

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 constraints, including time windows, vehicle capacity, package volume, and route stop limits. Unlike conventional systems that require pre-partitioning or large-scale infrastructure, the proposed system solves the full fleet planning problem as a single coherent optimization process on commodity hardware.

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], the architecture reduces aggregate measured route distance by 23.3% and route count by 13.6% relative to the released baseline under a shared external measurement protocol, with a mean depot-level distance improvement of 17.59%, while satisfying all operational constraints and servicing all stops [10, 11].

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.

This work explicitly challenges a common implicit assumption in both academic literature and commercial routing systems: that large-scale routing workloads require substantial computational infrastructure. In practice, this assumption is reflected in API limits, timeout policies, and the need for pre-decomposition of routing problems. The architecture presented here was developed under the opposite premise: that large-scale vehicle routing should remain solvable on commodity hardware, without requiring specialized infrastructure or artificial constraints on input size.

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.

The remainder of the paper is organized as follows. Section 2 reviews related work and positions the contribution. Section 4 defines the operational problem. Section 5 describes the system architecture. Section 6 presents the experimental setup, and Section 7 reports the results. Section 8 discusses implications, limitations, and directions for future work.

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.

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.

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 challenges that premise. Rather than treating hardware as effectively unbounded, the proposed system was developed under the opposite assumption: large-scale vehicle routing should remain solvable on commodity hardware, without requiring specialized infrastructure, large distributed clusters, or expensive cloud deployments.

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

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

3.1 Evidence from commercial systems

This premise becomes visible when examining how modern routing systems are exposed in practice.

Public route-optimization APIs typically impose limits or operational constraints that reflect underlying computational costs. For example, Google documents different timeout strategies for routing requests exceeding 10,000 shipments, indicating that large problem instances require careful resource management [5]. Other systems impose explicit caps on problem size: NextBillion documents limits on the order of thousands of locations per optimization problem, while RouteSavvy advertises routing limits of roughly one thousand locations per request [8, 9].

These limits do not necessarily imply algorithmic impossibility. Rather, they reflect the economic and operational constraints of centralized routing systems, where large instances must be tightly controlled to remain computationally and commercially viable.

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.

The contribution of this work is therefore not to “solve” the VRP in a theoretical sense, but to recast it as a systems problem. 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 many of the practical limits observed in commercial systems are not inherent properties of the routing problem itself, but consequences of specific architectural choices. 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 algorithmic results, but as evidence that the deployment model of large-scale routing can be fundamentally reconsidered.

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 $S = \{s_1, \dots, s_n\}$, a fleet of vehicles $V = \{v_1, \dots, v_m\}$, 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: vehicle capacity limits (volume and weight) 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 clustering under global constraints

Input stops are partitioned into chunks processed concurrently, but the objective is not to create disconnected subproblems. Each worker identifies route candidates while incorporating practical signals: package volume, vehicle capacity, and geographic density. The system identifies local grouping patterns that are later reconciled into a shared fleet plan, so that concurrent processes contribute to a globally consistent structure rather than operating in isolation.

5.2 Constraint-aware initial allocation

Stops are allocated into route candidates under operationally meaningful constraints driven by delivery volume, fleet capacity, and spatial density. This stage generates an initial fleet plan that is already plausible from a delivery perspective rather than merely a geometric partition. It acts as the first approximation to a coherent solution and largely determines the quality of the search space available to later refinement.

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. Only stops located near the frontier between adjacent clusters are eligible for transfer, preserving spatial

compactness while repairing constraint violations introduced during parallel clustering and initial allocation.

The negotiation is fully distributed: each region evaluates its boundary stops, proposes or accepts exchanges with adjacent regions based on local feasibility conditions, and converges through repeated local decisions. No centralized arbitration is required. This produces a distributed repair process that reduces pathological boundary allocations and improves both feasibility and spatial compactness before route optimization begins.

This mechanism is central to the architecture because it allows the planner to decompose first and repair before routing. At large scale, the initial clustered solution does not need to be perfect; it only needs to provide feasible region candidates that can be improved through constrained local exchanges at shared boundaries.

