DispatchEvolve: Autonomous Policy Evolution for Industrial Ride-Hailing Dispatch Engines with Multi-Objective Constraints

Zirui Yuan*
HKUST(GZ)
zyuan779@connect.hkust-gz.edu.cn

Tengfei Lyu*
HKUST(GZ)
tlyu077@connect.hkust-gz.edu.cn

Kai Wan
Didichuxing Co. Ltd
peterwan@didiglobal.com

Xu Liu
Didichuxing Co. Ltd
leoliuxu@didiglobal.com

Zhui Sun
Didichuxing Co. Ltd
sunzhui@didiglobal.com

Zihao Lu
Didichuxing Co. Ltd
luzihao@didiglobal.com

Li Ma
Didichuxing Co. Ltd
malimarey@didiglobal.com

Hao Liu†
HKUST(GZ)
liuh@ust.hk

Abstract

Ride-hailing dispatch engines must adapt continually to changing demand, supply, and driver behavior while protecting several coupled business objectives. Production improvement, however, still relies heavily on experts to identify a scenario, propose a policy change, validate it by offline replay, and advance it to an online A/B test. Automating this loop requires more than code generation. Around a mature engine, whole-engine edits provide little informative feedback, high-value scenarios and their responsible policies are not specified in advance, and locally beneficial edits may conflict when combined. We formulate the task as constrained multiobjective dispatch engine evolution and propose DispatchEvolve. Trace-Grounded Local Policy Evolution constructs scenario-specific opportunities from decision traces and replay outcomes, filters them with an opportunity critic, and evolves only the relevant policies under local guardrails. Pareto-Guided Global Engine Integration models relations among the resulting candidates, evaluates their combinations as complete engines, and retains globally feasible, non-dominated improvements in a cross-round archive. An assessment trained on historical A/B records is used only to rank archived engines for the next online test. Across held-out replay logs from four cities, DispatchEvolve is the only evaluated method to improve all six objectives over the incumbent in every city. Across cities, DispatchEvolve improves all six objectives on average after 15 engine evaluations, whereas the best-performing baseline in this comparison improves only five after 30. In four production A/B tests, the estimated CR and GMV changes are positive in every city. Nominally significant effects include gains of up to 1.07% in CR, 1.32% in GMV, and 0.97% in AR, together with a 3.09% reduction in DCAA, while City A incurs a 0.82% ETA regression. These results support the two-stage design while showing that offline feasibility does not guarantee online non-regression and must be followed by city-level monitoring. Project page: https://dispatchevolve.github.io/.

ACM Reference Format:
Zirui Yuan, Tengfei Lyu, Kai Wan, Xu Liu, Zhui Sun, Zihao Lu, Li Ma, and Hao Liu. 2026. DispatchEvolve: Autonomous Policy Evolution for Industrial Ride-Hailing Dispatch Engines with Multi-Objective Constraints. In . ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Ride-hailing platforms continuously match a stream of incoming passenger orders to available drivers, and the quality of this matching jointly determines the experience of passengers, the earnings of drivers, and the efficiency of the marketplace. In production, matching is carried out by a dispatch engine organized as a multi-stage pipeline that retrieves and filters candidate drivers, scores each order-driver pair by its predicted value, and solves an assignment to commit the final matches [16, 18]. These stages jointly optimize coupled and partly conflicting business metrics, including the completion rate (CR), the answer ratio (AR), ETA, gross merchandise value (GMV), and the passenger and driver cancel-after-acceptance rates (PCAA and DCAA). Production improvement is therefore a constrained multi-objective problem in which a candidate must improve at least one metric without degrading any of the others relative to the current engine.

In a ride-hailing marketplace, demand, supply, and driver behavior are non-stationary, shifting across morning and evening peaks, weekends, and one-off disruptions such as games and concerts [13, 17]. An engine tuned to one such regime does not remain effective in the next, so it must be improved continuously. In current practice, however, this improvement is carried out by hand, one scenario at a time. Each change must be hypothesized, hand-balanced against every other metric, screened in offline replay, and validated by a multi-week online A/B test [5]. This human-intensive workflow scales poorly because evaluating more candidate changes requires additional expert, engineering, and experimentation capacity, leaving many potentially valuable scenarios unexplored.

LLM-driven program evolution can automate candidate generation and refinement. FunSearch [14] demonstrated iterative program discovery from evaluator feedback, and AlphaEvolve [12] extended the paradigm to multi-objective system code. OpenEvolve

*Equal contribution. Work done during internship at Didichuxing Co. Ltd.
†Corresponding author.

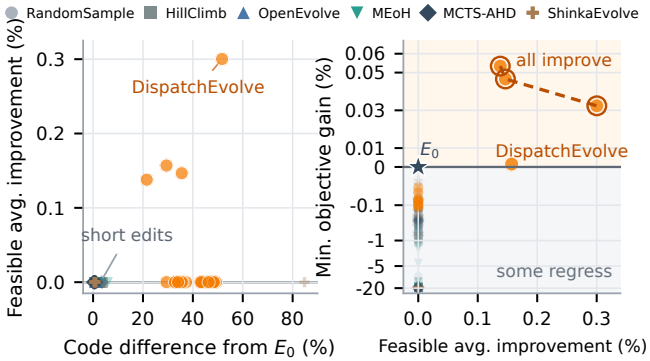
Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and ShinkaEvolve [6, 15] provide open implementations, while related loops optimize heuristics [10, 19–21, 23]. These methods work best when nearby edits provide directional feedback, the target is known, and useful edits accumulate without exhaustive interaction search. A production dispatch engine appears compatible: its editable regions can be bounded and evaluated on multiple objectives. However, industrial dispatch does not reliably satisfy these conditions.

Applying program evolution to the full dispatch engine exposes three challenges. First, direct whole-engine evolution rarely produces a candidate that clearly improves the current engine. Years of manual refinement leave gains concentrated in narrow conditions, which full-data replay dilutes with unaffected traffic. Broader edits often trade one objective for another and fail a non-regression guardrail. Figure 1 provides an empirical preview of this bottleneck. Under a matched budget, baselines remain near small edits of E_0 and fail to improve all six objectives. DispatchEvolve reaches farther variants by composing scenario-scoped edits. Focused evaluation could strengthen the search signal, but the engine does not identify which scenario or policy to target. Second, high-value opportunities are difficult to discover and attribute across heterogeneous scenarios. Policy effects vary across time, location, market conditions, order types, and user behavior. The resulting slices are too numerous to enumerate, and aggregate metrics can hide local losses. A weak slice may reflect exogenous conditions rather than an editable policy defect, while an actionable gap may arise from several policy decisions. Marketplace methods typically predefine a target such as scoring, matching, or repositioning [4, 7, 13, 16, 18, 24]. They do not identify an actionable scenario, its objective, and the responsible policies jointly. Third, the value of combining new policies cannot be inferred from their isolated gains. Complementary policies can offset isolated regressions and produce a Pareto improvement unavailable to any one policy. Their joint effects may reinforce, duplicate, cancel, or conflict, while subset search grows exponentially. These challenges require actionable-target discovery, focused edit evaluation, and joint policy testing under whole-engine guardrails.

DispatchEvolve connects two stages. *Trace-Grounded Local Policy Evolution* constructs opportunities from decision traces, each pairing a scenario, primary objective, and policy set. An opportunity critic rejects unactionable candidates, and an LLM program optimizer evolves only the selected policies on scenario-specific replay under metric guardrails. *Pareto-Guided Global Engine Integration* models candidate relations, assembles complete engines, and evaluates them on full-data replay. Feasible non-dominated engines enter a cross-round archive. An online-uplift assessment trained on historical A/B records selects each next incumbent and the final A/B candidate. Thus, the general optimizer supplies local mutations; DispatchEvolve contributes trace-grounded task construction and guarded global integration. We instantiate DispatchEvolve on the candidate-filtering and order-driver scoring stages of DiDi's production engine, holding the downstream matching solver fixed. On held-out replay logs from four cities, DispatchEvolve is the only evaluated method to improve all six objectives in every city. Across cities, it improves all six objectives on average after 15 engine evaluations, whereas the best-performing baseline in this comparison improves only five after 30. In four production



(a) Search distance (b) Multi-objective feasibility

Figure 1: Search reach and multi-objective feasibility. (a) Baseline candidates remain near E_0, whereas DispatchEvolve explores farther variants. (b) Only DispatchEvolve attains positive feasible average improvement while maintaining positive minimum-objective improvement.

A/B tests, the estimated CR and GMV changes are positive in every city, with nominally significant gains reaching 1.07% and 1.32%, respectively, although City A shows a nominally significant 0.82% ETA regression. We make four contributions.

- We define continuous improvement as constrained engine evolution. DispatchEvolve separates whole-engine search into opportunity-scoped evolution and Pareto-guided integration.
- We introduce Stage I. It pairs a trace-derived scenario with a primary objective and relevant policies, filters opportunities with an LLM critic, and evolves only the selected policies under metric guardrails.
- We propose Pareto-Guided Global Engine Integration. It represents candidate conflicts and interactions in a relation graph, uses a cross-round Pareto archive to guide combinatorial assembly, validates each complete engine under full-data non-regression, and ranks the surviving engines with an online-uplift assessment trained on historical A/B experiments.
- We evaluate DispatchEvolve on DiDi's production dispatch system through offline replay in four cities, component and composition analyses, a serving-latency pilot, and production A/B tests in four cities, while reporting protocol mismatches and deployment trade-offs explicitly.

2 Problem Formulation

Dispatch engine. The optimization target is a runnable production codebase rather than a single model or parameter vector. We represent a dispatch engine as

$$E = (P, L), \qquad E \in \mathcal{E}, \tag{1}$$

where P is the pipeline structure that orders and routes policy execution, L is the set of policy programs, and $\mathcal{E}$ is the search space of runnable engines within the allowed edit scope. Each $p_k \in L$ implements a filtering, scoring, or other dispatch rule behind a stable

interface. At each dispatch decision epoch, E receives the pending orders and available drivers, constructs and scores candidate order-driver pairs, and passes its outputs to a fixed downstream matching service that produces the assignment. We call the complete replay record of one epoch, including its inputs, policy executions, and resulting assignment, a *dispatch instance*. The formulation is stated at the engine level, while the reported instantiation operates on a bounded set of production policies.

Inputs and data partition. A problem instance contains the initial production engine E_0, offline order-driver replay logs D, the business objectives $\mathcal{M}$, a record H of historical online A/B experiments, and a maximum number of evolution rounds R. The objective set is $\mathcal{M} = \{\text{AR}, \text{CR}, \text{ETA}, \text{GMV}, \text{PCAA}, \text{DCAA}\}$. Before search, D is temporally partitioned into two disjoint sets. The evolution set D_{evo} supports trace construction, opportunity discovery, local policy evolution, complete-engine evaluation, and archive updates. The frozen test set D_{test} is not read during search and is used only to evaluate the selected engine afterward. The historical record H is separate from both replay sets and is used only to train the online-uplift assessment before evolution, as detailed in Section 3.4. It affects neither the optimization objective nor Pareto dominance.

Multi-objective performance. For any runnable candidate E and replay subset $D' \subseteq D$, a shared evaluator returns the raw metric vector $\mathbf{m}(E; D')$, whose component for objective j is $m_j(E; D')$. Because the objectives have different improvement directions, we orient each metric so that larger values are better and normalize it by a per-metric scale fixed before search. This gives the direction-unified performance vector $\widehat{\mathbf{m}}(E; D')$, with components $\widehat{m}_j(E; D')$ for $j \in \mathcal{M}$. The multi-objective performance change of E relative to a reference engine E_{ref} is

$$\Delta\widehat{\mathbf{m}}(E, E_{\text{ref}}; D') = \widehat{\mathbf{m}}(E; D') - \widehat{\mathbf{m}}(E_{\text{ref}}; D'). \tag{2}$$

Global evaluation uses E_0 as the reference, whereas local evolution compares a candidate with the incumbent engine E_r of the current round (Section 3.2).

Problem definition. We formalize continuous improvement as *constrained, multi-objective evolution of a dispatch engine*. Given E_0, D_{evo}, and $\mathcal{M}$, the goal is to find engines in $\mathcal{E}$ whose offline performance changes over E_0 are maximal under Pareto dominance,

$$\max_{E \in \mathcal{E}} \ \Delta\widehat{\mathbf{m}}(E, E_0; D_{\text{evo}}), \tag{3}$$

subject to global non-regression. We call an engine E globally feasible if it is no worse than E_0 on every business objective,

$$\forall j \in \mathcal{M}, \qquad \Delta\widehat{m}_j(E, E_0; D_{\text{evo}}) \geq 0. \tag{4}$$

This definition includes the reference engine E_0. A globally feasible engine is a *feasible improvement* if it additionally improves at least one objective,

$$\exists j \in \mathcal{M}, \qquad \Delta\widehat{m}_j(E, E_0; D_{\text{evo}}) > 0. \tag{5}$$

DispatchEvolve initializes the Pareto archive (Section 3.3) with E_0 and then admits only feasible improvements. At termination, the frozen online-uplift assessment trained from H ranks the final non-dominated engines. It returns the highest-ranked engine as $E^\star$ for online A/B testing.

