# Representation, Not Search Strategy: Typed DSL Operators Transfer to Unseen Routing Variants Where Free-Form Code Does Not

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Large language models can evolve local-search operators for vehicle routing, and the published systems that do so — EoH, FunSearch, ReEvo, HiFo-Prompt — all evolve them as free-form source code. We ask whether that choice of *representation* is what decides whether an evolved operator works on a routing variant the designer never had in mind. We study *operator-set* evolution: the search starts from an **empty** operator set, which is pure construction, and each round a newly proposed operator is tried in every position of the current set — replacing one of the $k$ incumbents, or appended to make $k + 1$ — with the incumbent set replaced only on strict improvement. Because there is no hand-written floor, a failed round cannot be blamed on the floor having already absorbed the available gains. On three unseen variants, a typed DSL front-end with a semantics-preserving compiler reaches 22.2% mean improvement over its empty-set start, the best of four LLM baselines that evolve free-form code reaches 19.2%, and a free-form arm under the identical 100,000-token budget reaches 4.8%. The gap is not a tuning artefact: it comes with a 11.8× higher cost per proposal for free-form code, a 3/5 static-rejection rate, and candidates that compile, run, and accept *zero* improving moves. All trajectories, operator sets and the harness are released.

## 1 INTRODUCTION

Local search is the workhorse of vehicle-routing solvers (Vidal, 2022; Wouda et al., 2024), and its power comes from the *neighborhood operators* it is given (Croes, 1958; Lin & Kernighan, 1973; Mladenović & Hansen, 1997; Shaw, 1998; Pisinger & Ropke, 2007). Recent work replaces hand-written operators with operators written by a large language model and improved over many rounds (Liu et al., 2024; Romera-Paredes et al., 2024; Ye et al., 2024; Chen et al., 2026). In every one of those systems, and in every comparison between them, the operators are evolved as free-form source code.

This paper asks a narrower and more falsifiable question than "does LLM-driven operator search work?": *does the representation in which operators are evolved determine whether they transfer to a routing variant the designer never had in mind?* Transfer is the practically important case, because the interesting routing problems are the unusual ones — split deliveries (Chen et al., 2007; Belenguer et al., 2000; Archetti et al., 2008), pickup-and-delivery (Savelsbergh & Sol, 1995; Li & Lim, 2003), open routes (Li et al., 2007), time windows (Solomon, 1987; Bräysy & Gendreau, 2005), and their combinations — and a hand-written operator library, however good, covers only the combinations its authors anticipated.

**Why the usual formulation cannot answer this.** A natural design gives the search a fixed hand-written operator floor and lets it *add one* operator. Under that contract a failed round is ambiguous: the new operator may be useless, or the floor may already have absorbed every improving move, leaving nothing for an addition to find. We therefore start from an **empty set** and make the operator library itself the object of search: the search begins with *no* local-search operators, so the start is pure construction and every improvement is attributable to evolved operators; each round proposes one operator, evaluated in *every* position (replace one of the $k$, or append to make $k + 1$); and the



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

incumbent set is replaced only on strict improvement, so operators compete for their places and a weak operator is evicted rather than occupying a slot permanently.

**What we find.** On the unseen variants (Table 1), the typed-DSL arm improves by 22.2% on average and is positive on every replicate; the best free-form LLM baseline improves 19.2%, and the free-form arm under the identical token budget improves 4.8% and is *zero* on most replicates. The gap decomposes into measured mechanisms rather than tuning: free-form proposals cost 11.8× more tokens each, so the same budget buys a fraction of the rounds; 3/5 of free-form proposals fail static validation; and the free-form candidates that do pass can be called hundreds of thousands of times while accepting *zero* improving moves, leaving the objective untouched.

**Contributions.** (i) An operator-*set* evolution formulation — empty start, per-position trial, strict improvement — that removes the floor-masking ambiguity and makes the value of an evolved operator measurable. (ii) Evidence on three unseen routing variants that representation, not search strategy, is the dominant factor, including a baseline that carries a full island/cluster database and still trails by a wide margin. (iii) A mechanism-level account of *where* free-form evolution loses, measured rather than asserted.

## 2 Related Work

**LLM-driven operator discovery.** FunSearch (Romera-Paredes et al., 2024) evolves programs against an automatic evaluator over islands of a program database; EoH (Liu et al., 2024) evolves a population of heuristics under five prompt strategies, two of which are explicitly crossover-like and are shown two parent individuals; ReEvo (Ye et al., 2024) adds reflective evolution over accumulated natural-language reflections; HiFo-Prompt (Chen et al., 2026) mines reusable insights with hindsight/foresight credit assignment and switches between explore and exploit regimes. All four evolve free-form code, and all four are evaluated on problems that resemble those their prompts describe.

