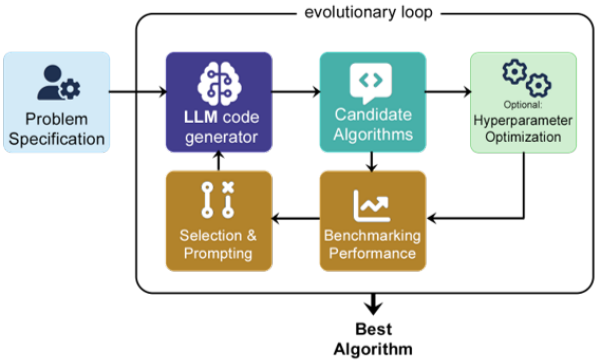

Figure 1: Evolutionary workflow of LLaMEA. Candidate algorithms are generated by prompt operators, evaluated on the optimization problem, and selected for subsequent generations.

Allowing prompt operators to use multiple parent solutions enables them to combine ideas from different algorithms, analogous to crossover in classical evolutionary computation.

This thesis investigates prompting strategies for mutation and crossover operators in LLM-based evolutionary algorithms. The central research question is:

What is the best prompting strategy for mutation and crossover operators in LLM-based evolutionary algorithms?

To answer this question, the following sub-questions are considered:

- How to effectively benchmark different prompt operators for automatic algorithm discovery?

- To what extent does the use of crossover operators affect the discovery process of optimization algorithms?
- What is the best strategy for selecting operators?

Addressing these questions, this thesis makes the following contributions:

- We introduce a methodology for evaluating prompt operators through isolated parent-child comparisons, enabling the analysis of operator behaviour independently of full evolutionary runs.
- We extend the operator set commonly used in LLM-based algorithm evolution with several new mutation and crossover (multi-parent) operators, and characterize their behaviour in terms of optimization performance, exploration-exploitation behaviour, semantic correctness and structural code modifications.
- We investigate online and offline operator selection strategies, evaluating how operator knowledge can be leveraged to improve automated algorithm discovery.

The code and data repository of this thesis is available via the following link:
https://doi.org/10.5281/zenodo.21080582
This thesis is organized as follows. Section 2 reviews related work on LLM-based automated algorithm discovery, prompt operator design, and operator selection. Section 3 presents a methodology for evaluating prompt operators. Section 4 evaluates operator selection strategies within LLaMEA. Finally, Sections 5 and 6 present the conclusions, discuss the implications and limitations of the work, and outline directions for future research.
This bachelor's thesis was conducted at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, under the supervision of Dr. A.V. Kononova and Dr. N. van Stein.

2 Related work

In this section, we discuss related research on AAD using LLMs, considering prompt-based operator design, operator selection, and the assessment of operator effectiveness.

2.1 Operators in LLM-based algorithm evolution

Early LLM-based AAD systems rely on fixed, problem-specific prompt templates, where the LLM is restricted to modifying only a limited part of the algorithm. FunSearch [EFTH+25] evolves only a critical program fragment within a predefined skeleton. It populates the LLM prompt with high scoring programs sampled from a database.

Algorithm Evolution using Large Language Model (AEL) [LTYZ23] extends this approach to full-algorithm evolution and explicitly defines specific prompt templates for initialization, crossover, and mutation. Evolution of Heuristics (EoH) [LTY+24] further formalizes operator design by introducing a predefined operator pool consisting of five operators, including three mutation operators focused on exploitation and two crossover operators for exploration. LLaMEA [vSB25] follows a similar

operator-pool design, but separates operator selection from prompt construction. Instead of defining separate prompt templates per operator, it maintains a set of mutation operators that are fitted into a shared prompt scaffold.

More recent approaches move away from manually specified operators and instead delegate operator construction to the LLM. ReEvo [YWC+24] makes use of a reflection mechanism in which the LLM generates improvements without explicitly defined operator templates. Similarly, AlphaEvolve [NVE+25] makes use of a prompt sampling strategy that constructs operators dynamically by considering previously generated programs and their evaluation feedback, removing the need for manually designed operators.

2.2 Approaches for operator selection

In contrast to classical evolutionary computation, where operator selection is a central design component, most LLM-based AAD methods do not explicitly model or adapt operator selection. In early approaches, operators are implicitly defined by a single prompt scaffold, as in FunSearch and AEL. Diversity in the population of algorithms is expected to develop naturally through selection. Alternatively, selection is bypassed by applying all available operators within each iteration, as is done in EoH.

AEL defines separate operators for initialization, crossover, and mutation, but applies them using fixed probabilities. This results in a static operator selection scheme without adaptation during the evolutionary process. LLaMEA further simplifies the operator space by restricting it to mutation operators. Operator selection is performed by random sampling from a predefined set, which remains unchanged throughout the evolutionary process.

2.3 Operator analysis

Several works have investigated adaptive operator selection in evolutionary algorithms. Boks [Bok21] proposes an adaptive framework for Differential Evolution that dynamically selects operators based on fitness improvements and population diversity, showing that different operator configurations are effective at different stages of the search and for different classes of optimization problems. Boel [Boe24] extends this idea by learning operator selection online using deep reinforcement learning. The results show that adaptive operator selection can outperform static strategies, although performance still varies considerably across optimization problems. Together, these works suggest that operator effectiveness is both problem dependent and dynamic throughout the search process.

While operator analysis has received attention in the context traditional evolutionary algorithms [Bok21, Boe24], comparable studies are absent in LLM-based automated algorithm discovery. Existing LLM-based AAD systems primarily focus on operator design with static selection methods. Operator effectiveness is typically evaluated only through aggregate performance over complete evolutionary runs, making it difficult to isolate the effects of individual operator applications from population dynamics and selection effects.

Overall, existing work leaves two important gaps. First, there is currently no established methodology

for characterizing prompt operators independently of full evolutionary runs. Second, adaptive operator selection methods that are common in evolutionary computation have received little attention in LLM-based algorithm evolution. This thesis addresses these gaps by introducing a methodology for evaluating prompt operators through isolated parent-child comparisons and by investigating operator selection strategies that leverage the resulting operator characteristics.

3 Operator comparison

We propose a methodology that evaluates prompt operators through isolated parent-child comparisons. Rather than analysing complete LLaMEA runs, we sample previously evaluated solutions, apply a selected prompt operator under the original problem configuration, and compare the resulting offspring directly to its parent. This allows operator behaviour to be observed independently of the evolutionary process.

3.1 Operator comparison methodology

To enable a direct comparison between mutation and crossover prompts, we represent prompt operators as a tuple consisting of a task message and the number of parent solutions provided to the LLM. This abstraction separates operator intent from prompt structure and allows both single-parent mutations and crossover operators to be studied within a common framework.

The set of operators considered in this work is summarized in Table 1. Among these, *Basic*, *New*, and *Simplify* have appeared in prior work [vSKYB25]. The remaining operators are newly introduced. Table 2 shows how these task-message operators are incorporated into the full mutation and crossover prompts. These templates follow the same layout as they have previously in LLaMEA.

Table 1: Definitions of the operators included in the experiments, with $|P|$ the parent count.

Identifier	$\|P\|$	Message
`Basic`	1	Refine the strategy of the selected algorithm to improve it.
`New`	1	Generate a new algorithm that is different from the algorithms you have tried before.
`Simplify`	1	Refine and simplify the selected algorithm to improve it.
`Refactor`	1	Generate a new algorithm using core ideas from the selected algorithm.
`Restructure`	1	Modify the selected algorithm by introducing a meaningful structural change that alters its overall search behaviour.
`Correct`	1	Correct any mistakes in the selected algorithm.
`Combine2`	2	Combine the selected algorithms by inheriting key traits from both parents.
`Combine3`	3	Combine the selected algorithms by inheriting key traits from both parents.
`Simplify2`	2	Simplify and recombine the selected algorithms, preserving only the most effective ideas from each.
`New2`	2	Generate a new algorithm that is different from the selected algorithms.

Table 2: Prompt templates for mutation and crossover operators

Mutation	Crossover
<role>	<role>
<problem context>	<problem context>
The current population of algorithms already evaluated (name, description, score) is:	The current population of algorithms already evaluated (name, description, score) is:
<summaries of solutions in population>	<summaries of solutions in population>
The selected solution to update is:	The selected algorithms are:
<selected solution with code and feedback>	<selected solutions with code and feedback>
<operator message>	<operator message>
<output format descriptor>	<output format descriptor>

3.1.1 Novel operators

The newly proposed mutation operators are designed as alternatives to the standard *Basic* improvement strategy. Instead of relying on a generic refinement instruction, they provide a more explicit direction for modifying the algorithm. In particular, *Refactor* encourages reorganization around core ideas, *Restructure* introduces deliberate structural changes affecting search behaviour, and *Correct* focuses on identifying and repairing potential flaws.

