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

Zirui Yuan*
The Hong Kong University of Science and Technology (Guangzhou)
zyuan779@connect.hkust-gz.edu.cn

Tengfei Lyu*
The Hong Kong University of Science and Technology (Guangzhou)
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†
The Hong Kong University of Science and Technology (Guangzhou)
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. On frozen test logs from four cities, DispatchEvolve is the only evaluated method that improves all six objectives over the incumbent in every city. Only 25 of 147 complete-engine compositions improve all six objectives while satisfying the broadcast-rate constraint, confirming that local gains are not reliably additive. Four production A/B tests yield six nominally significant improvements and one nominally significant ETA degradation. These results demonstrate the value of trace-grounded evolution and global integration while underscoring the need for city-level monitoring after deployment.

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, 16 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 [27, 30]. These stages jointly optimize coupled and partly conflicting business metrics, including the completion rate (CR), the accept rate (AR), the pickup 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 [23, 28]. 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

*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

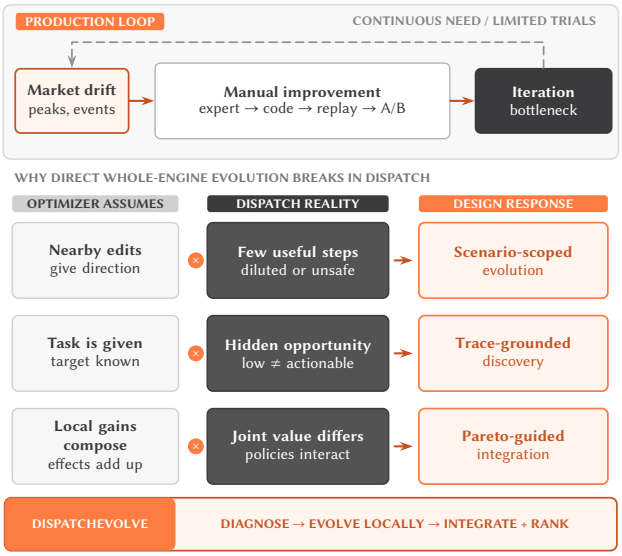

Figure 1: Motivation. Three assumptions behind direct whole-engine evolution fail in a throughput-limited dispatch loop, motivating scenario-scoped evolution, trace-grounded opportunity discovery, and Pareto-guided integration of interacting policies.

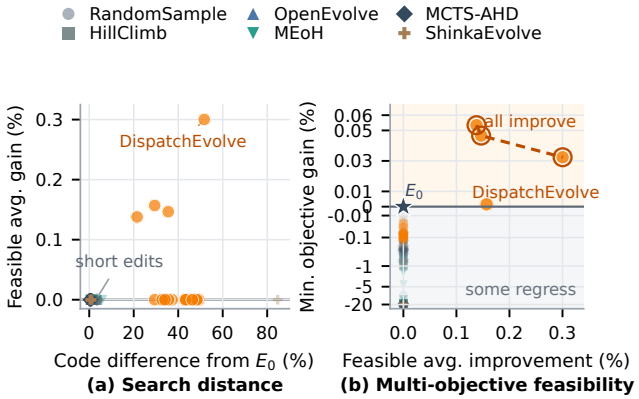

Figure 2: Escaping local optima in multi-objective dispatch-engine evolution. (a) Existing agentic methods predominantly explore short-edit neighborhoods around the expert-tuned engine E_0, whereas DispatchEvolve composes improvements evolved across multiple local regions to explore farther. (b) Only DispatchEvolve achieves positive feasible average improvement while maintaining positive minimum improvement across all six objectives.

by a multi-week online A/B test [12]. This human-intensive workflow scales poorly because evaluating more candidate changes requires additional expert, engineering, and experimentation capacity, leaving many potentially valuable scenarios unexplored.

Recent advances in LLM-driven program evolution offer a promising way to automate the generation and iterative refinement of candidate engine changes. FunSearch [24] demonstrated that LLM can discover new programs through iterative feedback from an evaluator, while AlphaEvolve [21] extended this paradigm to system-level code evolution under multiple metrics. Open implementations such as OpenEvolve and ShinkaEvolve [13, 25] have since made the paradigm broadly accessible. The same evaluator-guided loop has also been applied to automated heuristic design [17, 31, 32, 34, 37]. These approaches are most effective when nearby program edits yield directional feedback, the optimization target is specified, and promising edits can be accumulated without exhaustive search over their interactions. At first glance, a production dispatch engine appears compatible with this paradigm because it is a runnable codebase whose editable regions can be bounded and whose candidate changes can be evaluated across multiple objectives. Yet industrial dispatch does not reliably satisfy the conditions that make evaluator-guided program evolution effective.

Applying program evolution directly 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 have exhausted many broad improvements, leaving gains concentrated in narrow operating conditions that are diluted by unaffected traffic in full-data replay. Edits broad enough to move aggregate metrics, meanwhile, often improve one objective at the expense of another and fail a non-regression guardrail. The search therefore yields few useful candidates under a fixed replay budget; focusing the evaluation could strengthen the signal, but the engine does not specify which scenario or policy to target. Second, high-value opportunities are difficult to discover and attribute across heterogeneous scenarios. Policy effects vary across combinations of time, location, supply-demand conditions, order types, and driver or passenger behavior, creating too many slices to enumerate manually and allowing aggregate metrics to 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. Existing marketplace methods typically predefine a target such as scoring, matching, or repositioning [10, 14, 23, 27, 30, 38], leaving unresolved how to identify a material and actionable scenario, choose the objective to improve, and attribute its gap to the relevant policies. Third, the value of combining new policies cannot be inferred from their isolated gains. An individual policy may improve some objectives while degrading others, yet complementary policies can offset those regressions and achieve a Pareto improvement unavailable to any one policy. Because policies may overlap in scenarios, decision stages, or metrics, their joint effects can reinforce, duplicate, cancel, or conflict. The resulting subset search is exponential, making globally valuable combinations difficult to find within a fixed evaluation budget. Together, these challenges require mechanisms around a general program optimizer that discover actionable targets, evaluate edits where their effects remain visible, and test policy combinations jointly under whole-engine guardrails. Figure 1 maps each failed assumption of direct whole-engine evolution to the corresponding dispatch failure mode and design response.