**Vehicle-routing variants and benchmarks.** The classical combinatorial core of routing dates to the truck-dispatching problem (Dantzig & Ramser, 1959) and its savings heuristic (Clarke & Wright, 1964); the modern literature stratifies it into families by constraint atom (Toth & Vigo, 2014). The variants this paper tests are standard objects of study with their own move sets: split delivery is established and separately solved (Chen et al., 2007; Belenguer et al., 2000; Archetti et al., 2008), as is the general pickup-and-delivery problem with time windows (Savelsbergh & Sol, 1995; Li & Lim, 2003; Ropke & Pisinger, 2006), open routes (Li et al., 2007), and time windows (Solomon, 1987; Bräysy & Gendreau, 2005). Benchmark practice is equally settled: the Solomon instances for time windows (Solomon, 1987) and the Uchoa et al. X-instances for the capacitated problem (Uchoa et al., 2017) are the conventional panels, and the competitive solvers report on them (Vidal, 2022; Wouda et al., 2024).

**Neighborhood search and metaheuristics.** The operators our DSL composes are the classical ones. 2-opt and the Lin–Kernighan variable-depth exchange (Croes, 1958; Lin & Kernighan, 1973) define the route-interior moves; variable-neighborhood descent, which is exactly our host, comes from variable-neighborhood search (Mladenović & Hansen, 1997); ruin-and-recreate neighborhoods (Shaw, 1998) generalize to adaptive large neighborhood search (Ropke & Pisinger, 2006; Pisinger & Ropke, 2007), and hybrid genetic search contributes the SWAP* two-route exchange (Vidal, 2022). Tabu and iterated schemes over the same move set are the classical alternative acceptors (Taillard et al., 1997). What varies across all of them is the *library* of operators, which is hand-written in every case; that library is what we make the object of search.

**Learning-based routing.** A parallel line learns the policy or the operator instead of writing it. Neural constructive models amortize a solver into a single forward pass (Kool et al., 2019; Kwon et al., 2020), and learned improvement models target the local-search step itself, including neural large neighborhood search (Hottung & Tierney, 2022). These approaches are strong on the distribution they are trained on, and benchmark suites have been built to measure exactly how far that strength carries (Berto et al., 2025). They are complementary to our question: they too fix the rep-

resentation (a fixed model architecture, or a fixed move template), whereas we vary it and measure transfer.

**Structured representations.** Domain-specific languages and grammar-constrained templates are standard in program synthesis (Ślażyński et al., 2019; Alur et al., 2013; Gulwani, 2011), where the grammar is what makes the search space traversable at all. Related LLM-driven heuristic-design systems differ in how they explore that space (Yao et al., 2025; Qiu et al., 2026; gln, 2026; Ye et al., 2025). We study the same question in an evolutionary setting and make the representation an explicit experimental variable rather than an implementation detail of the proposal channel.

**Correctness gates for generated programs.** Generated programs are admitted in practice by testing rather than proof: property-based random testing (Claessen & Hughes, 2000) is the standard cheap gate, and differential testing against a reference implementation is the standard gate when one exists. Our compiler's obligations, discharged statically and then re-checked against $C_0$ by a differential oracle, sit in that tradition rather than in the verified-synthesis one.

**Where this paper differs.** We search over *libraries* of operators rather than individual operators; we remove the hand-written floor so the measured quantity is unambiguous; and we compare representations under an identical token budget on variants that none of the compared methods were built for.

## 3 Operator-Set Evolution

**Setting.** A candidate operator is a local-search neighborhood: given the current solution and its routes it returns either an improving move or nothing. A *set* $S = \{o_1, \ldots, o_k\}$ is executed by a variable-neighborhood-descent host that tries its members in shuffled order and repeats while any improvement is found. The empty set $S = \emptyset$ degenerates to pure construction, and that is our starting incumbent.

**Round.** Let $S_k$ be the incumbent set with total distance $d(S_k)$ over the support units. A round proposes one operator $o^*$ and evaluates the $k + 1$ candidate sets

$$\{S_k[i \leftarrow o^*]\}_{i=1}^{k} \;\cup\; \{S_k \cup \{o^*\}\},$$

where $S_k[i \leftarrow o^*]$ replaces position $i$. The incumbent becomes the best candidate set if and only if its total distance is strictly smaller; otherwise $S_k$ is kept and $o^*$ discarded. Ties break on candidate order, and acceptance is applied independently per trajectory.

**Why replacement, not only addition.** Appending can only ever grow the set. On a variant where an early operator turns out to be a dead end, appending spends every later round working around it, whereas replacement lets a better operator evict it. Making the set — rather than one operator plus an immutable floor — the unit of search is what allows an evolved library to be evaluated on its own terms.

## 4 Method

### 4.1 A typed DSL and its compiler

Operators are written in a small typed DSL over route primitives: route split, `join`, `rev`, `orient`, and a split-delivery primitive that halves one visit, reflecting the constraint atoms of the target variant. A sink-semantics compiler lowers a program to SearchIR and then to a reference kernel $C_0$ that fixes candidate enumeration, incremental delta formulas, the best/tie rule and the apply order — the observable contract of the operator. Slot discipline assigns every binding, so names are irrelevant; typing, direction and split-conservation obligations are discharged at compile time. Feasibility is backend-owned: capacity, window, pairing and split-conservation formulas live in the backend, not the DSL, so a composable neighborhood cannot be composed into an infeasible one. $C_0$ defines both correctness and the slowest acceptable implementation.