Scenario granularity. Within D_{evo}, a *scenario* s selects a subset $D_s \subseteq D_{\text{evo}}$ of dispatch instances through observable conditions such as city, time band, supply-demand regime, and order type. Scenarios are not specified as part of the problem input. DispatchEvolve discovers candidate scenarios automatically from the incumbent engine's decision traces, as described in Section 3.2.

3 The DispatchEvolve Framework

3.1 Overview

Under a fixed budget, direct whole-engine search over coupled objectives yields too few feasible improvements to guide evolution. DispatchEvolve instead runs for at most R rounds, with $E_1 = E_0$, and divides each round into two stages. Stage I (Section 3.2) uses E_r's decision traces to discover scenario-specific opportunities. It evolves the relevant policies against E_r on D_s under local guardrails. Stage II (Section 3.3) assembles these candidates and evaluates complete engines against E_0 on D_{evo}. Candidates that satisfy (4), improve strictly under (5), and pass operational checks are Pareto-filtered into the cross-round archive $\mathcal{A}^{\text{P}}$. A frozen online-uplift assessment trained from H selects E_{r+1}, and search stops when a round adds no archive entry.

Only policies in L are editable; the pipeline P and downstream matching service remain fixed. The language model proposes scenarios, opportunities, policy edits, candidate relations, and combinations. Deterministic procedures decide runnability, replay metrics, archive eligibility, and Pareto dominance, confining the general optimizer to local code evolution. Figure 2 summarizes the workflow. Appendix A provides the notation and algorithm, and Appendix C gives the finite-budget analysis.

3.2 Trace-Grounded Local Policy Evolution

This stage addresses actionable target discovery and sparse feasible feedback. It first discovers intervention targets from the engine's own behavior and then turns each accepted target into a localized search under coupled guardrails. Here, a *decision trace* is the structured execution record produced by the dispatch engine for one dispatch instance. It is distinct from an LLM reasoning trace. Running E_r on D_{evo} records the observable input context, the policies invoked by the engine in execution order, their outputs, and the resulting assignment. For dispatch instance t, let x_t denote its observable context, which may include pending orders, available drivers, time, location, and marketplace conditions. Let $y_{t,k}$ denote the output of policy p_k, such as a filtering decision, candidate score, or routing decision, and let a_t denote the resulting assignment. The decision trace is

$$\tau_t = \Big(x_t,\ (p_1, y_{t,1}) \to (p_2, y_{t,2}) \to \cdots \to (p_{K_t}, y_{t,K_t}),\ a_t\Big), \tag{6}$$

where K_t is the number of policies executed for instance t. The trace captures how the input passes through the engine and leads to the final assignment. It does not by itself contain the resulting business metrics. DispatchEvolve therefore joins the traces with replay outcomes to construct scenario summaries containing coverage, multiobjective performance, and policy execution statistics. These summaries reveal which policies were active in scenarios exhibiting a performance gap and narrow the set of plausible intervention targets.

Combining decision traces with replay outcomes localizes a performance gap to a scenario and reveals the policies active in that

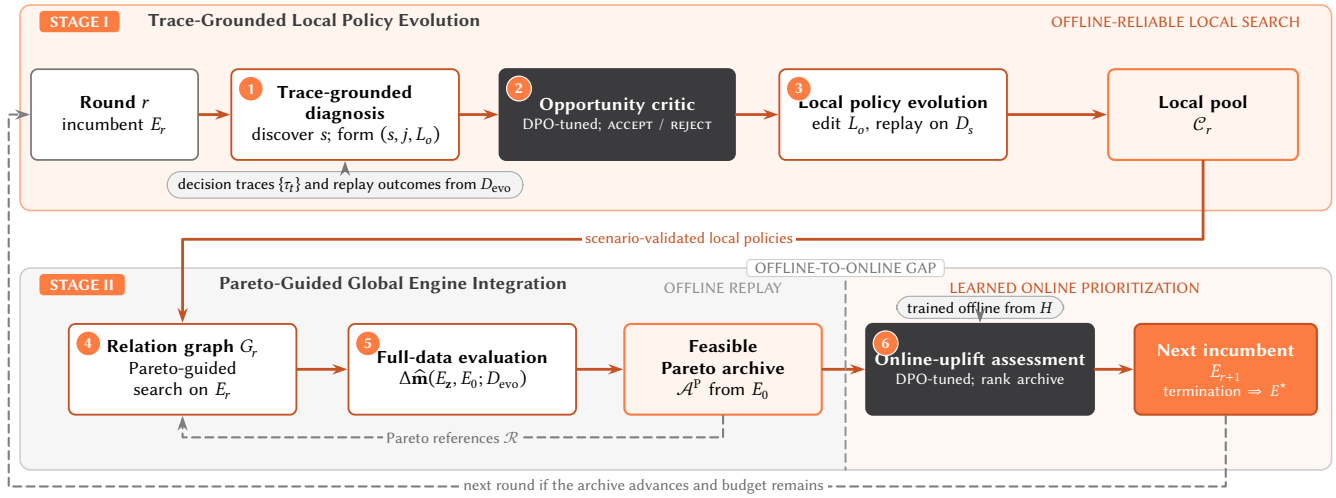


Figure 2: Overview of DispatchEvolve. Stage I discovers and locally evolves trace-grounded opportunities. Stage II assembles and validates complete engines, maintains a cross-round Pareto archive, and selects the next incumbent or final A/B-ready engine.

scenario. The next step turns this evidence into a concrete optimization target. The method maintains a cross round scenario memory M_r that accumulates the analyses of previously discovered scenarios and the outcomes of previous evolution attempts, allowing each round to build on earlier evidence. From the current traces and M_r, the language model first proposes *scenario partition rules*; each rule defines one scenario in the sense of Section 2 and yields a summary of its coverage, the incumbent's multiobjective performance on it, and policy execution statistics. Given a scenario summary, the model proposes which objective is most worth improving and which active policies are plausible intervention targets for that objective. That proposal is an *opportunity*: a scenario, the objective to lift, and the policies to touch.

$$o = (s, j, L_o), \qquad L_o \subseteq L, \tag{7}$$

that is, the scenario s, its main objective j, and the related policy set L_o drawn from the engine's policies L. The proposer is allowed to be optimistic, because a separate *opportunity critic* owns the improvability judgment: since a low metric can reflect an intrinsically hard scenario rather than a fixable one, the critic examines the triple against the scenario evidence, the size of the performance gap, and L_o, and returns ACCEPT or REJECT; only accepted opportunities proceed to evolution. Section 3.4 explains how successful and failed local-evolution trials provide preference supervision for this critic. Whether accepted or not, the scenario's summary and analysis are written back to memory, and the accepted opportunities form the round set $\mathcal{O}_r$. The per-round opportunity budget limits how many accepted opportunities proceed to local evolution.

An accepted opportunity defines a tightly scoped search: improve one objective on one scenario by editing only the policies responsible for it, and disturb nothing else. For $o = (s, j, L_o)$, local evolution edits only the related policies L_o while holding the pipeline structure P and every policy outside L_o fixed, producing a local candidate engine E'. Measured on the scenario subset D_s, its effect is the scenario-level change $\Delta\widehat{\mathbf{m}}(E', E_r; D_s) = \widehat{\mathbf{m}}(E'; D_s) - \widehat{\mathbf{m}}(E_r; D_s)$. The search is then to push the main objective as high as possible, subject to every other objective staying within a single unified non-regression tolerance ρ:

$$\begin{aligned} \max_{E'} \quad & \Delta\widehat{m}_j(E', E_r; D_s) \\ \text{s.t.} \quad & \Delta\widehat{m}_g(E', E_r; D_s) \geq -\rho, \quad \forall g \in \mathcal{M},\ g \neq j. \end{aligned} \tag{8}$$

In words, we maximize the gain on objective j over the scenario while the guardrail $\geq -\rho$ forbids any other objective g from slipping more than ρ, so a local win can never be bought by more than a bounded loss elsewhere. The local bar is deliberately the looser of the two: ρ buys the search room to trade off inside a scenario, whereas the strict global bar of (4), which allows no regression at all against the initial engine E_0, is re-imposed on every assembled engine in Section 3.3. Starting from the incumbent's code, an LLM-driven genetic program-evolution loop runs within a local budget B_o, repeatedly selecting parents and references, mutating the policy code, replaying on D_s, and reading back the multi-objective result. A run *succeeds* when it satisfies (8) with a strictly positive gain on j. Only successful runs produce local policy candidates; failed runs produce no candidate, but their diagnostics and replay outcomes are retained in M_{r+1}. Each successful result is recorded as the four-tuple

$$c_i = \left(s_i,\ j_i,\ L_i',\ \Delta\widehat{\mathbf{m}}(E_i', E_r; D_{s_i})\right), \tag{9}$$

namely its scenario s_i, main objective j_i, evolved policy set L_i', and scenario-level change, where E_i' is E_r with L_i' substituted for L_o. The candidates from all accepted opportunities form the candidate set $\mathcal{C}_r$, and analyses become the next round's memory M_{r+1}.

3.3 Pareto-Guided Global Engine Integration

This stage addresses reliable composition and deployment prioritization. A policy that helped in isolation need not survive combination with the others, nor extrapolate online. The first step is to make the couplings between candidates explicit rather than discover them by accident. Merging the per-opportunity candidates

gives the round set $\mathcal{C}_r = \{c_1, \dots, c_n\}$, over which the language model builds a candidate relation graph G_r. Each node records one candidate's scenario, main objective, evolved policies, and scenario-level change; each edge types how a pair interacts. A *hard conflict* joins two candidates that cannot coexist, for example because their edits are mutually exclusive or their required order is unsatisfiable. A *soft interaction* joins two that may influence each other through an overlapping scenario, a shared policy, or a shared metric, so their joint effect can be trusted only after a complete-engine replay. A *conflict-free* pair exhibits no structural or semantic conflict, which still does not make their gains additive. The graph thus turns an unstructured pile of candidates into a typed search space in which the impossible combinations are ruled out up front and the risky ones are flagged for measurement.

Given this typed space, integration becomes a constrained combinatorial multi-objective search over which candidates to install together. A binary vector $\mathbf{z} \in \{0,1\}^n$ names a subset of $\mathcal{C}_r$, and the integration operator Γ realizes that choice by installing each selected candidate's evolved policies at the pipeline positions of the policies they replace, yielding a complete candidate engine

$$E_{\mathbf{z}} = \Gamma(E_r, \mathcal{C}_r, \mathbf{z}); \tag{10}$$

at serving time, whenever the engine detects that an input matches a selected candidate's scenario, that position runs the candidate's evolved policy. The relation graph constrains which vectors are admissible: every hard conflict (c_i, c_ℓ) imposes $z_i + z_\ell \le 1$, and $\mathcal{Z}_r$ collects the combinations that satisfy all such constraints. Over this feasible set the search maximizes the full-data multi-objective change of the assembled engine against the baseline E_0,

$$\max_{\mathbf{z} \in \mathcal{Z}_r} \Delta\widehat{\mathbf{m}}(E_{\mathbf{z}}, E_0; D_{\text{evo}}), \tag{11}$$

read under Pareto dominance, so a vector is preferred when its engine improves the objective profile without conceding any coordinate. Crucially, because a candidate's scenario-level gain need not carry over once policies are combined, each $E_{\mathbf{z}}$ is re-evaluated on the whole evolution set D_{evo} and never scored by summing local results: the objective is measured on the complete engine rather than inferred from its parts.