DispatchEvolve organizes these mechanisms into two connected stages. The first stage, *Trace-Grounded Local Policy Evolution*, constructs opportunities from decision traces by specifying a spatiotemporal scenario, a primary objective, and a relevant policy set. An opportunity critic rejects unactionable candidates, and a general

LLM program optimizer evolves only the relevant policies for each accepted opportunity on scenario-specific replay under per-metric guardrails. The second stage, *Pareto-Guided Global Engine Integration*, represents conflicts and interactions among local candidates in a relation graph, uses a Pareto-guided combinatorial search to assemble complete engines, and re-evaluates each engine on full-data replay. Feasible non-dominated engines enter a cross-round archive, from which an online-uplift assessment trained on historical A/B records selects the next incumbent after each round and the A/B-ready output after the final round. The general program optimizer is thus only the mutation engine inside the first stage; DispatchEvolve contributes the trace-grounded task construction and guarded global integration around it. We instantiate DispatchEvolve on the order-driver scoring stage of DiDi's production engine and evaluate it with offline replay in four cities, component diagnostics, search and serving pilots, complete-composition analysis, a paired policy trace, and production A/B tests in four cities. We make four contributions.

- We formulate continuous dispatch-engine improvement as a constrained multi-objective dispatch-engine evolution problem and present DispatchEvolve, which decomposes direct whole-engine search into opportunity-scoped local evolution and Pareto-guided global integration.
- We propose Trace-Grounded Local Policy Evolution, which constructs optimization opportunities from decision traces by pairing a scenario with a primary objective and a relevant policy set, filters them with an LLM opportunity critic, and evolves only the selected policies on scenario-specific replay under per-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 a *constrained multi-objective dispatch engine evolution* problem. 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 performance archive (Section 3.3) with E_0 and thereafter admits only feasible improvements. At termination, the frozen online-uplift assessment trained from H ranks the non-dominated engines in the final archive and returns the highest-ranked one 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.

Table 1: Core notation used throughout Section 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

3 The DispatchEvolve Framework

3.1 Overview

Under a fixed evaluation budget, direct whole-engine search over coupled objectives yields too few feasible improvements to guide evolution. DispatchEvolve therefore runs for at most R rounds, with $E_1 = E_0$, and decomposes each round into two coupled stages. *Trace-Grounded Local Policy Evolution* (Section 3.2) uses the incumbent E_r's decision traces to discover scenario-specific opportunities and evolves the relevant policies against E_r on D_s under local guardrails. *Pareto-Guided Global Engine Integration* (Section 3.3) assembles the resulting candidates and evaluates complete engines against E_0 on D_{evo}. Candidates satisfying global non-regression in (4), strict improvement in (5), and the fixed 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} from the archive, and the search stops when a round adds no new archive entry.

Only P and L are editable, while the downstream matching service remains fixed. The language model proposes scenarios, opportunities, policy edits, candidate relations, and combinations; deterministic procedures decide runnability, replay metrics, archive eligibility, and Pareto dominance. The general program optimizer is thus confined to local code evolution. Figure 3 summarizes the workflow, Table 1 collects the core notation, and Algorithm 1 in Appendix A gives the complete procedure. Appendix C provides the corresponding finite-budget reachability 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

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

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 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.

$$o = (s, j, L_o), \qquad L_o \subseteq L, \quad (7)$$

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

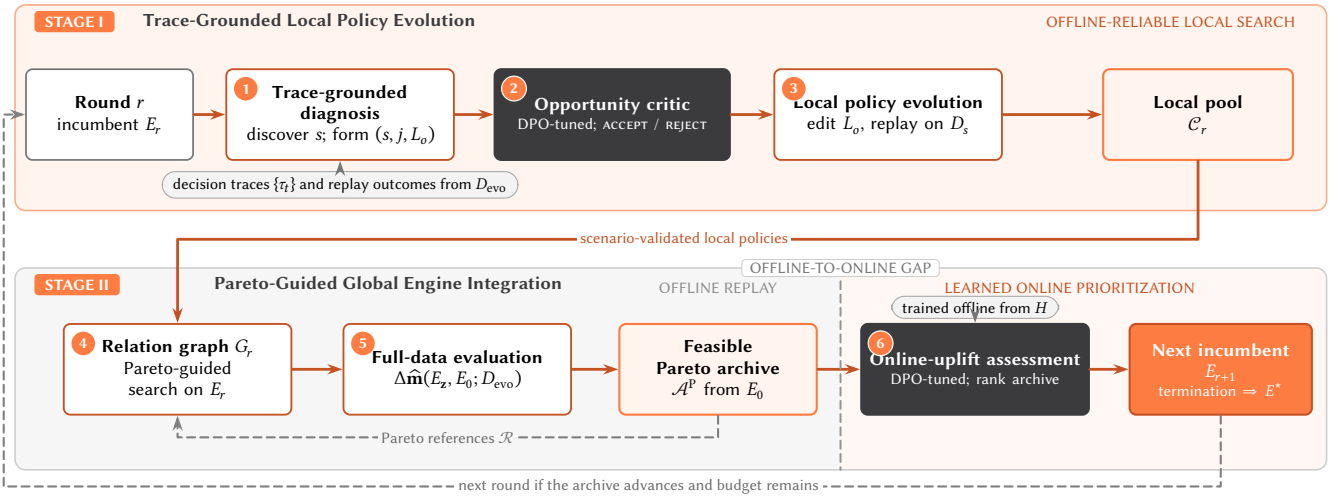

Figure 3: Overview of DispatchEvolve. Stage I discovers and filters trace-grounded opportunities, then evolves locally guardrail-constrained candidates $\mathcal{C}_r$ on scenario replay. Stage II uses cross-round Pareto references to assemble and evaluate complete engines on D_{evo}, retaining globally feasible, non-dominated improvements in $\mathcal{A}^P$. A frozen online-uplift assessment trained from H ranks the archive to select E_{r+1} and the final A/B-ready $E^\star$.