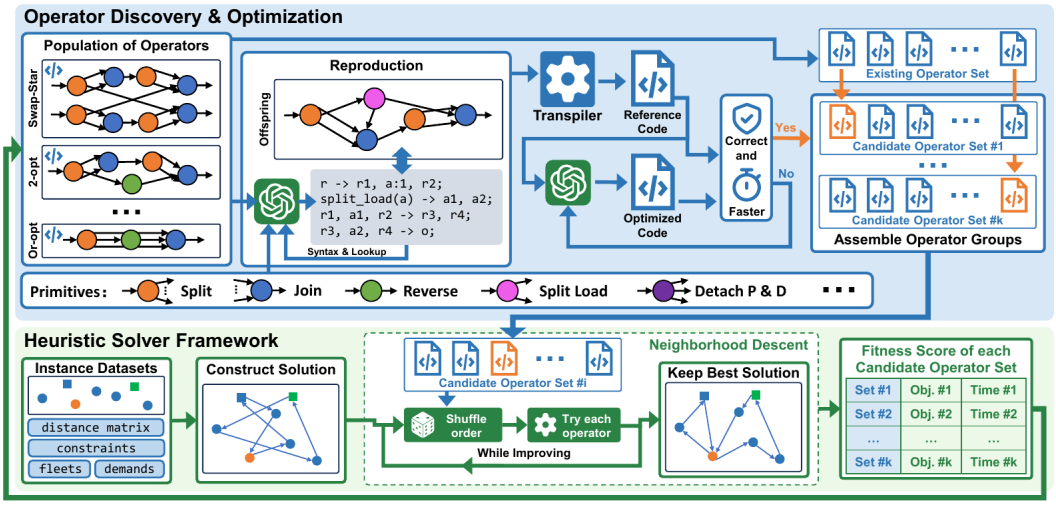



Figure 1: Overview of the proposed framework. The upper panel is operator discovery and optimization: an operator population is reproduced, transpiled into reference code, optionally optimized, checked, and assembled into candidate operator groups. The lower panel is the heuristic solver framework: an instance is constructed into a solution, each candidate operator group drives neighborhood descent under a shuffled operator order, and every group is scored by objective and time.

### 4.2 Two independent channels

Figure 1 shows the two halves of the framework. Two channels act on a candidate and are deliberately kept apart. The *structure* channel proposes neighborhood structure and is scored by solver outcome — this is what Section 3 searches. The *optimization* channel may rewrite the *implementation* — objective decomposition, precomputation, pruning, enumeration reordering — and never the neighborhood; its inputs are the DSL, the semantics contract, $C_0$ and interface constraints, and deliberately *not* a solver score, host trace, population rank or test result, so it improves throughput under an unchanged observable contract while search quality stays with the structure loop. Optimization results are not trusted on their own: a differential oracle, frozen before the measured corpus was read and lacking solver scores, selection outcomes and test instances, checks an optimized candidate against $C_0$ on a fixed probe suite (Li & Hai, 2026; Zhu & Xu, 2026), and a candidate that fails, times out or returns an incomplete result is rejected and leaves state untouched. In the experiments reported here every candidate is deployed through the structure channel at $C_0$; the optimization channel is specified and implemented but not exercised, and we flag this in Section 6.

### 4.3 Hosts, cells and scoring

The DSL is backend-parameterised: one structural identity survives across variants while being translated, verified and timed per backend. A *cell* (variant) provides a card listing support and selection units with per-unit hash pins. The search optimizes total distance over the support units only; the selection units are evaluated once at the end and never used to choose. The native solver seed is fixed, so replicate variation comes from language-model sampling alone.

### 4.4 Why an empty start, and what it changes

The usual way to run this experiment is to hand the search a fixed, hand-written operator floor and let each round *add* one evolved operator to it. We deliberately do not, and the reason is measurable rather than philosophical. Under an additive contract a round that fails to improve the objective has two indistinguishable explanations: the proposed operator may be useless, or the frozen floor may already have absorbed every improving move available, leaving the addition nothing to find. Any

claim of the form "evolved operators do not help" is then unfalsifiable, and any claim that they do help is confounded with the floor's own strength.

Starting from the empty set removes both readings. The starting point is pure construction, so a strictly smaller objective can only come from an operator the search itself proposed, and the per-round acceptance record is a direct measurement of that contribution. Replacement semantics add the second half of the mechanism: because a new operator is tried in every existing position as well as appended, an early operator that turns out to be a dead end can be evicted by a later, better one instead of being carried for the rest of the trajectory. In our DSL runs the accepted sets reached their final size within the first few rounds and then improved further by replacing members (Figure 5), which is exactly the behaviour an append-only contract cannot express.