In addition, we introduce several crossover operators. The operators *Combine2* and *Combine3* generalize classical crossover by combining multiple parent algorithms into a single offspring. We also adapt existing mutation-style prompts to the multi-parent setting. The operator *New2* investigates whether providing additional algorithmic context influences the generation of novel algorithmic ideas. Rather than relying on a single parent as a reference point, the operator is given multiple existing solutions from which it may infer alternative directions for exploration. The operator *Simplify2* combines simplification with recombination. The intuition is that combining multiple parent algorithms may introduce redundant or conflicting design choices. By explicitly emphasizing simplification, the operator is encouraged to retain only the most important ideas from each parent while discarding unnecessary complexity.

3.1.2 Evaluation procedure

We collect data by sampling previously evaluated solutions from completed LLaMEA runs. For each sampled solution, a selected prompt operator is applied using the original problem specification and evolutionary context. The resulting offspring is then evaluated independently and compared directly to its parent.

For all experiments, the AAOC [LIVDD25] anytime performance measure is used to determine algorithm fitness. Solutions that fail to evaluate are assigned fitness $-\infty$. To characterize the effect of each operator, we record the following fitness, reliability, and code-modification metrics.

Fitness Change The fitness improvement of an offspring is measured relative to its best parent:

$$\Delta f = \max(f_{\text{child}}, 0) - \max\left(\max_{p \in P} f_p, 0\right),$$

where P denotes the set of parent solutions. Positive values indicate improvement over the parent, while negative values indicate deterioration.

Recovery and Invalidation Rate Operator reliability is quantified using the recovery rate R and invalidation rate I:

$$R = \frac{N_{\text{recovered}}}{N_{\text{invalid parents}}}, \qquad I = \frac{N_{\text{invalidated}}}{N_{\text{valid parents}}}.$$

Here, $N_{\text{recovered}}$ denotes the number of invalid parent programs that produce a valid offspring, $N_{\text{invalidated}}$ the number of valid parent programs that produce an invalid offspring, and $N_{\text{invalid parents}}$ and $N_{\text{valid parents}}$ the corresponding total numbers of invalid and valid parent programs. For crossover operators, a parent set is considered invalid only if all parent programs are invalid, and valid only if all parent programs are valid.

Normalized Line Divergence The magnitude of code modifications is measured using the normalized line divergence:

$$D = \frac{1}{|P|} \sum_{p \in P} \frac{L_{\text{diff}}(p, \text{child})}{\max(L_p, L_{\text{child}})},$$

where P denotes the set of parent solutions, $L_{\text{diff}}(p, \text{child})$ is the number of inserted and deleted lines reported by a line-based diff algorithm between parent p and the offspring, and L_p and L_{child} denote the corresponding program lengths. Since modified lines are counted as both a deletion and an insertion, the normalized line divergence is bounded by $0 \leq D \leq 2$.

Relative Token and Complexity Change Changes in program size and complexity are measured relative to the average parent:

$$\Delta_{\text{tokens}} = \frac{T_{\text{child}} - \bar{T}_{\text{parent}}}{\bar{T}_{\text{parent}}}, \qquad \Delta_{\text{cc}} = \frac{C_{\text{child}} - \bar{C}_{\text{parent}}}{\bar{C}_{\text{parent}}},$$

where $\bar{T}_{\text{parent}}$ and $\bar{C}_{\text{parent}}$ denote the average token count and cyclomatic complexity of the parent solutions, respectively. Token counts and cyclomatic complexity are computed using the *Lizard* static analysis tool [Yin].

3.1.3 Experimental setup

To evaluate the proposed methodology, we apply it to solutions obtained from prior LLaMEA runs on the `MA_BBOB` problem set [VYBD23]. These intermediate solutions are sampled from existing runs [vS25], originally conducted to demonstrate the BLADE benchmark suite [vSKYB25]. All evaluations are performed under the same problem configuration as the sampled data: `MA_BBOB` with dimension 5 and a problem budget of 2000.

Since operator behaviour may depend on the underlying language model, the experiments are conducted using two different LLMs: `gemini-2.0-flash` and `qwen3-coder`. The selected models represent both a proprietary cloud-hosted model and an open-weight coding model. The corresponding model configurations are summarized in Table 3. `gemini-3.1-flash-lite` is used only in Section 4. For each operator, we run roughly 300 iterations, each time sampling $|P|$ parents from a single run generation from the selected dataset [vS25].

Table 3: LLM configurations used in the experiments.

Provider	**Model**	**Temp.**	**Top-p**	**Top-k**
Google AI	`gemini-2.0-flash`	1.0	0.95	64
Google AI	`gemini-3.1-flash-lite`	1.0	0.95	64
Ollama	`qwen3-coder` (30.5B)	0.7	0.80	20

3.2 Operator comparison results

This section evaluates the behaviour and effectiveness of the proposed prompt operators using isolated parent-child comparisons.

3.2.1 Aggregate operator performance

We compare the average fitness improvement obtained by each operator as a function of parent fitness. Table 4 reports average Δf values aggregated into eight equally spaced parent-fitness bins. To visualize the underlying trends, Figure 2 uses 20 equally spaced parent-fitness bins and applies a moving average with window size 4 to reduce noise in the observations.

Table 4: Average fitness change Δf per operator across eight equally spaced parent-fitness bins. Values represent the mean difference between offspring fitness and the fitness of the best parent.

Operator	0.00–0.12	0.12–0.25	0.25–0.38	0.38–0.50	0.50–0.62	0.62–0.75	0.75–0.88	0.88–1.00	Avg
gemini-2.0-flash									
Basic	0.094	-0.113	-0.074	-0.195	-0.235	-0.333	-0.762	–	-0.231
New	0.114	-0.173	-0.065	-0.234	-0.353	-0.476	-0.736	–	-0.275
Simplify	0.222	-0.004	0.091	0.004	-0.050	-0.166	-0.050	–	0.007
Refactor	0.054	-0.029	-0.023	-0.166	-0.215	-0.270	-0.536	–	-0.169
Restructure	0.055	-0.248	0.004	-0.142	-0.280	-0.319	-0.000	–	-0.133
Correct	0.080	0.171	0.050	0.072	-0.107	-0.102	0.029	–	0.027
Combine2	0.033	-0.155	-0.181	-0.167	-0.256	-0.346	-0.360	–	-0.205
Combine3	0.025	-0.117	-0.237	-0.243	-0.364	-0.349	-0.255	–	-0.220
Simplify2	0.091	-0.139	-0.100	-0.022	-0.230	-0.276	-0.273	–	-0.136
New2	0.129	0.090	-0.168	-0.209	-0.285	-0.392	-0.480	–	-0.188
qwen3-coder									
Basic	0.154	-0.117	-0.109	-0.315	-0.335	-0.350	-0.451	–	-0.218
New	0.042	-0.216	-0.286	-0.383	-0.357	-0.559	-0.492	–	-0.321
Simplify	0.193	0.145	0.040	-0.003	-0.099	-0.166	-0.321	–	-0.030
Refactor	0.104	-0.085	-0.126	-0.306	-0.428	-0.445	-0.434	–	-0.246
Restructure	0.032	-0.093	-0.257	-0.322	-0.529	-0.585	-0.693	–	-0.350
Correct	0.064	-0.055	-0.135	-0.266	-0.345	-0.403	-0.587	–	-0.247
Combine2	0.054	-0.196	-0.135	-0.266	-0.381	-0.449	-0.431	–	-0.257
Combine3	0.041	0.005	-0.257	-0.230	-0.426	-0.467	-0.292	–	-0.232
Simplify2	0.075	0.016	-0.167	-0.174	-0.221	-0.293	-0.227	–	-0.142
New2	0.076	-0.050	-0.240	-0.319	-0.469	-0.536	-0.544	–	-0.297