The feasible set can still be exponential, so the language model is guided toward promising combinations rather than left to enumerate the 2^n subsets. Before each proposal it receives a reference set $\mathcal{R}$ of at most $N_{\mathcal{R}}$ entries, drawn from the cross-round Pareto performance archive $\mathcal{A}^{\text{P}}$ of globally feasible engines already evaluated. To rank archived engines by how much objective space they cover, hypervolume is computed against a single point fixed for the whole search; since the baseline satisfies $\Delta\widehat{\mathbf{m}}(E_0, E_0; D_{\text{evo}}) = \mathbf{0}$, that reference point sits one regression tolerance below it on every objective,

$$\mathbf{r}_{\text{HV}} = \Delta\widehat{\mathbf{m}}(E_0, E_0; D_{\text{evo}}) - \rho\mathbf{1} = -\rho\mathbf{1}, \tag{12}$$

so every non-regressing engine contributes positive volume and the tolerance ρ sets a common floor across objectives. When the archive's non-dominated set fits within $N_{\mathcal{R}}$, all of it is used; otherwise $\mathcal{R}$ is assembled in two passes. First, for each objective it takes the archived engine best on that objective, so every objective direction is shown to the model. Second, it fills the remaining slots by decreasing hypervolume contribution, the volume an engine alone adds to the archive,

$$\Delta\text{HV}(E) = \text{HV}\big(\mathcal{A}^{\text{P}}; \mathbf{r}_{\text{HV}}\big) - \text{HV}\big(\{E' \in \mathcal{A}^{\text{P}} \mid E' \neq E\}; \mathbf{r}_{\text{HV}}\big), \tag{13}$$

Here, HV denotes dominated hypervolume above the fixed reference point (12). A large $\Delta\text{HV}(E)$ marks a trade-off that the rest of the archive cannot recover. Structural difference resolves ties and preserves distinct trade-offs. Guided by $\mathcal{R}$, G_r, and prior feedback, the model proposes feasible combinations that exploit complementarities and extend the archive.

The archive admits an engine only when it is both feasible and a genuine advance, keeping the reference set clean of regressions and redundancies. One evaluated engine E Pareto-dominates another E', written $E \succ_P E'$, when its change over E_0 is no worse on every objective and strictly better on at least one,

$$\begin{aligned} E \succ_P E' \iff & \big(\forall j,\ \Delta\widehat{m}_j(E, E_0; D_{\text{evo}}) \ge \Delta\widehat{m}_j(E', E_0; D_{\text{evo}})\big) \\ & \wedge\ \big(\exists j,\ \Delta\widehat{m}_j(E, E_0; D_{\text{evo}}) > \Delta\widehat{m}_j(E', E_0; D_{\text{evo}})\big). \end{aligned} \tag{14}$$

For every archive-eligible engine E, DispatchEvolve sets $\mathcal{A}^{\text{P}} \leftarrow \text{nd}\big(\mathcal{A}^{\text{P}} \cup \{E\}\big)$. All other evaluated engines leave the archive unchanged. The archive is initialized with E_0 before the first round and persists across rounds, so its trade-offs accumulate over the whole run and seed every subsequent reference selection.

Feasibility on D_{evo} certifies an engine only against the replay evidence used during evolution. Because replay cannot observe online distribution shift or long-horizon feedback, the deployment question is handled by a separate assessment. At the end of each round, the frozen assessment model $\pi_{\theta_{\text{up}}}$ ranks the non-dominated engines in $\mathcal{A}^{\text{P}}$ by their estimated online-uplift potential. Writing $\text{Rank}_{\theta_{\text{up}}}$ for the pairwise-aggregated ordering defined in Section 3.4, the next incumbent is its first element,

$$E_{r+1} = \text{Top}\Big(\text{Rank}_{\theta_{\text{up}}}\big(\text{nd}(\mathcal{A}^{\text{P}})\big)\Big), \tag{15}$$

where $\text{nd}(\cdot)$ returns the non-dominated set. At termination, the engine selected from the final archive by this rule is the delivered output $E^\star$. Historical records H are used to train $\pi_{\theta_{\text{up}}}$ offline, not supplied during this ranking step. The assessment never enters the objective vector or the dominance relation and cannot rewrite an offline result. It only chooses, among engines already globally feasible on D_{evo}, the one most worth an experiment, so the engine it advances is A/B-ready rather than a proven online win.

What is new in this stage is a Pareto-guided procedure that assembles complete engines from local candidates, validates them under all coupled objectives, and advances one by a learned online-uplift assessment. It is neither a new general-purpose Pareto or hypervolume algorithm nor a predictor that recovers realized online uplift from historical experiments. Section 4.3 evaluates the main mechanisms through component ablations, and Section 4.4 traces how local opportunities become complete engines. Appendix B sets out the design choices behind each component.

The local-then-global decomposition also admits a finite-budget reachability analysis. Write p_{loc} for the probability that the first stage supplies a candidate pool from which some complete, feasible improvement is constructible, and p_{P} for the probability that, given such a pool, the integration search generates and validates one. Under the per-history lower-bound assumptions in Appendix C, the

per-trial hit probability is at least $p_{\text{loc}} p_{\text{P}}$, and the T-trial hit probability is at least $1-(1-p_{\text{loc}} p_{\text{P}})^T$. These quantities require repeated equal-budget trials and matched candidate pools to estimate. The single-dataset component diagnostics in Section 4.3 probe mechanism behavior but do not estimate p_{loc}, p_{P}, or a causal advantage over whole-engine evolution. The bound is therefore conditional analysis, not an empirically verified guarantee.

3.4 Offline Preference Tuning

The opportunity critic in Stage I and the online-uplift assessment in Stage II make different decisions, but both depend on downstream outcomes unavailable to a general pretrained model. We therefore fine-tune a separate Qwen3-8B model for each role using Direct Preference Optimization (DPO) [**?**]; Appendix D.4 gives the objective. Tuning is performed once before evolution. The resulting models remain frozen, and the supervision trials never enter the reported candidate pools or Pareto archive.

Opportunity critic. We construct its preference data by sampling candidate scenarios and, within each scenario, treating each business metric in turn as the primary objective. The prompt contains the opportunity (s, j, L_o) together with its scenario distribution, performance gap, and policy evidence. During offline data construction, we run the corresponding local evolution to obtain outcome-based supervision. If evolution achieves a strictly positive gain on j while satisfying the guardrails in (8), `ACCEPT` is preferred to `REJECT`; otherwise the preference is reversed. Successful and failed trials sampled across scenarios enlarge the preference set and expose the critic to both actionable and intrinsically difficult slices. The tuned critic is used only as the Stage I gate and does not score policy code.

Online-uplift assessment. We construct a second preference set from historical A/B tests H. Each example combines an experiment's marketplace distribution and data characteristics with the tested dispatch-engine policy design and its observed online outcome. For each comparable pair, the preferred engine is the one favored by the original production decision after jointly considering the realized multi-metric changes, statistical significance, and operational guardrails; pairs without an unambiguous historical preference are excluded. These labels preserve the platform's multi-objective deployment judgment instead of introducing a new scalarization of online metrics.

At inference, the frozen model compares every pair of feasible, non-dominated engines under the same pre-deployment context. Let $\mathcal{S}_r = \text{nd}(\mathcal{A}^{\text{P}})$ and let $v_{\theta_{\text{up}}}(E, E') \in \{E, E'\}$ denote the model's preferred engine. We assign each engine the pairwise win count

$$w_{\theta_{\text{up}}}(E) = \sum_{\substack{E' \in \mathcal{S}_r \\ E' \neq E}} \mathbb{I}\left[v_{\theta_{\text{up}}}(E, E') = E\right]. \tag{16}$$

$\text{Rank}_{\theta_{\text{up}}}(\mathcal{S}_r)$ orders engines by decreasing $w_{\theta_{\text{up}}}(E)$, with ties broken by hypervolume contribution (13) and then by a fixed engine identifier. Equation (15) advances its top-ranked engine. Historical online outcomes thus supervise how to prioritize offline-feasible candidates, while offline replay remains solely responsible for feasibility and Pareto dominance.

4 Experiments

We evaluate DispatchEvolve from offline multi-objective engine evolution through component and mechanism analyses to deployment cost and production A/B tests.

4.1 Experimental Setup

4.1.1 Datasets. We use two consecutive weeks of real-world dispatch logs from four Brazilian cities with heterogeneous marketplace scales and dispatch densities: the first week is the evolution set D_{evo}, and the second is the held-out test set D_{test}. Production A/B tests cover Cities A and B and two additional anonymized markets, Cities E and F; absolute business volumes are withheld.

4.1.2 Evaluation Metrics. DiDi's replay simulator evaluates historical orders and driver availability, and all results are relative to the city-specific production engine E_0. The six objectives are completion rate (CR), answer ratio (AR), GMV, ETA, passenger cancel after acceptance (PCAA), and driver cancel after acceptance (DCAA); signs are direction-aligned so that positive values always indicate improvement. Feasible average improvement (FAI) is their arithmetic mean when all six improvements are strictly positive and BR improvement is greater than -0.5%; it is zero otherwise. BR is a feasibility constraint and is not averaged into FAI.

4.1.3 Baselines. We compare against six LLM-based program-evolution methods: RandomSample, HillClimb, MEoH, MCTS-AHD, OpenEvolve, and ShinkaEvolve (Appendix D.3). In each city, all methods optimize the same production dispatch-engine code E_0 under the same evaluation budget and replay simulator.

4.1.4 Implementation Details. All methods use Gemini 3 Flash as the base LLM; DispatchEvolve additionally fine-tunes Qwen3-8B as the opportunity critic. Each method receives at most 30 full-engine evaluations on D_{evo}, after which its selected engine is evaluated on D_{test}. DispatchEvolve runs for ten rounds, optimizes at most ten local opportunities per round, and supplies at most $N_{\mathcal{R}} = 14$ Pareto references to each combination proposal.

4.2 Overall Offline Effectiveness

Table 1 reports the test-set results of all methods. The central result is the consistency of joint improvement across cities: DispatchEvolve yields a positive change in every objective in all four cities. Among the baselines, only OpenEvolve satisfies the same criterion, reaching 0.352% FAI in City A, which is 0.264 percentage points below DispatchEvolve's 0.616%; no baseline does so in the other three cities. The production dispatch engine already provides a strong initial solution, making it difficult for existing methods to find a better solution through nearby edits. These results demonstrate the effectiveness of DispatchEvolve for automated multi-objective optimization of dispatch engines.

4.3 Ablation Study

We evaluate six variants of DispatchEvolve on City A. **w/o Critic Fine-tuning** replaces the fine-tuned Critic with Gemini 3 Flash. **w/o Trace Evidence** removes the decision-trace information of the dispatch engine from opportunity discovery and evaluation

Table 1: Held-out offline performance across four Brazilian cities. Values are direction-aligned relative changes (%) from E_0. FAI is zero unless all six objectives strictly improve and the BR change exceeds −0.5%.

City	Method	FAI	CR	AR	GMV	ETA	PCAA	DCAA
City A	RandomSample	+0.000	+0.065	+0.046	+0.010	−0.030	−0.083	+0.005
	HillClimb	+0.000	+0.263	+0.272	+0.184	−0.702	−0.871	−0.361
	OpenEvolve	+0.352	+0.656	+0.511	+0.024	+0.191	+0.003	+0.729
	MEoH	+0.000	+0.328	+0.196	−0.079	−0.074	−0.173	+0.780
	MCTS-AHD	+0.000	+0.646	+0.720	+0.211	−0.695	−0.850	−0.216
	ShinkaEvolve	+0.000	−0.113	−0.047	−0.003	+0.164	−0.082	−0.042
	DispatchEvolve	**+0.616**	+1.193	+1.475	+0.290	+0.011	+0.149	+0.578
City B	RandomSample	+0.000	+0.295	+0.241	−0.033	+0.075	−0.207	+0.261
	HillClimb	+0.000	+0.605	+0.463	−0.031	−0.093	−0.459	+0.929
	OpenEvolve	+0.000	+0.203	+0.139	−0.034	−0.012	−0.115	+0.387
	MEoH	+0.000	+0.706	+0.635	+0.043	−0.137	−0.446	+0.468
	MCTS-AHD	+0.000	+0.206	−0.023	−0.076	+0.021	−0.376	+2.120
	ShinkaEvolve	+0.000	+0.238	+0.138	−0.030	+0.115	−0.219	+0.778
	DispatchEvolve	**+0.300**	+0.717	+0.459	+0.032	+0.125	+0.084	+0.384
City C	RandomSample	+0.000	−0.063	−0.105	−0.164	+0.200	−0.014	+0.355
	HillClimb	+0.000	+0.170	+0.169	+0.038	−0.044	−0.155	+0.243
	OpenEvolve	+0.000	+0.766	+0.628	−0.282	−0.180	−0.386	+0.853
	MEoH	+0.000	+0.177	+0.241	+0.123	−0.336	−0.472	+0.630
	MCTS-AHD	+0.000	+0.000	+0.000	+0.000	−0.000	−0.000	−0.000
	ShinkaEvolve	+0.000	+0.702	+0.575	−0.044	−0.257	−0.153	+0.340
	DispatchEvolve	**+0.195**	+0.204	+0.209	+0.018	+0.367	+0.085	+0.285
City D	RandomSample	+0.000	+0.171	+0.185	−0.034	−0.002	−0.165	+0.178
	HillClimb	+0.000	+0.336	+0.461	−0.052	−0.189	−0.419	+0.393
	OpenEvolve	+0.000	−0.060	+0.091	−0.035	+0.033	−0.100	+0.134
	MEoH	+0.000	−0.058	−0.079	−0.023	−0.043	−0.107	+0.133
	MCTS-AHD	+0.000	+0.012	+0.084	−0.056	−0.063	−0.047	+0.616
	ShinkaEvolve	+0.000	+0.005	+0.005	+0.006	−0.023	+0.004	+0.016
	DispatchEvolve	**+0.290**	+0.124	+0.706	+0.395	+0.022	+0.196	+0.299

while retaining the validated editable policy scope. **w/o Opportunity Critic** admits every structurally valid opportunity without semantic quality filtering. **w/o Combination LLM** replaces LLM-based composition with a deterministic greedy strategy based on local objective gains. **w/o Candidate Relations** removes the explicit hard-conflict and soft-interaction relations used during composition and integration. **w/o Pareto References** withholds previously evaluated Pareto solutions from the Combination LLM while preserving archive maintenance and incumbent selection. All variants otherwise retain the full DispatchEvolve pipeline.