A practical advantage of the rebalancing mechanism is its extensibility. New operational constraints, such as requiring specific package categories to be assigned to compatible vehicle types, can be introduced as additional feasibility conditions within the same local negotiation process, without modifying the broader pipeline. Figure 1 illustrates the four stages of this process.

5.4 Route-level optimization as a primitive

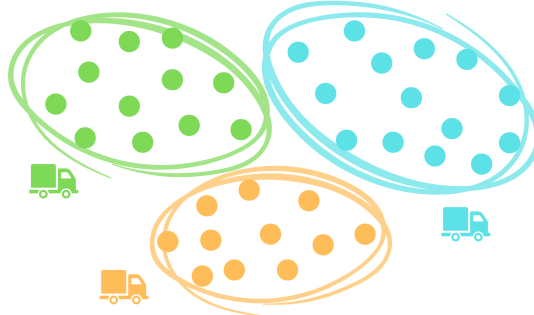
The first architectural requirement is that individual route optimization must be fast enough to serve as a reusable primitive. The system uses a probabilistic route-optimization procedure that typically completes in fractions of a second for routes ranging from a few dozen to several hundred stops. This bounded cost is what makes large-scale parallel dispatch feasible: once a single route is cheap to optimize, the broader problem shifts from solving isolated tours to coordinating many route candidates at fleet scale.

5.5 Parallel route construction

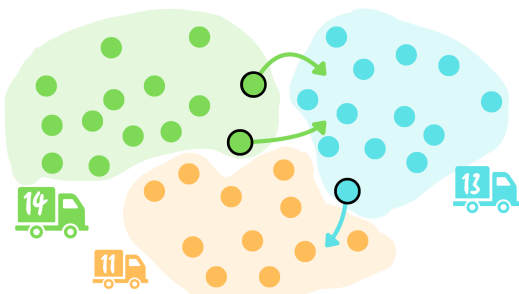
Once cluster candidates have been consolidated and rebalanced, each becomes an independent route-optimization job dispatched concurrently. Because route optimization is already cheap (Section 5.4), overall wall-clock time depends less on total stop count than on how efficiently jobs are scheduled. The planning cost is redistributed from a single centralized computation into many bounded parallel optimizations.



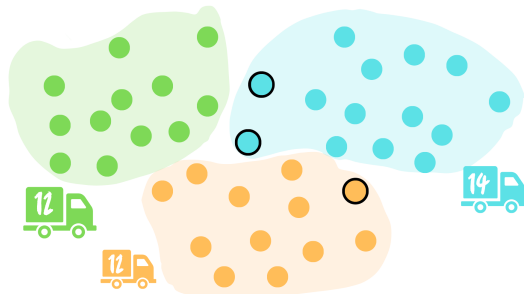
(a) Initial clustering with boundary imbalances.



(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.

5.6 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.

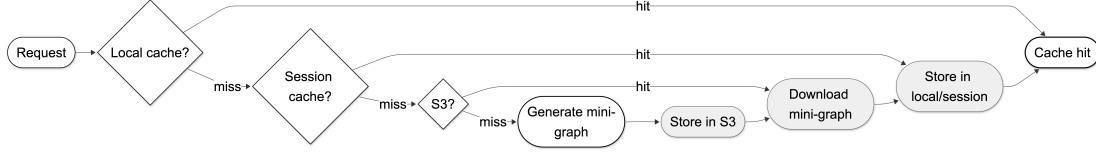


Figure 2: Cache hierarchy. Requests attempt reuse from instance-local memory, then session-level cache, then persistent storage. Only cache misses trigger graph generation, after which artifacts propagate upward for reuse.