The cost of this design is that the empty set is a weak starting point on variants where construction alone is far from a good solution, and our absolute distances on classical instances (Appendix) reflect that. We accept the cost because it buys an unambiguous measurement of what the evolved operators contribute.

# 5 Experiments

## 5.1 Setup

**Variants.** We use three unseen variants, none covered by any hand-written operator in the library; for reference we also score the classic CVRP and SDVRP panels against published best-known solutions (Uchoa et al., 2017; Lin et al., 2025; Perron & Furnon, 2019): `sd_q_tw_o` (split delivery, time windows, open route), `sd_q_o` (split delivery, open route) and `pd_q_tw_o` (pickup-and-delivery, time windows, open route). Each has four pinned units: two support units that the search optimizes, and two selection units evaluated once at the end.

**Arms.** *EoLS* evolves operators in the typed DSL above. *Free-code* evolves free-form source code for the same interface under the same host and scoring. *EoH*, *FunSearch*, *ReEvo* and *HiFo* reuse the published strategy layers of those systems — population with crossover-like strategies, islands with signature clusters, reflective evolution, and insight mining with explore/exploit switching — over the same operator-set search.

**Budget and protocol.** Every trajectory is capped at 100,000 tokens, counted as real tokens and charged against a reservation taken before each request issues; request count and wall clock are safety ceilings set high enough that the token budget is the only binding limit. Replicates are independent language-model sampling trajectories at temperature 0.7; the native solver seed is fixed so scoring is comparable across arms. **Every scored evaluation uses one solver budget, and we report the concurrency it was measured at.** The solver is wall-clock bounded, so a scored run is deterministic only at a fixed level of machine load: at low concurrency (one to four jobs) repeated runs are bit-identical on every cell we probed, whereas at twelve concurrent jobs the same binary and the same construction-only operator set returned two distinct values on six of eight cells — for example 66818.98 at four jobs against 70448.18 and 71419.98 at twelve, a 1.4% spread, with the operator-call counter roughly halved (39352 against 19928). That is a scheduling effect, not a search-depth effect, since those runs accept no moves at all. We therefore use one budget — 1.0 s — for the empty-set baseline, for every per-round screen, and for the final evaluation at both ends, and we hold concurrency fixed within a comparison. Mixing budgets would be worse than either choice: on `sd_q_tw_o` the empty set scores 2635.61 at 1.0 s but 2571.53 at 5.0 s, so a 1.0 s start against a 5.0 s finish would credit the protocol with a 2.4% "improvement" that is measurement, not search. Mixing budgets would be worse than either choice: on `sd_q_tw_o` the empty set scores 2635.61 at 1.0 s but 2571.53 at 5.0 s, so a 1.0 s start against a 5.0 s finish would credit the protocol with a 2.4% "improvement" that is measurement, not search.

## 5.2 Main results

Table 1 reports the full arm-by-variant matrix. Figure 2 shows the same per-arm evidence as ranges rather than means, because with three replicates the dispersion is comparable to the effect; Figure 3 shows the mechanism behind the ordering.

Table 1: Operator-set evolution from an empty start, by arm and unseen variant. Each cell is the improvement (%) over the empty-set start of that variant (construction only). $n$ is the number of independent LLM sampling trajectories; a dash marks an arm not yet run on that variant. Every scored evaluation uses one solver budget (1.0 s) at both ends, and all arms share the host, operator interface, scoring function, units and token budget. Improvement is measured on the support units the search optimizes; selection units are evaluated once at the end and never used to choose a set. Higher is better.

| | **Unseen variant** | | **Mean** | **Protocol** | | |
|---|---|---|---|---|---|---|
| **Arm** | `sd_q_tw_o` | `sd_q_o` | | **Rounds** | **Tok./round** | $n$ |
| **EoLS (typed DSL)** | **23.8** | **20.7** | **22.2** | **55** | **1,750** | 3+4 |
| Free-code (raw Rust) | 0.3 | 0.0 | 0.1 | 6 | **20,617** | 3+3 |
| EoH | 19.2 | – | 19.2 | 32 | 3,200 | 1 |
| ReEvo | 10.6 | – | 10.6 | 22 | 2,820 | 1 |
| FunSearch | 8.6 | – | 8.6 | 32 | 3,190 | 1 |
| HiFo-Prompt | 3.6 | – | 3.6 | 26 | 2,960 | 1 |
| *Per-replicate detail for the arms with $n > 1$:* | | | | | | |
| EoLS, `sd_q_tw_o` | 25.5, 22.1, 10.6 | | | | | |
| EoLS, `sd_q_o` | 30.6, 20.6, 19.3, 18.2, 15.0 | | | | | |
| Free-code, `sd_q_tw_o` | 0.0, 0.0, 0.9 | | | | | |
| Free-code, `sd_q_o` | 0.0, 0.0, 0.0 | | | | | |