Full DispatchEvolve attains an FAI of 0.616%, outperforming every ablated variant. Removing candidate relations reduces FAI to zero, while greedy composition reaches 0.542%, demonstrating the value of explicit relation modeling and relation-aware LLM composition. Replacing the fine-tuned Critic with Gemini 3 Flash reduces FAI to 0.364%; removing the Opportunity Critic and trace evidence lowers it to 0.498% and 0.457%, respectively, supporting Critic fine-tuning, opportunity filtering, and trace grounding. Withholding Pareto references further reduces FAI to 0.398%, confirming the value of prior trade-offs during global composition.

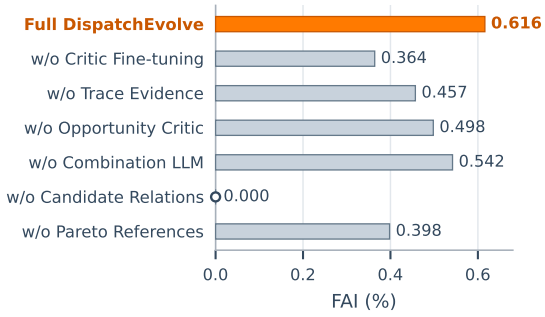


Figure 3: City A component ablation measured by FAI.

4.4 In-depth Analysis

4.4.1 Opportunity Actionability and Critic Accuracy. Across all completed diagnostic runs in the four cities, DispatchEvolve discovers 982 unique opportunities. The opportunity critic retains 771, of which 560 undergo local search and 335 succeed, giving an overall success rate of 59.8%. Figure 4(a) characterizes each opportunity by its target-metric performance gap and scenario coverage. A positive gap means that the scenario underperforms

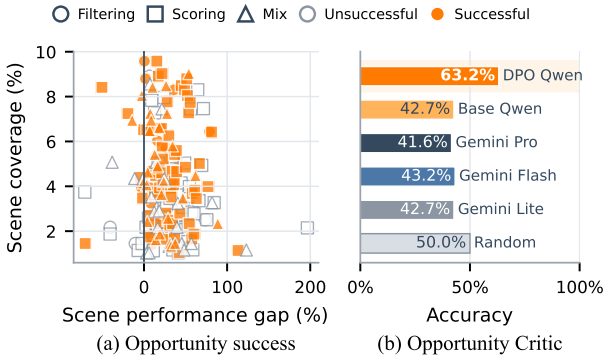


Figure 4: Opportunity outcomes and Critic accuracy.

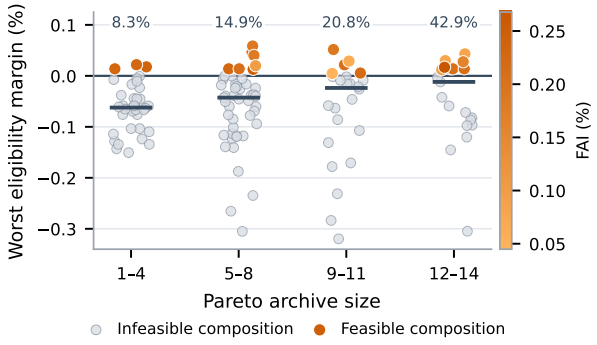


Figure 5: Engine feasibility by Pareto archive size.

the engine-wide value, while coverage is the fraction of dispatch batches captured by the scenario. Successful and unsuccessful opportunities overlap across both dimensions, showing that neither a large target-metric gap nor broad coverage alone predicts local-search success. Panel (b) shows that the DPO-tuned critic more accurately distinguishes actionable opportunities from apparent performance gaps than the untuned and general-purpose critics or random decisions. Overall, scenario gaps expose where improvement may be available but do not establish actionability; local replay remains the deterministic test of whether a corresponding policy edit is effective.

4.4.2 Global Feasibility through Composition. Successful local searches provide policy edits for global composition. We analyze all 147 full-composition evaluations from the completed diagnostic runs. A composition is feasible when all six objectives improve strictly and BR degradation does not exceed 0.5%, consistent with Section 4.1. Only 25 compositions (17.0%) satisfy both requirements. Figure 5 focuses on the 128 evaluations proposed while the Pareto archive contained 1–14 entries; 24 of these compositions are feasible. As the Pareto archive grows from 1–4 to 12–14 entries, the feasibility rate rises from 8.3% to 42.9%, while the median worst feasibility margin moves from −0.062% to −0.012%. Thus, the probability of proposing a feasible engine increases with the number of archived Pareto references. Pareto guidance improves the context for combination search.

4.5 Deployment Cost and Serving Overhead

We evaluate deployment practicality through the one-time offline expenditure required to evolve an engine and the latency overhead of serving the evolved engine. Appendix D.5 reports one complete matched-budget run for each of the four offline cities. Using Gemini 3 Flash Standard pricing, the runs cost USD 103.32–115.37 per city and complete within 11 hours 37 minutes. These LLM costs are incurred offline and do not enter the serving path. On the same machine and identical test data, mean serving latency changes by 0.68%–4.98% (macro average: 3.09%), a negligible deployment overhead. The offline expenditure is incurred once, whereas the benefit of a deployed engine recurs with production traffic. The results establish modest evolution cost and low serving overhead, but withheld absolute business volumes preclude a complete monetary cost–benefit calculation.

4.6 Production A/B Tests

We use DispatchEvolve to evolve the incumbent dispatch engines in Cities A, B, E, and F and deploy the resulting engines in online A/B tests. The retained analysis windows contain 14, 14, 12, and 7 valid days, respectively. Table 2 reports the experimental results. As shown in the table, the engines evolved by DispatchEvolve achieve significant improvements in both CR and GMV in Cities A and E. City A further improves the driver experience. Although its ETA increases by 0.82%, PCAA, the more important passenger-experience outcome, still decreases. Together, these results show that DispatchEvolve balances the interests of the platform, drivers, and passengers. Before deployment, our trained Online Critic also predicts the online performance of each evolved engine. It assigns higher scores to Cities A, E, and F, and Cities A and E indeed achieve significant improvements; City B receives a lower score and shows no significant improvement. This alignment suggests that the Online Critic can learn from historical online experiments to assess the likely online performance of a new dispatch engine.

Table 2: Production A/B-test changes from E_0 (%). Parentheses give p-values; bold marks nominal significance.

Stakeholder	Metric	City A	City B	City E	City F
Platform	Completion rate (CR) ↑	$+\mathbf{0.72}^{*}$ (0.0191)	+0.58 (0.1477)	$+\mathbf{1.07}^{**}$ (0.0010)	+0.46 (0.2870)
	GMV ↑	$+\mathbf{1.32}^{***}$ (0.0004)	+0.18 (0.7497)	$+\mathbf{0.94}^{**}$ (0.0032)	+0.48 (0.3110)
	Broadcast rate (BR) ↑	+0.20 (0.2555)	−0.27 (0.3520)	+0.20 (0.0781)	+0.22 (0.2505)
Driver	Answer ratio (AR) ↑	+0.49 (0.1720)	+0.51 (0.1206)	$+\mathbf{0.97}^{***}$ (0.0002)	+0.37 (0.1441)
	DCAA ↓	$-\mathbf{3.09}^{**}$ (0.0074)	+1.04 (0.6833)	+1.98 (0.2855)	−0.82 (0.7384)
Passenger	ETA ↓	$+\mathbf{0.82}^{***}$ (0.0003)	−0.71 (0.0562)	−0.03 (0.9270)	+0.32 (0.5541)
	PCAA ↓	−0.80 (0.6153)	−0.73 (0.5101)	+0.03 (0.9896)	−0.01 (0.9961)
Selection	Online-uplift assessment score ↑	0.75	0.65	0.75	0.75

$^{*}p < 0.05$, $^{**}p < 0.01$, and $^{***}p < 0.001$.

5 Related Work

LLM-driven program evolution and automated heuristic design. A growing line of work places a language model in a loop with an evaluator to discover programs and heuristics. Building on the code-generation ability of large models [1, 8], FunSearch [14] evolves programs against a scored objective, AlphaEvolve [12] extends the paradigm to system-level code artifacts under multiple metrics, and OPRO [19] casts the model itself as an optimizer. Related methods automate heuristic design: EoH [10] co-evolves ideas and code, ReEvo [21] adds reflective evolution, MCTS-AHD [23] organizes search with Monte Carlo Tree Search, and MEoH [20] retains multi-objective trade-offs. OpenEvolve [15] and ShinkaEvolve [6] provide reusable implementations of this paradigm. These methods build on classical Pareto and hypervolume tools [2, 11, 25], but generally presuppose a given task, a faithful offline score, and freedom to rewrite and re-evaluate the program. DispatchEvolve treats the program optimizer as an interchangeable component and supplies what a production engine lacks: trace-grounded discovery of what to optimize, guarded local evaluation, and Pareto-guided integration of complete engines.

Learning-based dispatching in ride-hailing marketplaces. Order dispatching has mostly been formulated as learning or planning within a fixed model class. Early production systems use combinatorial assignment [22]; Xu et al. [18] combine learning with large-scale combinatorial planning, and Tang et al. [16] learn a deep value network for multi-driver assignment. Multi-agent reinforcement learning further addresses mean-field coordination [7], order–vehicle distribution matching [24], joint dispatch and fleet management [4], and driver repositioning [9], as surveyed by Qin et al. [13]. These methods improve a preselected component such as scoring, matching, or repositioning and assume that both the intervention target and model form are given. DispatchEvolve instead discovers which spatiotemporal scenarios and policy code to change under coupled non-regression constraints. Because the target is ultimately online marketplace performance, we also distinguish replay from deployment evidence: controlled A/B tests remain the standard for measuring online uplift [5], while offline evaluation can only screen and prioritize candidates [3]. Accordingly, replay determines feasibility and Pareto dominance, whereas historical A/B evidence ranks which offline-feasible engine should proceed to a production test.

6 Conclusion

DispatchEvolve formulates the continuous improvement of a production engine as constrained multi-objective program evolution. It places a general program optimizer within a two-stage workflow. Trace-grounded local evolution identifies actionable scenarios and the policies responsible for them, while relation-aware Pareto integration tests which locally successful edits remain beneficial when assembled into a complete engine. Historical A/B evidence is used only to prioritize engines already shown to be feasible offline and does not affect replay objectives or Pareto dominance.

Across held-out replay logs from four cities, DispatchEvolve is the only evaluated method to improve all six objectives over the incumbent in every city. It reaches all six improved objectives on average after 15 full-engine evaluations, whereas the strongest baseline reaches five after 30. Only 25 of 147 complete-engine compositions satisfy all six improvement conditions and the broadcast-rate constraint, demonstrating why isolated local gains require global integration. Four production A/B tests yield six nominally significant improvements and positive CR and GMV estimates in every city, but also expose a nominally significant ETA regression in City A. These findings support the two-stage design while showing that offline feasibility is a disciplined promotion criterion rather than a guarantee of online non-regression. The evidence remains limited to the reported engine scope, markets, model configurations, and evaluation budgets. Future work should extend the method to additional pipeline stages and marketplace regimes, repeat component comparisons under matched random seeds, and incorporate calibrated uncertainty and post-deployment feedback into engine prioritization.