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 $\Delta\hat{\mathbf{m}}(E', E_r; D_s) = \hat{\mathbf{m}}(E'; D_s) - \hat{\mathbf{m}}(E_r; D_s)$. 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:

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

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

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

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 $\mathcal{C}_r = \{c_1, \ldots, c_n\}$, 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 $z_i + z_\ell \leq 1$, 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,

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

where $\text{HV}(\cdot; \mathbf{r}_{\text{HV}})$ is the dominated hypervolume above the fixed reference point (12), so a large $\Delta\text{HV}(E)$ marks an engine whose trade-off the rest of the archive cannot recover; structural difference breaks ties, so that near-duplicate combinations do not crowd out distinct trade-offs. Guided by $\mathcal{R}$, by G_r, and by the offline feedback accumulated so far, the model proposes relation-feasible combinations meant to exploit complementarities and extend the archive with new non-dominated trade-offs.

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,

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

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) [**?**]. 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 $r_\theta(x, y) = \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$. DPO learns π_θ by minimizing

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y^+,y^-)} \log \sigma\big(\beta\left[r_\theta(x, y^+) - r_\theta(x, y^-)\right]\big), \tag{16}$$

where β controls deviation from the reference model. Preference tuning is a one-time offline preprocessing step performed before the engine-evolution loop in Algorithm 1. The trials used to construct preference pairs contribute only training supervision and never enter the candidate pools or Pareto archive of the reported search. After tuning, both decision modules remain frozen throughout all evolution rounds.

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 $\mathcal{S}_r = \mathrm{nd}(\mathcal{A}^{\mathrm{P}})$ and let $v_{\theta_{\mathrm{up}}}(E, E') \in \{E, E'\}$ denote the model's preferred engine. We assign each engine the pairwise win count

$$w_{\theta_{\mathrm{up}}}(E) = \sum_{\substack{E' \in \mathcal{S}_r \\ E' \neq E}} \mathbb{I}\left[v_{\theta_{\mathrm{up}}}(E, E') = E\right]. \tag{17}$$

$\mathrm{Rank}_{\theta_{\mathrm{up}}}(\mathcal{S}_r)$ orders engines by decreasing $w_{\theta_{\mathrm{up}}}(E)$, 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 along the path from offline dispatch-engine evolution to online deployment. The experiments address five questions: (1) whether DispatchEvolve discovers engines that improve multiple dispatch objectives simultaneously; (2) how each component contributes to DispatchEvolve's effectiveness; (3) how locally evolved policies become globally feasible engines; (4) what deployment cost and serving overhead the resulting engines introduce; and (5) whether they remain effective in live marketplaces.

4.1 Experimental Setup

4.1.1 Datasets. We evaluate DispatchEvolve on real-world order-dispatch data from four Brazilian cities spanning heterogeneous marketplace scales and dispatch densities. For each city, we collect two consecutive weeks of dispatch logs. The first week forms the evolution set, D_{evo}, used for dispatch-engine evolution; the second forms the held-out test set, D_{test}.

The production study covers Cities A and B and two additional Brazilian markets, denoted Cities E and F. City identities and absolute business volumes are withheld.

4.1.2 Evaluation Metrics. We use DiDi's replay simulator to evaluate each dispatch engine on historical orders and driver availability. We denote the production dispatch engine for each city by E_0 and report all metrics as relative changes from E_0. We evaluate six dispatch objectives: completion rate (CR), driver answer rate (AR), GMV, estimated time of arrival (ETA), passenger cancel after acceptance (PCAA), and driver cancel after acceptance (DCAA). Higher values are preferred for CR, AR, and GMV, whereas lower values are preferred for ETA, PCAA, and DCAA. Positive values indicate improvement after accounting for metric direction.

We define feasible average improvement as the arithmetic mean of the direction-aligned relative improvements in CR, AR, GMV, ETA, PCAA, and DCAA for an engine that strictly improves all six objectives. If no such engine is found, feasible average improvement is zero. Broadcast rate (BR) is used only as an eligibility constraint: a feasible engine must limit BR degradation relative to E_0 to at most 0.5%.

4.1.3 Baselines. We compare DispatchEvolve with E_0 and six program-search methods spanning distinct optimization paradigms. RandomSample and HillClimb represent stochastic and local search. MEoH is an LLM-guided multi-objective heuristic evolution method that maintains non-dominated candidates. MCTS-AHD uses LLM-guided Monte Carlo tree search for heuristic design, while OpenEvolve and ShinkaEvolve perform population-based program evolution. All methods start from the same city-specific engine E_0, optimize the same dispatch-engine code, receive the same full-engine evaluation budget, and are evaluated by the same replay simulator.

4.1.4 Implementation Details. We use Gemini 3 Flash as the base LLM for all methods. DispatchEvolve additionally fine-tunes Qwen3-8B as the Opportunity Critic. DispatchEvolve runs for ten evolution rounds and optimizes at most ten local Opportunities per round.

4.2 Overall Offline Effectiveness

All engines reported in Table 2 satisfy the BR constraint. DispatchEvolve finds a feasible engine in each of the four cities.

DispatchEvolve attains feasible average improvements of 0.616%, 0.300%, 0.107%, and 0.248% in Cities A–D, respectively, and yields a positive change in every objective. Among the baselines, only OpenEvolve finds a feasible engine, in City A, with a feasible average improvement of 0.352%; no baseline finds a feasible engine in the other three cities. Partial gains do not suffice under the joint objective: for example, MCTS-AHD improves CR, AR, and GMV in City A, but regresses ETA, PCAA, and DCAA, and therefore has zero feasible average improvement. Across the four cities, DispatchEvolve achieves a macro-average feasible improvement of 0.318%, compared with 0.088% for the strongest baseline. Its smallest objective gains are 0.011%, 0.032%, 0.008%, and 0.013% in Cities A–D, showing that the aggregate improvements do not conceal an objective regression.

Table 2: Metric-level offline performance in four Brazilian cities with Gemini 3 Flash. Metric columns report direction-aligned relative percentage improvements over E_0; percentage signs are omitted. Feasible average improvement is zero when no engine satisfies all six objective-improvement conditions.

City	Method	Feasible average improvement	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	**+0.107**	+0.163	+0.210	+0.032	+0.068	+0.008	+0.160
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	**+0.248**	+0.347	+0.516	+0.063	+0.209	+0.013	+0.342

Figure 4 further compares evaluation efficiency using the average best-so-far number of improved objectives among BR-eligible engines. DispatchEvolve improves all six objectives on average after 15 dispatch-engine evaluations, whereas the strongest baseline reaches five after 30 evaluations. Its four-city feasible average improvement emerges after six evaluations and increases from 0.045% to 0.187% by the 24th evaluation, indicating that additional evaluations continue to improve solution quality after feasibility has been reached.

