Rethinking Last-Mile Routing at Scale

Near-Linear Planning on Commodity Hardware

Martin Vizzolini

Abstract

This paper presents a practical architecture for last-mile delivery routing at scales reaching one million stops under realistic operational conditions, including vehicle capacity, package volume, route stop limits, and time windows. Unlike conventional systems that require pre-partitioning or large-scale infrastructure, the proposed system addresses the full fleet planning problem through a single coherent planning pipeline on commodity hardware. Physical feasibility constraints (vehicle loading limits and full stop coverage) are strictly enforced; soft operational constraints (time windows, route stop limits, route duration) are subject to measurable, bounded violation rates, in line with standard last-mile practice where perfect compliance is not physically achievable under realistic travel-time variability.

The system combines parallel constraint-aware clustering, constraint-aware initial allocation, distributed neighbor-based rebalancing, and fast route-level optimization to produce fleet plans that preserve global coherence without a centralized monolithic planner. Evaluated against the public Amazon Last Mile Routing Research Challenge dataset [6, 7], and under a shared external measurement protocol based on OSRM and Google Maps, the architecture reduces aggregate measured route distance by 23.3% and route count by 11.1% relative to the released Amazon baseline routes, with a mean depot-level distance reduction of 17.59%. The comparison is conducted on the same set of stops, under the same measurement procedure, and reflects differences in routing structure under an external metric; it does not claim equivalence with Amazon's internal operational objective, which is not publicly specified.

In an extended scaling experiment, the same system processes one million stops in approximately 20 minutes on a commodity laptop, exhibiting near-linear empirical runtime growth. The contribution is architectural rather than theoretical: we show that large-scale routing becomes tractable when computation is organized into bounded, composable stages, enabling efficient planning without input-size caps or specialized infrastructure.

1 Introduction

Large-scale vehicle routing is commonly implemented through centralized optimization pipelines whose practical limits are shaped not only by algorithmic complexity but also by how computation is organized. Many commercial systems impose explicit caps on input size, and even enterprise-grade solvers typically require pre-zoning or staged dispatch workflows to handle large workloads [5, 9, 8].

This work begins from a different premise: the number of delivery stops should be treated as a scaling dimension rather than a fixed upper bound, and large routing workloads should remain solvable on commodity hardware without specialized infrastructure. Under this premise, the proposed system was progressively developed from a fast single-route optimizer into a parallel planning pipeline that preserves global fleet coherence across clustering, route construction, and boundary-level rebalancing.

The system was initially developed and evaluated in the context of the Amazon Last Mile Routing Research Challenge [6, 7], a public benchmark comprising 17 depots with instances ranging

from approximately 8,000 to 174,000 stops, totaling over one million stops and more than 2.5 million packages. Beyond this benchmark, the same architecture was extended to a single-depot instance of one million stops, completing the workload in approximately 20 minutes on a standard laptop.

The experimental results show two main outcomes. In the benchmark setting, the proposed system reduces total measured route distance and route count relative to the released Amazon baseline under a shared external measurement protocol. In the scaling experiment, wall-clock runtime grows near-linearly from benchmark scale to one million stops, with the system processing approximately 821 stops per second.

The contribution of this paper is architectural. Rather than proposing a new exact solver or a refinement of existing metaheuristics, it documents a practical planning system in which route-level optimization, parallel clustering, caching, and distributed rebalancing are organized as cooperating bounded stages. This structure explains both the observed scaling behavior and the compatibility with commodity hardware. The paper does not claim linear-time complexity for the general Vehicle Routing Problem; the near-linear characterization is empirical and refers to the observed runtime pattern under the tested architecture and workload range.

2 Related Work and Positioning

Vehicle Routing Problems (VRPs) have been studied extensively in both exact and heuristic forms. Because the general VRP family remains computationally hard, exact methods are typically limited to small or specially structured instances, and large routing systems rely on heuristics, decomposition strategies, or hybrid pipelines [3].

The literature relevant to this work spans three directions. The first focuses on local-search and metaheuristic methods for classical variants such as the Capacitated VRP (CVRP). The second develops fast subroutines for specific pipeline stages, such as linear-time split procedures whose scalability is limited to restricted components of a broader solver [4]. The third emphasizes engineering scalability: controlling memory growth, pruning neighborhood evaluations, and reorganizing computation so that runtime remains tractable as instance size increases [3, 1]. The present work belongs primarily to this third direction.

2.1 Large-scale routing

Prior work has established that very large routing instances become tractable when scalability is treated as an architectural property. Arnold et al. showed that constraining neighborhood search, avoiding dense cost representations, and reducing the cost of key optimization stages yields strong performance on instances up to 30,000 customers [3]. Accorsi and Vigo extended this perspective to instances of up to one million customers under CVRP-style constraints [1]. These results demonstrate that million-scale routing is computationally feasible, but remain closer to classical CVRP than to operational last-mile planning with heterogeneous fleets, package volume, time windows, and route-boundary rebalancing.

The present work is aligned with this systems-oriented research direction but differs in emphasis: it targets operational last-mile planning under a globally coherent architecture rather than benchmark CVRP alone.

2.2 Linear-time claims in routing

The phrase “linear time” requires careful use in routing contexts. In the literature, it typically refers to a linear-time subroutine within a larger pipeline, a solver whose dominant operations

scale approximately linearly under fixed assumptions, or an empirical runtime curve that appears near-linear over a tested range [4, 3]. This paper adopts the third interpretation: the claim is empirical, based on observed wall-clock growth from benchmark-scale depots to one million stops, and does not imply a proof of linear-time complexity for the general VRP.

2.3 The Amazon benchmark context

The Amazon Last Mile Routing Research Challenge is primarily framed around route sequencing quality and practical driver knowledge rather than pure total-distance minimization [6, 7]. The comparison reported in this paper is therefore performed under a shared external measurement protocol rather than under Amazon's internal objective function, which is not fully specified. The reported improvements reflect differences in solution structure under a constraint-aware formulation, not a strict objective-for-objective comparison. This distinction is developed further in Section 6.

2.4 Deployment constraints in commercial systems

Beyond algorithmic difficulty, the practical routing landscape is shaped by deployment constraints. Commercial routing APIs impose per-request limits that reflect underlying computational and economic costs rather than inherent algorithmic impossibility. Table 1 summarizes publicly documented constraints across representative providers.

Table 1: Publicly documented limits or practical scaling constraints in representative commercial routing APIs (verified April 2026).

Provider / product	Constraint (documented or practical)	Pricing / posture	Implication
Google Route Optimization API	Size-dependent timeouts; guidance for requests above 10,000 shipments	Per-shipment billing; execution deadlines	Large requests remain sensitive to both timeout policy and cost
Onfleet	Approximately 2,000 tasks per optimization request	SaaS dispatch platform	Optimization remains bounded at the request level
NextBillion.ai	Limits documented in the low-thousands range for tasks / locations per problem	API with quotas and rate limits	Practical planning units remain in the low-thousands
RouteSavvy API Plus	Approximately 1,000 locations, with support for up to approximately 50 vehicles	Bounded API product	Typical planning units remain in the low-thousands
Mapbox Optimization API	Small optimization units in the stable API; larger-scale support exposed separately by version	Usage-based API pricing / evolving product surface	Practical usage depends strongly on endpoint/version
openrouteservice optimization	Public optimization endpoint with strict caps	Public API with rate limits	Compute is intentionally bounded on the shared public service
HERE Tour Planning API	Enterprise tour-planning offering with workflow-dependent limits	Enterprise routing platform	Larger problems are supported, but practical usage depends on product mode and deployment workflow

These constraints matter because they shape how routing is carried out in practice. In many operational settings, the limiting factor is not that routing is impossible, but that available tools require the problem to be subdivided before optimization begins. This introduces artificial territorial boundaries, weakens global coherence, and creates avoidable inefficiencies near route borders.

2.5 Positioning

Table 2 positions the present work relative to publicly reported large-scale routing systems.

Table 2: Comparison with publicly reported very-large-scale routing work.

Work	Problem class	Scale	Hardware	Relevance to the present work
Accorsi & Vigo [1]	CVRP with engineered neighborhoods	Up to 1M customers	AMD Ryzen 5, 16 GB, single thread	Strong prior art for scale; differs in constraint richness and last-mile framing
Arnold et al. [3]	CVRP with pruning and restricted memory	Up to 30K customers	Single machine	Conceptual prior art: scalability is architectural
Gibbons et al. [4]	Linear-time split subroutine	Subroutine only	Algorithmic focus	Supports careful positioning of near-linear claims
PyVRP / HGS	Benchmark VRPTW/CVRP (hybrid genetic search)	Benchmark scale	C++ package	State-of-the-art quality, not positioned for million-stop scale
This work	Last-mile with capacity, volume, TW, route limits, rebalancing	1M stops	Commodity laptop	Combines operational constraints, large planning units, and commodity hardware

The distinguishing contribution lies in combining three properties within a single architecture: fast route-level optimization enabling large-scale parallel dispatch, constraint-aware clustering that preserves global coherence, and distributed boundary rebalancing that repairs inconsistencies after parallel clustering and initial allocation, before route construction.

3 Reframing the Deployment Problem: The Hardware Assumption

A largely implicit premise in both the academic literature and commercial routing practice is that very large routing workloads require substantial computational infrastructure. Although rarely stated explicitly, this premise is embedded in how routing systems are designed, deployed, and exposed to users.

This work examines that premise from a different systems perspective. Rather than treating hardware as effectively unbounded, the proposed system was developed under the working assumption that large-scale vehicle routing can remain practical on commodity hardware, without requiring specialized infrastructure, large distributed clusters, or expensive cloud deployments.

3.1 Evidence from commercial systems

These limits do not imply algorithmic impossibility. They reflect the economic and operational constraints of routing systems that expose optimization as bounded API requests. In practice,

large instances must often be subdivided before optimization begins, introducing artificial territorial boundaries and weakening global coherence.

3.2 From algorithmic difficulty to deployment constraint

Classical formulations of the Vehicle Routing Problem emphasize combinatorial hardness. In deployed systems, however, the limiting factor is often not the mathematical formulation alone, but the way computation is organized and executed. When routing is implemented as a centralized, monolithic optimization process, both memory usage and runtime tend to scale unfavorably, making large instances impractical without significant infrastructure.

By restructuring computation into parallelizable layers, bounding route-level optimization, and eliminating redundant work through caching, the proposed system changes the practical deployment characteristics of large routing workloads.

3.3 Implications for large-scale routing

Under this reframing, hardware is no longer the primary determinant of solvability. Instead, it becomes an accelerator: additional compute reduces wall-clock time, but is not required to make the problem tractable.

This perspective suggests that some of the practical limits observed in commercial systems may reflect architectural choices as much as inherent properties of the routing problem itself. If very large routing workloads can be executed efficiently on commodity hardware, then the design space for routing systems expands substantially, both in cost and in operational flexibility.

The experimental results presented later should be interpreted in this context: not only as optimization results, but as evidence that the deployment model of large-scale routing can be reconsidered from a systems-design perspective.

4 Problem Definition and Constraints

The proposed system targets last-mile delivery planning under realistic operational conditions. The objective is not merely to construct feasible routes in an abstract graph, but to generate deployable fleet plans that reflect the constraints and trade-offs of real logistics operations.

Given a set of delivery stops $S = \{s_1, \ldots, s_n\}$, a fleet of vehicles $V = \{v_1, \ldots, v_m\}$, and package-level attributes associated with each stop (including volume, weight, and optional time windows), the planner must produce a set of routes that ensures full stop coverage while satisfying a mixed constraint model.

Hard constraints are strictly enforced and define feasibility: physical vehicle loading limits (volume, weight, and package-size compatibility) and full stop coverage (no dropped deliveries). Soft constraints guide optimization and may be relaxed when necessary: geographic compactness, time-window adherence, and route duration and stop limits. This distinction reflects operational reality, where enforcing every requirement as a hard constraint often produces infeasible or highly fragmented solutions. By combining strict feasibility conditions with flexible optimization signals, the planner remains both robust and adaptable.