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[3] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline A/B Testing for Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. ACM, 198–206. https://doi.org/10.1145/3159652.3159687

[4] Jiarui Jin, Ming Zhou, Weinan Zhang, Minne Li, Zilong Guo, Zhiwei Qin, Yan Jiao, Xiaocheng Tang, Chenxi Wang, Jun Wang, Guobin Wu, and Jieping Ye. 2019. CoRide: Joint Order Dispatching and Fleet Management for Multi-Scale Ride-Hailing Platforms. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*. 1983–1992. https://doi.org/10.1145/3357384.3357978

[5] Ron Kohavi, Diane Tang, and Ya Xu. 2020. *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing*. Cambridge University Press, Cambridge, United Kingdom. https://doi.org/10.1017/9781108653985

[6] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. 2025. ShinkaEvolve: Towards Open-Ended and Sample-Efficient Program Evolution. arXiv preprint arXiv:2509.19349. arXiv:2509.19349 [cs.NE] https://arxiv.org/abs/2509.19349

[7] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. 2019. Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning. In *The World Wide Web Conference (WWW)*. ACM, 983–994. https://doi.org/10.1145/3308558.3313433

[8] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-Level Code Generation with AlphaCode. *Science* 378, 6624 (2022), 1092–1097. https://doi.org/10.1126/science.abq1158

[9] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. 2018. Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. ACM, London, United Kingdom, 1774–1783. https://doi.org/10.1145/3219819.3219993

[10] Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2024. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *Proceedings of the 41st International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 32201–32223. https://proceedings.mlr.press/v235/liu24bs.html

[11] Kaisa Miettinen. 1999. *Nonlinear Multiobjective Optimization*. International Series in Operations Research & Management Science, Vol. 12. Kluwer Academic Publishers, Boston. https://users.jyu.fi/~miettine/book/

[12] 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. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv preprint arXiv:2506.13131. arXiv:2506.13131 [cs.AI] https://arxiv.org/abs/2506.13131

[13] Zhiwei (Tony) Qin, Hongtu Zhu, and Jieping Ye. 2022. Reinforcement learning for ridesharing: An extended survey. *Transportation Research Part C: Emerging Technologies* 144 (2022), 103852. https://doi.org/10.1016/j.trc.2022.103852

[14] 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. 2024. Mathematical discoveries from program search with large language models. *Nature* 625, 7995 (2024), 468–475. https://doi.org/10.1038/s41586-023-06924-6

[15] Asankhaya Sharma. 2025. OpenEvolve: an open-source evolutionary coding agent. https://github.com/algorithmicsuperintelligence/openevolve. Open-source implementation of AlphaEvolve.

[16] Xiaocheng Tang, Zhiwei (Tony) Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. 2019. A Deep Value-network Based Approach for Multi-Driver Order Dispatching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. ACM, 1780–1790. https://doi.org/10.1145/3292500.3330724

[17] Yongxin Tong, Yuqiang Chen, Zimu Zhou, Lei Chen, Jie Wang, Qiang Yang, Jieping Ye, and Weifeng Lv. 2017. The Simpler The Better: A Unified Approach to Predicting Original Taxi Demands based on Large-Scale Online Platforms. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1653–1662. https://doi.org/10.1145/3097983.3098018

[18] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. 2018. Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 905–913. https://doi.org/10.1145/3219819.3219824

[19] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large Language Models as Optimizers. In *The Twelfth International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=Bb4VGOWELI

[20] Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. 2025. Multi-Objective Evolution of Heuristic Using Large Language Model. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 39. 27144–27152. https://doi.org/10.1609/aaai.v39i25.34922

[21] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*. https://proceedings.neurips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html

[22] Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. 2017. A Taxi Order Dispatch Model based On Combinatorial Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, Halifax, NS, Canada, 2151–2159. https://doi.org/10.1145/3097983.3098138

[23] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. 2025. Monte Carlo Tree Search for Comprehensive Exploration in LLM-Based Automatic Heuristic Design. arXiv preprint arXiv:2501.08603. arXiv:2501.08603 [cs.AI] https://arxiv.org/abs/2501.08603

[24] Ming Zhou, Jiarui Jin, Weinan Zhang, Zhiwei (Tony) Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, and Jieping Ye. 2019. Multi-Agent Reinforcement Learning for Order-dispatching via Order-Vehicle Distribution Matching. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*. 2645–2653. https://doi.org/10.1145/3357384.3357799

[25] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (1999), 257–271. https://doi.org/10.1109/4235.797969

A The DispatchEvolve Algorithm

Table 3 collects the core notation used throughout Sections 2 and 3; auxiliary symbols are defined inline at first use. Algorithm 1 condenses the full two-stage round loop of Section 3 into pseudocode; the one-time preference tuning in Section 3.4 precedes this procedure, which assumes both decision modules are frozen.

Table 3: Core notation used throughout Sections 2 and 3. Auxiliary symbols are defined inline at first use.

Symbol	Meaning
E_0	initial engine; reference for global feasibility
E_r, $E^\star$	round-r incumbent; final selected output engine
E=(P, L), $\mathcal{E}$	engine (pipeline, policy set); search space, Eq. (1)
D_{evo}, D_{test}	evolution and frozen test replay sets; data partition of Section 2
D_s	scenario-s subset of D_{evo}
H, R	historical online A/B records for preference tuning; maximum evolution rounds
$\widehat{\mathbf{m}}$, $\Delta\widehat{\mathbf{m}}$	direction-unified metric vector; change relative to a reference, Eq. (2)
$\mathcal{M}$, j, ρ	objective set; objective index (primary objective within an opportunity); shared local regression tolerance
o=(s, j, L_o), $\mathcal{O}_r$	opportunity (scenario, objective, related policies); accepted set
c_i, $\mathcal{C}_r$	local policy candidate; round candidate set
M_r	scenario memory (carried across rounds)
$\mathbf{z}$, Γ, $E_{\mathbf{z}}$	combination vector; integration operator; assembled engine
G_r, $\mathcal{Z}_r$	candidate relation graph; relation-feasible combination space
$\mathcal{A}^{\text{P}}$	cross-round Pareto archive initialized with E_0
$\mathbf{r}_{\text{HV}}$, $\mathcal{R}$	hypervolume reference point; Pareto reference set for combination search

B Design Rationale

This appendix explains why the two stages are built the way they are; the bounded scope of the claim and the ablations that substantiate it are stated at the end of Section 3.3. The two stages answer two different questions, and each design choice below exists to keep those questions from contaminating each other: the first stage asks whether a scenario admits a safe local gain, the second asks whether such gains can be assembled into a better complete engine.

The first stage is deliberately conservative, because a local edit that cannot be attributed or bounded is worse than no edit at all. Two choices enforce this. First, the opportunity critic returns a binary ACCEPT/REJECT verdict on the triple $o = (s, j, L_o)$ rather than a calibrated score: a yes-or-no judgment is defensible from the scenario evidence and the observed performance gap, whereas a probability the deterministic layer would then threshold merely relocates a miscalibrated number into a decision that layer must own. Second, local evolution edits only the related policies L_o, holding the pipeline structure and every other policy fixed. This makes a scenario-level gain attributable to a small, named set of policies, and it lets the unified tolerance ρ protect the remaining objectives, so that a local improvement can never quietly regress the rest of the engine.

The second stage cannot simply sum local gains, because a sum of scenario wins is not an engine win, and three coupled problems force it to evaluate complete engines instead. First, local gains are not additive: two individually beneficial candidates may act on the same traffic, share a policy, or jointly regress a metric, so the relation graph removes clearly incompatible pairs and every retained combination is re-evaluated on the full evolution set D_{evo}, never by adding local results. Second, a single weighted score would collapse a genuinely multi-objective problem into a groundless ranking, so the search is Pareto-guided: a cross-round Pareto performance archive $\mathcal{A}^{\text{P}}$ records the feasible non-dominated engines, and reference combinations are drawn from it by target-extremal and hypervolume-contribution selection, which shows the language model distinct trade-off directions rather than one scalarization.

Algorithm 1 DispatchEvolve

Require: initial engine E_0; evolution set D_{evo} (D_{test} never read); objectives $\mathcal{M}$; unified tolerance ρ; frozen preference-tuned decision modules; reference-combination limit; per-stage budgets; max rounds R
Ensure: selected engine $E^\star$

$E_1, E^\star \leftarrow E_0, E_0$; $\mathcal{A}^{\text{P}} \leftarrow \{E_0\}$ with reference point $\mathbf{r}_{\text{HV}} = -\rho\mathbf{1}$; feedback, $M_1 \leftarrow \emptyset, \emptyset$
for $r \leftarrow 1, \dots, R$ **do**
  run E_r on D_{evo} to obtain decision traces and current performance
  ▷ *Stage I: Trace-Grounded Local Policy Evolution*
  $\widetilde{\mathcal{O}}_r \leftarrow$ propose scenario rules, summaries, and opportunities o=(s, j, L_o) from the traces and M_r
  $\mathcal{O}_r, \widetilde{M}_r \leftarrow$ OpportunityCritic$(\widetilde{\mathcal{O}}_r, M_r)$
  $\mathcal{C}_r \leftarrow \emptyset$
  for all o selected from $\mathcal{O}_r$ within the per-round opportunity budget **do**
    $\mathcal{C}_o, \widetilde{M}_r \leftarrow$ LocalPolicyEvolution$(E_r, o, D_s, \widetilde{M}_r)$ ▷ $\mathcal{C}_o$: successes only
    $\mathcal{C}_r \leftarrow \mathcal{C}_r \cup \mathcal{C}_o$
  end for
  $M_{r+1} \leftarrow \widetilde{M}_r$
  if $\mathcal{C}_r = \emptyset$ **then**
    $E^\star \leftarrow E_r$; **break** ▷ no archive improvement is possible
  end if
  ▷ *Stage II: Pareto-Guided Global Engine Integration*
  $G_r \leftarrow$ relation graph over $\mathcal{C}_r$
  while combination budget remains **do**
    $\mathcal{R} \leftarrow$ reference combinations from $\mathcal{A}^{\text{P}}$ (target-extremal + hypervolume)
    $\mathbf{z} \leftarrow$ LLM proposal from $\mathcal{R}, G_r$, feedback; $E_{\mathbf{z}} \leftarrow \Gamma(E_r, \mathcal{C}_r, \mathbf{z})$
    if $E_{\mathbf{z}}$ not runnable or violates G_r **then continue**
    end if
    $\Delta\widehat{\mathbf{m}}(E_{\mathbf{z}}, E_0; D_{\text{evo}}) \leftarrow$ full-data evaluation; append to feedback
    if $\forall j, \Delta\widehat{m}_j \geq 0$, $\exists j, \Delta\widehat{m}_j > 0$, and fixed operational constraints pass **then**
      $\mathcal{A}^{\text{P}} \leftarrow \text{ND}\big(\mathcal{A}^{\text{P}} \cup \{E_{\mathbf{z}}\}\big)$
    end if
  end while
  $E_{r+1} \leftarrow \text{Top}\Big(\text{Rank}_{\theta_{\text{up}}}\big(\text{nd}(\mathcal{A}^{\text{P}})\big)\Big)$
  $E^\star \leftarrow E_{r+1}$
  if $\mathcal{A}^{\text{P}}$ gained no new engine this round **then break**
end for
return $E^\star$

Third, whether an engine is worth an online test is a different question from whether it is offline-feasible, so it is answered separately by an online-uplift assessment trained from the historical records H, kept out of the objective vector and out of the dominance relation so that a deployment guess can never rewrite an offline result.

C Finite-Budget Reachability: Formal Statements

This appendix makes precise the finite-budget reachability bound summarized at the end of Section 3.3, and its conditional-advantage form, so that the analysis is transparently about the two stages of this method rather than about search in general. Throughout, the target is defined only on the evolution set D_{evo}, since that is the evidence on which a round's archive decisions are made; the frozen D_{test} is never read during search, the historical online evidence H is used only for preference tuning and enters neither the objective vector nor the dominance relation, and each objective is first normalized by the scale fixed before search begins.