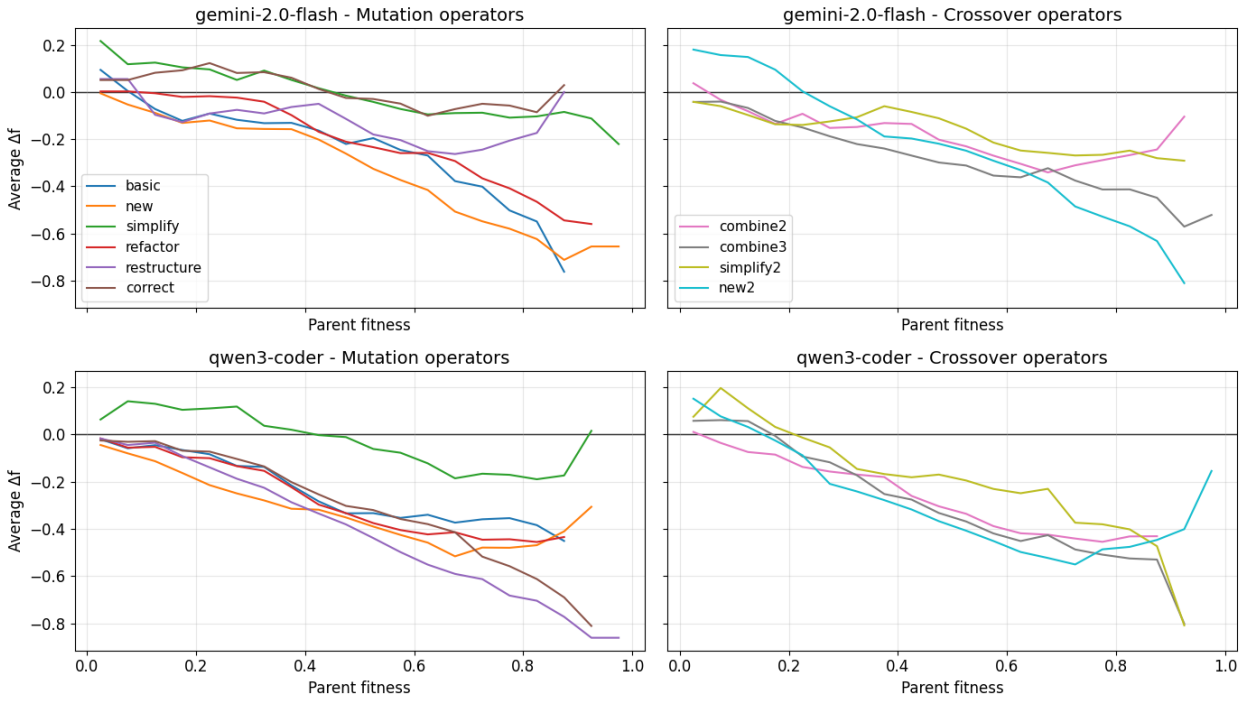

Figure 2: Average fitness change Δf as a function of parent fitness for all evaluated operators. Parent fitness values are grouped into 20 equally spaced bins and smoothed using a moving average with window size 4.

Figure 2 reveals substantial differences between prompt operators for both language models. While operator effectiveness generally decreases as parent fitness increases, the magnitude of this decline differs considerably across operators. This indicates that prompt operators are not interchangeable and that operator effectiveness strongly depends on the current stage of the search process.

Most operators achieve positive average Δf values only for low parent fitness levels and become increasingly negative as parent quality improves. However, several operators show improved relative performance in the highest fitness regions, including *Restructure* and *Combine2* for Gemini, and *New* and *New2* for Qwen. The operator rankings therefore change throughout the search process, suggesting that no single operator is uniformly optimal across all fitness levels.

Among the mutation operators, *Simplify* is the strongest overall performer. For Gemini, it achieves the highest improvement in the first fitness bin (0.222), while for Qwen, it achieves an overall average close to zero (-0.030). Interestingly, *Correct* performs comparably to *Simplify* across all fitness levels, indicating it may have a similar effect on the underlying solutions. The novelty-oriented *New* operator performs only marginally better than the baseline *Basic* operator and does not emerge as a particularly strong low-fitness operator.

The crossover operators exhibit similar behaviour. None consistently outperform the strongest mutation operators, indicating that additional parent information alone does not guarantee improved offspring quality. However, several crossover operators remain competitive and appear less affected by increasing parent fitness. For Gemini, *New2* achieves the strongest low-fitness performance

among the crossover operators, while *Combine2* performs best in the highest populated fitness region. For Qwen, *Simplify2* is the strongest crossover operator, achieving the highest average Δf among the crossover operators (-0.142). This relative stability at higher parent fitness levels suggests that crossover operators may be particularly useful for exploiting already strong solutions.

Overall, the results indicate that operator effectiveness varies substantially both between operators and across parent fitness levels. The two language models show a remarkably consistent overall picture. In both cases, *Simplify* is the strongest operator, operator effectiveness decreases with parent fitness, and the crossover operators are competitive with the established mutation operators.

3.2.2 Exploration and exploitation

We observe the relationship between improvement frequency and improvement magnitude. Unlike the previous results, which aggregate improvement behaviour into a single outcome measure, this separates how often an operator improves a solution from how much it improves when successful. This distinction is particularly relevant in LLaMEA, where the commonly used elitist selection retains only offspring that outperform their parents. Consequently, the frequency of improvements may be more important than the magnitude of unsuccessful modifications.

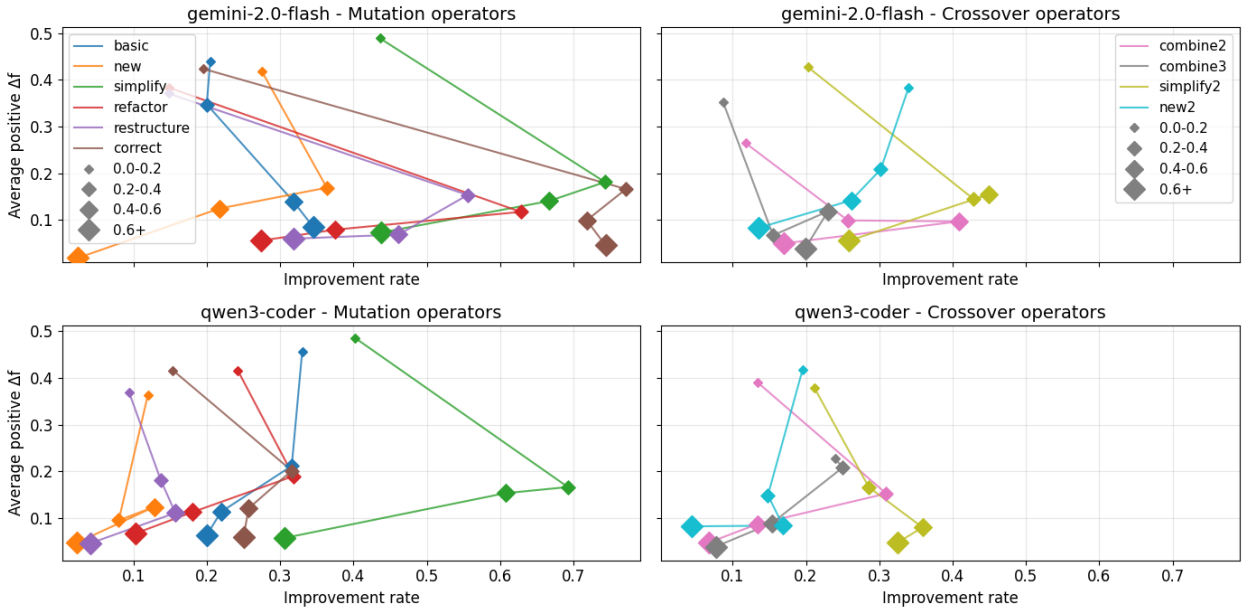

Figure 3: Exploration–exploitation trade-off for each operator and parent-fitness bin. The x-axis shows improvement rate, while the y-axis shows the average positive fitness improvement among successful offspring. Marker size indicates parent-fitness bin.

Figure 3 largely reproduces the operator ranking observed previously. Across almost all operators, average improvement magnitude decreases as the parent fitness increases. Improvement rates typically increase from the lowest fitness bins to intermediate fitness levels before declining again. Together, these trends indicate a shift from larger exploratory improvements toward smaller exploitative refinements as the search progresses.

Mutation operators achieve substantially higher improvement rates for *Gemini* than for *Qwen*, whereas the crossover operators occupy similar regions of the plot for both models. Among the mutation operators, *Simplify* and *Correct* are the strongest performers. Interestingly, for *Gemini*, *Correct* maintains the highest improvement rate across all but the first fitness bin, exceeding 0.7 in all but the lowest fitness region. Among the crossover operators, *Simplify2* is the most consistent, maintaining competitive improvement rates throughout the search and outperforming *Simplify* at high fitness levels for *Qwen*.

The results confirm that operator behaviour changes substantially throughout the search process. The crossover operators show less competitive performance in general than their mutation counterparts.

3.2.3 Reliability and code modifications

Reliability measures an operator's ability to preserve valid programs during evolution. Since invalid offspring still consume evaluations in LLaMEA, reliability directly affects the effective search budget.

To explain differences in reliability, we also examine the magnitude of the code modifications introduced by each operator. Together, these metrics characterize both how aggressively an operator modifies existing solutions and how likely those modifications are to preserve program validity.

Figure 4 and Table 5 show substantial operator-dependent differences in reliability. Among the mutation operators, *Simplify* consistently achieves the highest recovery rate while maintaining one of the lowest invalidation rates. For Gemini it achieves a recovery rate of 0.419 and an invalidation rate of 0.175, while for Qwen the corresponding values are 0.394 and 0.208. By contrast, the novelty-oriented operators, particularly *New* and *New2*, are substantially less reliable, especially for Qwen where invalidation rates reach 0.750 and 0.688, respectively.