Unlike classical VRP formulations focused primarily on minimizing total distance, the problem is treated here as a multi-objective planning task balancing distance, fleet utilization, route compactness, and constraint satisfaction. The system is accordingly parameterized: load-balancing targets, time-window strictness, and the primary optimization criterion (distance, time, or a weighted

combination) can be configured per scenario. Section 6 describes the configuration used in the reported experiments.

These design choices also have architectural implications. Route-level optimization must remain sufficiently fast to serve as a reusable primitive; clustering cannot be treated as purely geometric preprocessing, since it must incorporate signals such as volume, capacity, and density; and the planner requires a repair mechanism capable of correcting inconsistencies introduced by parallel decomposition. These considerations motivate the architecture described in the next section.

5 System Architecture

The planning pipeline is organized as a sequence of cooperating bounded stages: parallel constraint-aware clustering, constraint-aware initial allocation and consolidation, distributed neighbor-based rebalancing, parallel route construction, and a multi-level caching layer that eliminates redundant computation. Fast route-level optimization serves as the reusable primitive that makes the final routing stage scalable. Figure 4 provides a high-level overview of the pipeline structure.

5.1 Parallel Constraint-Aware Clustering

Intuition. The clustering stage does not aim to produce a final fleet plan. Its role is to generate a set of *operationally plausible region candidates* that subsequent stages (allocation, rebalancing, route construction) can refine. Concretely, the input stops are first split into spatial chunks that can be processed concurrently; within each chunk, a constrained partitioning routine produces sub-clusters whose size and aggregate package volume already lie within bounds derived from the available fleet. Constraints are not treated as a post hoc filter: they enter the partitioning objective itself, so that purely geometric (but operationally infeasible) groupings are never produced.

Notation. Throughout this subsection we use the following symbols.

S the full set of input stops; $|S| = n$.

$\{S_1, \ldots, S_B\}$ a disjoint partition of S into B spatial chunks of comparable size, satisfying $S = S_1 \uplus \cdots \uplus S_B$. The number of chunks B is set as $B = \min(\lceil n/\tau \rceil, P)$, where τ is a target chunk size (a configuration parameter, on the order of 10^4 stops in this work) and P is the number of available worker processes. This rule keeps each chunk small enough for the inner partitioning routine to remain cache-friendly while exploiting the available parallelism.

$\mathbf{vol}(s) \in \mathbb{R}_{\geq 0}$ the package volume of stop $s \in S$.

V the heterogeneous fleet, with per-vehicle upper bounds u_v^{vol} (volume capacity) and other capacity attributes defined as in Section 5.4.

$\bar{u}^{\mathbf{vol}}, \bar{\ell}^{\mathbf{vol}} \in \mathbb{R}_{\geq 0}$ per-cluster upper and lower bounds on aggregate package volume, derived from the fleet capacity profile:

$$\bar{u}^{\mathrm{vol}} \;=\; \alpha \operatorname*{aggr}_{v \in V}\!\left(u_v^{\mathrm{vol}}\right), \qquad \bar{\ell}^{\mathrm{vol}} \;=\; \beta \cdot \bar{u}^{\mathrm{vol}},$$

where $\alpha \in (0, 1]$ is the configurable loading-target coefficient described in Section 6.3, $\beta \in (0, \alpha)$ is the minimum-utilization coefficient that prevents underloaded clusters, and $\mathrm{aggr}(\cdot)$ is a fleet-level aggregation operator. In this work aggr is the fleet mean of vehicle volume capacities; the convergence and complexity arguments below require only that aggr satisfies $\min_v u_v^{\mathrm{vol}} \leq \mathrm{aggr}(\cdot) \leq \max_v u_v^{\mathrm{vol}}$ and is monotone in its arguments.

$\bar{u}^{\mathbf{count}}, \bar{\ell}^{\mathbf{count}} \in \mathbb{N}$ per-cluster upper and lower bounds on the *number of stops* admissible in a single cluster, derived analogously from fleet route-stop-limit profiles. We use the symbol *count* (rather than *size*) to disambiguate from the package-size physical constraint introduced in Section 5.4, which refers to a per-package bounding-box dimension.

$k_b \in \mathbb{N}$ the desired number of sub-clusters for chunk S_b, set as the smallest integer that simultaneously respects both bounds:

$$k_b \;=\; \max\Big(\Big\lceil \frac{\sum_{s\in S_b} \mathrm{vol}(s)}{\bar{u}^{\mathrm{vol}}} \Big\rceil,\; \Big\lceil \frac{|S_b|}{\bar{u}^{\mathrm{count}}} \Big\rceil\Big).$$

This is a coarse but safe initial estimate; the inner routine self-adjusts k_b when feedback indicates infeasibility (see Θ_b below).

$\bar{c}_j \in \mathbb{R}^2$ the spatial centroid of sub-cluster C_j^b in planar stop coordinates.

$\Theta_b \in \mathbb{N}_{\geq 0}$ the count of sub-clusters in chunk S_b that violate the volume bounds after a partitioning attempt:

$$\Theta_b \;=\; \sum_j \mathbf{1}\Big[\mu_{\mathrm{vol}}(C_j^b) > \bar{u}^{\mathrm{vol}} \text{ or } \mu_{\mathrm{vol}}(C_j^b) < \bar{\ell}^{\mathrm{vol}}\Big].$$

It is the feedback signal that drives adjustment of k_b.

$J_{\max} \in \mathbb{N}$ the maximum number of k_b adjustments per chunk before falling through to consolidation. It bounds the worst-case cost of the inner loop and prevents non-termination when the chunk's volume profile is intrinsically incompatible with the fleet bounds.

$I \in \mathbb{N}$ the cap on internal iterations of the inner ConstrainedPartition subroutine. It guarantees bounded per-call cost.

Constrained partitioning objective. For a chunk S_b with k_b sub-clusters, the inner routine returns a partition $\{C_1^b, \ldots, C_{k_b}^b\}$ that minimizes a within-cluster spatial dispersion objective subject to per-cluster stop-count bounds:

$$\min_{\{C_j^b\}} \sum_{j=1}^{k_b} \sum_{s\in C_j^b} d\big(s, \bar{c}_j\big)^2 \quad \text{s.t.} \quad \bar{\ell}^{\mathrm{count}} \leq |C_j^b| \leq \bar{u}^{\mathrm{count}} \;\; \forall j,$$

where $d(\cdot,\cdot)$ is planar distance on stop coordinates. Volume bounds are enforced as a post-condition of the partition: if the resulting partition has $\Theta_b > 0$, the chunk re-enters the loop with k_b adjusted upward (when overload dominates) or downward (when every cluster falls below $\bar{\ell}^{\mathrm{vol}}$), up to $J_{\max}$ adjustments.

In simple terms, this objective favors clusters whose stops remain close to their own spatial center, while preventing clusters from becoming too large or too small in stop count.

This feedback step is important. The initial value of k_b is only a safe first estimate, not a final decision. If one or more clusters are too large, k_b is increased so that the chunk is repartitioned into more clusters and the load is spread more evenly. If all clusters are too small, k_b is decreased so that the chunk is repartitioned into fewer, better-utilized groups. In this sense, the clustering stage does not assume that the first partition is correct; it uses bounded feedback to refine the number of sub-clusters until the chunk is brought into a more operationally plausible regime.

Algorithm 1 Parallel Constraint-Aware Clustering

Require: Stop set S, fleet V, capacity bounds $\bar{\ell}^{\text{vol}}, \bar{u}^{\text{vol}}, \bar{\ell}^{\text{count}}, \bar{u}^{\text{count}}$, max adjustments $J_{\max}$
Ensure: Initial partition $\mathcal{P}_0$

```
Split S into chunks S_1, ..., S_B
for all chunk S_b in parallel do
    k_b ← max(⌈Σ_{s∈S_b} vol(s)/ū^vol⌉, ⌈|S_b|/ū^count⌉)
    for j = 1, ..., J_max do
        {C_1^b, ..., C_{k_b}^b} ← ConstrainedPartition(S_b, k_b, ℓ̄^count, ū^count)
        Θ_b ← Σ_j 1[μ_vol(C_j^b) > ū^vol or μ_vol(C_j^b) < ℓ̄^vol]    ▷ number of clusters violating volume bounds
        if Θ_b = 0 then break
        end if
        adjust k_b upward if any cluster exceeds ū^vol, downward if all fall below ℓ̄^vol
    end for
end for
P_0 ← deterministic consolidation of all {C_j^b} under a global numbering
return P_0
```

Concurrency and consolidation. The B chunks are processed in parallel. Per-chunk partitions are not treated as final assignments: their cluster identifiers are remapped to a global numbering during a deterministic consolidation step that preserves the within-chunk grouping, so that the global partition $\mathcal{P}_0 = \bigcup_b \{C_1^b, \ldots, C_{k_b}^b\}$ is reproducible across runs given the same input. Clustering itself does *not* negotiate across chunk borders; stops near a chunk boundary that would naturally belong to a neighboring chunk's cluster remain in their original chunk at this stage, and any resulting boundary inconsistencies are repaired downstream by the rebalancing stage (Section 5.4). This is a deliberate division of labor: clustering is responsible for producing locally coherent regions cheaply and in parallel; fixing inter-chunk artefacts is delegated to a stage that already has the machinery (boundary-restricted exchanges, Φ-descent) to do so without redoing the partition.

The inner partitioning subroutine. ConstrainedPartition is a size-constrained spatial partitioning routine: given a stop set, a target number of clusters k_b, and per-cluster count bounds, it returns a partition that approximates the dispersion objective above while respecting the count bounds. Conceptually, the routine assigns stops to a fixed number of spatially compact groups while keeping the number of stops per group within admissible bounds.

The convergence and complexity arguments of this section depend only on three published properties of the subroutine: (a) it terminates in at most I internal iterations; (b) it produces a partition into exactly k_b non-empty clusters whenever $|S_b| \geq k_b \cdot \bar{\ell}^{\text{count}}$; and (c) its inner cost per chunk is $\mathcal{O}(k_b \cdot |S_b| \cdot I)$. Any size-constrained variant satisfying these properties is admissible; the specific implementation in this work is left implicit.

Termination and complexity. The outer $J_{\max}$-loop is bounded by construction. Per chunk, the cost is $\mathcal{O}(J_{\max} \cdot k_b \cdot |S_b| \cdot I)$. Since chunks are disjoint, $\sum_b |S_b| = n$, and each chunk is bounded by the target size τ up to load-balancing effects, the dominant per-worker cost is controlled by the largest chunk rather than by the full input. With bounded $J_{\max}$, bounded I, and bounded per-cluster count ranges, the clustering stage scales with the amount of work assigned to each worker, rather than requiring a single global partitioning pass over all stops. This is the property that lets the

clustering stage track input size near-linearly under increasing parallelism.

Operationally, this means that clustering cost grows with the amount of input, but remains manageable because work is distributed across independent chunks processed in parallel.

Implementation-level variations. The exact ConstrainedPartition subroutine, the precise feedback rule that adjusts k_b between attempts, and the heuristics that handle small or low-density chunks are implementation-level choices and are not specified here. The convergence and complexity bounds depend only on the published properties stated above: bounded k_b, bounded $J_{\max}$, bounded I, and the three subroutine properties (a)–(c).

5.2 Constraint-Aware Initial Allocation

Intuition. Once the initial partition $\mathcal{P}_0$ is available, each cluster must be matched to exactly one vehicle of the fleet V. This is a *generalized assignment problem* (GAP) in which a cluster's *demand vector* (volume, package size, weight) must fit within the *capacity vector* of the assigned vehicle. The aim of this stage is not to produce a final, perfectly feasible allocation; rather, it is to produce an allocation that is sufficiently coherent that the downstream rebalancing stage can repair residual violations through local boundary exchanges. The allocation is therefore allowed to be slightly infeasible (e.g. a 10%–20% overload on a small subset of clusters), provided that the resulting violation residual $\Phi(\mathcal{P}_0)$ is within the convergence regime of the rebalancing stage.

Notation. We use the following symbols throughout this subsection.