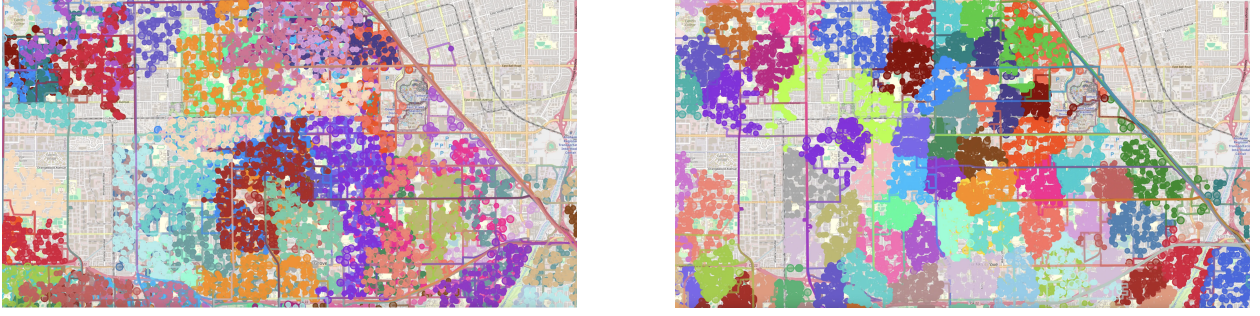


Figure 3: Overlapping clusters from the Amazon dataset (left) versus spatially compact clusters generated by the proposed system (right) for the same set of stops. Compact clustering reduces route interference and improves fleet-level efficiency.

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.

5.7 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 5.3.

This spatial behavior can be quantified through *inter-cluster interference*. For a partition $\mathcal{P} = \{C_1, \dots, C_m\}$ 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.

5.8 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.