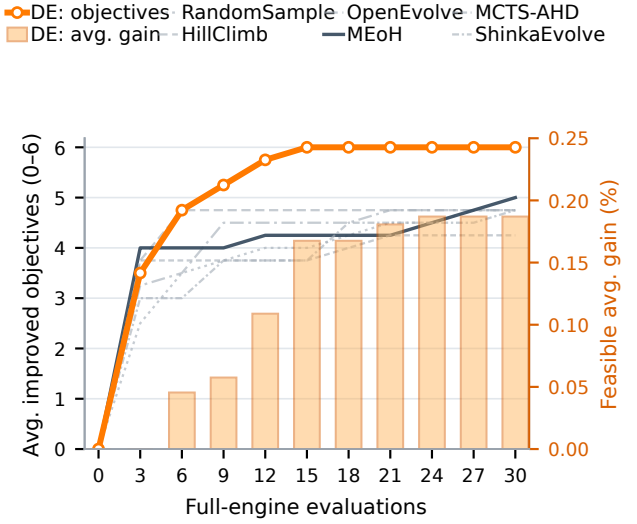

Figure 4: Offline evaluation efficiency across Cities A–D. Curves report the macro-average best-so-far number of improved objectives under the BR constraint; bars report DispatchEvolve's macro-average best-so-far feasible average improvement. Hollow DispatchEvolve markers denote checkpoints containing carried-forward runs.

4.3 Ablation Study

We evaluate five variants of DispatchEvolve on City A. **w/o Trace Evidence** removes policy activity, decision-effect, and performance traces from Opportunity discovery and evaluation 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 explicit conflict and precedence relations during composition and integration. **w/o Pareto References** withholds previously evaluated Pareto solutions from the Combination LLM while preserving Pareto archive maintenance and incumbent selection. All variants otherwise retain the full DispatchEvolve pipeline.

Table 3 reports Avg. Δ for six configurations: the full method and five ablated variants.

The complete method achieves the strongest result. Full DispatchEvolve attains an Avg. Δ of 0.616%, outperforming every ablated variant. The consistent decrease after removing any component indicates that opportunity discovery, candidate assessment,

Table 3: Component ablation on City A. Avg. Δ is the direction-aligned average percentage improvement relative to E_0; percentage signs are omitted.

Variant	Avg. Δ
Full DispatchEvolve	**+0.616**
w/o Trace Evidence	+0.457
w/o Opportunity Critic	+0.498
w/o Combination LLM	+0.542
w/o Candidate Relations	+0.000
w/o Pareto References	+0.398

and global composition make complementary contributions to the final engine.

Candidate relations are essential for integration. Removing candidate relations reduces Avg. Δ to zero, indicating that independently evolved candidates cannot be reliably integrated without explicit conflict and precedence information. Replacing the Combination LLM with greedy composition retains an Avg. Δ of 0.542%, but remains below the full method, showing that relation-aware LLM composition improves upon fixed local-gain ordering.

Evidence and evaluation improve candidate quality. Removing the Opportunity Critic and trace evidence lowers Avg. Δ to 0.498% and 0.457%, respectively, confirming the value of filtering Opportunities and grounding their discovery in execution evidence. Removing Pareto references produces the largest non-degenerate reduction, from 0.616% to 0.398%, highlighting the importance of previously evaluated trade-offs when composing local candidates into a globally effective engine.

4.4 In-depth Analysis

The preceding experiments establish the effectiveness and component contributions of DispatchEvolve. We next examine three questions: what distinguishes an apparent performance gap from an actionable Opportunity and how accurately the Critic identifies it, how the resulting policy edits compose into globally feasible engines, and how an evolved policy changes dispatch execution.

4.4.1 Opportunity Actionability and Critic Accuracy. Across the completed diagnostic runs, 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 5(a) examines whether the observed scene performance gap identifies Opportunities that are easier to optimize. The gap aligns the direction of every objective so that a positive value indicates that the scene underperforms the engine-wide value; scene coverage is the fraction of dispatch batches captured by the scene. Successful and unsuccessful Opportunities overlap across both dimensions. From the lowest to highest gap quartile, the success rates are 63.6%, 66.4%, 60.0%, and 49.3%, respectively. A larger gap therefore indicates greater apparent improvement headroom, but does not predict whether local search will succeed.

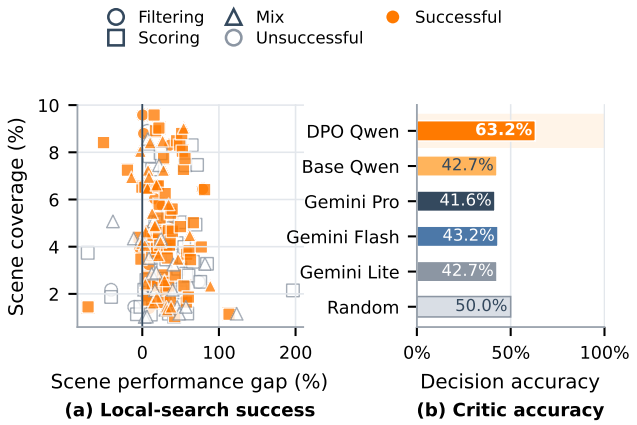

Figure 5: Opportunity actionability and Critic accuracy. Panel (a) shows a deterministic stratified sample of 180 Opportunities from the 560 local searches; marker shape denotes policy scope and color denotes search success. Panel (b) compares the decision accuracy of the DPO-tuned Critic, untuned and general-purpose LLM Critics, and random binary decisions.

Panel (b) evaluates the Critic's outcome-based Opportunity judgment. The DPO-tuned Qwen3-8B Critic reaches 63.2% decision accuracy, compared with 42.7% for the base Qwen3-8B Critic, 41.6%–43.2% for the Gemini Critics, and 50.0% for random binary decisions. This result indicates that supervision from completed local-evolution outcomes improves the Critic's ability to separate actionable Opportunities from apparent gaps that do not yield a valid local improvement.