$\mathcal{P}_0 = \{C_1, \ldots, C_m\}$ the initial partition produced by Algorithm 1.

$q(C_i) \in \mathbb{R}^3_{\geq 0}$ the physical demand vector of cluster C_i: $q(C_i) = (\mu_{\text{vol}}(C_i),\ \mu_{\text{size}}(C_i),\ \mu_{\text{weight}}(C_i))$.

$U_v \in \mathbb{R}^3_{\geq 0}$ the capacity vector of vehicle v: $U_v = (u_v^{\text{vol}},\ u_v^{\text{size}},\ u_v^{\text{weight}})$. Componentwise comparison $q(C_i) \leq U_v$ is the strict feasibility check used in the allocation criterion.

$r(C_i) \in \mathbb{R}_{\geq 0}$ the depot distance of cluster C_i: the planar distance from the depot to the centroid of C_i. It drives sorting in the distance-aware mode.

$p_v \in \mathbb{Z}_{\geq 0}$ a configurable per-vehicle priority (lower is preferred). It allows operators to express preferences over the fleet (e.g. owned vehicles before contractor vehicles) and is used as the primary key in vehicle ordering.

$\mathbf{slack}_v(C_i) = u_v^{(c)} - \mu_c(C_i)$ the residual capacity of vehicle v in the dominant active capacity dimension c if C_i were assigned to it. Used as the secondary key in the matching policy.

$\lambda \in \mathbb{R}_{>0}$ a large overload penalty, finite but $\lambda \gg 1$. It discourages (but does not forbid) overloads, and is what makes the allocation usable as input to the rebalancing stage even when the fleet is tight.

$\pi : \mathcal{P}_0 \to V$ the final assignment from clusters to vehicles produced by this stage. The assignment is injective ($\sum_{C_i} \mathbf{1}[\pi(C_i) = v] \leq 1$ for every v): each vehicle serves at most one cluster.

Operating modes. The allocation stage exposes two operating modes whose selection is determined by the active constraint set declared in the request.

1. **Capacity-first mode.** Used when distance and time constraints are inactive. Clusters are sorted by demand (descending in the dominant active capacity dimension); vehicles are sorted by capacity (descending) within each priority tier; a one-pass best-fit matching is performed.

Algorithm 2 Constraint-Aware Initial Allocation (greedy lexicographic policy)

Require: Initial partition $\mathcal{P}_0$, fleet V, active constraint set $\mathcal{F}$, mode $M \in \{\text{capacity-first}, \text{distance-aware}\}$, overload penalty λ
Ensure: Assignment $\pi : \mathcal{P}_0 \to V$

$L_C \leftarrow$ sort $\mathcal{P}_0$ according to mode M: descending in dominant demand dimension if $M =$ capacity-first, descending in $r(C_i)$ if $M =$ distance-aware.
$L_V \leftarrow$ sort V by (p_v, cap order), where cap order is descending for capacity-first and ascending for distance-aware.
$\pi \leftarrow \emptyset$; $V_{\text{free}} \leftarrow V$
for all $C \in L_C$ **do**
    $v^\star \leftarrow$ first $v \in L_V \cap V_{\text{free}}$ such that $q(C) \le U_v$ ▷ strict best-fit under mode order
    if $v^\star$ exists **then**
        $\pi(C) \leftarrow v^\star$; $V_{\text{free}} \leftarrow V_{\text{free}} \setminus \{v^\star\}$
    else
        $\pi(C) \leftarrow$ first $v \in V_{\text{free}}$ ▷ overload accepted, weighted by λ
        $V_{\text{free}} \leftarrow V_{\text{free}} \setminus \{\pi(C)\}$
    end if
end for
return π

2. **Distance-aware mode.** Used when distance or time constraints are active. Clusters are sorted by depot distance $r(C_i)$ (descending); vehicles are sorted by capacity *ascending* within priority tier. The pairing therefore assigns farther clusters to smaller vehicles. The operational rationale is asymmetric pressure absorption: a small vehicle serving a far cluster is naturally constrained to a low stop count, which keeps the route's total time under control; conversely, large vehicles are reserved for nearby clusters where their stop-absorption capacity is most useful and where the marginal travel cost per stop is low.

Allocation as a heuristic GAP approximation. The exact GAP problem corresponding to this stage is

$$\min_{\pi} \sum_{C_i \in \mathcal{P}_0} \left[p_{\pi(C_i)} \; + \; \lambda \cdot \mathbf{1}[q(C_i) \not\le U_{\pi(C_i)}] \right] \quad \text{s.t. } \pi \text{ injective into } V,$$

which is NP-hard in general. We do *not* solve this problem exactly. Instead, the allocation stage executes a one-pass *greedy lexicographic policy* that approximates the GAP objective: clusters and vehicles are ordered according to the active mode (Algorithm 2), and each cluster in order is assigned to the first available vehicle that satisfies $q(C_i) \le U_v$, falling back to the first available vehicle with overload penalty λ if no strictly feasible vehicle remains.

This greedy policy is not globally optimal in general. We adopt it deliberately, on the grounds that exact GAP minimization is unnecessary in this pipeline: the only property required of the allocation is that the resulting $\Phi(\mathcal{P}_0)$ lies within the convergence regime of the rebalancing stage (Section 5.4), which empirically holds for the workloads evaluated in Section 7. Allocation is thus deliberately fast and bounded; the residual responsibility of moving from a coherent allocation to a feasible plan is discharged by rebalancing, which provably reduces Φ.

Cost. The dominant cost of Algorithm 2 is the initial sort of $\mathcal{P}_0$ and V, followed by an $\mathcal{O}(m \cdot |V|)$ scan in the worst case. Total cost is $\mathcal{O}(m \log m + |V| \log |V| + m \cdot |V|)$, dominated by the $m \cdot |V|$

matching scan when both sets are large. This is linear in m for fixed fleet size and does not contribute a super-linear term to the overall pipeline.

Implementation-level variations. The exact tie-breaking rule used when multiple vehicles share the same priority and slack, the policy that handles heterogeneous fleets where some vehicles cannot serve any feasible cluster on their own (and must instead absorb sub-clusters carved off larger ones), and the precise calibrated value of the overload penalty λ are implementation-level details. They affect constant factors in the quality of $\mathcal{P}_0$ but not the convergence or complexity properties on which the pipeline relies.

5.3 Distributed neighbor-based rebalancing

After initial cluster candidates are formed and consolidated, a rebalancing stage allows neighboring regions to exchange stops near their shared geographic boundaries before route construction. The purpose of this stage is to repair residual constraint violations and reduce boundary artefacts introduced by parallel clustering and initial allocation, without recomputing the global partition.

Only frontier stops are eligible for exchange, which preserves spatial compactness and keeps the repair local. The formal residual, feasibility conditions, acceptance rule, and complexity of this stage are defined in Section 5.4.

5.4 Operational Formalization of Distributed Neighbor-Based Rebalancing

Intuition. After parallel clustering and initial allocation, some clusters may slightly violate operational bounds. Instead of recomputing the global partition, the system performs local exchanges between neighboring clusters: overloaded clusters transfer frontier stops to neighbors that can absorb them, while underloaded clusters request frontier stops from neighbors that can spare them. A move is accepted only if it is feasible for both clusters and strictly reduces the current constraint violation without degrading already stabilized higher-priority constraints. Because each accepted move decreases a non-negative residual, the process terminates in finite time.

Notation. Let $S = \{s_1, \dots, s_n\}$ be the set of delivery stops and $\mathcal{P} = \{C_1, \dots, C_m\}$ a partition of S into m clusters, where each cluster is assigned to one vehicle. Let $\pi : \mathcal{P} \to V$ denote the cluster-to-vehicle assignment, fixed during rebalancing.

Constraints are processed in a fixed priority order

$$\mathcal{C} = \langle \text{volume}, \text{size}, \text{weight}, \text{distance}, \text{time} \rangle,$$

where the first three are physical bounds and the last two are operational bounds. For each vehicle v and constraint c, let $\ell_v^{(c)}$ and $u_v^{(c)}$ denote the lower and upper bounds, and let $\mu_c(C_i)$ denote the load of cluster C_i under constraint c (e.g. total volume, weight, route distance, or route time).

Let $\mathcal{N}(C_i)$ be the bounded set of geographic neighbors of cluster C_i, with $|\mathcal{N}(C_i)| \le K$. For adjacent clusters C_i and C_j, define the frontier stop of C_i relative to C_j as

$$s^*(C_i, C_j) \;=\; \arg\min_{s \in C_i} \min_{t \in C_j} d(s, t),$$

where d is the spatial distance function used to identify frontier candidates. Intuitively, this is the border stop of C_i that lies closest to cluster C_j, making it the most natural local candidate for transfer.

Violation residual. For a partition $\mathcal{P}$ and constraint c, define the violation residual as

$$\Phi_c(\mathcal{P}) \;=\; \sum_{C_i \in \mathcal{P}} \Big[\max\!\big(0, \mu_c(C_i) - u^{(c)}_{\pi(C_i)}\big) + \max\!\big(0, \ell^{(c)}_{\pi(C_i)} - \mu_c(C_i)\big)\Big].$$

Thus $\Phi_c(\mathcal{P}) \geq 0$, with equality if and only if every cluster is feasible under constraint c.

Interpretation of the residual. A high value of $\Phi_c(\mathcal{P})$ means that the current partition is substantially infeasible with respect to constraint c, while $\Phi_c(\mathcal{P}) = 0$ means that all clusters satisfy that constraint. This gives the rebalancing stage a simple progress signal: every accepted move must reduce the current residual.

Lexicographic descent (sequential constraint optimization). Constraints are processed sequentially in the order of $\mathcal{C}$. While handling a constraint c, a move is accepted only if it strictly decreases Φ_c and does not increase $\Phi_{c'}$ for any higher-priority constraint c' that has already been stabilized. This avoids combining heterogeneous quantities such as litres, kilograms, kilometres, and minutes into a single weighted scalar objective. A higher-priority constraint is considered stabilized once its phase finishes without further accepted moves, and subsequent phases are not allowed to reintroduce violations on it.

Δ-feasibility. A candidate move $s : C_i \to C_j$ is Δ-*feasible* if, after transferring s, both clusters remain within bounds for every constraint active for their assigned vehicles:

$$\mu_c(C_i \setminus \{s\}) \in [\ell^{(c)}_{\pi(C_i)}, u^{(c)}_{\pi(C_i)}] \quad \text{and} \quad \mu_c(C_j \cup \{s\}) \in [\ell^{(c)}_{\pi(C_j)}, u^{(c)}_{\pi(C_j)}],$$

for all relevant c. A move that improves Φ_c but violates any active bound is rejected.

Boundary restriction. Only frontier stops are eligible for transfer. For a proposed move $C_i \to C_j$, the candidate is $s^*(C_i, C_j)$; in the reverse direction, the candidate is $s^*(C_j, C_i)$. This restriction keeps the repair local and preserves spatial compactness.

Local exchange operator. We abstract pairwise negotiation into a local operator LocalExchange(C_i, C_j, c) that either applies one boundary-restricted move satisfying the acceptance rule below or returns *no-op*. This operator is intentionally left abstract at this level: the focus here is on the orchestration logic and the descent properties of the rebalancing stage, not on the internal heuristics used to evaluate competing local candidates. In practice, LocalExchange attempts a small local correction between two neighboring clusters, typically by testing a boundary stop transfer and accepting it only if it satisfies the current feasibility and descent rules.

Acceptance criterion. LocalExchange(C_i, C_j, c) applies a move iff: (i) the candidate is Δ-feasible, and (ii) it decreases Φ_c by at least $\varepsilon_c > 0$ without increasing any already stabilized higher-priority residual $\Phi_{c'}$. If no such move exists, the operator returns *no-op*.

Termination and fallback. Each accepted move reduces Φ_c by at least ε_c, and Φ_c is non-negative. Therefore, each constraint phase terminates after a finite number of accepted moves, bounded by $\Phi_c(\mathcal{P}_0)/\varepsilon_c$, with an outer safeguard of $T_{\max}$ iterations. If the violating set remains unchanged across two consecutive iterations, the orchestrator exits the local stage and invokes FallbackRepair, which performs stronger corrective actions such as cluster splitting or reassignment beyond local adjacency.