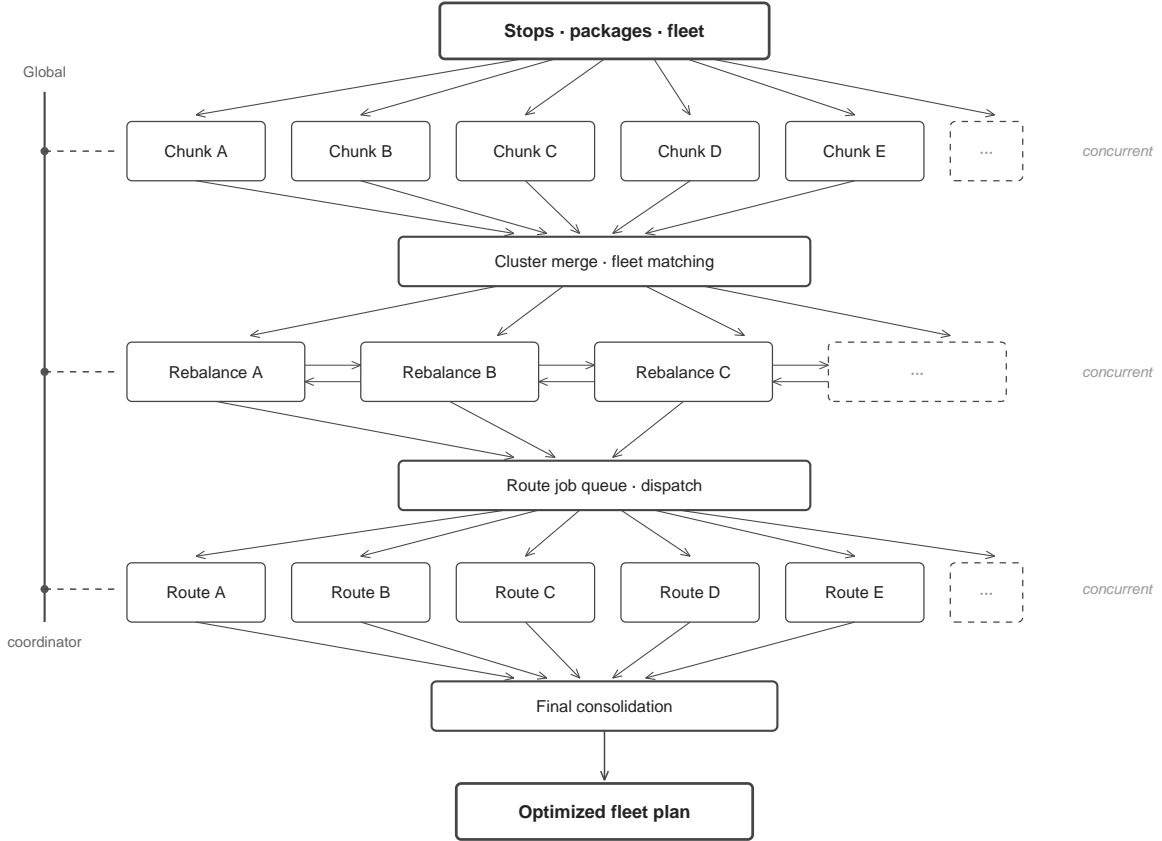


Figure 4: Architecture of the proposed routing pipeline. Three concurrent processing layers — chunk-level clustering, distributed rebalancing, and route-level optimization — are connected through queues between stages and supervised by a shared orchestration layer that preserves global consistency without 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, including capacity, volume, time windows, and route stop limits. Both the released Amazon routes and the proposed routes are then evaluated under the shared measurement protocol described below.

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.

6.3.3 Interpretation of parameter effects

These controls are part of the operational meaning of the system itself rather than minor implementation details. Increasing loading targets may reduce fleet size while increasing route length and time-window pressure. Prioritizing time over distance may improve punctuality while allowing a modest increase in route count. Dense urban service areas may tolerate more aggressive consolidation, whereas sparse regions often benefit from smaller routes and slightly lower loading targets in order to avoid zig-zag patterns and excessive detours.

For this reason, the benchmark and scaling results reported here should be interpreted as outcomes under a specific practical configuration rather than as universal maxima under every possible parameter setting. This is a strength rather than a weakness of the architecture: the system is scalable, but also operationally tunable.

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.

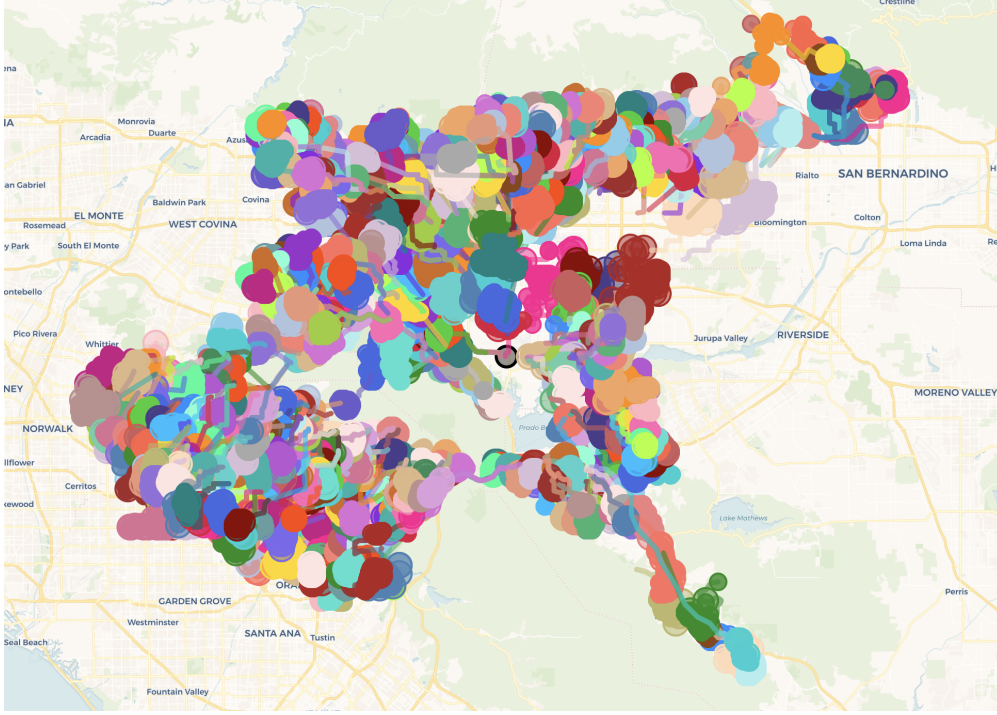


Figure 5: Routing output for depot DLA7 (173,738 stops, 977 routes). Each color represents a distinct route. The compact cluster structure serves as the starting point for the extended scaling experiment.

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.