C.1 Target and macro-trial

We first fix what "progress in one round" means, because the finite-budget question only has an answer once the target is pinned down. Let $\mathcal{B}_\eta$ be the near-Pareto set of empirically global-feasible candidates whose distance to the empirical Pareto front is at most η. For two feasible candidates, one *Pareto dominates* the other, written $\succ_P$, when its multi-objective result is no worse on every coordinate and strictly better on at least one. Writing $\mathcal{A}_r^{\mathrm{P}}$ for the cross-round archive $\mathcal{A}^{\mathrm{P}}$ of Section 3.3 as it stands at the start of round r, and fixing it there, the target is the set of near-Pareto candidates that strictly improve on that archive,

$$\mathcal{B}_{r,\eta}^{+} = \left\{ E \in \mathcal{B}_\eta \;\middle|\; \begin{array}{l} \nexists\, A \in \mathcal{A}_r^{\mathrm{P}} \;:\; A \succ_P E, \\ \exists\, A \in \mathcal{A}_r^{\mathrm{P}} \;:\; E \succ_P A \end{array} \right\}. \tag{17}$$

In words, a member of $\mathcal{B}_{r,\eta}^{+}$ is an engine the archive neither already beats nor already contains: it is undominated by the archive, yet it dominates at least one archived engine, so accepting it strictly advances the archive. Counting only such domination-advancing candidates, rather than every front-enlarging one, is deliberate: it makes a hit strictly harder to score and so keeps the bounds below conservative. The archive is held fixed until the first hit and only updated afterward, so "the next Pareto improvement" does not drift within one analysis. We assume $\mathcal{B}_{r,\eta}^{+}$ is nonempty, since a round with no reachable improvement has nothing to reach.

The unit of budget is one full pass through both stages, which we call a macro-trial. A *macro-trial* is one equal-cost pass through both stages of Algorithm 1: a Stage-I candidate-supply step followed by a Stage-II integration-and-replay step. To keep reachability well defined, the analysis is anchored at a fixed archive: indexing macro-trials by k, we hold $\mathcal{A}_r^{\mathrm{P}}$ and hence $\mathcal{B}_{r,\eta}^{+}$ fixed throughout, abstracting the interleaving by which the running algorithm updates the archive. An equal-cost macro-trial comprises the first stage's opportunity search and local evolution together with the second stage's integration search and full-data replay. When comparing full frameworks, each method's macro-trial obeys a common cost cap; when comparing integration modules, the local candidate pool and the integration budget are held fixed. Any run that fails, violates a hard constraint, or yields no complete candidate counts as a miss. To attach a probability to each stage, let $\mathcal{C}_k$ be the local candidate pool produced by the first stage in the k-th macro-trial, let L_k be the event that, under the second stage's admissible operators, at least one complete candidate constructible from $\mathcal{C}_k$ lies in $\mathcal{B}_{r,\eta}^{+}$ after full-data replay, let P_k be the event that the second stage actually generates and validates such a candidate, and let $\mathcal{J}_{k-1}$ be the search history that has not yet hit the target. Assuming uniform positive per-history lower bounds,

$$\begin{aligned} \Pr\big(\mathsf{L}_k \mid \mathcal{J}_{k-1}\big) &\geq p_{\text{loc}} > 0, \\ \Pr\big(\mathsf{P}_k \mid \mathsf{L}_k, \mathcal{C}_k, \mathcal{J}_{k-1}\big) &\geq p_{\mathrm{P}} > 0, \end{aligned} \tag{18}$$

the conditional-probability decomposition gives $\Pr(\mathsf{P}_k \mid \mathcal{J}_{k-1}) \geq p_{\text{loc}} p_{\mathrm{P}}$, since P_k implies L_k. Read plainly, p_{loc} is the chance that Stage I supplies material capable of supporting a global improvement, and p_{P}, taken uniform over all pools satisfying L_k and all unhit histories, is the chance that Stage II's Pareto-guided integration then identifies a valid combination and passes it through full-data replay. Both are conditional-probability bounds to be estimated empirically, not properties guaranteed by the module names; the decomposition requires neither stage to be independent, only that the bounds hold for every unhit history. Finally, write $\Phi_T(x) = 1-(1-x)^T$ for the finite-budget hit function; it is strictly increasing on $[0,1]$, since $\Phi_T'(x) = T(1-x)^{T-1} > 0$ for $x < 1$.

C.2 Bounds

The first proposition says that if each stage clears a fixed positive bar, enough equal-cost rounds make a hit near-certain. It composes the two per-stage bounds into a single finite-budget guarantee.

Proposition C.1 (Hit bound). *If $\mathcal{B}_{r,\eta}^{+} \neq \emptyset$ and (18) holds, then*

$$\Pr\Big(\bigcup_{k=1}^{T} \mathsf{P}_k\Big) \geq \Phi_T\big(p_{\text{loc}} p_{\mathrm{P}}\big) \xrightarrow[T\to\infty]{} 1. \tag{19}$$

Proof. For any history that has not yet hit the target, the conditional probability that the next macro-trial again misses is at most $1 - p_{\text{loc}} p_{\mathrm{P}}$, by the decomposition of (18). Chaining this bound over $k = 1, \ldots, T$, the probability that all T trials miss is at most $(1 - p_{\text{loc}} p_{\mathrm{P}})^T$, and the complement is exactly $\Phi_T(p_{\text{loc}} p_{\mathrm{P}})$. Since $p_{\text{loc}} p_{\mathrm{P}} > 0$, this tends to 1 as $T \to \infty$. The argument does not require the trials to be independent. □

Read back, Proposition C.1 splits reachability into two necessary links: Stage I decides whether material capable of supporting a global improvement is produced, and Stage II decides whether a correct combination is then found and validated on that material. If either lower bound approaches zero the overall hit bound is throttled accordingly, which is precisely why the two factors are worth measuring separately.

The second proposition turns the same bound into two comparisons, each isolating one design choice of the second stage. It says that whenever Pareto-guided integration clears a higher per-history bar than its alternative, its finite-budget hit probability is strictly higher.

Proposition C.2 (Conditional advantage). *Fix a pool $\mathcal{C}$ with integrable material and give both methods the same budget. Let E_k^{P}*

and E_k^{B} be their validated candidates at trial k. Suppose Pareto integration has per-history hit lower bound p_{P}. Suppose the greedy or fixed-weight baseline has per-history hit upper bound $q_{\mathrm{B}} < p_{\mathrm{P}}$. Then over T integration trials,

$$\begin{aligned}\Pr\big(\exists k \le T : E_k^{\mathrm{P}} \in \mathcal{B}^{+}_{r,\eta} \mid \mathcal{C}\big) &\ge \Phi_T(p_{\mathrm{P}}) \\ &> \Phi_T(q_{\mathrm{B}}) \\ &\ge \Pr\big(\exists k \le T : E_k^{\mathrm{B}} \in \mathcal{B}^{+}_{r,\eta} \mid \mathcal{C}\big).\end{aligned} \tag{20}$$

Moreover, let G_k be the event that global direct evolution hits the same target in an equal-cost macro-trial, with per-history upper bound q_{G}. If $p_{\mathrm{loc}} p_{\mathrm{P}} > q_{\mathrm{G}}$, then

$$\begin{aligned}\Pr\Big(\bigcup_{k=1}^{T} \mathrm{P}_k\Big) \ge \Phi_T(p_{\mathrm{loc}} p_{\mathrm{P}}) > \Phi_T(q_{\mathrm{G}}) \\ \ge \Pr\Big(\bigcup_{k=1}^{T} \mathrm{G}_k\Big).\end{aligned} \tag{21}$$

Proof. Φ_T is strictly increasing on $[0,1]$. Under the fixed pool and budget, Proposition C.1 propagates the per-trial lower bounds. For the upper-bounded side, $\Pr(\text{no hit in } T \text{ trials}) \ge (1-q)^T$. Substituting $p_{\mathrm{P}} > q_{\mathrm{B}}$ and $p_{\mathrm{loc}} p_{\mathrm{P}} > q_{\mathrm{G}}$ into Φ_T yields both strict comparisons. □

The two inequalities in Proposition C.2 name exactly the alternatives the second stage was built against: (20) against greedy or fixed-weight combination on a shared pool, and (21) against blind whole-engine evolution at equal cost. Both are advantages only under their stated hypotheses on q_{B} and q_{G}, which is what the next subsection ties back to measurement.

C.3 Scope

The bounds are conditional.

Estimating p_{loc} requires repeated Stage-I trials under a shared budget. Each trial must be tested for a feasible improvement. Estimating p_{P} against q_{B} requires matched hit frequencies on a shared pool and budget. The ablation in Section 4.3 reports one frozen run per variant on one dataset, so it is diagnostic and does not estimate these probabilities. The target $\mathcal{B}^{+}_{r,\eta}$ is an offline replay surrogate, not the true online Pareto front. The propositions guarantee neither recovery of the full front nor a globally optimal combination, and replay never replaces a production A/B test.

D Additional Results and Implementation Details

D.1 Evaluation Efficiency

Figure 6 further compares four-city evaluation efficiency under a matched budget. At each checkpoint, the curves report the four-city mean of the best-so-far objective count among BR-compliant engines. DispatchEvolve reaches all six after 15 evaluations; the strongest baseline reaches five after 30. Its mean FAI becomes positive after six evaluations and increases from 0.045% to 0.187% by the 24th evaluation, showing that further search improves solution quality after feasibility is reached.

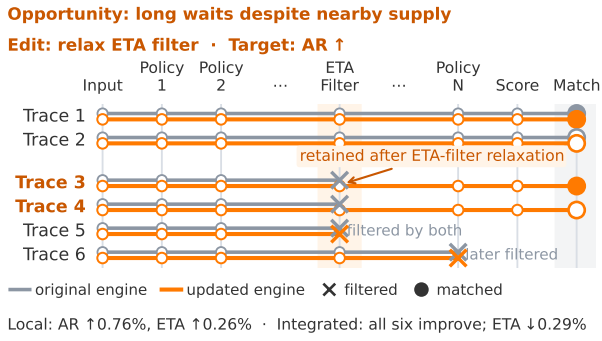


Figure 7: Policy-trace case study of local editing and full-engine integration.

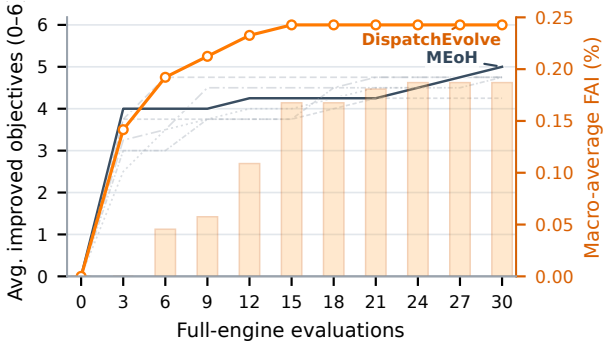


Figure 6: Four-city mean improved-objective count and FAI.

D.2 Policy-Trace Case Study

To make the end-to-end mechanism concrete, we follow one admitted opportunity from discovery to global integration. DispatchEvolve identifies a scenario in which orders wait substantially longer despite ample nearby driver supply and selects AR as the primary objective. Joining this outcome gap with the execution trace localizes the candidate loss to an ETA filter: under the original engine, this stage removes 246 of the 968 source order–driver pairs, and 33 orders are left without an eligible driver. This evidence suggests that the ETA boundary is overly restrictive for this scenario, rather than calling for an indiscriminate expansion of the candidate pool.

Local evolution conditionally relaxes only this ETA filter while preserving the remaining policy sequence and matching solver. We replay the original and updated engines on the same frozen scenario with identical inputs. Figure 7 summarizes the execution change using six illustrative traces. The gray and orange paths follow the same anonymized policy chain until the ETA filter. Traces 3 and 4 show candidates retained for later policies and scoring. Trace 3 reaches matching; Trace 4 remains unmatched. Trace 5 is filtered by both engines, while Trace 6 is removed by a later policy. Thus, the edit changes which candidates reach scoring and matching without changing the solver.

Paired replay validates the intervention while exposing its local trade-off. ETA-filter removals fall from 246 to 237, finally eligible pairs rise from 719 to 728, and orders without an eligible driver

fall from 33 to 31. The number of matched pairs remains 151, but the retained candidate set changes. Relative to E_0, AR increases by 0.756%, CR by 0.570%, and DCAA improves by 1.090%; mean ETA, however, regresses by 0.255%. The candidate is useful for its target scenario but is not by itself a globally balanced engine.

DispatchEvolve subsequently composes this candidate with 15 others: eight target AR, three PCAA, two DCAA, and two CR. Full-engine replay of the resulting 16-candidate composition improves all six objectives relative to E_0: CR by 0.325%, AR by 0.751%, GMV by 0.138%, ETA by 0.285%, PCAA by 0.148%, and DCAA by 0.526%, with signs reported in favorable directions. The composition enters the Top-5 evaluation set but is not the final selected engine. Its global ETA improvement shows that integration can absorb the local ETA cost while preserving the AR gain; because the result is evaluated jointly, we do not attribute that compensation to any single constituent.

This case closes the loop behind the aggregate analyses: trace-grounded discovery turns a scenario-level gap into a localized policy hypothesis, paired replay verifies the execution change and its trade-off, and Pareto-guided composition tests whether the candidate can participate in a globally improving engine.

D.3 Baseline Mechanisms

All baselines share the task, code, base LLM, replay evaluator, and full-engine budget. **RandomSample** proposes independent edits to E_0.