Algorithm 3 Distributed Boundary-Restricted Rebalancing

Require: Partition $\mathcal{P}$, assignment π, neighbor map $\mathcal{N}$, ordered constraint set $\mathcal{C}$, max iterations $T_{\max}$
Ensure: Rebalanced partition $\mathcal{P}^\star$ with $\Phi_c(\mathcal{P}^\star) \le \Phi_c(\mathcal{P})$ for all c

for each constraint $c \in \mathcal{C}$ in priority order **do**
  $t \leftarrow 0$; *progress* $\leftarrow$ TRUE
  while *progress* **and** $t < T_{\max}$ **do**
    progress $\leftarrow$ FALSE; $t \leftarrow t + 1$
    for all $C_i \in \mathcal{P}$ **in parallel do**
      if $\mu_c(C_i) \notin [\ell^{(c)}_{\pi(C_i)}, u^{(c)}_{\pi(C_i)}]$ **then**
        for all $C_j \in \mathcal{N}(C_i)$ **ordered by spatial proximity do**
          moved $\leftarrow$ LOCALEXCHANGE(C_i, C_j, c)
          if *moved* **then**
            progress $\leftarrow$ TRUE; **break**
          end if
        end for
      end if
    end for
    if the set of clusters violating c has not changed for two consecutive iterations **then**
      invoke FALLBACKREPAIR; **break**
    end if
  end while
end for
return $\mathcal{P}$

Complexity. With m clusters and bounded neighborhood size K, one outer iteration performs at most $O(mK)$ invocations of LOCALEXCHANGE. Assuming incremental maintenance of constraint loads and frontier candidates, each invocation has amortized constant cost at the orchestration level. The total rebalancing cost is therefore

$$O\big(|\mathcal{C}| \cdot T_{\max} \cdot m \cdot K\big).$$

Under these assumptions, the repair stage depends on the number of clusters and their local adjacency structure rather than directly on the total stop count n, which helps prevent the rebalancing phase from reintroducing a super-linear bottleneck after partitioning.

5.5 Repair Hierarchy from Clustering to Routing

Cluster-first architectures raise a natural concern: a cluster that is geographically compact may still be infeasible for the vehicle to which it is assigned. The proposed pipeline does not assume that clustering alone produces independently routable units. Instead, feasibility is established progressively through a bounded repair hierarchy applied between clustering and route construction.

Hard and soft constraints. The pipeline distinguishes between *hard physical constraints* (package volume, package size, vehicle weight) and *soft operational constraints* (route distance, route time, optional time windows). Hard constraints must hold for a cluster to be loadable onto its assigned vehicle. Soft constraints are optimized and reported, but may be relaxed depending on configuration.

Repair hierarchy. Feasibility is enforced through four levels, each invoked only on the residual infeasibility left by the previous one.

Level 0: constrained clustering. The clustering stage already incorporates structural bounds such as cluster size and aggregate volume, so many infeasible assignments are prevented before they reach the global partition.

Level 1: post-allocation structural split. After vehicles are assigned to clusters, any cluster that cannot be served by its assigned vehicle under the hard physical constraints is split into smaller subclusters and reassigned across additional vehicles if available. This is the main mechanism for resolving internal infeasibility that cannot be repaired by local exchanges.

Level 2: boundary-restricted rebalancing. Once clusters are structurally compatible with their assigned vehicles, the distributed rebalancing stage performs local exchanges of frontier stops between adjacent clusters. Its role is to correct residual fine-grained imbalances while preserving spatial compactness.

Level 3: fallback repair. If local boundary repair stalls, the pipeline escalates to a fallback stage that performs stronger corrective actions, including forced splits and reassignment beyond local adjacency. This guarantees that the partition passed to route construction is feasible under hard constraints.

Route-level optimization as final verification. Route-level optimization is applied only after the repair hierarchy has driven hard-constraint infeasibility to zero. At that point, route construction serves two roles: it produces the final stop sequence for each cluster, and it evaluates soft operational constraints whose exact value depends on sequencing, such as route duration and route distance. If a route exceeds a configured soft bound, the violation is reported as part of the final plan rather than causing rejection of the entire solution.

Implication for the cluster-first claim. The architecture therefore does not claim that clustering directly produces independently routable clusters in a strict graph-theoretic sense. The narrower claim is that, subject to the available fleet and the bounded repair hierarchy above, the pipeline progressively converts an initial partition into a set of route candidates that are feasible under hard physical constraints before route construction begins, while soft operational constraints are evaluated and surfaced at routing time. The assignment-of-stops-to-vehicles problem is thus addressed across multiple bounded stages rather than hidden inside clustering alone.

Scope and limitation. This design choice is central to the architecture: it preserves a bounded and scalable planning pipeline by resolving most structural infeasibility before route construction, while avoiding repeated global replanning after routes are built. The trade-off is that soft violations discovered during route sequencing are currently reported rather than fed back into a new global repair cycle.

5.6 Route-level optimization as a primitive

The first architectural requirement is that individual route optimization must be fast enough to serve as a reusable primitive. The system therefore uses a lightweight route-improvement procedure rather than a heavy global metaheuristic. Starting from an initial stop ordering, the optimizer

applies a bounded sequence of local improvements designed to reduce route cost while preserving the hard feasibility conditions established by the upstream repair hierarchy.

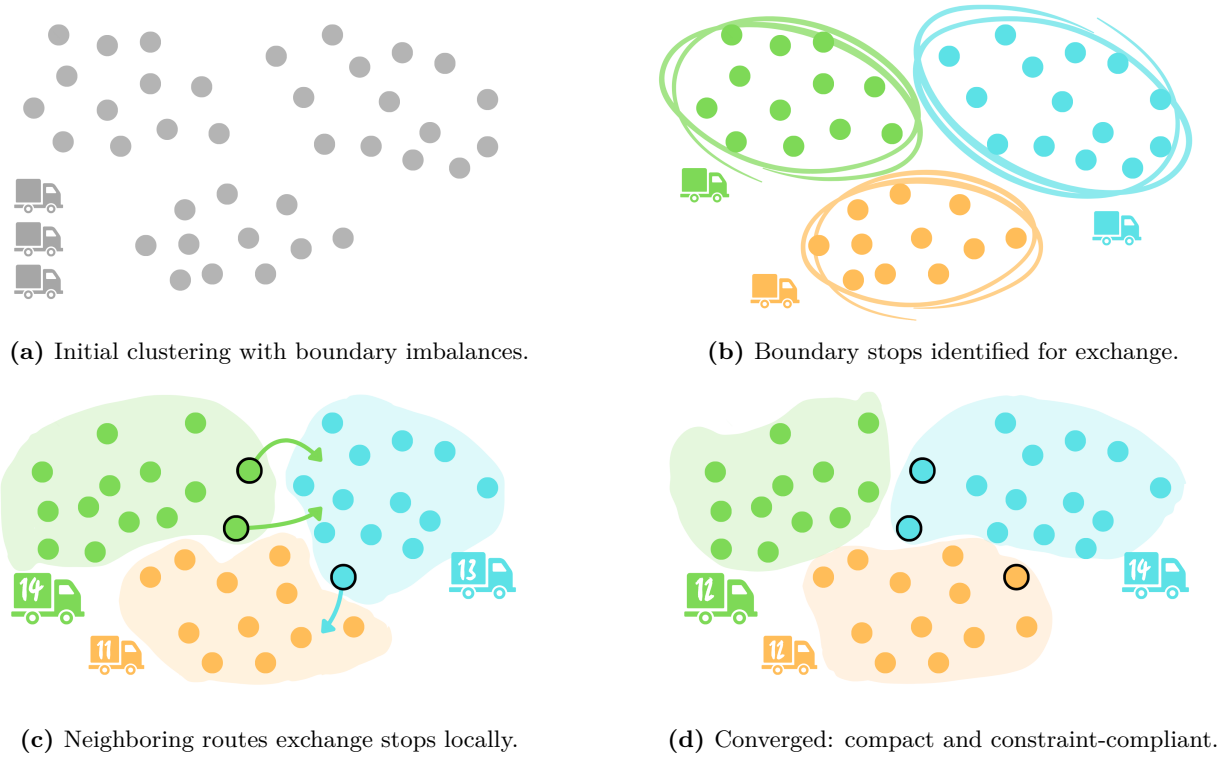
At a high level, the route optimizer combines two components. First, it applies a linear pass over the route to identify local inconsistencies and fix obvious ordering defects in $O(n)$ time, where n is the number of stops in the route. Second, it performs a small-neighborhood improvement stage based on a restricted 2-opt-style exchange policy, accepting only moves that improve the active route objective (distance, time, or a configured weighted combination) without violating the route-level feasibility conditions active for that scenario.

The route primitive is intentionally bounded: it does not attempt exhaustive tour optimization, but a fast local refinement whose role is to convert a hard-feasible cluster into a good route at low computational cost. The search terminates when no improving local move is found within the bounded candidate set or when the configured iteration cap is reached. In practice, this keeps per-route optimization in the sub-second regime for routes ranging from a few dozen to several hundred stops.

This bounded cost is what makes large-scale parallel dispatch feasible: once a single route is cheap to improve, the broader problem shifts from solving isolated tours to coordinating many route candidates at fleet scale.

5.7 Parallel route construction

Once cluster candidates have been consolidated, repaired, and brought into hard-feasible form by the hierarchy described in Section 5.5, each becomes an independent route-optimization job dispatched concurrently. Because route optimization is already cheap (Section 5.6), overall wall-clock time depends less on total stop count than on how efficiently jobs are scheduled. The planning cost is redistributed from a single centralized computation into many bounded parallel optimizations.



(a) Initial clustering with boundary imbalances.

(b) Boundary stops identified for exchange.

(c) Neighboring routes exchange stops locally.

(d) Converged: compact and constraint-compliant.

Figure 1: Neighbor-based rebalancing. Exchanges are restricted to boundary regions, enabling distributed convergence toward a globally coherent fleet plan.

5.8 Caching and localized graph lifecycle

A major contributor to the observed near-linear scaling is a multi-level caching strategy that minimizes redundant computation across pipeline stages.

In-memory graph and matrix caching. Localized routing graphs, distance matrices, and precomputed artifacts are retained in memory for reuse across nearby route calculations.

Distance query caching. Frequently evaluated spatial relations are cached across clustering, route construction, and rebalancing, avoiding repeated lookups.

Localized graph generation. Rather than loading a single graph for an entire territory, the planner creates smaller graphs focused on the active service region, preserving routing fidelity while remaining cache-friendly.

Dynamic cache lifecycle. Graphs are created on demand, stored under location-aware keys, reused when later requests overlap with the same service area, and released once inactive. Memory usage is therefore bounded by active routing regions rather than by the full geographic extent of the dataset. Figure 2 illustrates this hierarchy.

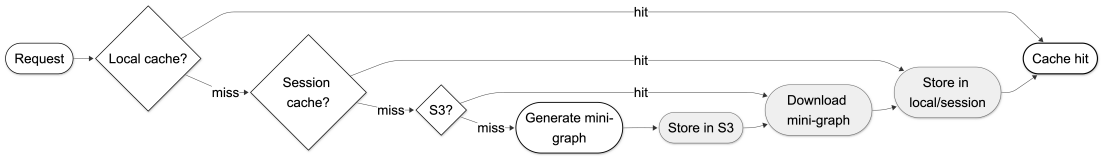


Figure 2: Cache hierarchy. Requests attempt reuse from instance-local memory, then session-level cache, then persistent storage. Only cache misses trigger graph generation, after which artifacts propagate upward for reuse.