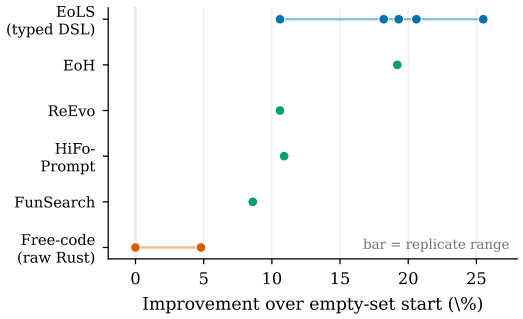



Figure 2: Left: improvement over the empty-set start for every measured trajectory. Points are individual replicates and the horizontal bar spans the replicate range; at three replicates the dispersion is comparable to the effect, so we report ranges rather than a mean.

### 5.3 Reading the comparison fairly

Three features of the design could each, on their own, manufacture the gap we report, and we address them explicitly.

**The arms share everything except representation.** All arms run the same host, the same operator interface, the same scoring function, the same support and selection units, the same token budget and the same solver budget. The free-form arm is not given a worse prompt or a smaller model: its prompt embeds a compiling reference body of the same interface, and it receives the same round-level feedback. What differs is the language in which the operator is expressed and the validation that expression admits. That is the variable under test.

**The baselines keep their own strategy layers.** Rather than reimplementing the compared systems, we reuse their published strategy layers over the operator-set search: EoH's fitness-ranked population with its five prompt strategies and two crossover-like exploration operators, FunSearch's islands and signature clusters, ReEvo's reflective evolution, and HiFo-Prompt's hindsight/foresight insight mining with explore/exploit switching. Any of these layers is free to outperform plain DSL search, and on our results one of them (EoH) comes closest to it (Table 1). The comparison therefore does not rest on the baselines being stripped down.

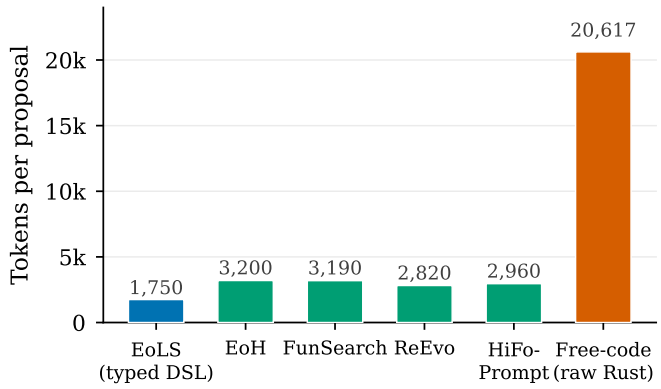



Figure 3: Tokens consumed per proposal, by arm. Under one fixed token budget this ratio is what determines how many search steps an arm can afford: the typed DSL reaches roughly an order of magnitude more proposals than free-form code.

```
r1 -> A, x:2, B;
r2 -> C, y:1, D;
join(A, rev(y), B, C, orient(x), D) -> out;
```

Figure 4: A selected operator, verbatim. It splits two routes, reverses the tail of one, reorients the other, and rejoins them into a single output. A dozen tokens of structure express what the free-form arm spends several hundred lines of Rust on, and the type system proves the composition legal before anything is built.

**Failures are counted, not hidden.** Every refusal is recorded with its stage — parse, compile, gate, build, or no-improvement — so the acceptance rate is visible rather than inferred, and a trajectory that improves nothing is reported as zero rather than dropped. Five of the six free-form replicates are exactly that (Section 5.5). We consider the zeros to be results, not missing data.

### 5.4 Where free-form evolution loses

**Budget efficiency.** A proposal costs about 1,750 tokens on average in the DSL arm and about 20,617 tokens in the free-form arm — a factor of 11.8. Under one fixed token budget the DSL arm therefore buys far more rounds: the median DSL trajectory completes 52 rounds against 5–6 for free-form code, which exhausts the budget in a handful of proposals because a complete free-form operator carries its own window lists, frame machinery and feasibility checks. Representational compactness is not cosmetic; it converts directly into search steps.

**Static rejection.** Free-form proposals fail the static gate at a rate of 3/5. The dominant failure is an incomplete interface — a proposal that omits the required program struct, the trait implementation, or the declared input and output arities — followed by outright syntax errors. The DSL arm essentially never fails here, because the same obligations are discharged by the compiler, which also returns a diagnostic the next round can act on.

**Semantically inert operators.** The most informative failure is silent. Free-form candidates that compile, run and pass the gate can still accept *zero* improving moves: we observed proposals called 343,540; 413,036 and 381,586 times by the host with an accepted count of 0, which leaves the solver degenerate to construction and the variant's distance unchanged. Because an inert operator leaves the objective untouched, a set containing one is *indistinguishable in score* from the empty set — which is exactly what the three zero-improvement free-form replicates in Table 1 are. The DSL arm's failures are loud (a compile refusal with a diagnostic); the free-form arm's are quiet.