HillClimb replaces the incumbent only after a scalar-reward gain.

MEoH uses LLM operators to retain non-dominated reward–similarity trade-offs [20].

MCTS-AHD balances evolution and exploration in a Monte Carlo tree [23].

OpenEvolve uses multiple islands, diverse references, and candidate migration [15].

ShinkaEvolve uses island evolution, evaluator feedback, novelty sampling, and migration [6].

D.4 Preference-Tuning Objective

For either module, let x be its evidence-conditioned prompt and let y^+ and y^- be the preferred and dispreferred responses. Starting from reference model π_{ref}, define

$$r_\theta(x, y) = \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}. \tag{22}$$

Direct Preference Optimization minimizes

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y^+,y^-)} \log \sigma\big(\beta \left[r_\theta(x, y^+) - r_\theta(x, y^-)\right]\big), \tag{23}$$

where β controls deviation from the reference model.

D.5 Deployment Cost and Serving Overhead

Table 4 gives the measurements summarized in Section 4.5. Each row uses one complete matched-budget run; latency compares E_0 and the evolved engine on the same machine and identical test data.

D.6 Detailed Performance

Tables 5 and 6 expand the complete four-city main comparison in Table 1, including all six baselines and DispatchEvolve. Each objective-metric cell reports the selected performance followed by the sample standard deviation across the five highest-ranked candidate attempts. In the aggregate column, the central value is the selected FAI, whereas the displayed SD and variance are computed from the ungated six-objective average improvement of those Top-5 candidates. All changes are direction-aligned, so positive values indicate improvement; BR remains a feasibility constraint rather than an optimization objective. For completed runs that emit FQS, that output occupies the CR column under the frozen overall performance reporting protocol.

The two tables contain 28 method–city results: seven methods in each of four cities. DispatchEvolve is the only method with positive FAI in every city. The dispersion is a search-process diagnostic rather than uncertainty across independent reruns: Top-5 candidates share data, ancestry, caches, and adaptive search history, and the post-search ranking conditions the sample on strong observed performance. Moreover, a baseline with selected FAI equal to zero can still have nonzero Top-5 dispersion because zero denotes failure of the joint six-objective feasibility rule rather than an absence of candidate variation.

D.7 Detailed Ablation Results

Table 7 gives the metric-level search result behind Figure 3. The six structural configurations use the same City A search sample, base model, opportunity budget, and local-evolution budget. The additional **w/o Critic Fine-tuning** row reports the selected City A result obtained with Gemini 3 Flash serving as the Critic. A composition is counted as feasible only when all six objectives improve and BR improvement is strictly greater than −0.5%.

The ablation artifacts name the sixth objective FQS; it is shown in the CR column under the reporting mapping used throughout the paper. The Feasible/eval. entry for **w/o Critic Fine-tuning** is omitted because only its selected engine is used in this comparison. Removing candidate relations is the only ablation that produces no strictly feasible composition in this run. Greedy composition remains competitive on the sampled search set, but its previously evaluated held-out average falls to 0.037%, compared with 1.080% for the full method. The remaining ablations still discover feasible engines but reduce the best balanced improvement. The variant without Critic fine-tuning reaches an FAI of 0.364%. Together, these results indicate that Critic fine-tuning, trace evidence, critic filtering, and Pareto references affect candidate quality and balance.

D.8 Sensitivity to the Base LLM

We test sensitivity to the general Base LLM. Table 8 compares DispatchEvolve and six baselines under three models using the main evaluation's six-objective FAI. DispatchEvolve retains its Qwen3-8B DPO opportunity critic, isolating the Base LLM used by the remaining components.

Among 21 method–model results, eight have positive FAI and thirteen are zero. DispatchEvolve alone is positive under all three Base LLMs and leads each column. All baselines remain at zero

Table 4: Offline evolution cost and serving overhead in Cities A–D. Tokens are billed totals; latency changes are relative to E_0.

City	Calls	Tokens (M)	Time	Cost (USD)	Latency Δ
City A	1,872	142.0	06:52:48	115.37	+3.38%
City B	1,916	130.1	11:33:42	109.12	+0.68%
City C	1,880	117.4	05:50:59	103.32	+4.98%
City D	1,952	132.6	11:36:50	112.43	+3.33%

Table 5: Detailed performance for Cities A and B. Objective metrics are reported as selected result ± Top-5 sample standard deviation; the aggregate column pairs selected FAI with the SD and variance of the ungated six-objective average improvement (percentage points, $n = 5$).

City	Method	Selected FAI ± SD(raw avg.)	Var(raw avg.)	CR ± SD	AR ± SD	GMV ± SD	ETA ± SD	PCAA ± SD	DCAA ± SD	BR ± SD
A	RandomSample	+0.000000 ± 0.004050	0.0000164	+0.064868 ± 0.018446	+0.045998 ± 0.014844	+0.010097 ± 0.001054	−0.029612 ± 0.003286	−0.083390 ± 0.007476	+0.005272 ± 0.005626	+0.005057 ± 0.004523
	HillClimb	+0.000000 ± 0.034024	0.0011576	+0.263301 ± 0.030948	+0.271710 ± 0.023110	+0.184143 ± 0.018841	−0.702498 ± 0.073161	−0.870778 ± 0.034130	−0.360673 ± 0.061660	+0.232629 ± 0.038539
	OpenEvolve	+0.352458 ± 0.018499	0.0003422	+0.656306 ± 0.011103	+0.511395 ± 0.012057	+0.023672 ± 0.011514	+0.190796 ± 0.014270	+0.003338 ± 0.014812	+0.729241 ± 0.080960	−0.257914 ± 0.042725
	MEoH	+0.000000 ± 0.203687	0.0414884	+0.328242 ± 0.459487	+0.195903 ± 0.437895	−0.079349 ± 0.013427	−0.074266 ± 0.106515	−0.173063 ± 0.092779	+0.779718 ± 0.348867	−0.010114 ± 0.156111
	MCTS-AHD	+0.000000 ± 0.001361	0.0000019	+0.645631 ± 0.001880	+0.719505 ± 0.002382	+0.211298 ± 0.000948	−0.695191 ± 0.002501	−0.850455 ± 0.003523	−0.216012 ± 0.001972	+0.139072 ± 0.002116
	ShinkaEvolve	+0.000000 ± 0.001827	0.0000033	−0.112603 ± 0.009304	−0.046510 ± 0.008154	−0.002967 ± 0.000369	+0.164378 ± 0.009295	−0.081681 ± 0.001012	−0.042034 ± 0.000652	+0.068271 ± 0.002770
	DispatchEvolve	**+0.616000 ± 0.058765**	0.0034533	+1.193054 ± 0.098919	+1.474880 ± 0.182875	+0.290000 ± 0.090386	+0.010930 ± 0.015925	+0.149035 ± 0.125708	+0.578101 ± 0.037853	−0.373932 ± 0.102753
B	RandomSample	+0.000000 ± 0.044354	0.0019673	+0.295077 ± 0.246778	+0.241438 ± 0.220470	−0.032548 ± 0.013218	+0.074896 ± 0.156156	−0.207397 ± 0.201061	+0.260647 ± 0.123697	−0.043968 ± 0.015174
	HillClimb	+0.000000 ± 0.075189	0.0056534	+0.604964 ± 0.196730	+0.462686 ± 0.149861	−0.031403 ± 0.014287	−0.093416 ± 0.088408	−0.459458 ± 0.146224	+0.928795 ± 0.308179	−0.073279 ± 0.020789
	OpenEvolve	+0.000000 ± 0.009320	0.0000869	+0.202905 ± 0.057539	+0.138989 ± 0.053981	−0.034343 ± 0.007968	−0.011553 ± 0.030171	−0.114888 ± 0.051893	+0.386579 ± 0.020135	−0.023449 ± 0.002622
	MEoH	+0.000000 ± 0.074936	0.0056154	+0.705702 ± 0.143189	+0.634956 ± 0.145299	+0.042630 ± 0.009057	−0.137216 ± 0.176848	−0.446457 ± 0.323684	+0.467596 ± 0.165435	−0.140696 ± 0.062090
	MCTS-AHD	+0.000000 ± 0.127368	0.0162226	+0.205547 ± 0.122120	−0.022692 ± 0.082100	−0.075903 ± 0.031178	+0.021041 ± 0.105612	−0.375917 ± 0.145292	+2.120313 ± 0.773955	−0.058624 ± 0.024295
	ShinkaEvolve	+0.000000 ± 0.044677	0.0019960	+0.237817 ± 0.052584	+0.138060 ± 0.027189	−0.030079 ± 0.019086	+0.115042 ± 0.080196	−0.219481 ± 0.079889	+0.777540 ± 0.254379	−0.026381 ± 0.013811
	DispatchEvolve	**+0.300223 ± 0.009249**	0.0000855	+0.717051 ± 0.077614	+0.459291 ± 0.033795	+0.032399 ± 0.046837	+0.124880 ± 0.035584	+0.083626 ± 0.050003	+0.384093 ± 0.032772	−0.171821 ± 0.015368

Table 6: Detailed performance for Cities C and D, following the convention of Table 5.

City	Method	Selected FAI ± SD(raw avg.)	Var(raw avg.)	CR ± SD	AR ± SD	GMV ± SD	ETA ± SD	PCAA ± SD	DCAA ± SD	BR ± SD
C	RandomSample	+0.000000 ± 0.007927	0.0000628	−0.062651 ± 0.089530	−0.105416 ± 0.114142	−0.164251 ± 0.077318	+0.199880 ± 0.114957	−0.014381 ± 0.048804	+0.354582 ± 0.140946	−0.125805 ± 0.049369
	HillClimb	+0.000000 ± 0.019938	0.0003975	+0.170150 ± 0.110295	+0.168582 ± 0.103295	+0.037630 ± 0.023198	−0.043845 ± 0.062858	−0.155014 ± 0.159048	+0.242568 ± 0.062118	−0.041935 ± 0.054624
	OpenEvolve	+0.000000 ± 0.002381	0.0000057	+0.765614 ± 0.008854	+0.628376 ± 0.009094	−0.281557 ± 0.003323	−0.180386 ± 0.002503	−0.386119 ± 0.007532	+0.852833 ± 0.003886	−0.144867 ± 0.002088
	MEoH	+0.000000 ± 0.093483	0.0087391	+0.176684 ± 0.136618	+0.240809 ± 0.108760	+0.122759 ± 0.064511	−0.335784 ± 0.192599	−0.472302 ± 0.195178	+0.630285 ± 0.283587	+0.243986 ± 0.275747
	MCTS-AHD	+0.000000 ± 0.109530	0.0119968	+0.000000 ± 0.190258	+0.000000 ± 0.188287	+0.000000 ± 0.043746	−0.000000 ± 0.162462	−0.000000 ± 0.191473	−0.000000 ± 0.133881	+0.000000 ± 0.293761
	ShinkaEvolve	+0.000000 ± 0.027588	0.0007611	+0.702183 ± 0.093446	+0.574927 ± 0.082606	−0.044479 ± 0.015741	−0.256690 ± 0.032601	−0.152660 ± 0.039639	+0.339729 ± 0.047697	−0.133430 ± 0.015948
	DispatchEvolve	**+0.194502 ± 0.107547**	0.0115664	+0.203514 ± 0.114776	+0.208864 ± 0.118031	+0.018022 ± 0.012590	+0.366632 ± 0.199862	+0.084523 ± 0.053579	+0.285459 ± 0.155220	−0.268324 ± 0.143534
D	RandomSample	+0.000000 ± 0.038511	0.0014831	+0.170619 ± 0.244903	+0.184855 ± 0.234888	−0.034479 ± 0.020752	−0.001542 ± 0.125712	−0.165347 ± 0.203037	+0.178409 ± 0.062224	−0.026032 ± 0.030874
	HillClimb	+0.000000 ± 0.063078	0.0039788	+0.335913 ± 0.123746	+0.461154 ± 0.238857	−0.052319 ± 0.027524	−0.188693 ± 0.082516	−0.418569 ± 0.145913	+0.392768 ± 0.228633	−0.015805 ± 0.020899
	OpenEvolve	+0.000000 ± 0.013883	0.0001927	−0.060003 ± 0.066908	+0.090503 ± 0.194571	−0.035288 ± 0.021716	+0.033411 ± 0.119289	−0.100484 ± 0.169794	+0.134288 ± 0.111702	−0.063222 ± 0.031292
	MEoH	+0.000000 ± 0.055051	0.0030306	−0.057540 ± 0.220913	−0.079254 ± 0.195484	−0.022764 ± 0.011196	−0.043358 ± 0.149748	−0.107486 ± 0.335583	+0.132817 ± 0.140915	−0.024173 ± 0.040839
	MCTS-AHD	+0.000000 ± 0.049157	0.0024164	+0.011983 ± 0.086057	+0.084331 ± 0.188741	−0.055508 ± 0.020814	−0.063294 ± 0.088047	−0.047217 ± 0.062034	+0.616363 ± 0.231037	−0.029751 ± 0.017060
	ShinkaEvolve	+0.000000 ± 0.007994	0.0000639	+0.004901 ± 0.056863	+0.005439 ± 0.049881	+0.006423 ± 0.004830	−0.023425 ± 0.040535	+0.003576 ± 0.045953	+0.016306 ± 0.021047	−0.000930 ± 0.012577
	DispatchEvolve	**+0.290314 ± 0.062975**	0.0039659	+0.124375 ± 0.056074	+0.706001 ± 0.075793	+0.394880 ± 0.150035	+0.022071 ± 0.073624	+0.195865 ± 0.161989	+0.298692 ± 0.079592	−0.056794 ± 0.014198

Table 7: Detailed City A ablation. Values are direction-aligned changes (%) from E_0 on the search set.