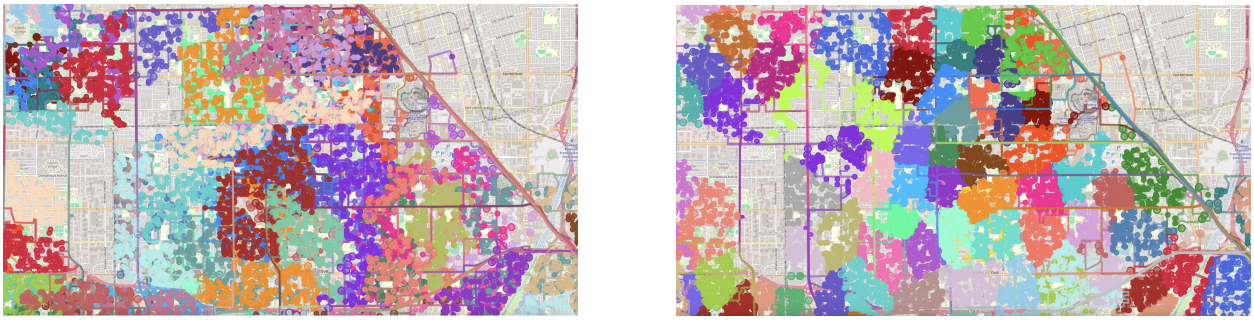


Figure 3: Overlapping clusters from the Amazon dataset (left) versus spatially compact clusters generated by the proposed system (right) for the same set of stops. Compact clustering reduces route interference and improves fleet-level efficiency.

These caching layers complement the parallel design: the system scales both by dispatching more work concurrently and by ensuring that graph preparation and distance evaluation are reused rather than recomputed at each pipeline stage.

5.9 Spatial coherence and cluster compactness

A central design objective is to produce spatially compact clusters with minimal overlap. The system aims to generate *atomic clusters*: regions that are internally dense, externally separated, and only weakly entangled with neighboring routes. Figure 3 illustrates this property using the Amazon benchmark data: the left panel shows overlapping clusters from the released Amazon routes, while the right panel shows compact clusters produced by the proposed system for the same set of stops.

When clusters overlap substantially, vehicles from different routes operate in the same areas, increasing crossings, duplicated coverage, and total distance. Compact clusters reduce these interactions, yielding higher stop density per route and lower fleet distance. The tension between geometric compactness and constraint feasibility is resolved through the boundary-restricted rebalancing described in Section 5.4.

This spatial behavior can be quantified through *inter-cluster interference*. For a partition $\mathcal{P} = \{C_1, \ldots, C_m\}$ of the stop set S, let $N_k(s)$ denote the k nearest neighbors of stop $s \in S$, and let $C(s)$ denote the cluster in $\mathcal{P}$ that contains s. The interference measure

$$\Psi(\mathcal{P}) = \frac{1}{|S|} \sum_{s \in S} \frac{|\{u \in N_k(s) : u \notin C(s)\}|}{k}$$

captures the average proportion of each stop's local neighborhood assigned to clusters different from its own. Low Ψ values indicate self-contained neighborhoods; high Ψ values indicate entangled assignments that tend to increase detours and route crossings. In the reported experiments, compact

clustering and boundary-constrained rebalancing together reduce inter-cluster interference relative to the baseline allocation, as illustrated qualitatively in Figure 3.

5.10 Pipeline overview

Figure 4 summarizes the execution architecture of the proposed system. The pipeline consists of three concurrent processing layers, chunk-level clustering, distributed rebalancing, and route-level optimization, connected through intermediate queueing stages that coordinate data flow between layers.

The first concurrent layer partitions the input into multiple chunks that are processed independently during clustering. These workers do not solve isolated subproblems in a fully disconnected way; rather, each one identifies local grouping patterns while remaining compatible with the broader problem context. Their outputs are then merged into a shared cluster pool, where consolidation and fleet matching restore a coherent global view before the next stage begins.

The second concurrent layer performs distributed rebalancing. At this stage, neighboring cluster regions exchange boundary stops in parallel in order to repair infeasibilities and improve spatial compactness before route construction. The important point is that rebalancing is not a centralized global correction pass applied after routing, but a distributed stage in which multiple local negotiations can proceed simultaneously across different parts of the solution.

The third concurrent layer dispatches rebalanced clusters as independent route jobs. Each job is solved in parallel using the route-level optimization primitive described earlier, after which the resulting routes are consolidated into the final fleet plan. Because each route remains a bounded optimization unit, wall-clock runtime depends primarily on the coordination of many jobs rather than on a single monolithic optimization process.

What the global coordinator is and is not. The global coordinator shown in Figure 4 should not be interpreted as a centralized optimizer or a single decision-making authority. Its role is logical rather than monolithic: it provides the shared orchestration context through which concurrent layers remain mutually consistent across the full pipeline. In practice, this coordination is reconstructed locally at each stage through common queues, shared state abstractions, and reconciliation rules, without collapsing the system into a single centralized planning process. Conceptually, this is closer to a distributed-consensus mechanism than to a classical top-down controller: globally coherent behavior emerges from multiple concurrent processes that operate independently while preserving compatibility with a shared problem context. In that sense, the coordinator is best understood as a distributed orchestration layer that preserves global coherence without centralizing the optimization itself.

This distinction is important for interpreting the architecture. The system does not rely on one central process to compute the entire routing solution end-to-end. Instead, it maintains coherence through lightweight orchestration across bounded stages, allowing large routing workloads to be processed in parallel without sacrificing consistency at fleet scale.

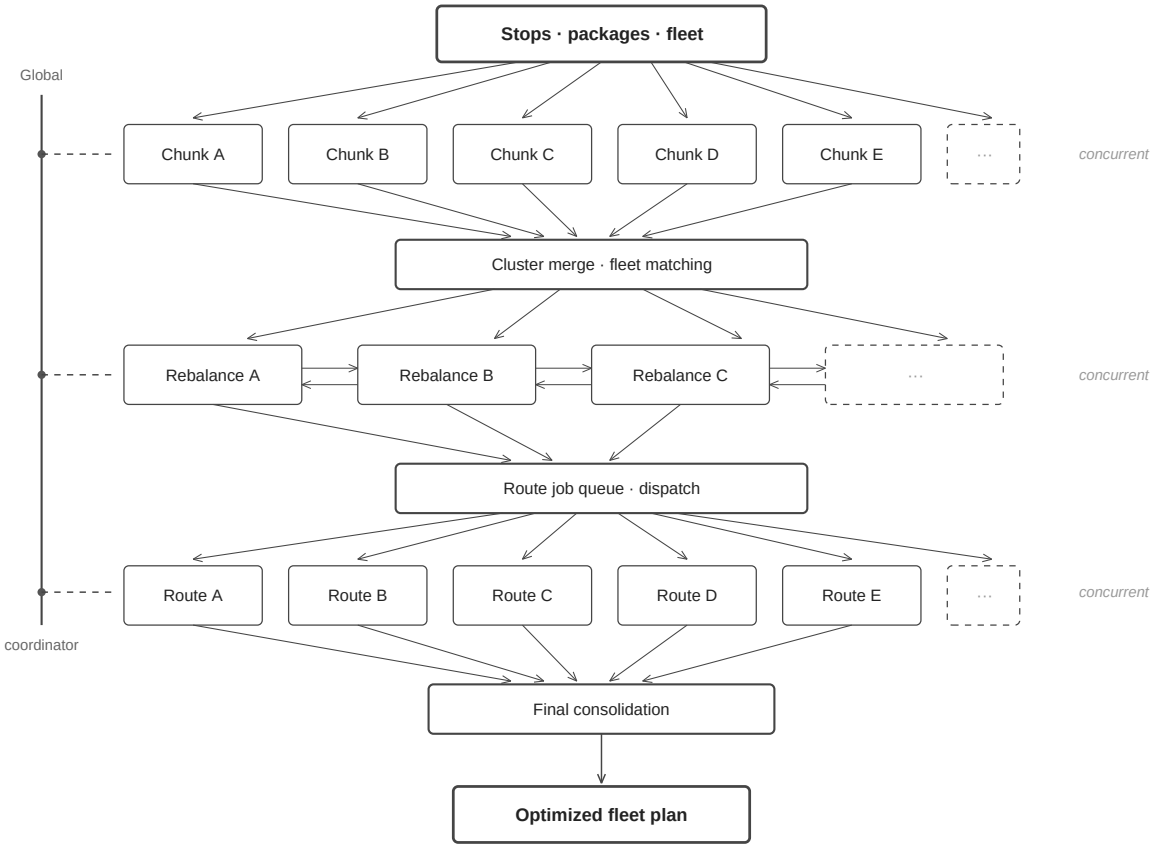


Figure 4: Architecture of the proposed routing pipeline. Three concurrent processing layers — chunk-level clustering, distributed rebalancing, and route-level optimization — are connected through queues between stages and supervised by a shared orchestration layer that preserves global consistency without centralizing the optimization itself.

6 Experimental Setup

The proposed architecture is evaluated in two complementary settings: a benchmark evaluation based on the Amazon Last Mile Routing Research Challenge [6, 7], and a scaling experiment extending to one million stops. The experiments serve two purposes. First, they evaluate whether the proposed architecture can produce operationally coherent routing plans under realistic delivery conditions. Second, they test whether runtime remains predictable as input size increases by an order of magnitude beyond the original benchmark scale.

6.1 Amazon Last Mile Routing Challenge dataset

The benchmark evaluation uses the public dataset released as part of the Amazon Last Mile Routing Research Challenge [6, 7], comprising real-world delivery scenarios from Amazon logistics operations. The evaluation covers 17 depot-level scenarios: DAU1, DBO1, DBO2, DBO3, DCH1, DCH2, DCH3, DCH4, DLA3, DLA4, DLA5, DLA7, DLA8, DLA9, DSE2, DSE4, and DSE5. Across these scenarios, the released Amazon baseline contributes 6,112 routes, totaling 1,048,575 stops and more than 2.5 million packages.

The proposed system re-optimizes the same stops under the planning assumptions described in Section 4, with strict enforcement of hard physical feasibility constraints and configurable treatment of soft operational constraints such as time windows and route stop limits.

6.2 Hardware

The experiments were executed on a MacBook Pro equipped with a 14-core Apple M4 Pro CPU and 48 GB of unified memory. This hardware context is deliberate: the purpose is to show that routing at large scale remains practical on commodity hardware rather than requiring specialized infrastructure.

6.3 System parameterization and operational trade-offs

An important property of the proposed system is that it is highly configurable. The routing outcomes reported here should therefore be understood not as the result of a single rigid objective, but as the output of a parameterized planning architecture whose behavior can be tuned according to operational priorities.

This is relevant both in practice and in scientific interpretation. Different organizations prioritize different trade-offs: minimizing fleet size, minimizing distance, maximizing punctuality, or maintaining higher vehicle utilization. As a result, route count, average distance, volume utilization, and time-window compliance are not independent outcomes; they depend on the selected configuration.

By default, the optimizer is configured toward a balanced objective that combines three goals: high fleet utilization, low total travel distance, and acceptable on-time performance under time-window constraints. The same architecture can nevertheless be tuned to emphasize different operating strategies when required.

6.3.1 Smart load balancing

The parameterization includes both route-level and system-level controls for load balancing.

Per-vehicle load bounds. Each route can be constrained by minimum and maximum loading ratios in order to avoid both underfilled and overloaded trips. This prevents the optimizer from creating routes that are technically feasible but operationally undesirable because of poor cargo utilization or excessive load concentration.

System-wide load target. In addition to per-vehicle bounds, the planner can be guided toward a global utilization target across the fleet. This acts as a fleet-level balancing signal and helps the optimizer converge toward a desired average occupancy pattern rather than optimizing routes in isolation.

These controls affect the structure of the final plan. Increasing the load target generally reduces the total number of routes, but may also increase route length, intensify boundary pressure between routes, and reduce flexibility for time-window compliance.

6.3.2 Routing objective and time-window policy

The system also exposes routing-policy controls that affect the trade-off between distance efficiency and service quality.

Primary optimization criterion. The routing stage can prioritize total route length, travel time, or a balanced objective combining both. This matters because the configuration that minimizes geometric or road distance is not always the one that best preserves punctuality or operational robustness.

Time-window handling. Time-window optimization can be enforced more or less aggressively depending on the scenario. When time windows are emphasized, the optimizer may require additional routes or lower consolidation to protect punctuality. When they are relaxed, route count may decrease, but schedule adherence becomes less strict.