The searched Opportunities span both major editable stages of the dispatch engine: 372 target Scoring, 36 target Filtering, and 152 target both through Mix. These groups contain 218, 27, and 90 successful searches, respectively. The matching solver remains fixed in these runs; the categories describe the policies evolved by DispatchEvolve. Overall, scene gaps expose where improvement may be available but do not establish actionability. The tuned Critic improves this judgment by learning from prior evolution outcomes, while 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 the evaluation protocol in Section 4.1. Only 25 compositions (17.0%) satisfy both requirements. To examine the mechanism before the 14-reference selection cap becomes binding, Figure 6 focuses on the 128 evaluations proposed while the six-objective Pareto reference archive contained 1–14 entries; 24 of these compositions are feasible.

Within the displayed archive range, the observed feasibility rate rises from 8.3% at 1–4 entries to 14.9%, 20.8%, and 42.9% in the

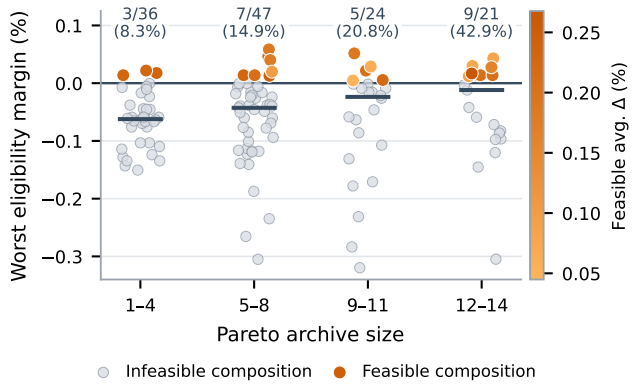

Figure 6: Pareto-archive context and global composition feasibility. The figure shows 128 full-composition evaluations proposed while the archive contained 1–14 entries. Each point's vertical position is the worst margin among the six objective-improvement conditions and the BR constraint; positive values indicate feasible engines. Labels report feasible evaluations over all evaluations in each archive-size bin, navy segments mark median margins, and color encodes the six-objective average improvement for feasible compositions.

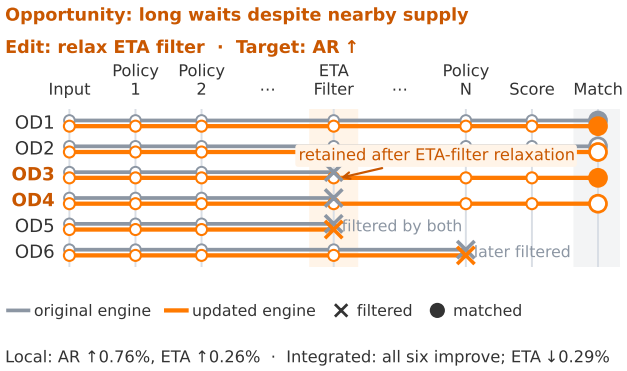

Figure 7: Trace-centered view of one admitted local candidate. Gray and orange denote the original and updated engines, respectively. Policy 1, Policy 2, and Policy N anonymize production policies, while ellipses suppress intermediate policies not material to this case; the full filtering pipeline contains more than ten production policies. The six OD lanes schematically encode representative path types because the retained evidence provides aggregate stage counts rather than row identities. Scene-level effects come from paired replay of the same six batches and 968 source OD pairs; the integrated effect in the footer comes from full-engine composition evaluation.

successive bins. The median worst eligibility margin simultaneously moves from −0.062% to −0.043%, −0.024%, and −0.012%, so later proposals are progressively closer to satisfying all seven conditions. This is a diagnostic association rather than a controlled archive-size intervention, because the archive co-evolves with the search history. Nevertheless, the joint movement of the full margin distribution and feasibility yield shows how a richer Pareto reference context accompanies more globally compatible compositions. The median remains negative even at 12–14 entries, and 12 of 21 compositions in that bin are still infeasible; Pareto guidance therefore improves the search context but does not replace full-engine evaluation.

4.4.3 Policy-Trace Case Study. To make the end-to-end mechanism concrete, we follow one admitted Opportunity from its discovery to global integration. DispatchEvolve identifies a scene 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 an actionable intervention—the ETA boundary is overly restrictive for this scene—rather than an indiscriminate expansion of the candidate pool.

Local evolution therefore conditionally relaxes only this ETA filter while preserving the remaining policy sequence and the matching solver. We replay the original and updated engines on the same frozen scene, comprising six complete batches with identical inputs. Figure 7 makes the resulting execution change explicit. The gray and orange paths follow the same anonymized policy chain until the ETA filter. In the schematic encoding, the updated engine retains path types OD3 and OD4 for downstream policies and scoring; OD3 eventually reaches a match, whereas OD4 remains unmatched. In contrast, OD5 is filtered by both engines and OD6 is removed by a later policy. Thus, the edit changes which candidates reach scoring and matching without changing the solver itself.

The paired replay validates the intervention while exposing its local trade-off. ETA-filter removals fall from 246 to 237, final eligibility rises from 719 to 728 pairs, 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 increases by 0.570%, and DCAA decreases by 1.090%; mean ETA, however, increases by 0.255%. The candidate is therefore useful for its target scene but is not by itself a globally balanced engine.

DispatchEvolve subsequently composes this candidate with 15 complementary candidates: eight further candidates target AR, three target PCAA, two target DCAA, and two target CR. Full-engine replay of the resulting 16-candidate composition improves all six primary 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 their favorable directions. The composition is retained on the Pareto frontier and enters the Top-5 evaluation set, although it is not the final selected engine. Its global ETA reduction shows that integration can absorb the local ETA cost while preserving the candidate's 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 preceding aggregate analyses: trace-grounded discovery turns a scene-level performance gap into a localized policy hypothesis, paired replay verifies the induced execution change and its trade-off, and Pareto-guided composition tests whether the resulting local candidate can participate in a globally improving engine.

4.5 Deployment Cost and Serving Overhead

We evaluate deployment practicality along two dimensions: the one-time offline expenditure required to evolve an engine and the latency overhead of serving the selected engine. Table 4 reports both quantities for the same four cities used in the main offline evaluation.

Table 4: Offline evolution cost and serving overhead in Cities A–D. Tokens are billed tokens. The final column reports the relative percentage change in mean engine latency from E_0; percentage signs are omitted.

City	LLM calls	Billed tokens (M)	Elapsed time	LLM cost (USD)	Mean latency Δ vs. E_0
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

Using Gemini 3 Flash Standard pricing, the four evolution 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.