Variant	Feasible/eval.	FAI	AR	GMV	ETA	PCAA	DCAA	CR	BR
Full DispatchEvolve	3/16	**+0.616**	+1.475	+0.290	+0.011	+0.149	+0.578	+1.193	−0.374
w/o Critic Fine-tuning	—	+0.364	+0.477	+0.003	+0.570	+0.191	+0.453	+0.492	−0.275
w/o Trace Evidence	13/16	+0.457	+0.826	+0.001	+0.200	+0.361	+0.562	+0.791	−0.454
w/o Opportunity Critic	2/22	+0.498	+1.116	+0.175	+0.004	+0.296	+0.581	+0.817	−0.454
w/o Combination LLM	9/10	+0.542	+1.235	+0.281	+0.082	+0.229	+0.457	+0.970	−0.374
w/o Candidate Relations	0/5	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000
w/o Pareto References	10/24	+0.398	+1.059	+0.247	+0.061	+0.036	+0.246	+0.741	−0.427

with Gemini 3.1 Flash Lite. OpenEvolve leads the Gemini 3 Flash baselines, and RandomSample leads those using Gemini 3.1 Pro.

The Base LLM affects the methods differently. Gemini 3 Flash gives DispatchEvolve its largest point estimate, while Gemini 3.1 Pro yields positive FAI for four of the six baselines; Gemini 3 Flash and Gemini 3.1 Flash Lite do so for one and zero baselines, respectively. An FAI of zero means that the method did not produce a candidate satisfying the joint six-objective feasibility rule, not that every candidate had zero raw metric change. Because the table reports point estimates without independent reruns or dispersion, the observed Base-LLM ordering is descriptive rather than statistically resolved.

Table 8: Base-LLM sensitivity of DispatchEvolve and six baselines. Entries are FAI (%); bold marks the largest value within each method, with all ties bold. DispatchEvolve retains the fixed Qwen3-8B DPO opportunity critic.

Method	Gemini 3 Flash	Gemini 3.1 Flash Lite	Gemini 3.1 Pro
DispatchEvolve	**+0.350260**	+0.193091	+0.293590
RandomSample	0.000000	0.000000	**+0.128590**
HillClimb	**0.000000**	**0.000000**	**0.000000**
MEoH	**0.000000**	**0.000000**	**0.000000**
MCTS-AHD	0.000000	0.000000	**+0.037165**
OpenEvolve	+0.088115	0.000000	**+0.100335**
ShinkaEvolve	0.000000	0.000000	**+0.063429**

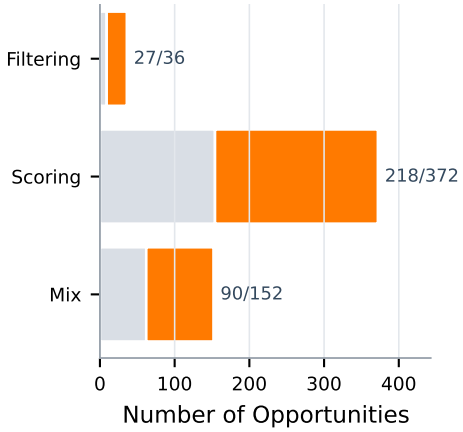


Figure 8: Recorded local-evolution outcomes by policy scope. Bars split successful and unsuccessful outcomes, and labels show successful/total Opportunities.

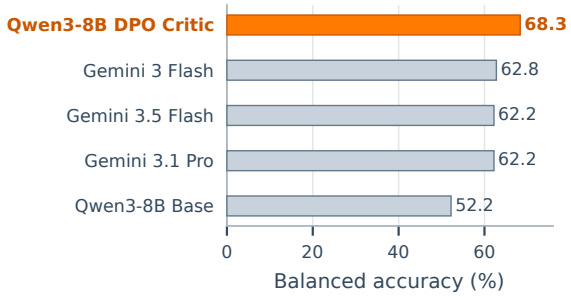


Figure 9: Historical leave-one-out online-uplift assessment performance. Bars report balanced accuracy for five model/deployment interfaces, ordered from highest to lowest. The DPO Critic is highlighted in orange. The retained corpus contains 9 positive and 10 negative realized deployment events.

D.9 Opportunity Distribution across Policy Scopes

We classify each discovered Opportunity from its related policy paths using the frozen analysis rule: a filter-only scope is *Filtering*, a scope containing both a filter and the scoring-parameter configuration is *Mixed*, and the remaining non-filter scope is *Scoring*. Across 982 unique Opportunities, the three groups contain 107 Filtering (10.9%), 603 Scoring (61.4%), and 272 Mixed (27.7%) Opportunities. The Opportunity Critic retains 66, 525, and 180 from these groups, respectively.

Among 560 Opportunities with a local-evolution outcome, Scoring has 372 cases (66.4%), Mixed 152 (27.1%), and Filtering 36 (6.4%). They yield 218, 90, and 27 successful candidates, or 58.6%, 59.2%, and 75.0%, respectively. The Filtering rate rests on only 36 outcomes and varies across cities, so it does not establish a general advantage.

The distribution shows that the search is not confined to a single policy family: most evolution targets scoring behavior, while more than one quarter of evaluated Opportunities require coordinated filter and scoring-parameter changes. Scoring contributes 218 of the 335 successful local candidates (65.1%), Mixed contributes 90 (26.9%), and Filtering contributes 27 (8.1%). Policy scope therefore describes where the search effort and successful candidates arise, while deterministic local replay remains the test of whether an individual edit is effective.

D.10 Online-Uplift Assessment Performance

We evaluate the online-uplift assessment on 19 structurally complete historical A/B experiments. For each target, its complete outcome record is withheld, the input contains only pre-outcome candidate fields, and the model receives eight of the other 18 experiments selected without using their outcomes. Five exact binary calls are aggregated by majority vote. Figure 9 reports balanced accuracy, which averages recall over the realized positive and negative outcomes, for five model/deployment interfaces.

The DPO Critic provides the highest balanced accuracy (68.33%), exceeding the next-best interface by 5.55 percentage points. This small leave-one-out study measures discrimination within the retained historical corpus, not prospective deployment accuracy, and the resulting vote fractions are not calibrated probabilities.

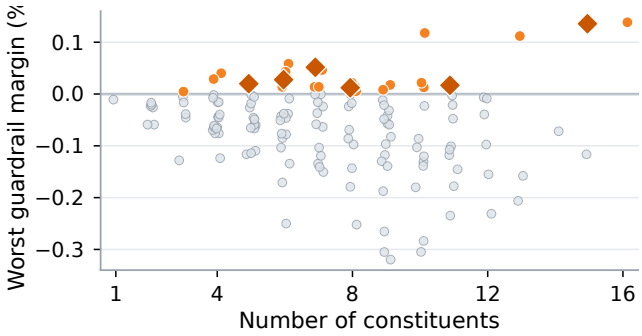


Figure 10: Feasibility of 147 full-engine compositions. Orange points are feasible, navy outlines mark Top-5 evaluations, and diamonds mark final selections.

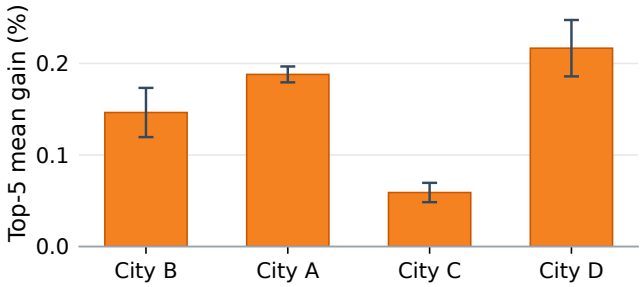


Figure 11: Performance across city batch loads. Cities are ordered by mean OD pairs per batch; bars and error bars show the Top-5 mean seven-metric improvement and sample standard deviation.

D.11 Composition Feasibility

Figure 10 plots all 147 persisted full-engine composition evaluations from the seven completed diagnostic runs. The horizontal coordinate is the number of local candidates assembled into the engine; the vertical coordinate is the worst frozen guardrail margin. Positive margin therefore indicates that every frozen acceptance condition passes.

Under the original acceptance rule, 29 of 147 compositions are feasible. The grouped feasibility rates are 3/30 for 1–4 constituents, 17/64 for 5–8, 6/46 for 9–12, and 3/7 for 13–16. Under the stricter paper criterion requiring all six objectives to improve together with the BR constraint, 25 of 147 compositions remain feasible. The scatter does not show a monotonic size effect: larger candidate sets can create a balanced engine, but they also introduce additional interactions that full replay must reject.

D.12 Performance across Average Batch Sizes

The four cities differ in the order–driver pairs processed per dispatch batch. Figure 11 orders them by average load and reports the mean and sample standard deviation of each city's five strongest compositions.

Performance is not monotonic in average batch size: City D has the largest batch and highest Top-5 mean, while medium-load City C has the lowest. Across four cities, the descriptive Spearman correlation is $\rho = 0.40$ ($p = 0.60$). Without a batch-size intervention, this describes heterogeneity rather than a causal effect.

D.13 Abstracted Prompt Templates

Following the compact presentation style used by the ProfiLLM online appendix,[1] Table 9 reports only the stable prompt structure for five modules. Concrete production feature legends, policy source, data rows, and full instructions are omitted.

[1] https://profillm.github.io/appendix.html#prompts

Table 9: Abstracted prompt templates used by DispatchEvolve. Braced terms denote runtime-injected evidence.

Module	Key prompt structure
Opportunity construction	**Role:** production dispatch analyst. **Task:** identify an actionable scenario and form $o = (s, j, L_o)$. **Inputs:** {TRACE_SUMMARY}, {METRIC_GAPS}, {POLICY_ACTIVITY}, and {SCENARIO_MEMORY}. **Guidelines:** use observable conditions, retain material coverage, distinguish exogenous weakness from an editable policy gap, and name one primary objective. **Output:** scenario rule, objective, related policy set, and concise evidence.
Opportunity critic	**Role:** conservative semantic gate. **Task:** decide whether the proposed opportunity should enter local evolution. **Inputs:** {OPPORTUNITY}, scenario distribution, performance gap, policy evidence, and prior outcomes. **Checks:** actionability, attribution to L_o, evidence sufficiency, and guardrail risk. **Output:** ACCEPT or REJECT with a short evidence-grounded reason.
Local policy evolution	**Role:** bounded dispatch-policy programmer. **Task:** improve primary objective j on D_s. **Inputs:** editable policy source {POLICIES}, scenario definition, replay metrics, previous candidate, and evaluator feedback. **Constraints:** edit only L_o, preserve interfaces and runnability, and respect all local metric guardrails. **Output:** a complete candidate implementation plus a concise change summary.
Relation and composition	**Role:** full-engine integrator. **Task:** identify candidate interactions and propose a globally promising combination. **Inputs:** {LOCAL_CANDIDATES}, modified policies, local effects, relation graph G_r, Pareto references $\mathcal{R}$, and prior full-replay feedback. **Constraints:** exclude hard conflicts, respect required ordering, and never infer global metrics by summing local gains. **Output:** candidate subset, integration order, and short rationale for full-engine evaluation.
Online-uplift assessment	**Role:** pre-deployment experiment reviewer. **Task:** compare two offline-feasible engines for online-test priority. **Inputs:** market context, engine summaries, offline multi-metric results, and historical A/B evidence H. **Constraints:** rank deployment value only; do not alter offline feasibility or Pareto dominance, and do not assume offline gains transfer online. **Output:** preferred engine and a concise evidence summary.