The code-modification metrics help explain these differences. Operators that make larger modifications generally exhibit higher invalidation rates, whereas operators making smaller, more targeted changes are more likely to preserve program validity. This relationship is particularly visible for *Simplify*, which combines relatively small code modifications with high reliability, and for *New* and *New2*, which produce the largest modifications together with the highest invalidation rates. Despite being explicitly designed to repair invalid programs, *Correct* does not stand out in terms of recovery performance.

The two language models exhibit similar qualitative trends, although Qwen produces substantially more variable outcomes. Across nearly all operators, Gemini applies more consistent modifications, whereas Qwen exhibits greater variation in both reliability and code-modification metrics, as reflected by the larger standard deviations in Figure 5. This difference may in part be explained by the lower sampling temperature used for Qwen (0.7) compared to Gemini (1.0). Despite this increased variability, the overall operator tendencies remain similar.

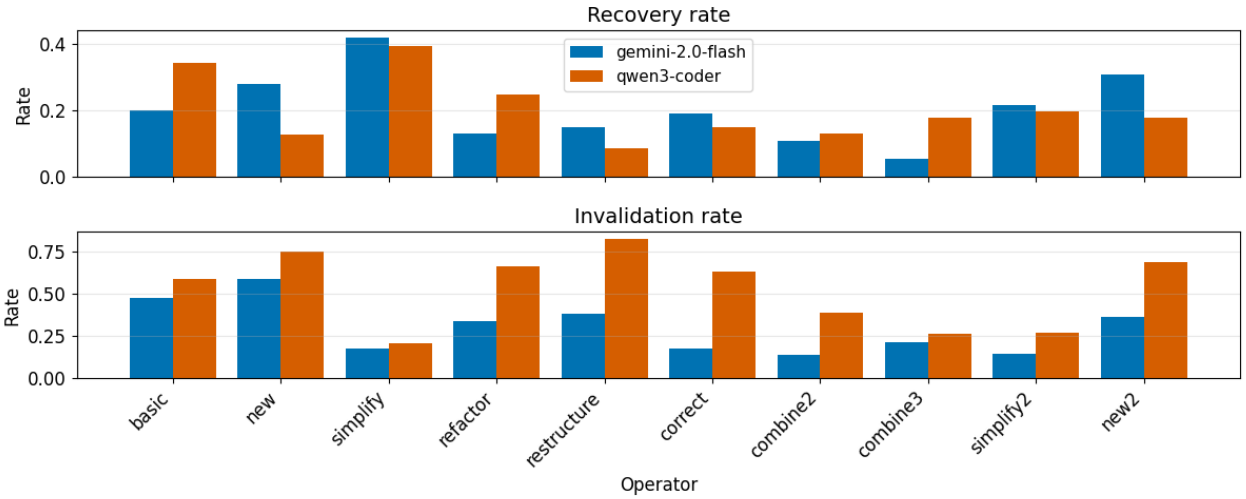

Figure 4: Recovery and invalidation rates for each operator and language model. The recovery rate measures the proportion of invalid parent programs that are transformed into valid offspring, while the invalidation rate measures the proportion of valid parent programs that become invalid after operator application.

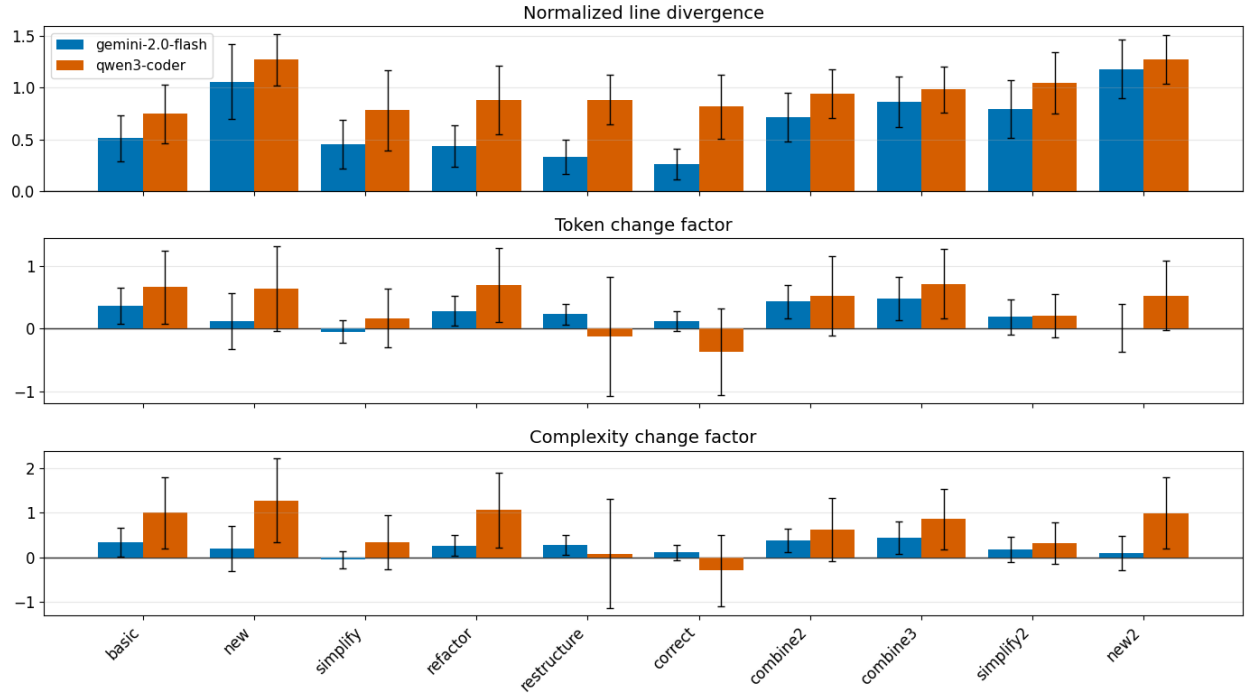

Figure 5: Structural effects of each operator measured by code divergence, relative token-count change, and relative cyclomatic-complexity change. Values are reported relative to the parent solution, or the average parent characteristics for multi-parent operators. Error bars indicate one standard deviation.

Overall, reliability is closely related to the magnitude of the generated code modifications. Operators producing smaller, targeted edits tend to preserve valid programs, whereas operators performing larger rewrites primarily contribute exploration at the cost of reliability. Since invalid offspring directly consume evaluation budget in LLaMEA, these reliability characteristics form an important consideration when selecting prompt operators.

Table 5: Operator reliability and code-modification characteristics for Gemini 2.0 Flash and Qwen3-Coder. Reliability is quantified using the recovery and invalidation rates. Code modifications are quantified using normalized line divergence, relative token-count change, and relative cyclomatic-complexity change.

Model	Operator	Recovery	Invalidation	Line Div.	Token Δ	Complexity Δ
gemini-2.0-flash	`Basic`	0.200	0.477	0.511	0.366	0.341
	`New`	0.279	0.588	1.057	0.118	0.189
	`Simplify`	0.419	0.175	0.455	-0.048	-0.058
	`Refactor`	0.129	0.338	0.440	0.283	0.264
	`Restructure`	0.148	0.377	0.330	0.231	0.278
	`Correct`	0.190	0.172	0.264	0.125	0.107
	`Combine2`	0.106	0.137	0.714	0.435	0.383
	`Combine3`	0.053	0.214	0.864	0.484	0.435
	`Simplify2`	0.215	0.140	0.791	0.190	0.170
	`New2`	0.307	0.359	1.179	0.009	0.091
qwen3-coder	`Basic`	0.344	0.587	0.747	0.665	0.994
	`New`	0.125	0.750	1.267	0.643	1.273
	`Simplify`	0.394	0.208	0.781	0.166	0.330
	`Refactor`	0.247	0.662	0.879	0.695	1.056
	`Restructure`	0.085	0.826	0.884	-0.124	0.078
	`Correct`	0.148	0.633	0.816	-0.366	-0.299
	`Combine2`	0.128	0.387	0.940	0.521	0.617
	`Combine3`	0.176	0.262	0.981	0.714	0.854
	`Simplify2`	0.196	0.269	1.042	0.207	0.316
	`New2`	0.179	0.688	1.275	0.531	0.989

4 Operator selection strategies

Having characterized the prompt operators in isolation, we now evaluate them by means of complete LLaMEA runs. The operator selection strategy determines how variation operators are allocated throughout the evolutionary search process. As shown in the previous section, prompt operators offer substantial differences in effectiveness, reliability, and structural behaviour. These differences suggest that operator choice may have a significant impact on optimization performance.