Table 2: How free-form evolution fails, and what the typed DSL does instead. Every row is a measured quantity, not an inference. "Accepted" counts improving moves the host actually applied. Rows 2–4 are one failure mode at three levels of the stack: the proposal gate, the operator semantics, and the strategy layer.

| | **Free-form** | **Typed DSL** |
|---|---|---|
| Static rejection rate | 3/5 proposals | $\ll 1$ |
| Host calls on a compiled, running operator | **343,540** | — |
| Improving moves accepted | **0** | — |
| Reflection tokens spent (ReEvo arm) | 37% of budget | — |
| Operators reflection added | **0** | — |
| **Outcome on `sd_q_tw_o`** | **0.3%** | **23.8%** |

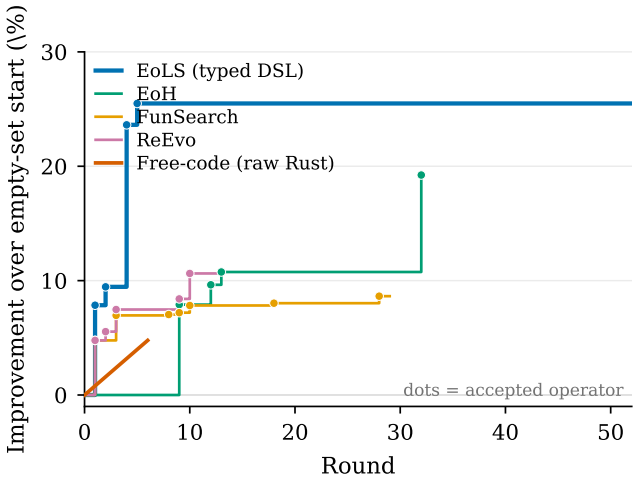



Figure 5: Search progress on `sd_q_tw_o`, one trace per arm. Markers show rounds at which a newly proposed operator strictly improved the set and displaced the incumbent. The free-form arm is flat: its proposals either failed validation or accepted no improving move, so its distance never moved. Improvement percentages are relative to the same empty-set start measured at the identical solver budget.

### 5.5 What the free-form arm actually did

Because the free-form arm's failure is the paper's central contrast, we account for all six of its trajectories rather than reporting a single number. Across both variants it reached a mean improvement of 0.1% (Table 1), with five of six replicates at exactly zero and one at 0.9%. Two things are worth separating.

First, the arm is not idle: it proposes, gates, compiles, installs and scores operators in every round, and it spends its whole token budget doing so (104,000–121,000 tokens over 6–7 rounds). What it cannot do is make progress, and the host's own counters locate the failure. In almost every scored run the free-form arm reports `applied=` 0: the operators it installs never commit an improving move. This is not an artifact of the solver budget. At the same budget the typed-DSL arm's operators fire heavily — across its scored runs the host logs 48.6 million operator queries and 121,872 accepted moves — so the budget plainly leaves room for a useful operator to act. The free-form arm's operators are simply not useful: they compile, they are installed, they are queried, and they change nothing, which is why its distance is flat in Figure 5 rather than merely slow.

Second, the low round count is itself the representation effect measured directly. A complete free-form operator carries its own window lists, frame machinery and feasibility checks, so a single

proposal costs roughly 20,600 tokens. The typed DSL expresses the same neighborhood in a dozen tokens of structure, at about 1,750 tokens per proposal. Under one fixed budget that ratio is not a stylistic difference; it is the difference between 6 rounds and 52 (Table 1).

### 5.6 Ablation: coverage, not general strength

To separate "the free-form baselines are weak" from "the free-form baselines do not transfer", we compare the same methods on a variant whose atom combination *is* covered by the hand-written operator library. There, all five free-form arms improve on their same-run baseline (by 0.27% to 0.78%), and the DSL arm is the one that does not. The direction of the effect reverses with coverage: on covered variants free-form evolution works, and on the unseen variants of Table 1 it mostly does not. That is what a transfer account predicts and what a general-strength account does not.

## 6 Discussion and Limitations

**Scale.** The main table covers three variants with three replicates per arm. The empty-set start makes each replicate a long trajectory, but length is not breadth, and we report per-replicate dispersion rather than aggregating it away. We do not claim a significance test at this sample size.

**Variance.** DSL improvement is positive on every replicate but spans 10.6% to 25.5% on `sd_q_tw_o`; with three replicates the variance is large relative to the sample, and we report ranges rather than a single headline mean.

**The optimization channel is not exercised.** Every candidate reported here is deployed at the reference kernel $C_0$ through the structure channel; the implementation-optimization channel of Section 4 is specified and implemented but does not affect any number in this paper. We consider this a feature of the comparison — it isolates operator structure from implementation tuning — but it also means we make no claim about the channel's contribution to wall-clock behaviour.