The empirical effect of these controls on time-window violation rates is reported in Section 7.5.

6.3.3 Interpretation of parameter effects

The reported benchmark and scaling results should be interpreted as outcomes under a specific practical configuration rather than as universal optima under all possible parameter settings. This configurability is part of the system's operational value: the same architecture can be tuned toward lower distance, higher utilization, fewer routes, or stronger time-window adherence depending on the deployment scenario.

6.4 Evaluation metrics

Two families of metrics are reported.

Runtime metrics measure wall-clock execution time as the number of stops and routes increases. At larger scale, runtime is interpreted relative to total stop count to assess whether growth remains approximately proportional to input size.

Operational quality metrics measure the structure and efficiency of the resulting fleet plan: total measured distance, route count, stops per route, and cargo efficiency (defined as required delivery volume divided by total deployed vehicle capacity).

6.5 Distance measurement protocol

The published Amazon routes and the routes generated by the proposed system are compared using a shared post hoc measurement procedure rather than the optimizer's internal graph distance. This design avoids dependence on any single internal routing graph and ensures that both solutions are evaluated under a common external measurement rule.

Importantly, both solutions operate on the same underlying set of delivery stops. The comparison therefore does not involve different problem instances, but different structural solutions to the same routing problem. The two solutions differ only in how stops are grouped into routes (clustering) and how stops are ordered within each route (sequencing). Differences in total distance are thus attributed to routing structure rather than to differences in input data.

For each benchmark scenario, route distances are recomputed using external road-engine measurements. When Google Maps measurements are available, those values are used directly; otherwise, route distance is computed as the average of OSRM and OpenRouteService measurements. The same rule is applied to both solutions.

The primary validation path uses a locally hosted OSRM instance with preprocessed regional OpenStreetMap data, ensuring that both solutions are evaluated under the same routing engine and map dataset. Routes are represented as ordered lists of segments (up to 50 coordinates each), chained sequentially. This segmented representation allows long routes to be evaluated within engine request limits while preserving the exact route structure.

The segment size of 50 coordinates was selected as a practical compromise for local OSRM validation, reducing the number of routing queries required for large instances while remaining within engine constraints. Segmentation does not alter the routing solution; it is only a technical mechanism for evaluating long routes.

Total distance is computed as

$$\text{TotalDistance} = \sum_{\text{routes}} \sum_{\text{segments}} d_{\text{segment}}.$$

It is important to note that the reported distances correspond to this shared external measurement procedure rather than to Amazon's internal evaluation metric. The results therefore demonstrate relative improvements under a consistent and reproducible measurement protocol, rather than strict equivalence with proprietary routing objectives.

The validation framework, including example datasets and execution scripts, is publicly available at:

`https://github.com/vizzito/last-mile-route-verifier`

This framework enables transparent and reproducible comparison by ensuring that both solutions are evaluated on the same set of stops, under the same measurement procedure, and using identical segmentation and routing-engine configurations.

6.6 Scaling experiment beyond benchmark size

The second experiment extends the architecture beyond benchmark scale. Starting from the largest Amazon depot scenario (DLA7, 173,738 stops across 977 routes), progressively larger synthetic scenarios were generated, culminating in a one-million-stop run.

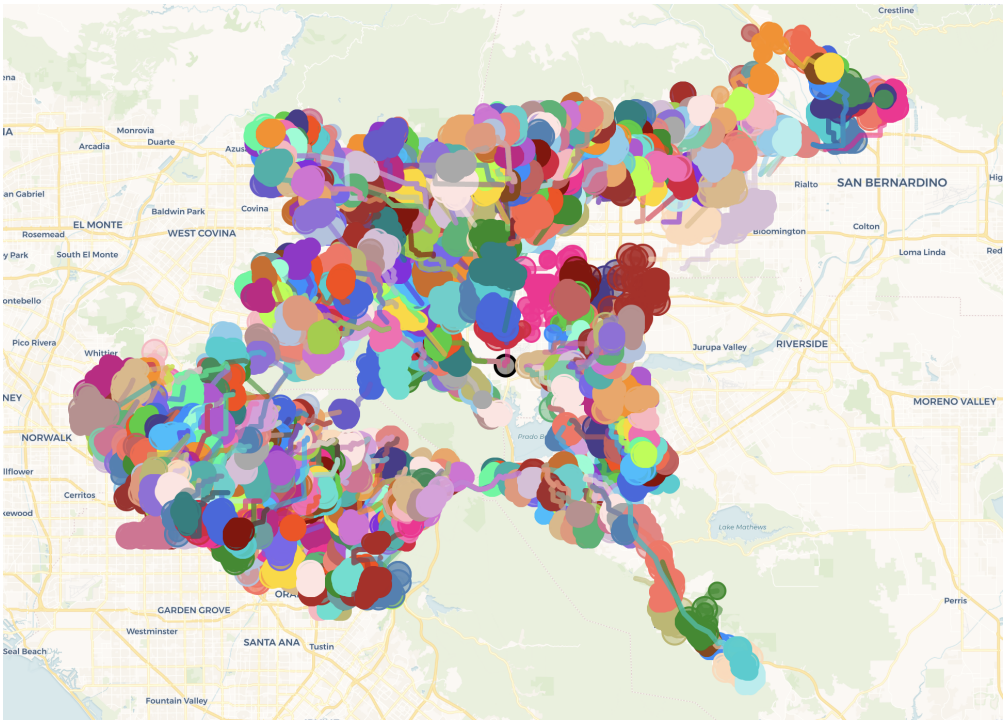


Figure 5: Routing output for depot DLA7 (173,738 stops, 977 routes). Each color represents a distinct route. The compact cluster structure serves as the starting point for the extended scaling experiment.

6.7 Synthetic generation of the one-million-stop scenario.

The one-million-stop scenario was constructed by extending the DLA7 service area to 1,000,000 stops across 4,961 routes. The original spatial distribution of the DLA7 depot (173,738 stops) was used as the reference template. Additional stops were generated by sampling random geographic points within an approximately 35 km radius around the depot center, thereby preserving the same service-area footprint.

Package attributes, including volume and dimensions, were not generated from arbitrary synthetic distributions. Instead, they were sampled from the real package data associated with the original 174K-stop scenario. This preserves the statistical properties of delivery volume and package composition under the enlarged instance.

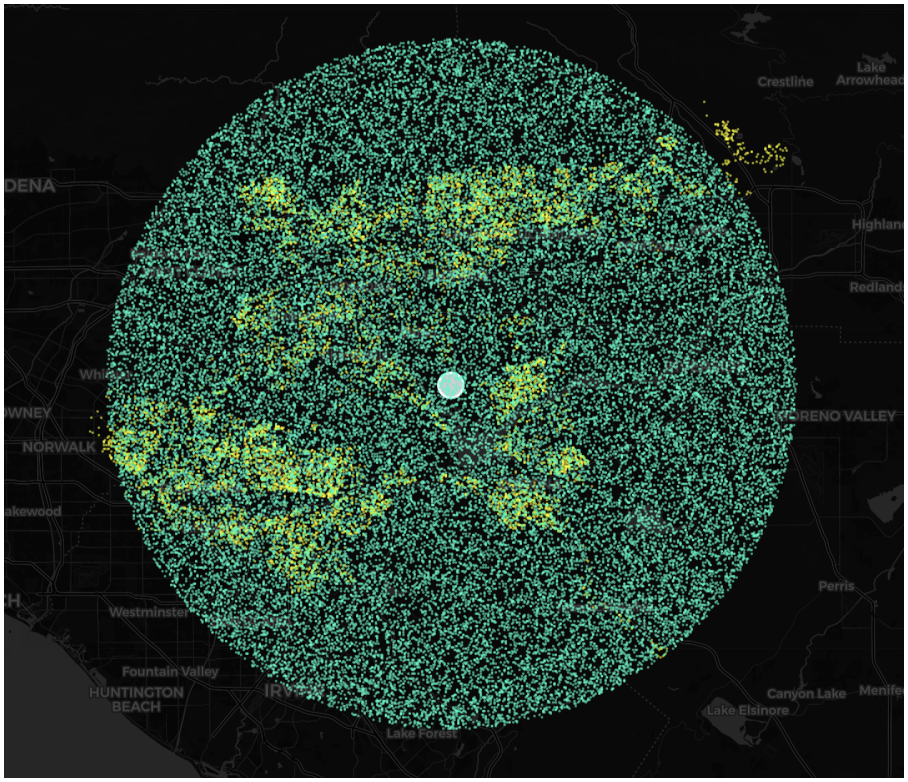


Figure 6: Spatial distribution of the one-million-stop scenario prior to routing. The generated instance combines dense urban clusters with lower-density peripheral regions.

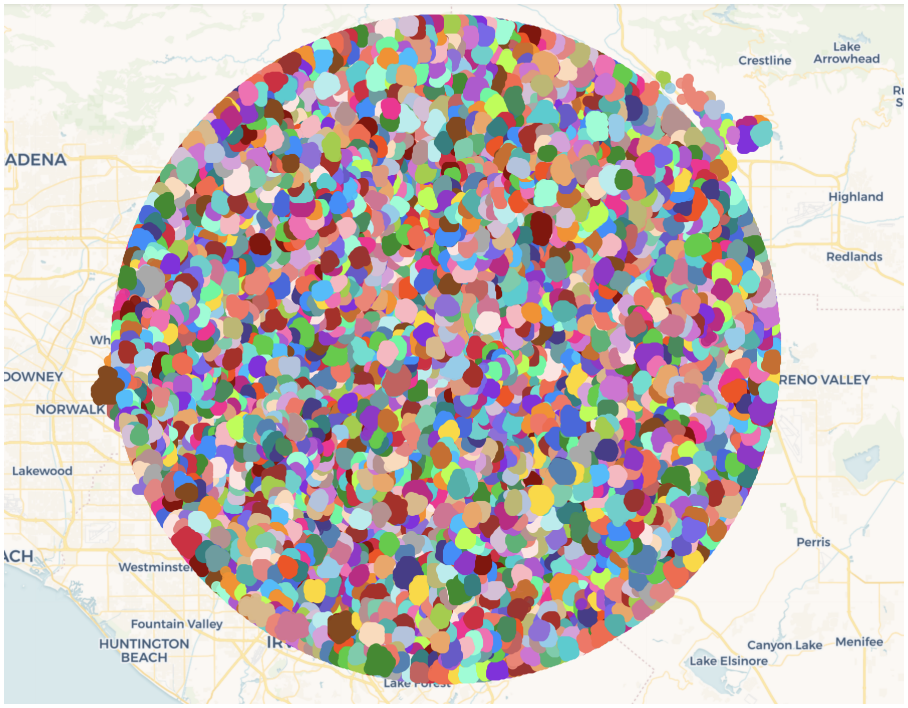


Figure 7: Full routing output for the one-million-stop experiment (1,000,000 stops, 4,961 routes). The solution covers the complete service area without visible fragmentation.

Figures 8a and 8b provide progressive zoom levels where individual routes become visible and remain spatially compact, preserving the structural property observed at benchmark scale despite the order-of-magnitude increase in problem size.

(a) First zoom level: compact clustering across a city-scale subregion.

(b) Second zoom level: individual routes remain clearly separated.

Figure 8: Progressive zoom into the one-million-stop routing output. Spatial compactness and route separation are preserved at finer granularity.