The selected engines increase mean engine latency by 0.68%–4.98% across the four cities, with a macro average of 3.09%. The benchmark isolates `engine.run_batch` on matched complete batches; the deployed policies require no LLM inference, model loading, or additional service during online dispatch.

The offline expenditure is incurred once, whereas the benefit of a deployed engine recurs with production traffic. Table 5 reports positive relative GMV changes in all four production cities, with nominally significant gains in Cities E and A. These results establish modest evolution cost and low serving overhead, but the withheld absolute business volumes preclude a complete monetary cost–benefit calculation.

4.6 Production A/B Tests

We deploy frozen DispatchEvolve policies in Cities A, B, E, and F and compare them with E_0. The retained analysis windows contain 14, 14, 12, and 7 valid days for Cities A, B, E, and F, respectively. We report the same seven outcomes in every city. Each table entry is the platform-reported signed relative change from E_0, with its reported p-value below. Positive changes are favorable for CR, GMV, BR, and AR; negative changes are favorable for DCAA, ETA, and PCAA. The retained records do not specify randomization units, sample sizes, confidence intervals, or a multiple-testing correction, so we describe $p < 0.05$ results as nominally significant.

Table 5 reports all 28 city–metric estimates: six nominally significant improvements, one nominally significant degradation, and 21 nonsignificant changes. CR and GMV increase in all four cities. The gains are nominally significant in City E (CR +1.07%, $p = 0.0010$; GMV +0.94%, $p = 0.0032$) and City A (CR +0.72%, $p = 0.0191$; GMV +1.32%, $p = 0.0004$), whereas the positive estimates in Cities B and F are not significant. BR increases in Cities A, E, and F but decreases nonsignificantly by 0.27% in City B.

Driver-side gains are more selective. AR increases in all four cities and is nominally significant in City E (+0.97%, $p = 0.0002$). DCAA decreases by 3.09% in City A ($p = 0.0074$), while its changes in the other three cities are not significant. Passenger-side outcomes expose the main deployment trade-off: PCAA does not change significantly in any city. Broadcast ETA remains statistically unchanged in Cities B, E, and F; City B reports a 0.71% decrease ($p = 0.0562$), whereas City A shows a significant 0.82% increase ($p = 0.0003$).

City B adds production coverage but no new nominally significant outcome: its CR, GMV, and AR estimates are positive and its ETA and PCAA estimates decrease, while all seven reported p-values exceed 0.05. In a leakage-controlled retrospective simulation, we withheld each city's online outcome and supplied only its offline evidence and experiments completed earlier than its deployment date. Five repeated Online-Critic calls produced vote fractions of 1.00, 0.20, 1.00, and 1.00 for Cities A, B, E, and F, respectively. At a 0.5 decision threshold, the Critic agrees with the strict realized event—at least one nominally significant improvement and no nominally significant degradation—in two of four cities. Its false positives in City A (significant ETA degradation) and City F (no significant gain) show that this small retrospective test is diagnostic rather than evidence of calibrated deployment reliability.

Overall, the online results show that evolved dispatch policies can deliver nominally significant platform and driver gains under live supply–demand interactions, but not a uniform Pareto improvement. The ETA regression in City A demonstrates why offline promotion must be followed by city-level guardrail monitoring. Across offline replay, mechanism analysis, and production A/B tests, the evidence supports DispatchEvolve's central design principle: useful policy changes are discovered locally, but their value and safety are determined only after global integration and deployment-level evaluation.

Table 5: Relative changes from E_0 in four production A/B tests. Each business-metric cell reports the signed relative change (%) and the platform-reported p-value in parentheses. Arrows mark the favorable direction; boldface marks nominal significance. The Online-Critic score is the fraction of five repeated pre-deployment evaluations that returned YES; it is a vote fraction rather than a calibrated probability.

Stakeholder	Metric	City A	City B	City E	City F
Platform	Completion rate (CR) ↑	**+0.72*** (0.0191)	+0.58 (0.1477)	**+1.07**** (0.0010)	+0.46 (0.2870)
	GMV ↑	**+1.32***** (0.0004)	+0.18 (0.7497)	**+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 rate (AR) ↑	+0.49 (0.1720)	+0.51 (0.1206)	**+0.97***** (0.0002)	+0.37 (0.1441)
	DCAA ↓	**−3.09**** (0.0074)	+1.04 (0.6833)	+1.98 (0.2855)	−0.82 (0.7384)
Passenger	Broadcast ETA ↓	**+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-critic score (pre-deployment) ↑	1.00 (5/5)	0.20 (1/5)	1.00 (5/5)	1.00 (5/5)

$^{*}p < 0.05$, $^{**}p < 0.01$, and $^{***}p < 0.001$. Parentheses in the Online-Critic row report YES/valid calls.

5 Related Work

LLM-driven program evolution and automated heuristic design. A fast-growing line of work places a large language model in a loop with an evaluator to discover programs and heuristics. Building on the code-generation ability of large models [3, 15], FunSearch [24] evolves programs against a scored objective, AlphaEvolve [21] extends this to system-level code artifacts under multiple metrics, and OPRO [31] casts the model itself as an optimizer. A parallel thread automates heuristic design: EoH [17] co-evolves natural-language ideas and code, ReEvo [34] adds reflective evolution, MCTS-AHD [37] searches the heuristic space with Monte Carlo Tree Search, MEoH [32] makes the search explicitly multi-objective, and EvoPrompt [7] evolves prompts rather than code; open frameworks such as OpenEvolve [25] and ShinkaEvolve [13], together with a recent survey [18], document the breadth of the paradigm, which in turn connects to a wider move toward self-improving language agents that iterate with feedback [19, 26, 33]. All of these presuppose a given task, a faithful offline score, and freedom to rewrite and re-score offline. DispatchEvolve treats such an optimizer as one interchangeable component and supplies what a live engine lacks: trace-grounded discovery of what to optimize, Pareto-guided integration under coupled guardrails, and an offline-to-online judgment for what to deploy.