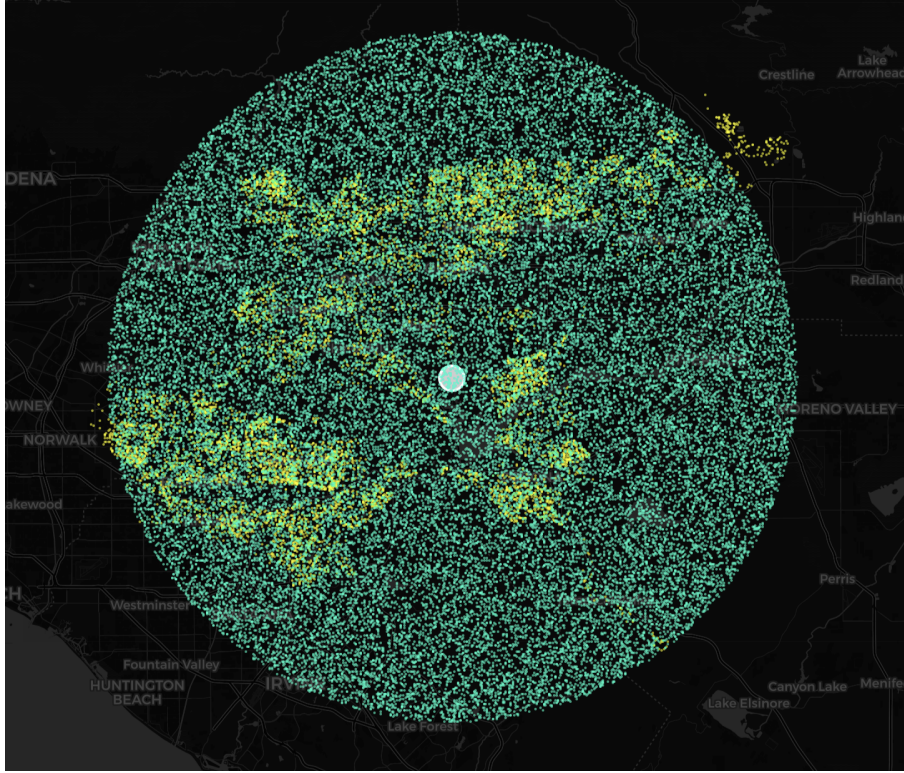


Figure 6: Spatial distribution of the one-million-stop scenario prior to routing. The generated instance combines dense urban clusters with lower-density peripheral regions.

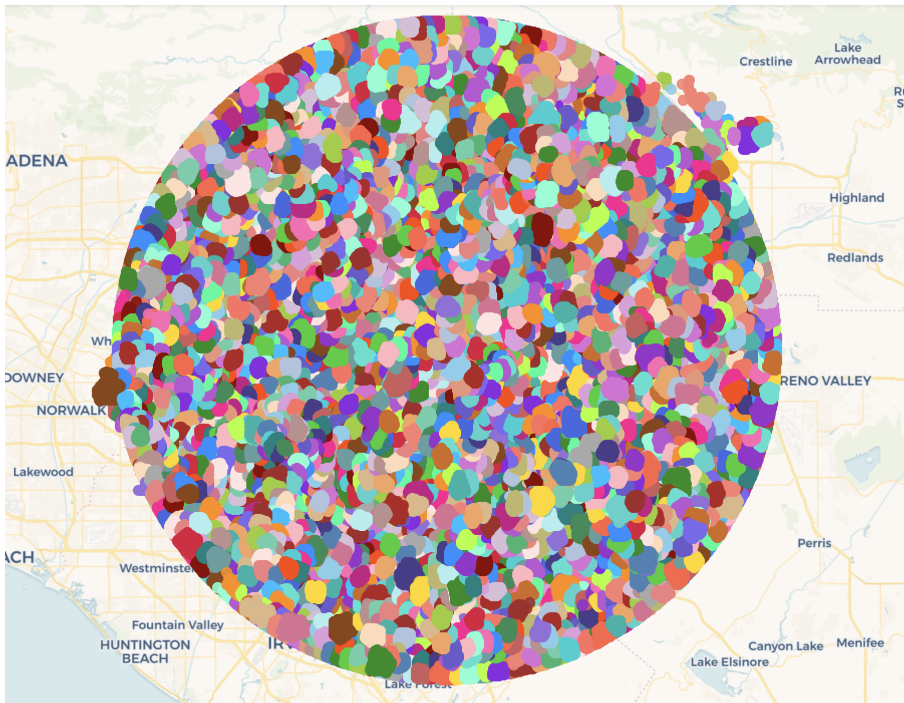
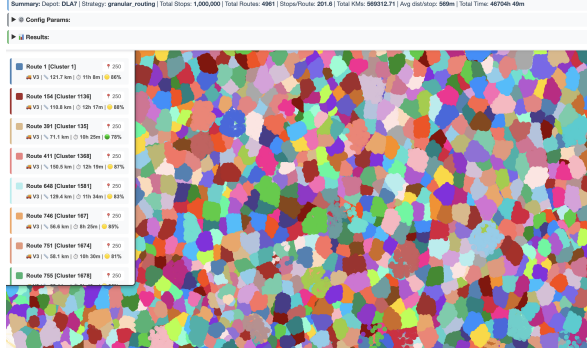