We compare both adaptive online methods that learn operator preferences during evolution and preprocessing-based methods that make use of operator characteristics identified in the previous section. The goal is to determine whether informed operator selection can improve automated algorithm discovery relative to standard random sampling.

4.1 Operator selection methodology

We consider three classes of operator selection strategies. The first consists of static methods, represented by uniform random sampling. The second consists of adaptive online methods, which estimate operator quality during evolution and continuously update their selection preferences. The third consists of preprocessing-based methods that utilize operator performance data obtained prior to the evolutionary run.

Together, these strategies allow us to compare the value of prior operator knowledge against information gathered online during the search process.

4.1.1 Sliding Window UCB

To dynamically adapt operator selection during evolution, we implement the Sliding Window Upper Confidence Bound (SW-UCB) algorithm [GM08]. Unlike static sampling strategies, SW-UCB continuously updates its estimate of operator effectiveness using recent observations. This makes it suitable for the non-stationary evolutionary process.

SW-UCB models operator selection as a multi-armed bandit problem, in which each available action is represented by an *arm*. The objective is to maximize cumulative reward by balancing exploration of less frequently selected arms with exploitation of high-performing arms. In our setting, each prompt operator corresponds to one arm. After applying an operator and evaluating the resulting offspring, the observed fitness improvement is used as the reward signal. Rewards are normalized to the interval $[0, 1]$, where non-improving operators receive reward 0.

For each operator i, the algorithm maintains the number of times it has been selected within the sliding window, denoted by n_i, together with the sum of observed rewards s_i. The empirical mean reward is then defined as

$$\mu_i = \frac{s_i}{n_i}.$$

Operators are selected according to the standard Upper Confidence Bound criterion

$$\mathrm{UCB}_i = \mu_i + c\sqrt{\frac{\log(\min(t, \tau))}{n_i}},$$

where t denotes the current iteration, τ is the sliding window size, and c is the exploration coefficient controlling the trade-off between exploration and exploitation.

The first term favours operators with high observed rewards, while the second term favours operators that have been sampled less frequently. At each iteration, the operator with the highest UCB score is selected, i.e.,

$$i^* = \arg\max_i \mathrm{UCB}_i$$

To adapt to changes during evolution, only the most recent τ operator applications are retained. When the window becomes full, the oldest observation is removed before inserting the new reward.

This allows the selector to gradually forget outdated information and react to shifts in operator effectiveness over time.

Initially, each operator is sampled once to ensure valid reward estimates before UCB-based selection begins.

4.1.2 Top3 selection

Unlike SW-UCB, which adapts during evolution, the following strategies make use of preprocessing data obtained using the operator comparison methodology described in Section 3. Their goal is to estimate which operators perform best at different stages of the search process.

We construct these selectors by collecting a large number of individual operator applications and grouping them according to the fitness of their parent solution. The parent fitness range is segmented into bins, producing groups of operator applications associated with similar parent quality levels.

For each fitness bin b and operator i, we compute the average positive fitness improvement

$$q(b, i) = \frac{1}{N_{b,i}} \sum_{k=1}^{N_{b,i}} \max(\Delta f_k, 0),$$

where $N_{b,i}$ denotes the number of sampled applications of operator i within bin b.

This produces a mapping from parent fitness ranges to estimated operator effectiveness. During evolution, the current parent fitness determines the corresponding bin, after which an operator is selected using the preprocessed statistics.

The `Top3` strategy selects the three operators with the highest values of $q(b, i)$ within the corresponding fitness bin. One of these operators is then sampled uniformly at random.

Formally, let T_b denote the set containing the three highest-ranked operators for bin b. The probability of selecting operator i is then

$$P(i \mid b) = \begin{cases} \frac{1}{3}, & i \in T_b \\ 0, & \text{otherwise} \end{cases}.$$

At each iteration, the current parent fitness determines the corresponding fitness bin b, after which an operator is sampled according to $P(i \mid b)$ This strategy preserves some fixed amount of exploration while restricting selection to operators that performed well during preprocessing.

4.1.3 Geometric selection

The *Geometric* strategy uses the same preprocessing procedure as *Top3*, but instead of restricting selection to a fixed subset of operators, it constructs a probability distribution over the full ranked operator list.

For each fitness bin b, operators are sorted according to their estimated effectiveness $q(b, i)$. Let $r_b(i)$ denote the rank of operator i within bin b, where rank 1 corresponds to the best-performing operator.

Selection probabilities are assigned according to a geometrically decreasing weighting scheme

$$w_i = \lambda^{-(r_b(i)-1)},$$

where $\lambda > 1$ is the decay factor controlling selection pressure.

The resulting operator probabilities are obtained through normalization

$$P(i \mid b) = \frac{w_i}{\sum_j w_j}.$$

As with *Top3*, the current parent fitness determines the corresponding fitness bin, after which an operator is sampled according to $P(i \mid b)$. Higher-ranked operators are exponentially more likely to be selected than lower-ranked operators. An operator ranked one position higher is λ times more likely to be selected.

Compared to `Top3`, this strategy offers a non-zero probability for all operators while still strongly favouring operators that demonstrated high effectiveness during preprocessing. The decay factor λ directly controls the balance between exploration and exploitation.

4.1.4 Evaluation procedure

To quantify the extent to which an operator selection strategy deviates from uniform random selection, we report the following distribution metric.

Operator Selection Divergence The deviation of an operator selection distribution from uniform random selection is measured using the normalized total variation distance:

$$D_{\text{sel}} = \frac{n}{2(n-1)} \sum_{i=1}^{n} \left| p_i - \frac{1}{n} \right|,$$

where n is the number of available operators and p_i denotes the probability of selecting operator i. The measure satisfies $0 \leq D_{\text{sel}} \leq 1$, where $D_{\text{sel}} = 0$ corresponds to uniform random selection and $D_{\text{sel}} = 1$ indicates deterministic selection of a single operator.

4.1.5 Experimental setup

The proposed operator selection strategies are evaluated using full LLaMEA runs within the BLADE benchmarking suite [vSKYB25] on the `MA_BBOB` problem set [VYBD23]. We compare the static *Random* baseline, the adaptive online method *SW-UCB*, and the preprocessing-based methods *Top3* and *Geometric*.

All experiments use the `gemini-3.1-flash-lite` configuration from Table 3, dimensionality 5, and a budget of 2000. Each strategy is evaluated over 6 independent runs. LLaMEA is configured

with a population size of 4, an offspring size of 4, elitist survival selection, and a total evaluation budget of 400, corresponding to 100 generations.

For the preprocessing-based selectors, operator preferences are derived from the operator comparison data presented in Section 3. Parent fitness values are grouped into 8 equally sized bins and used to estimate operator quality within different regions of the search space.

Table 6 summarizes the parameter settings used for the selection strategies.

Table 6: Evaluated operator selection strategies and their parameter settings.

Identifier	**Selection method**	**Parameters**
`Random`	Uniform random sampling	–
`UCB`	UCB selection	$\tau = 400,\ c = 1.0$
`SW-UCB-L`	Sliding Window UCB	$\tau = 120,\ c = 0.9$
`SW-UCB-M`	Sliding Window UCB	$\tau = 60,\ c = 1.0$
`SW-UCB-S`	Sliding Window UCB	$\tau = 30,\ c = 1.2$
`Top3`	Top-k selection	$k = 3$
`Geometric-L`	Geometric ranking selection	$\lambda = 2$
`Geometric-S`	Geometric ranking selection	$\lambda = 1.3$

4.2 Operator selection results

This section evaluates whether leveraging knowledge obtained from the operator analysis improves performance during full LLaMEA runs.

4.2.1 Preprocessed selection distributions

Figure 6 visualizes the operator probability distributions generated by the offline preprocessing-based selection methods. These distributions are derived directly from the operator evaluation data presented in Section 3.
The figure reflects the operator performance trends observed previously. Operators with stronger average Δf within a parent-fitness bin receive a larger proportion of the selection probability mass. Consequently, the *Simplify* and *Correct* appear most frequently throughout.