Learning-based dispatching in ride-hailing marketplaces. Order dispatching has been studied mostly as a learning or planning problem over a fixed model class. Early production systems framed it as combinatorial assignment [35]; Xu et al. [30] combine learning and combinatorial planning for large-scale dispatch, and Tang et al. [27] learn a deep value network for multi-driver assignment. A rich multi-agent reinforcement learning literature follows, including mean-field formulations [14], order-vehicle distribution matching [38], knowledge transfer across cities [29], and joint dispatching with fleet management and driver repositioning [8–10, 16], with the DiDi deployment and the broader area surveyed by Qin et al. [22, 23]; related pipeline components such as passenger-demand forecasting [11, 28] feed the same system. These methods improve a preselected component such as scoring, matching, or repositioning, and they assume the intervention target and its model form are given in advance. We instead automatically discover which spatiotemporal scenarios are worth optimizing and evolve the engine's own policy code, subject to non-regression across all coupled business objectives.

Multi-objective and Pareto-based optimization. Our integration stage builds on classical multi-objective optimization [20]. NSGA-II [4], SPEA2 [39], and MOEA/D [36] are canonical evolutionary algorithms for approximating a Pareto front, and the hypervolume indicator [40] is a standard quality measure that also drives selection in methods such as SMS-EMOA [1]; we use it to rank reference combinations. We do not propose a new general-purpose multi-objective solver; our contribution is a framework that embeds Pareto dominance and hypervolume selection inside an LLM-guided engine-integration loop, in which the candidates are complete dispatch engines validated on full-data replay.

Offline evaluation and online experimentation. Because the true objective is a multi-metric online uplift, our design is shaped by the gap between offline replay and online behavior. Online controlled experiments (A/B tests) are the accepted standard for measuring that uplift [12], but they are slow and scarce, which motivates estimating deployment value offline. Off-policy and counterfactual evaluation [2, 5] and offline A/B testing for large systems [6] study exactly this estimation problem. DispatchEvolve keeps the two roles separate by construction: offline replay decides feasibility and dominance, while an online-uplift assessment trained on past experiments only ranks which offline-feasible engine is most worth an actual A/B test, so a replay never stands in for one.

6 Conclusion

DispatchEvolve reframes the continuous improvement of a production dispatch engine as a constrained multiobjective evolution

problem. Its central design choice is to place a general program optimizer inside a larger production workflow. Trace-grounded opportunity construction identifies where an actionable gap exists and which policies should be edited. Scenario-specific replay then supplies focused evidence for local evolution, while relation-aware Pareto integration determines which candidates remain beneficial when assembled into a complete engine. An assessment trained from historical A/B records is used only for final prioritization among engines already shown to be feasible offline.

The empirical evidence supports each part of this design at a different level. On frozen test logs from four cities, DispatchEvolve is the only evaluated method that improves all six objectives over the incumbent in every city. It improves all six objectives on average after fifteen dispatch-engine evaluations, while the strongest baseline reaches five after thirty; its four-city feasible average improvement continues to increase through the twenty-fourth evaluation. Only 25 of 147 complete-engine compositions improve all six objectives while satisfying the broadcast-rate constraint, which confirms that isolated policy gains cannot simply be accumulated. Four production A/B tests produce six nominally significant improvements and one nominally significant ETA degradation. The online results show that evolved policies can create live marketplace gains, but they also establish an important boundary. Offline feasibility is a disciplined promotion criterion rather than a guarantee of online non-regression.

The current evidence is 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 budgets and random seeds, and incorporate calibrated uncertainty and post-deployment feedback into engine prioritization.

References

[1] Nicola Beume, Boris Naujoks, and Michael T. M. Emmerich. 2007. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research* 181, 3 (2007), 1653–1669. https://doi.org/10.1016/j.ejor.2006.08.008

[2] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. 2013. Counterfactual Reasoning and Learning Systems: The Example of Computational Advertising. *Journal of Machine Learning Research* 14, 101 (2013), 3207–3260. https://www.jmlr.org/papers/v14/bottou13a.html

[3] 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

[4] 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

[5] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*. 1097–1104. https://arxiv.org/abs/1103.4601

[6] 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

[7] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers. In *The Twelfth International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2309.08532

[8] John Holler, Risto Vuorio, Zhiwei Qin, Xiaocheng Tang, Yan Jiao, Tiancheng Jin, Satinder Singh, Chenxi Wang, and Jieping Ye. 2019. Deep Reinforcement Learning for Multi-Driver Vehicle Dispatching and Repositioning Problem. In *2019 IEEE International Conference on Data Mining (ICDM)*. 1090–1095. https://doi.org/10.1109/ICDM.2019.00129

[9] Yan Jiao, Xiaocheng Tang, Zhiwei (Tony) Qin, Shuaiji Li, Fan Zhang, Hongtu Zhu, and Jieping Ye. 2021. Real-world ride-hailing vehicle repositioning using deep reinforcement learning. *Transportation Research Part C: Emerging Technologies* 130 (2021), 103289. https://doi.org/10.1016/j.trc.2021.103289

[10] 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

[11] Jintao Ke, Hongyu Zheng, Hai Yang, and Xiqun (Michael) Chen. 2017. Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. *Transportation Research Part C: Emerging Technologies* 85 (2017), 591–608. https://doi.org/10.1016/j.trc.2017.10.016

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Zhe Zhao, Xi Lin, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, Mingxuan Yuan, and Qingfu Zhang. 2024. A Systematic Survey on Large Language Models for Algorithm Design. https://doi.org/10.48550/arXiv.2410.14716 arXiv:2410.14716 [cs.NE]

[19] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html

[20] 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/

[21] 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

[22] Zhiwei (Tony) Qin, Xiaocheng Tang, Yan Jiao, Fan Zhang, Zhe Xu, Hongtu Zhu, and Jieping Ye. 2020. Ride-Hailing Order Dispatching at DiDi via Reinforcement Learning. *INFORMS Journal on Applied Analytics* 50, 5 (2020), 272–286. https://doi.org/10.1287/inte.2020.1047

[23] 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

[24] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 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

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

[26] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html

[27] 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

[28] 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

[29] Zhaodong Wang, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. 2018. Deep Reinforcement Learning with Knowledge Transfer for Online Rides Order Dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*. 617–626. https://doi.org/10.1109/ICDM.2018.00077

[30] 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

[31] 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

[32] 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

[33] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2210.03629

[34] 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

[35] 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

[36] Qingfu Zhang and Hui Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (2007), 712–731. https://doi.org/10.1109/TEVC.2007.892759

[37] 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

[38] 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

[39] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. TIK-Report 103. Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland. https://doi.org/10.3929/ethz-a-004284029

[40] 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

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. The notation follows Table 1.

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. 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.