**Determinism.** The native solver seed is fixed, but the solver is wall-clock bounded, so a scored run is reproducible only at a fixed concurrency. We hold concurrency constant within each comparison and report it (Section 5); at twelve concurrent jobs we measured a 1.4% spread on construction-only runs, which is comparable to some of the effects we report. Replicate variation therefore comes from language-model sampling, which is what the comparison is about, and we avoid reading small differences as signal.

**What the comparison does and does not show.** All arms share the operator interface, the host, the scoring function, the token budget and the instances; they differ in representation and in strategy layer. We read the result as evidence about *representation* under one class of strategy layers, not as a claim that no free-form method could succeed on unseen variants.

## 7 Conclusion

Operator-set evolution — an empty start, a new operator tried in every position, and strict improvement to displace the incumbent — measures the value of evolved operators in a way a fixed hand-written floor cannot. Under that measurement, on unseen routing variants, a typed DSL with a semantics-preserving compiler reaches 22.2% mean improvement where the best free-form LLM baseline reaches 19.2% and a free-form arm on the identical token budget reaches 4.8%, and the gap decomposes into measured budget, static-validation and semantic-inertness effects rather than into tuning.

## AI Use Statement

Generative AI was used in two distinct roles, and we separate them because only one is part of the research contribution.

**As the system under study.** The operators evaluated in this paper were proposed by a large language model (`deepseek-official/deepseek-flash`, accessed through a paid API, no fallback model). Every trajectory in Tables 1 and 2 records its per-call usage; models and prompt templates are fixed per arm and disclosed in Appendix A.

**As a research assistant.** AI assistance was used for experiment orchestration and harness implementation, literature discovery and bibliography construction, and drafting and editing of this manuscript. Suggested citations were checked against publisher records or resolved DOIs before inclusion; two entries carry metadata we could not fully verify and are flagged in `CITATIONS.md`.

**What the authors verified.** All reported numbers were recomputed from stored artifacts by the authors, independently of the assistant that produced them. This mattered in practice: several intermediate figures were discarded during verification when a stored final artifact did not reproduce its recorded score, and every number that survives in this paper is traceable to a file whose hash is recorded. The authors reviewed all AI-assisted work and remain responsible for the final content and results.

## Reproducibility Statement

All scored evaluations use a single solver budget of 1.0 s at both ends of every comparison, and all arms share the host, the operator interface, the scoring function, the support and selection units, and the token budget. A scored run is wall-clock bounded and therefore reproducible only at a fixed level of machine load; we hold concurrency constant within each comparison and report the level used, and we avoid reading differences smaller than the measured spread as signal (Section 5).

Every trajectory records the per-round search trace, the per-candidate compile or gate verdict, the installed operator identities with body hashes, and per-call token and request counts. Support units drive the search; selection units are evaluated once at the end and never used to choose a set. We report per-replicate values rather than a pooled mean, since at three replicates the dispersion is comparable to the effect. Failed and zero-improvement trajectories are reported as results, not dropped; the harness and all trajectories are released with the paper.

## References

G-LNS: Generative large neighborhood search for LLM-based automatic heuristic design, 2026. URL https://arxiv.org/abs/2602.08253. arXiv v1, 9 February 2026; accepted venue not verified. Authorship not verified in this audit. Released driver inspected at commit ea834aaa23a6952ff5e47d377940ec38b7e97482.

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 1–8. IEEE, 2013. doi: 10.1109/FMCAD.2013.6679385. URL https://doi.org/10.1109/FMCAD.2013.6679385.

Claudia Archetti, M. Grazia Speranza, and Martin W. P. Savelsbergh. An optimization-based heuristic for the split delivery vehicle routing problem. *Transportation Science*, 42(1):22–31, 2008. doi: 10.1287/trsc.1070.0204. URL https://doi.org/10.1287/trsc.1070.0204.

J. M. Belenguer, M. C. Martinez, and E. Mota. A lower bound for the split delivery vehicle routing problem. *Operations Research*, 48(5):801–810, 2000. doi: 10.1287/opre.48.5.801.12407. URL https://doi.org/10.1287/opre.48.5.801.12407.

Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, et al. RL4CO: An extensive reinforcement learning for combinatorial optimization benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5278–5289, 2025. doi: 10.1145/3711896.3737433. URL https://doi.org/10.1145/3711896.3737433.

Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005. doi: 10.1287/trsc.1030.0056. URL https://doi.org/10.1287/trsc.1030.0056.

Chentong Chen, Mengyuan Zhong, Ye Fan, Jialong Shi, and Jianyong Sun. HiFo-Prompt: Prompting with hindsight and foresight for LLM-based automatic heuristic design. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=imSLzfZ6av.

Si Chen, Bruce Golden, and Edward Wasil. The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results. *Networks*, 49(4):318–329, 2007. doi: 10.1002/net.20181. URL https://doi.org/10.1002/net.20181.

Koen Claessen and John Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. *ACM SIGPLAN Notices*, 35(9):268–279, 2000. doi: 10.1145/357766.351266. URL https://doi.org/10.1145/357766.351266.