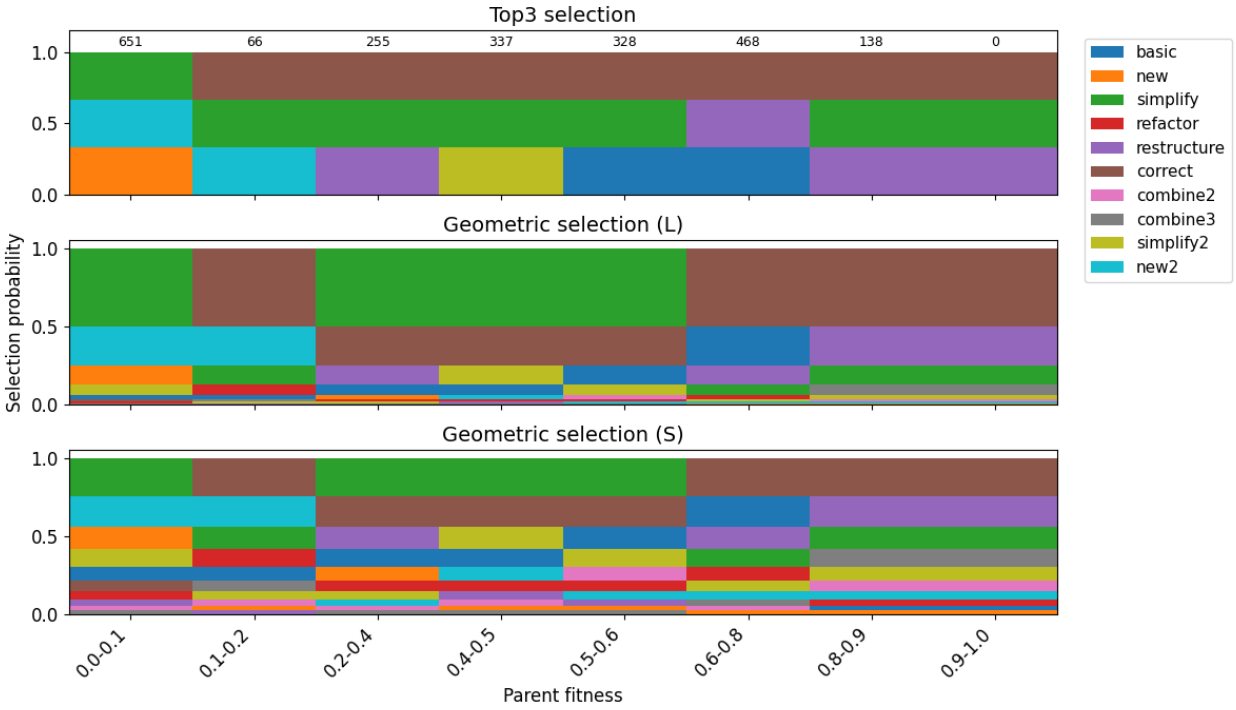

Figure 6: Operator selection distributions generated by the *Top3* and *Geometric* selection methods for each parent-fitness bin. Stacked bars represent the probability of selecting each operator. Numbers above the bars indicate the number of evaluation samples within the corresponding parent fitness bin.

4.2.2 Optimization performance

We compare the optimization performance of the different operator selection strategies during full LLaMEA runs. Figure 7 shows the average best-so-far fitness over the course of evolution, while Table 7 reports the final fitness values obtained in the individual runs.

Table 7: Final best fitness obtained by each operator selection strategy. The best result for each method is highlighted.

Method	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Mean	Std
Random	0.8217	0.8185	0.8450	**0.8488**	0.8088	0.8397	0.8304	0.0148
UCB	0.7985	0.8296	**0.8911**	0.8346	0.8703	0.8475	0.8453	0.0296
SW-UCB-L	0.8318	0.8270	0.8073	0.8060	0.8324	**0.8368**	0.8235	0.0123
SW-UCB-M	0.8409	**0.8737**	0.8311	0.8471	0.8320	0.8157	0.8401	0.0179
SW-UCB-S	0.8295	0.8542	**0.8550**	0.8374	0.8409	0.8331	0.8417	0.0098
Top3	0.8568	0.8476	**0.8666**	0.8115	0.8189	0.8087	0.8350	0.0229
Geometric-L	0.8299	0.8222	0.8181	**0.8460**	0.8292	0.8312	0.8294	0.0087
Geometric-S	**0.8601**	0.8177	0.8497	0.8447	0.8527	0.8570	0.8470	0.0140

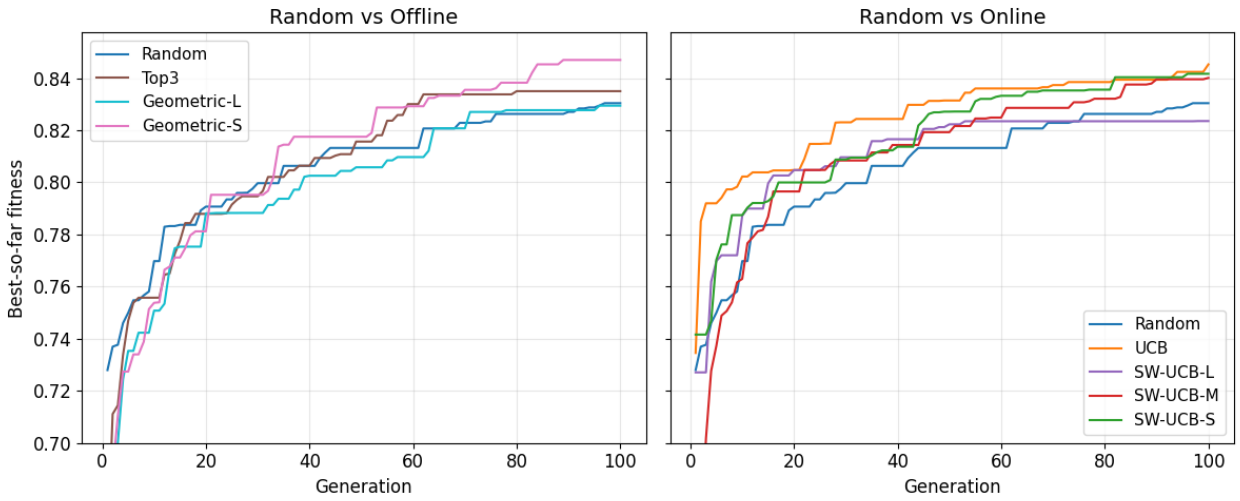

Figure 7: Average best-so-far fitness over 6 independent LLaMEA runs for each operator selection strategy. Offline methods (left) and online methods (right) both compared against random selection.

Figure 7 shows noticeable differences in both convergence behaviour and final optimization performance between the evaluated operator selection strategies. Several informed selection strategies achieve higher average final fitness than uniform random sampling, although considerable variation between runs remains visible.

Among the offline methods, *Geometric-S* achieves the highest average final fitness of 0.8470 (Table 7). Although its convergence is not noticeably faster than the random baseline during the early stages of evolution, it appears more effective at generating improvements at higher fitness levels. In contrast, *Geometric-L* performs similarly to random selection, with a mean final fitness of 0.8294 compared to 0.8304 for the baseline. This suggests that the less strongly biased probability distribution used by *Geometric-S* may provide a more favourable balance between exploiting highly ranked operators and continuing to explore lower-ranked alternatives.

The online methods exhibit a distinct optimization behaviour. Compared to the random baseline, the UCB-based methods improve more rapidly during the early stages of evolution. This behaviour is consistent with the adaptive nature of UCB, which is able to quickly adjust operator preferences during the rapidly changing early stages of the search. Among the online methods, the standard *UCB* strategy achieves the highest average final fitness (0.8453), although it also exhibits the largest variation between runs (standard deviation 0.0296). The sliding-window variants obtain comparable average performance for the smaller window configurations, with *SW-UCB-S* and *SW-UCB-M* achieving mean fitness values of 0.8417 and 0.8401, respectively, while *SW-UCB-L* performs slightly below the random baseline with a mean final fitness of 0.8235.

Overall, the results indicate that both preprocessing-based and adaptive operator selection are capable of outperforming uniform random sampling. While the observed differences are relatively small and exhibit noticeable variation between runs, the results suggest that incorporating information about operator effectiveness can improve optimization performance.

4.2.3 Operator utilization

Figure 8 summarizes the distribution of operator applications across methods throughout the evolutionary runs. The first best-so-far fitness interval is intentionally much wider than the remaining intervals. Since LLaMEA typically reaches a best-so-far fitness above 0.75 within the first 10 generations (Figure 7), relatively few operator applications occur below this threshold. As expected, the random baseline maintains an approximately uniform operator distribution, reflected by the low selection divergence values in the higher fitness intervals, where substantially more operator applications are observed.