Figure 7: Full routing output for the one-million-stop experiment (1,000,000 stops, 4,961 routes). The solution covers the complete service area without visible fragmentation.

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

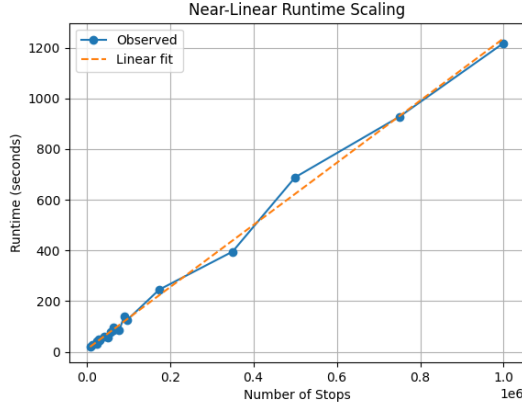
7.1 Runtime scaling

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.

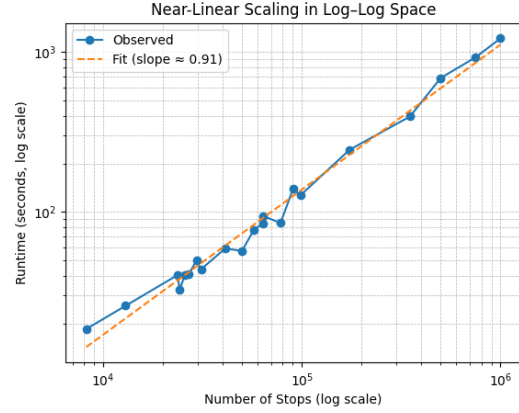
Table 3: Runtime progression from benchmark-scale depots to extended large-scale runs.

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.