7 Results

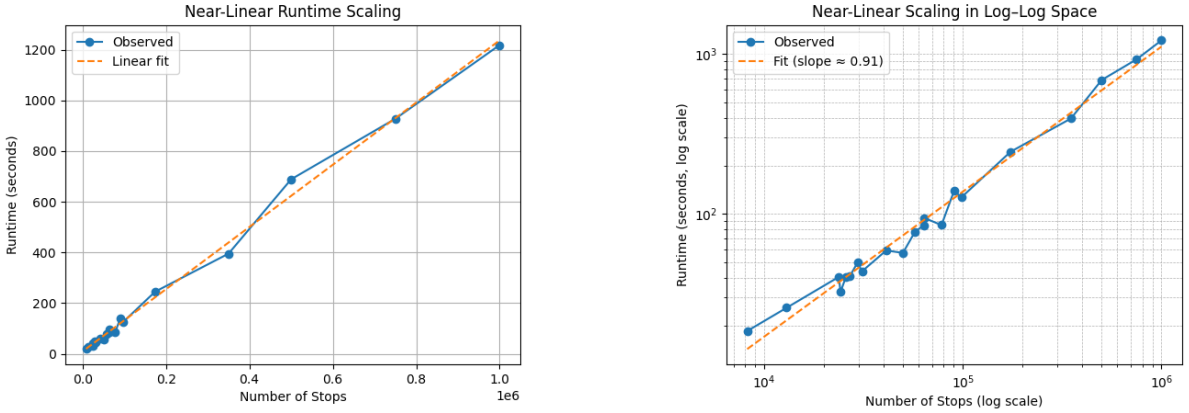
7.1 Runtime scaling

Table 3 summarizes runtime progression from benchmark-scale depots through extended synthetic scenarios up to one million stops. The system processes one million stops in 1,218.84 seconds (approximately 20 minutes), corresponding to roughly 821 stops per second. Smaller benchmark depots are solved in tens of seconds within the same unified pipeline. The transition from the largest real depot (DLA7) to the extended 350K–1M scenarios is continuous, supporting the interpretation that the scaling behavior is architectural rather than incidental.

Table 3: Runtime progression from benchmark-scale depots to extended large-scale runs.

Depot	Routes	Stops	Time (s)	Time (min)
DBO1	55	8,205	18.53	0.31
DSE2	103	12,962	25.91	0.43
DCH2	155	23,751	40.43	0.67
DCH1	180	24,198	32.43	0.54
DLA5	143	25,742	40.34	0.67
DLA4	193	27,035	41.03	0.68
DLA3	214	29,497	49.75	0.83
DAU1	178	31,060	43.80	0.73
DBO2	292	41,126	59.15	0.99
DCH3	251	49,909	57.07	0.95
DLA8	413	57,359	77.33	1.29
DCH4	351	63,550	85.04	1.42
DSE4	398	63,701	94.61	1.58
DSE5	406	78,039	85.19	1.42
DBO3	486	90,362	139.85	2.33
DLA9	636	98,181	126.88	2.11
DLA7	977	173,738	244.88	4.08
DLA7_350K	2250	350,000	395.37	6.59
DLA7_500K	2898	500,000	688.75	11.48
DLA7_750K	3681	750,000	927.89	15.46
DLA7_1M	4961	1,000,000	1218.84	20.31

Figure 9 visualizes the scaling behavior. The linear-scale view shows the gradual runtime growth, while the log–log representation confirms a near-linear pattern (slope ≈ 0.91) across two orders of magnitude in problem size.



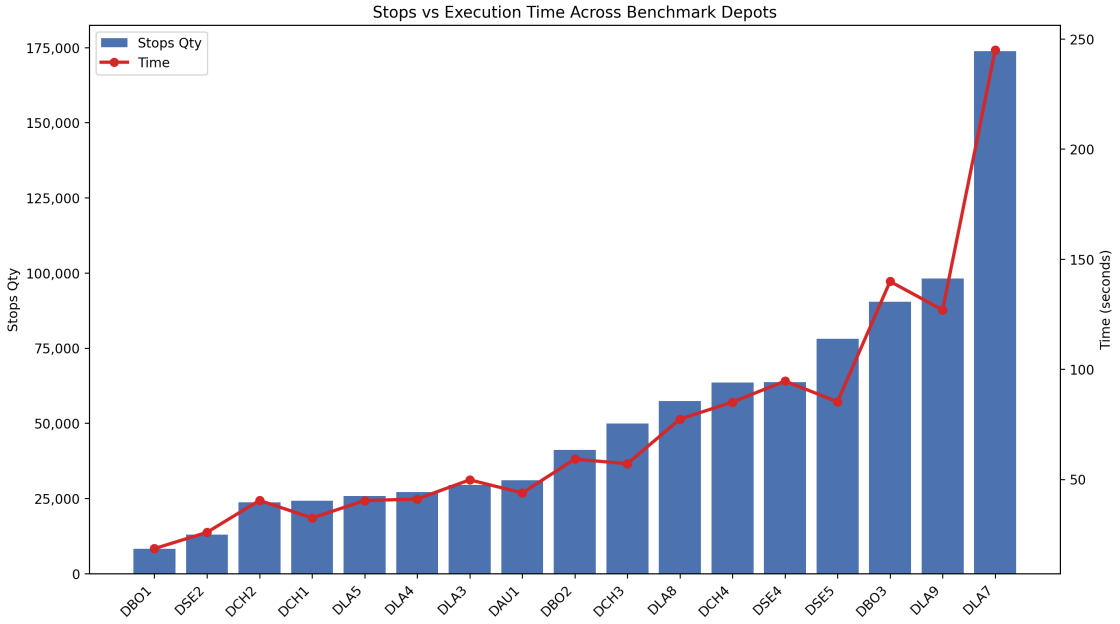
(a) Runtime vs. stops in linear scale.

(b) Log–log representation (slope ≈ 0.91).

Figure 9: Empirical runtime scaling of the proposed routing architecture. The linear-scale view provides an intuitive interpretation, while the log–log view highlights the near-linear growth pattern over increasing problem sizes.

Visual scaling behavior. While Table 3 provides the numerical progression of runtime, the scaling behavior becomes clearer when visualized.

Figure 10 summarizes this behavior at two complementary levels. The upper panel shows execution time across the benchmark depots, where the relationship between stops and runtime remains smooth and stable despite heterogeneous depot sizes and spatial distributions. The lower panel extends the same view to the synthetic scaling experiment, from benchmark scale up to one million stops. The final four scenarios (350K–1M) are highlighted to emphasize the transition into the extended large-scale regime.



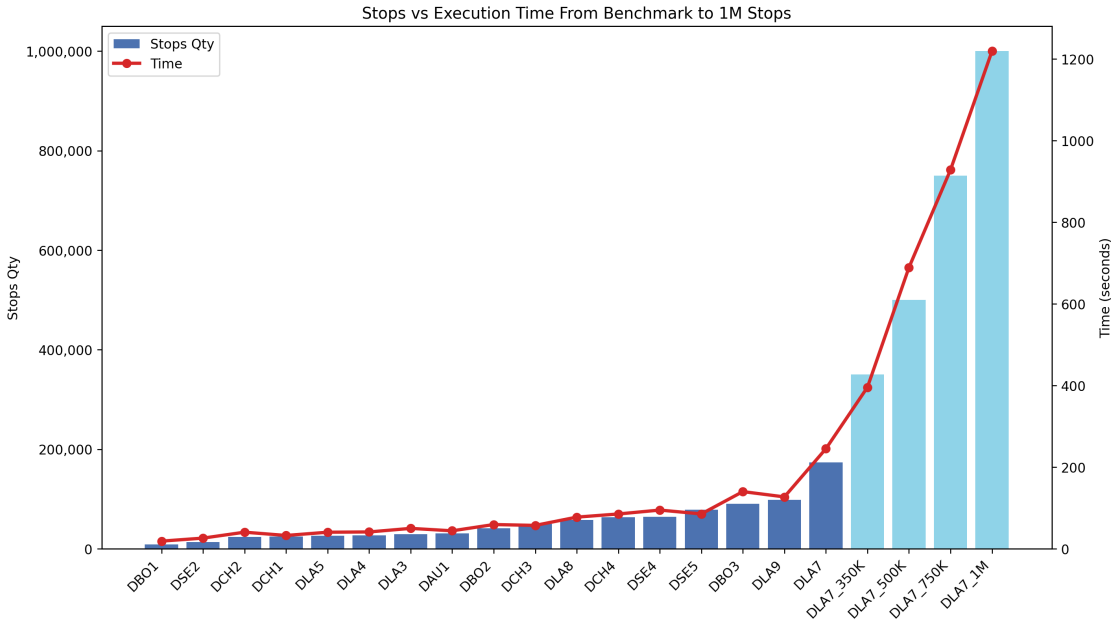


Figure 10: Visual runtime scaling of the proposed architecture. The upper plot shows execution time across benchmark depots, while the lower plot extends the same view to the synthetic scaling experiment up to one million stops. In both cases, bars represent stops quantity and the line represents execution time in seconds. The progression suggests stable runtime behavior at benchmark scale and continued near-linear empirical scaling in the extended large-scale scenarios.

7.2 Benchmark distance comparison

Table 4 reports the aggregate comparison across all benchmark depots under the shared external measurement protocol. The proposed system reduces total measured route distance from 605,606.90 km to 464,296.89 km (−23.33%), while the total number of routes decreases from 6,112 to 5,431 (−11.14%) and average route length drops from 99.08 km to 85.49 km (−13.72%).

Table 4: Aggregate benchmark comparison using average measured distance.

Metric	Baseline	Solver	Delta
Total distance (km)	605,606.90	464,296.89	−23.33%
Total routes	6,112	5,431	−11.14%
Avg. route length (km)	99.08	85.49	−13.72%

7.3 Depot-level comparison

Table 5 and Figure 11 present the depot-by-depot breakdown. The improvement is not driven by a single favorable instance: the proposed system reduces measured distance in every depot, with gains ranging from −2.13% (DCH1) to −34.26% (DSE5) and a mean depot-level improvement of 17.59%. Larger gains tend to appear in depots with higher route counts and greater spatial complexity, consistent with the hypothesis that the architectural benefits become more visible as route-boundary interactions increase. Both solutions service the same set of stops under comparable operational constraints, ensuring that the observed differences reflect routing structure rather than input variation.

Table 5: Depot-level benchmark comparison using average measured distance.

Depot	Amazon km	Solver km	Amazon routes	Solver routes	Δ km
DAU1	22,616	19,877	214	178	−12.11%
DBO1	3,712	3,624	60	55	−2.39%
DBO2	22,083	21,177	296	292	−4.10%
DBO3	84,938	59,940	573	486	−29.43%
DCH1	12,244	11,983	196	180	−2.13%
DCH2	10,754	9,742	154	155	−9.41%
DCH3	30,886	25,755	271	251	−16.61%
DCH4	46,473	35,584	381	351	−23.43%
DLA3	17,046	13,422	254	214	−21.26%
DLA4	16,346	15,201	197	193	−7.00%
DLA5	18,640	16,059	155	143	−13.85%
DLA7	124,697	84,896	1133	977	−31.92%
DLA8	34,112	30,007	448	413	−12.03%
DLA9	55,709	45,914	701	636	−17.58%
DSE2	5,414	3,723	125	103	−31.24%
DSE4	42,728	29,782	446	398	−30.30%
DSE5	57,211	37,611	508	406	−34.26%

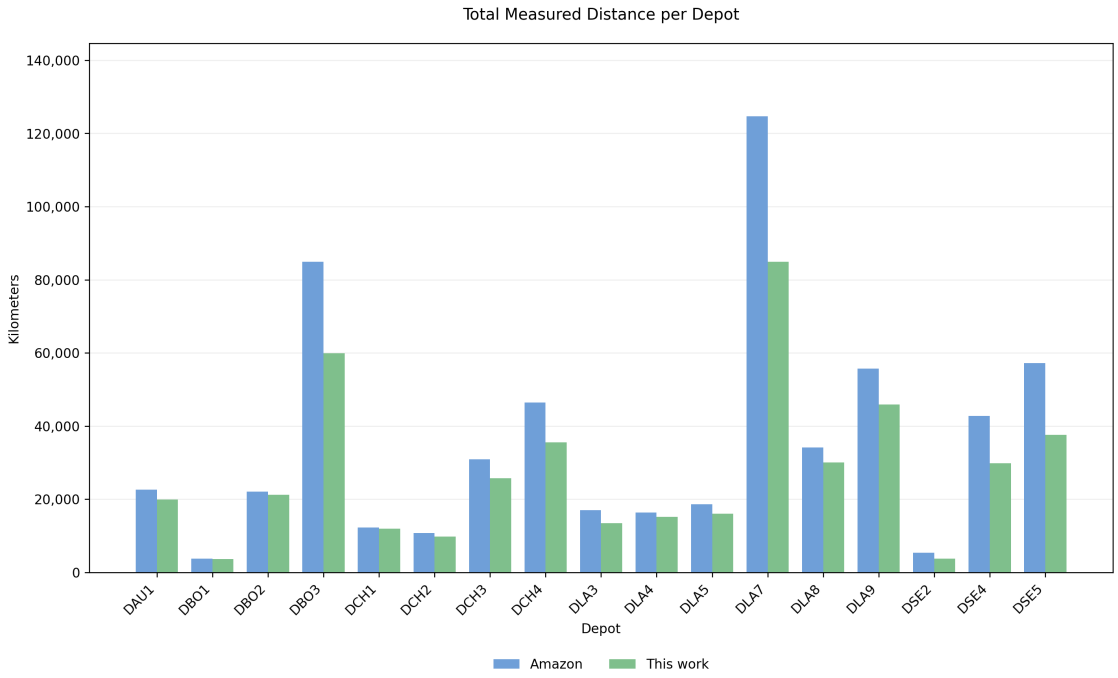


Figure 11: Total measured distance per depot: Amazon baseline vs. proposed system under the shared measurement protocol.