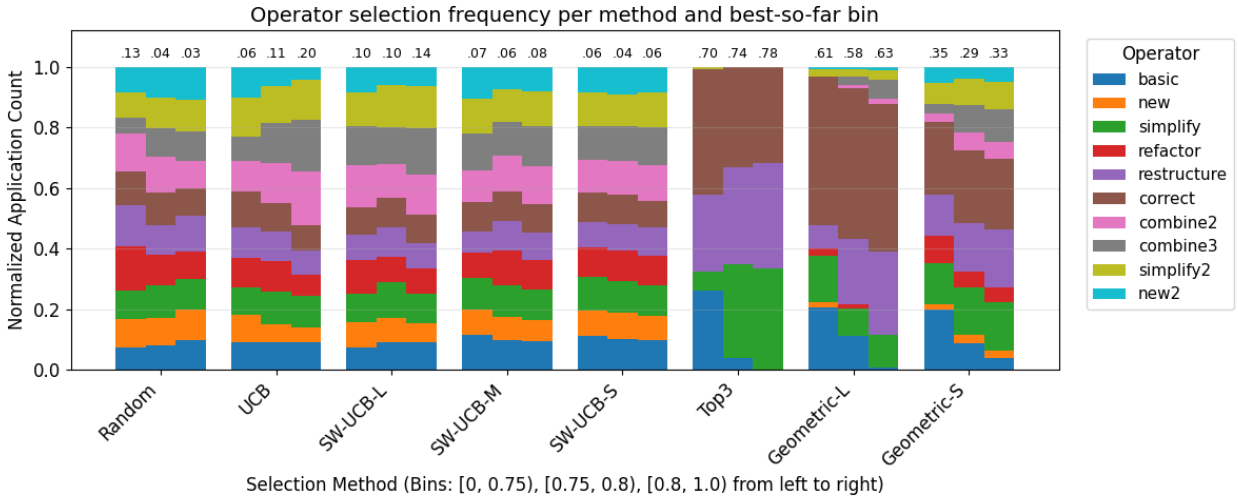

Figure 8: Operator selection frequencies during full LLaMEA runs for each selection strategy. Within each method, the three stacked bars correspond to the best-so-far fitness intervals $[0, 0.75)$, $[0.75, 0.8)$, and $[0.8, 1.0)$. Bars show the normalized operator application frequencies, while the numbers above indicate the operator selection divergence D_{sel} relative to uniform random selection. Larger values of D_{sel} indicate stronger deviation from uniform random selection.

The offline preprocessing-based methods follow their intended selection policies. The adaptive UCB-based methods occupy an intermediate position between random and preprocessing-based selection. The standard *UCB* strategy develops a clear preference for a subset of operators while still maintaining substantial exploration. Among the sliding-window variants, operator selection becomes progressively closer to uniform random selection as the window size decreases. In particular, *SW-UCB-S* approaches an almost uniform operator distribution in the later stages of the search, suggesting that the small sliding window is unable to retain sufficient reward information to establish strong operator preferences. Overall, the observed operator distributions are consistent with the expected behaviour of the evaluated selection strategies.

4.2.4 Operator contributions during optimization

We analyse the role of individual prompt operators during optimization by aggregating operator applications across all LLaMEA runs and operator selection strategies. Unlike the isolated parent-child comparisons presented in Section 3, this analysis measures how frequently operators contribute

to new best-so-far solutions during complete evolutionary runs. Table 8 summarizes the aggregated improvement statistics, while Figure 9 visualizes the corresponding improvement rates and average improvement sizes.

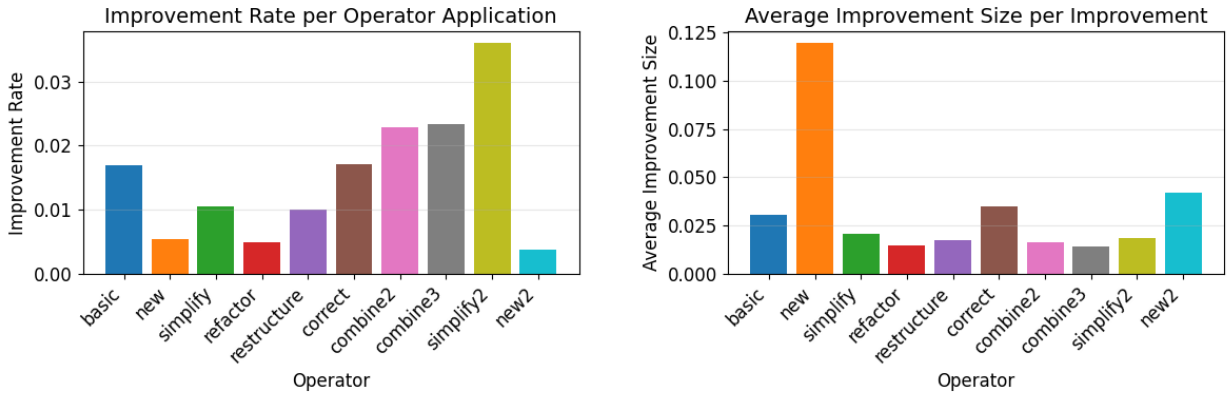

Figure 9: Aggregated operator improvement statistics across all LLaMEA runs and operator selection strategies. Left: Improvement rate, measured as the fraction of operator applications that produced a new best-so-far solution. Right: Average fitness improvement size, measured only over improving offspring.

Table 8: Aggregated operator statistics across all LLaMEA runs and operator selection strategies. Improvement rate denotes the fraction of operator applications that produced a new best-so-far solution. Average improvement size is computed only over successful improvements.

Operator	**Times Applied**	**Successful Improvements**	**Improvement Rate**	**Improvement Rate** (> 0.80)	**Avg Improvement Size** Δf
`basic`	1483	25	0.0169	0.0045	0.0304
`new`	921	5	0.0054	0.0000	0.1195
`simplify`	2581	27	0.0105	0.0054	0.0204
`refactor`	1243	6	0.0048	0.0024	0.0147
`restructure`	2874	29	0.0101	0.0039	0.0175
`correct`	3621	62	0.0171	0.0046	0.0351
`combine2`	1658	38	0.0229	0.0162	0.0162
`combine3`	1881	44	0.0234	0.0158	0.0142
`simplify2`	1694	61	0.0360	0.0233	0.0187
`new2`	1052	4	0.0038	0.0000	0.0419

The results reveal substantial differences in operator behaviour. Among all evaluated operators, the crossover operator *Simplify2* achieves the highest improvement rate (0.0360), followed by *Combine3* (0.0234) and *Combine2* (0.0229). These results demonstrate that crossover operators are highly effective within LLaMEA and confirm that extending the framework beyond mutation-only prompting is beneficial. Interestingly, the strong performance of *Simplify* observed in the isolated operator comparison experiments does not translate directly to complete evolutionary runs, where its improvement rate falls below both *Basic* and *Correct*. At the opposite end of the spectrum, the

novelty-oriented operators *New* and *New2* achieve the lowest improvement rates. However, *New* produces by far the largest average improvement size (0.1195), indicating that although successful improvements are rare, they can occasionally produce substantial jumps in solution quality.

The improvement rates obtained after a best-so-far fitness of 0.80 further highlight the effectiveness of crossover during the later stages of optimization. Both *Simplify2* and the *Combine* operators substantially outperform the mutation operators in this high-fitness region, suggesting that combining information from multiple parent solutions becomes increasingly beneficial once high-quality algorithms have been discovered. Interestingly, *Combine3* performs comparably to *Combine2*, demonstrating that crossover can be effectively extended beyond two parent solutions.

Figure 10 illustrates these observations for the two highest-performing optimization runs, both obtained using UCB-based operator selection. Nearly all improvements beyond a fitness of approximately 0.85 originate from crossover operators. Notably, the large and final improvement in the best-performing run is produced by *Combine3*.

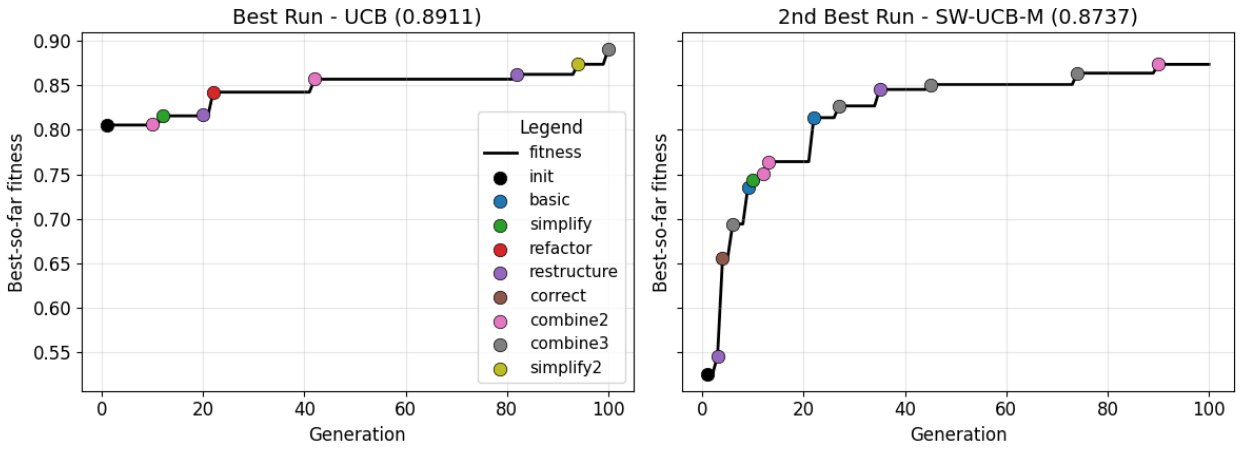

Figure 10: Optimization trajectories of the two highest-performing LLaMEA runs. Colored markers indicate the operator responsible for each accepted improvement, illustrating how different operators contribute throughout the search.