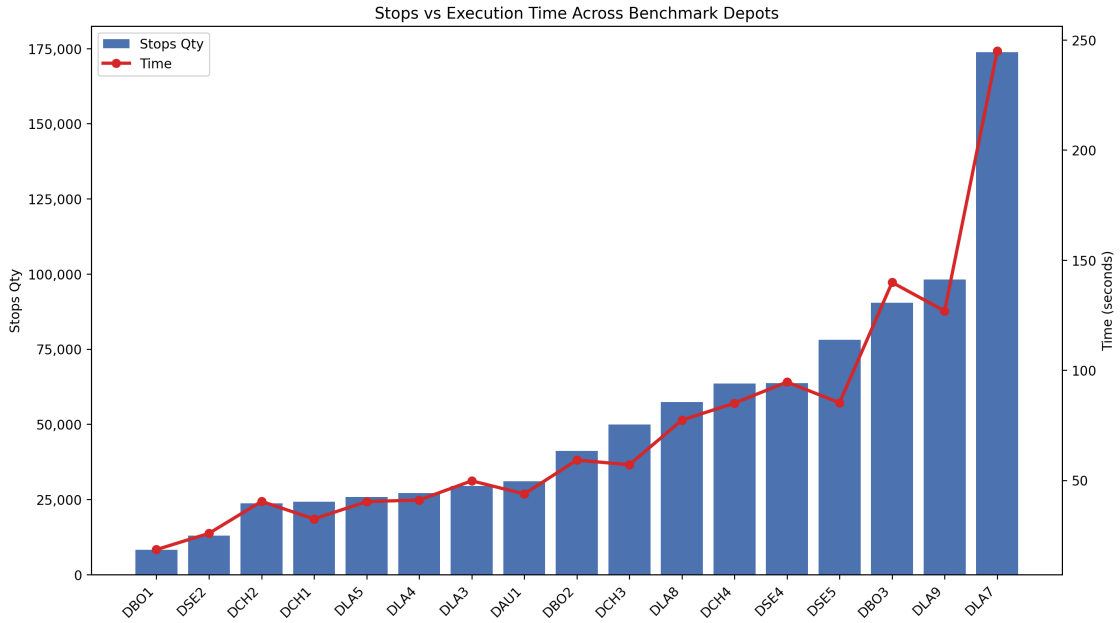


(b) Log-log representation (slope ≈ 0.91).

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.



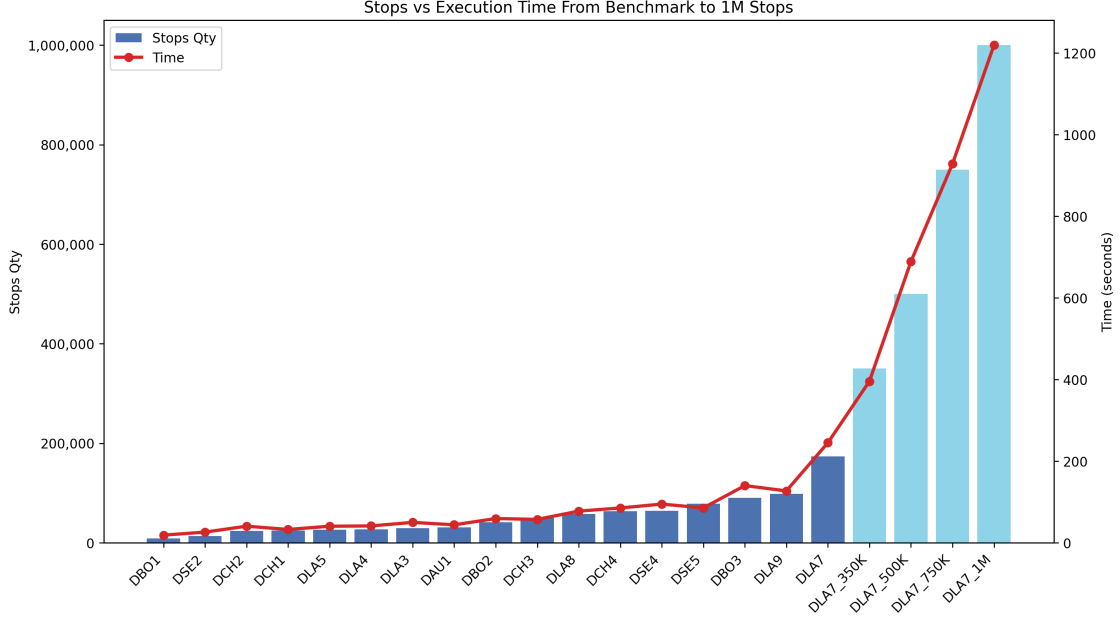


Figure 10: Visual runtime scaling of the proposed architecture. The upper plot shows execution time across benchmark depots, while the lower plot extends the same view to the synthetic scaling experiment up to one million stops. In both cases, bars represent stops quantity and the line represents execution 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 5,283 (-13.56%) and average route length drops from 99.08 km to 87.89 km (-11.30%).

Table 4: Aggregate benchmark comparison using average measured distance.