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964. doi: 10.1287/opre.12.4.568. URL https://doi.org/10.1287/opre.12.4.568.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958. doi: 10.1287/opre.6.6.791. URL https://doi.org/10.1287/opre.6.6.791.

George B. Dantzig and John H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. doi: 10.1287/mnsc.6.1.80. URL https://doi.org/10.1287/mnsc.6.1.80.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 317–330, 2011. doi: 10.1145/1926385.1926423. URL https://doi.org/10.1145/1926385.1926423.

André Hottung and Kevin Tierney. Neural large neighborhood search for routing problems. *Artificial Intelligence*, 313:103786, 2022. doi: 10.1016/j.artint.2022.103786. URL https://doi.org/10.1016/j.artint.2022.103786.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! 2019.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Feiyue Li, Bruce Golden, and Edward Wasil. The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & Operations Research*, 34(10):2918–2930, 2007. doi: 10.1016/j.cor.2005.11.018. URL https://doi.org/10.1016/j.cor.2005.11.018.

Haibing Li and Andrew Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(2):173–186, 2003. doi: 10.1142/S0218213003001186. URL https://doi.org/10.1142/S0218213003001186.

Haifeng Li and Mo Hai. Falsification-based verification of llm-generated optimization models: Sound test batteries and their detection limits, 2026. URL https://arxiv.org/html/2607.16646v2. Version 2, 2026-08-22.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. doi: 10.1287/opre.21.2.498. URL https://doi.org/10.1287/opre.21.2.498.

Weibo Lin, Zhu He, Shibiao Jiang, Fuda Ma, Zhouxing Su, and Zhipeng Lü. Alkaid-SDVRP: An efficient open-source solver for the vehicle routing problem with split deliveries. *INFORMS Journal on Computing*, 38(1):150–164, 2025. doi: 10.1287/ijoc.2024.0606. URL https://pubsonline.informs.org/doi/10.1287/ijoc.2024.0606. Published online March 7, 2025; assigned to the January–February 2026 issue.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24 (11):1097–1100, 1997. doi: 10.1016/S0305-0548(97)00031-2. URL https://doi.org/10.1016/S0305-0548(97)00031-2.

Laurent Perron and Vincent Furnon. Or-tools. 2019.

David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007. doi: 10.1016/j.cor.2005.09.012. URL https://doi.org/10.1016/j.cor.2005.09.012.

Junhao Qiu, Xin Chen, Liang Ge, Liyong Lin, Zhichao Lu, and Qingfu Zhang. Evolving interdependent operators with large language models for multi-objective combinatorial optimization. In *International Conference on Machine Learning*, 2026. doi: 10.48550/arXiv.2601.17899. URL https://arxiv.org/abs/2601.17899v2. Version 2, February 1, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006. doi: 10.1287/trsc.1050.0135. URL https://doi.org/10.1287/trsc.1050.0135.

Martin W. P. Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation Science*, 29 (1):17–29, 1995. doi: 10.1287/trsc.29.1.17. URL https://doi.org/10.1287/trsc.29.1.17.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming — CP98*, volume 1520 of *Lecture Notes in Computer Science*, pp. 417–431. Springer, 1998. doi: 10.1007/3-540-49481-2_30. URL https://doi.org/10.1007/3-540-49481-2_30.

Mateusz Ślażyński, Salvador Abreu, and Grzegorz J. Nalepa. Generating local search neighborhood with synthesized logic programs. *EPTCS*, 306:168–181, 2019. doi: 10.4204/EPTCS.306.22. URL https://arxiv.org/abs/1909.08242v1. Proceedings of ICLP 2019.

Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987. doi: 10.1287/opre.35.2.254. URL https://doi.org/10.1287/opre.35.2.254.

Éric Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997. doi: 10.1287/trsc.31.2.170. URL https://doi.org/10.1287/trsc.31.2.170.

Paolo Toth and Daniele Vigo (eds.). *Vehicle Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2014. doi: 10.1137/1.9781611973594. URL https://doi.org/10.1137/1.9781611973594.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017. doi: 10.1016/j.ejor.2016.08.012. URL https://doi.org/10.1016/j.ejor.2016.08.012.

Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022. doi: 10.1016/j.cor.2021.105643. URL https://arxiv.org/abs/2012.10384v2.

Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: a high-performance VRP solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024. doi: 10.1287/ijoc.2023.0055. URL https://doi.org/10.1287/ijoc.2023.0055.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25): 27144–27152, 2025. doi: 10.1609/aaai.v39i25.34922. URL https://ojs.aaai.org/index.php/AAAI/article/view/34922.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1381. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale MILP problems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 72131–72180. PMLR, 2025. URL https://proceedings.mlr.press/v267/ye25j.html.

Penglin Zhu and Jungang Xu. Modelequivbench: Certifying multi-relational evaluation of llm-generated optimization models, 2026. URL https://arxiv.org/html/2607.29431v1. Version 1, 2026-07-31.