5 Discussion

This thesis investigated prompt operators from two complementary perspectives. The proposed parent-child comparison methodology characterizes the intrinsic behaviour of individual operators in isolation, whereas complete LLaMEA runs evaluate how these operators contribute within an evolutionary search process. Together, these experiments show that prompt operators exhibit distinct search characteristics and can be regarded as evolutionary search operators rather than interchangeable prompt variants.

The isolated operator comparison experiments reveal substantial differences in operator behaviour. No operator performs best across all parent-fitness regions, and clear trade-offs exist between reliability, exploration behaviour, and structural code modifications. These observations are consistent

with those reported for traditional evolutionary algorithms, where different variation operators become effective at different stages of the search process.

A notable observation is that the operator rankings obtained from isolated parent-child comparisons do not completely match those observed during complete evolutionary runs. In particular, *Simplify* consistently emerges as one of the strongest mutation operators during isolated evaluation, yet contributes comparatively few improvements during optimization. Conversely, the operator *Simplify2*, a crossover operator, becomes the single most successful operator during full LLaMEA runs.

A likely explanation is that operator effectiveness depends not only on the current solution, but also on the sequence of previously applied operators. Repeatedly applying a simplification operator gradually reduces the opportunities for further simplification, whereas an algorithm that has undergone many improvement or restructuring steps may again benefit substantially from simplification. This effect may also explain the discrepancy between the isolated operator analysis and the complete evolutionary runs. The operator comparison dataset was collected from multiple LLaMEA configurations, some of which did not include simplification operators. Consequently, sampled solutions likely contained considerably more redundant structure than the solutions encountered during the operator selection experiments, where simplification operators were applied throughout evolution.

The operator selection experiments suggest that exploiting operator characteristics can improve optimization performance, although the observed differences between selection strategies remain relatively small and noisy. The preprocessing-based methods leverage information obtained through offline operator analysis, while the adaptive UCB-based methods rapidly adjust operator preferences during the search. The strengths of these approaches appear complementary rather than contradictory, suggesting that future operator selection methods may benefit from combining offline operator knowledge with online adaptation.

5.1 Implications

The results indicate that prompt operators are not interchangeable. Different operators exhibit distinct exploration, exploitation, and reliability characteristics that can be characterized and exploited during evolution through informed operator selection.

The observed discrepancy between isolated operator evaluation and complete evolutionary runs indicates that operator effectiveness depends on evolutionary context. While offline operator characterization provides useful prior knowledge, the contribution of an operator also depends on previously applied transformations and the current state of the search. This suggests that future operator selection methods should consider not only the current solution quality, but also the evolutionary history of candidate algorithms. More generally, the complementary behaviour of the preprocessing-based and adaptive selection strategies suggests that combining offline operator knowledge with online adaptation may provide a promising direction for future research.

Finally, the proposed crossover operators demonstrate that relying exclusively on mutation may unnecessarily restrict the search capabilities. The experiments show that crossover prompting integrates naturally into LLaMEA, remains competitive throughout optimization, and contributes

disproportionately to improvements at high fitness levels. Furthermore, the comparable performance of *Combine2* and *Combine3* indicates that LLM-based crossover naturally extends beyond two parent solutions. These results encourage treating crossover as a first-class component of future LLM-based evolutionary algorithms.

5.2 Limitations

Several limitations should be considered when interpreting these results.

The experiments were conducted exclusively on LLaMEA using the `MA_BBOB` benchmark suite and three language models. Although many qualitative trends remain consistent across the evaluated models, it is unclear to what extent the observed operator characteristics generalize to other optimization problems, automated algorithm discovery frameworks, or future generations of language models. Furthermore, the operator selection experiments were conducted using a different language model than the one used to generate the offline operator evaluation data. This may reduce the effectiveness of the evaluated preprocessing-based selection strategies.

The proposed parent-child comparison methodology intentionally evaluates operators in isolation. While this provides a controlled setting for analysing intrinsic operator behaviour, it does not capture historical dependencies between successive operator applications or interactions between different operators. As demonstrated by the discrepancy between the isolated experiments and the complete evolutionary runs, these evolutionary effects can substantially influence practical operator effectiveness.

Finally, the operator selection experiments were performed using a relatively small number of independent runs. Although consistent trends are visible, the observed differences between several selection strategies remain noisy. A larger experimental study would be required to determine whether these differences are statistically robust and to establish stronger conclusions regarding the relative merits of the evaluated operator selection methods.

6 Conclusions and Further Research

This thesis investigated prompt operators for mutation, crossover, and operator selection in LLM-based evolutionary algorithms.

A parent-child comparison methodology was introduced to evaluate prompt operators independently of complete evolutionary runs. The proposed methodology provides a practical way to characterize operator behaviour and reveals substantial differences in operator effectiveness, reliability, exploration behaviour, and structural code modifications.

The experiments demonstrate that prompt operators are not interchangeable. Operator effectiveness varies considerably throughout the search process, and no single operator performs best across all parent-fitness regions. Furthermore, the observed differences between the isolated operator analysis and complete evolutionary runs suggest that operator effectiveness depends not only on the current solution, but also on the evolutionary context in which an operator is applied.

The proposed crossover operators integrate naturally into LLaMEA. Crossover operators are competitive with mutation operators during isolated evaluation and become particularly effective during complete evolutionary runs, accounting for many improvements at high fitness levels. Furthermore, the comparable performance of *Combine2* and *Combine3* demonstrates that effective LLM-based crossover extends naturally beyond two parent solutions.

The operator selection experiments suggest that exploiting operator characteristics can improve optimization performance. Both preprocessing-based and adaptive online selection performed better on average than uniform random sampling, although the observed differences remain relatively small and noisy. The complementary behaviour of these approaches suggests that combining offline operator characterization with online adaptation may provide an effective direction for future operator selection strategies.

Future work could investigate history-aware operator selection methods that explicitly account for previously applied operators in addition to current solution quality. More extensive studies across additional optimization problems, language models, and evolutionary frameworks are also needed to determine how well the observed operator characteristics generalize.

Overall, this work demonstrates that prompt operators can be systematically analysed, compared, and exploited during evolutionary search. Together, operator design, crossover prompting, and informed operator selection provide promising directions for improving LLM-based automated algorithm discovery.

References

[Boe24] Marc Boel. Online mutation strategy selection in differential evolution through deep reinforcement learning. Master's thesis, Leiden University, Leiden, The Netherlands, 2024. LIACS, Master Computer Science.

[Bok21] Rick Boks. Dynamic configuration of operators and parameters in differential evolution through combined fitness and diversity-driven adaptation methods. Master's thesis, Leiden University, Leiden Institute of Advanced Computer Science (LIACS), Leiden, The Netherlands, July 2021.

[EFTH+25] Jordan S. Ellenberg, Cristofero S. Fraser-Taliente, Thomas R. Harvey, Karan Srivastava, and Andrew V. Sutherland. Generative modeling for mathematical discovery, 2025.

[GM08] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems, 2008.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[LIVDD25] Manuel López-Ibáñez, Diederick Vermetten, Johann Dreo, and Carola Doerr. Using the empirical attainment function for analyzing single-objective black-box optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 29(5):1774–1782, October 2025.

[LTY+24] 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, 2024.

[LTYZ23] Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using Large Language Model, 2023.

[NVE+25] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025.

[vRWvLB16] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. Evolving the structure of evolution strategies. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

[vS25] N. van Stein. BLADE - code and results for the paper, 2025. Dataset https://zenodo.org/records/15119985.

[vSB25] Niki van Stein and Thomas Bäck. LLaMEA: A Large Language Model evolutionary algorithm for automatically generating metaheuristics, 2025.

[vSKYB25] 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*, GECCO '25 Companion', pages 2336–2344, New York, NY, USA, 2025. Association for Computing Machinery.

[VYBD23] Diederick Vermetten, Furong Ye, Thomas Bäck, and Carola Doerr. MA-BBOB: A problem generator for black-box optimization using affine combinations and shifts, 2023.

[WM97] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[Yin] Terry Yin. Lizard: An extensible cyclomatic complexity analyzer. https://github.com/terryyin/lizard. Version 1.22.2, accessed 2026-06-29.

[YKBvS25] Haoran Yin, Anna V. Kononova, Thomas Bäck, and Niki van Stein. Optimizing photonic structures with Large Language Model driven algorithm discovery. Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '25 Companion), 2025.

[YWC+24] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large Language Models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, 2024. https://github.com/ai4co/reevo.