7.4 Fleet efficiency and capacity utilization

Beyond distance, the architecture improves fleet-level operational efficiency (Table 6). The average route reduction across depots is 9.96%, stops per route increase by 11.31%, and average cargo efficiency rises from 74.34% to 83.88% (+9.54 pp). Average optimizer runtime is 0.24 seconds per route, confirming that the route-level primitive remains cheap even at benchmark scale. These results indicate that the architecture reduces distance, reduces route count, and improves load utilization while maintaining a route-level execution cost that remains small relative to planning scale.

Table 6: Cross-scenario summary of fleet efficiency and route-level performance.

Metric	Baseline	Proposed	Delta
Avg. cargo efficiency	74.34%	83.88%	+9.54 pp
Avg. stops per route	144.01	160.30	+11.31%
Avg. route length (km)	99.08	85.49	−13.72%
Avg. runtime per route (s)	—	0.24	—
Avg. route reduction	—	—	−9.96%
Avg. distance improvement	—	—	−17.59%

7.5 Time-window adherence and operating-mode trade-offs

The time-window behavior of the system should be interpreted together with the operating mode selected in Section 6.3. In more consolidation-oriented configurations, the planner reduces route count and increases fleet utilization by accepting tighter route packing and lower temporal slack. This improves structural efficiency, but tends to increase time-window pressure.

In the one-million-stop experiment, 55,040 stops carried time windows, of which 53,113 were served within the target window and 1,927 were served outside it, corresponding to a violation rate of approximately 3.5% [11]. By comparison, a secondary analysis of the released Amazon dataset reports an observed time-window violation rate of approximately 1.81% in the historical routes [10].

Additional internal runs under more standard operating conditions, where the planner does not aggressively minimize route count or maximize fleet consolidation, have produced time-window violation rates closer to 2%. Across the evaluated scenarios, the observed time-window violation regime is therefore approximately 1% to 3.5%, depending on how strongly the planner is tuned toward consolidation versus punctuality.

This behavior is consistent with the architecture described in Section 6.3: route count, total distance, fleet utilization, and time-window adherence are coupled outputs rather than independently optimizable objectives. Reducing time-window violations generally requires deploying more vehicles, increasing route count, and accepting a higher total traveled distance.

8 Discussion

8.1 Why the architecture works in practice

The observed scaling behavior and quality improvements do not arise from a single optimization technique, but from the way several bounded stages are composed. The planner avoids treating the full instance as one monolithic optimization problem. Instead, it decomposes the workload

into route-sized and region-sized units, repairs inconsistencies before route construction, and reuses graph and distance information across stages.

Four mechanisms are particularly important.

Bounded route-level optimization. Individual route optimization is treated as a reusable primitive with bounded cost. Because each route can be improved quickly, the global problem can be reorganized as the coordination of many route candidates rather than as one centralized tour-construction problem. This shifts the practical bottleneck from combinatorial search over all stops to the orchestration of many bounded tasks.

Constraint-aware clustering with global intent. Clustering is not used as purely geometric preprocessing. It incorporates signals such as package volume, vehicle capacity, stop density, and operational bounds, producing route candidates that are already meaningful from a logistics perspective. This reduces the amount of correction required downstream and prevents obviously infeasible decompositions from becoming final routes.

Boundary-restricted rebalancing. Parallel clustering and initial allocation can introduce local inconsistencies near cluster borders. The rebalancing stage repairs these inconsistencies through local exchanges restricted to neighboring clusters and frontier stops. This preserves spatial compactness while reducing residual infeasibility, avoiding the need to recompute the global partition after every local correction.

Caching and localized graph reuse. A significant portion of routing cost in practice comes from repeated graph construction, distance evaluation, and preprocessing. The multi-level caching strategy reduces this hidden recomputation by reusing localized routing graphs, distance matrices, and spatial artifacts across clustering, rebalancing, and route construction. This helps runtime scale with newly introduced work rather than with repeated reconstruction of the same information.

Taken together, these mechanisms explain why wall-clock growth remains comparatively gradual as input expands from benchmark-scale depots to one million stops. The observed near-linear behavior should therefore be interpreted as an empirical property of the tested architecture and workload range, not as a claim of linear-time complexity for the general Vehicle Routing Problem. This interpretation is consistent with prior very-large-scale routing research, which emphasizes that scalability depends not only on local-search quality, but also on memory control, bounded neighborhoods, and the organization of computation [3, 1].

8.2 Relation to prior large-scale routing work

Accorsi and Vigo demonstrated that very large CVRP-class instances, including instances with up to one million customers, can be processed in minutes using carefully engineered heuristics [1]. Arnold et al. similarly showed that performance improves substantially when time and space complexity are controlled at the level of local-search execution, memory usage, and neighborhood evaluation [3]. These studies establish that very large routing is computationally feasible when scalability is treated as a systems property rather than only as an algorithmic property.

The present work follows that systems-oriented direction, but differs in emphasis. Its focus is operational last-mile planning rather than benchmark CVRP alone: the pipeline combines package volume, heterogeneous fleet constraints, configurable time-window handling, route-boundary rebalancing, and commodity-hardware deployment. The contribution is therefore not a new exact

solver, but a practical architecture for organizing large-scale last-mile planning into bounded, cooperating stages.

8.3 Limitations and threats to validity

Several limitations should be stated explicitly.

First, the Amazon benchmark is a large public dataset based on real operational routes, but it does not represent all production settings. The public challenge was framed around learning from driver behavior rather than purely minimizing geometric distance [6, 2]. The reported distance reductions are meaningful under the shared external measurement protocol used in this paper, but they should not be interpreted as equivalence with Amazon's internal operational objective.

Second, the one-million-stop experiment is an extended scaling study rather than a direct public benchmark with a pre-existing industry baseline. Its purpose is architectural: to test whether the same planning stack continues to operate under substantially larger inputs.

Third, the system's performance is closely tied to implementation choices. As prior work has noted, the boundary between "algorithm" and "system" blurs at scale [3, 1]. The observed results depend on execution strategy, data structures, concurrency design, caching policy, and the implementation of route-boundary rebalancing.

Fourth, the reported experiments correspond to single runs under fixed configurations. Additional validation through repeated runs, variance analysis, broader third-party replication, and direct comparison against open-source solvers such as PyVRP or VROOM would further strengthen the empirical evaluation.

Finally, not all implementation details are disclosed. The paper therefore should be read primarily as a systems-oriented technical report with externally verifiable measurement results, rather than as a fully reproducible algorithmic specification. The public validation framework supports reproducibility of the distance comparison, but not full reproduction of the solution-generation pipeline.

Despite these limitations, the results indicate that large-scale last-mile routing can be addressed effectively through system-level design: by decomposing the workload into bounded stages, repairing locally before route construction, and avoiding redundant computation through caching and localized graph reuse.

9 Conclusion

This work shows that large-scale last-mile routing can be made practical by treating it as a systems problem rather than a purely algorithmic one. The proposed architecture enables large routing workloads to be planned in seconds or minutes under real operational constraints, directly affecting delivery distance, fleet utilization, planning time, and operational cost.

Under the evaluated conditions, the system reduces total measured route distance by 23.3% and route count by 11.1% relative to the released Amazon baseline routes, while achieving a mean depot-level distance improvement of 17.59%. In the extended scaling experiment, the same architecture processes one million stops in approximately 20 minutes on commodity hardware, exhibiting near-linear empirical runtime growth over the tested range.

The main result is architectural. Large-scale routing performance is not determined only by the local quality of a routing heuristic, but also by how computation is organized across clustering, allocation, rebalancing, route construction, and graph reuse. By decomposing the workload into bounded, coordinated stages, the system keeps each planning operation tractable and prevents any single monolithic optimization step from dominating the full workload.

As a result, additional hardware acts primarily as an accelerator of response time rather than as a prerequisite for feasibility. The same planning system can operate across a wide range of scales without specialized infrastructure, making large-scale last-mile routing more accessible as a practical, on-demand service.

References

[1] Luca Accorsi and Daniele Vigo. "Routing One Million Customers in a Handful of Minutes". In: *Computers & Operations Research* (2024). DOI: 10.1016/j.cor.2024.106562. URL: https://www.sciencedirect.com/science/article/pii/S0305054824000340.

[2] Amazon Science. *Winning Last Mile Challenge Team Addresses Problem of Combining Mathematical Routes with Driver Knowledge*. 2021. URL: https://www.amazon.science/academic-engagements/winning-last-mile-challenge-team-addresses-problem-of-combining-mathematical-routes-with-driver-knowledge (visited on 04/02/2026).

[3] Florian Arnold, Michel Gendreau, and Kenneth Sörensen. "Efficiently Solving Very Large-Scale Routing Problems". In: *Computers & Operations Research* 107 (2019), pp. 32–42. DOI: 10.1016/j.cor.2019.03.006. URL: https://publications.polymtl.ca/38921/.

[4] Ethan Gibbons, Mario Ventresca, and Beatrice M. Ombuki-Berman. *Split Algorithm in Linear Time for the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows*. 2026. URL: https://arxiv.org/abs/2601.17572 (visited on 04/02/2026).

[5] Google for Developers. *Configure Timeouts and Deadlines — Route Optimization API*. 2026. URL: https://developers.google.com/maps/documentation/route-optimization/timeouts (visited on 04/02/2026).

[6] Daniel Merchán, Nitish Arora, et al. "2021 Amazon Last Mile Routing Research Challenge: Data Set". In: *Amazon Science / arXiv* (2022). URL: https://www.amazon.science/publications/2021-amazon-last-mile-routing-research-challenge-data-set.

[7] MIT Center for Transportation and Logistics. *Amazon Last-Mile Routing Research Challenge*. 2021. URL: https://routingchallenge.mit.edu/ (visited on 04/02/2026).

[8] NextBillion.ai. *Route Optimization API*. 2026. URL: https://docs.nextbillion.ai/optimization/route-optimization-api (visited on 04/02/2026).

[9] RouteSavvy. *RouteSavvy API PLUS Overview*. 2026. URL: https://www.routesavvy.com/routesavvy-api-plus/ (visited on 04/02/2026).

[10] Ankur Singh. *Understanding Last-Mile Delivery: An Analysis of the Amazon Last Mile Routing Dataset*. Master's dissertation, University of Liverpool. 2023. URL: https://www.researchgate.net/publication/373923950_Understanding_Last-Mile_Delivery_An_Analysis_of_the_Amazon_Last_Mile_Routing_Dataset.

[11] Martin Vizzolini. *Last-Mile Route Optimization at 1 Million Stops, With Near-Linear Scaling*. 2026. URL: https://medium.com/@martinvizzolini/last-mile-route-optimization-at-1-million-stops-with-near-linear-scaling-e4d4b0118e80 (visited on 04/02/2026).