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 $E^\star$

$E_1, E^\star \leftarrow E_0, E_0$; $\mathcal{A}^{\mathrm{P}} \leftarrow \{E_0\}$ with reference point $\mathbf{r}_{\mathrm{HV}} = -\rho\mathbf{1}$; 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
  $\mathcal{O}_r, \widetilde{M}_r \leftarrow \textsc{OpportunityCritic}(\widetilde{\mathcal{O}}_r, M_r)$
  $\mathcal{C}_r \leftarrow \emptyset$
  for all o selected from $\mathcal{O}_r$ within the per-round opportunity budget **do**
    $\mathcal{C}_o, \widetilde{M}_r \leftarrow \textsc{LocalPolicyEvolution}(E_r, o, D_s, \widetilde{M}_r)$ ▷ $\mathcal{C}_o$: 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 $\mathcal{A}^{\mathrm{P}}$ (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**
      $\mathcal{A}^{\mathrm{P}} \leftarrow \mathrm{ND}\big(\mathcal{A}^{\mathrm{P}} \cup \{E_{\mathbf{z}}\}\big)$
    end if
  end while
  $E_{r+1} \leftarrow \mathrm{Top}\Big(\mathrm{Rank}_{\theta_{\mathrm{up}}}\big(\mathrm{nd}(\mathcal{A}^{\mathrm{P}})\big)\Big)$
  $E^\star \leftarrow E_{r+1}$
  if $\mathcal{A}^{\mathrm{P}}$ gained no new engine this round **then break**
end for
return $E^\star$

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,

$$\mathcal{B}^+_{r,\eta} = \left\{ E \in \mathcal{B}_\eta \;\middle|\; \begin{array}{l} \nexists\, A \in \mathcal{A}_r^{\mathrm{P}} \;:\; A \succ_P E, \\ \exists\, A \in \mathcal{A}_r^{\mathrm{P}} \;:\; E \succ_P A \end{array} \right\}. \tag{18}$$

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 $\mathcal{I}_{k-1}$ be the search history that has not yet hit the target. Assuming uniform positive per-history lower bounds,

$$\begin{aligned} \Pr\big(\mathsf{L}_k \mid \mathcal{I}_{k-1}\big) &\geq p_{\mathrm{loc}} > 0, \\ \Pr\big(\mathsf{P}_k \mid \mathsf{L}_k, \mathcal{C}_k, \mathcal{I}_{k-1}\big) &\geq p_{\mathrm{P}} > 0, \end{aligned} \tag{19}$$

the conditional-probability decomposition gives $\Pr(\mathrm{P}_k \mid \mathcal{J}_{k-1}) \geq p_{\mathrm{loc}} p_{\mathrm{P}}$, 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 (Finite-budget hit bound). *If $\mathcal{B}^+_{r,\eta}$ is nonempty and the bounds in* (19) *hold, then over the first T equal-cost macro-trials*

$$\Pr\Big(\bigcup_{k=1}^{T} \mathrm{P}_k\Big) \geq \Phi_T(p_{\mathrm{loc}} p_{\mathrm{P}}) \xrightarrow[T\to\infty]{} 1. \tag{20}$$

Proof. For any history that has not yet hit the target, the conditional probability that the next macro-trial again misses is at most $1-p_{\mathrm{loc}}p_{\mathrm{P}}$, by the decomposition of (19). Chaining this bound over $k = 1, \ldots, T$, the probability that all T trials miss is at most $(1-p_{\mathrm{loc}}p_{\mathrm{P}})^T$, and the complement is exactly $\Phi_T(p_{\mathrm{loc}}p_{\mathrm{P}})$. Since $p_{\mathrm{loc}}p_{\mathrm{P}} > 0$, 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 finite-budget advantage). *Fix a pool $\mathcal{C}$ containing integrable material and give Pareto integration and a comparison baseline the same integration budget; let E_k^{P} and E_k^{B} be their validated candidates at trial k. If Pareto integration has per-history hit lower bound p_{P} and the baseline, greedy or fixed-weight integration, has per-history hit upper bound q_{B} with $p_{\mathrm{P}} > q_{\mathrm{B}}$, then over T integration trials*

$$\begin{aligned}\Pr\big(\exists k \leq T : E_k^{\mathrm{P}} \in \mathcal{B}^+_{r,\eta} \mid \mathcal{C}\big) &\geq \Phi_T(p_{\mathrm{P}}) \\ &> \Phi_T(q_{\mathrm{B}}) \\ &\geq \Pr\big(\exists k \leq T : E_k^{\mathrm{B}} \in \mathcal{B}^+_{r,\eta} \mid \mathcal{C}\big).\end{aligned} \tag{21}$$

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

$$\begin{aligned}\Pr\Big(\bigcup_{k=1}^{T} \mathrm{P}_k\Big) &\geq \Phi_T(p_{\mathrm{loc}}p_{\mathrm{P}}) > \Phi_T(q_{\mathrm{G}}) \\ &\geq \Pr\Big(\bigcup_{k=1}^{T} \mathrm{G}_k\Big).\end{aligned} \tag{22}$$

Proof. Φ_T is strictly increasing on $[0,1]$. The per-trial bounds propagate to the cumulative statements by Proposition C.1 applied under the fixed pool and budget, using $\Pr(\text{no hit in } T \text{ trials}) \geq (1-q)^T$ for the upper-bounded side, so substituting $p_{\mathrm{P}} > q_{\mathrm{B}}$ and $p_{\mathrm{loc}}p_{\mathrm{P}} > q_{\mathrm{G}}$ into Φ_T yields the two strict comparisons. □

The two inequalities in Proposition C.2 name exactly the alternatives the second stage was built against: (21) against greedy or fixed-weight combination on a shared pool, and (22) 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

These statements are conditional by design. Estimating p_{loc} requires repeated equal-budget first-stage trials judged on whether they supply a constructible feasible improvement; estimating p_{P} and comparing it with q_{B} requires matched hit-frequency comparisons on a shared candidate pool and common integration budget. The current ablation in Section 4.3 instead reports one frozen run per variant on one dataset, so it is a component diagnostic and does not estimate these probabilities. The target $\mathcal{B}^+_{r,\eta}$ is also an offline surrogate defined on replay data rather than the true online Pareto front. The propositions guarantee neither recovery of the full front nor discovery of a globally optimal combination, and full replay never replaces a production A/B test.