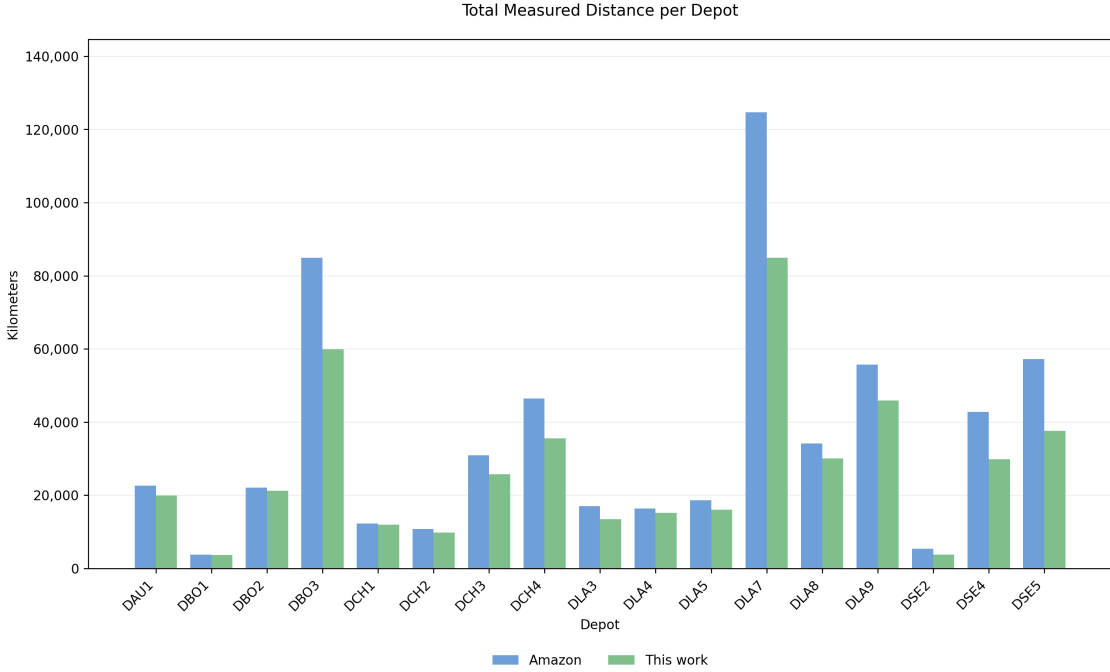
Metric	Baseline	Solver	Delta
Total distance (km)	605,606.90	464,296.89	-23.33%
Total routes	6,112	5,283	-13.56%
Avg. route length (km)	99.08	87.89	-11.30%

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	283	−4.10%
DBO3	84,938	59,940	573	467	−29.43%
DCH1	12,244	11,983	196	175	−2.13%
DCH2	10,754	9,742	154	150	−9.41%
DCH3	30,886	25,755	271	235	−16.61%
DCH4	46,473	35,584	381	321	−23.43%
DLA3	17,046	13,422	254	210	−21.26%
DLA4	16,346	15,201	197	196	−7.00%
DLA5	18,640	16,059	155	140	−13.85%
DLA7	124,697	84,896	1133	946	−31.92%
DLA8	34,112	30,007	448	422	−12.03%
DLA9	55,709	45,914	701	624	−17.58%
DSE2	5,414	3,723	125	96	−31.24%
DSE4	42,728	29,782	446	385	−30.30%
DSE5	57,211	37,611	508	400	−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 12.31%, stops per route increase by 14.54%, 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 simultaneously reduces distance, 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	164.95	+14.54%
Avg. route length (km)	99.08	87.89	−11.30%
Avg. runtime per route (s)	—	0.24	—
Avg. route reduction	—	—	−12.31%
Avg. distance improvement	—	—	−17.59%

8 Discussion

8.1 Why the observed behavior appears near-linear

The central observation is that wall-clock growth remains comparatively gradual as input expands from benchmark-scale depots to one million stops. This behavior is consistent with the architectural design: fast route-level optimization, concurrent clustering, localized graph reuse, and distributed rebalancing prior to route construction within a shared planning structure.

From a systems perspective, the architecture shifts the practical bottleneck from monolithic centralized planning to the orchestration of many bounded tasks. This interpretation aligns with prior very-large-scale routing research, which has shown that scalability depends less on a single algorithmic breakthrough than on how the solver controls memory growth, limits neighborhood expansion, and structures incremental computation [3, 1].

Caching and localized graph reuse are particularly important. By eliminating repeated preprocessing and bounding memory growth, the system avoids having its scaling behavior dominated by hidden recomputation costs. The observed near-linear behavior should therefore be interpreted as the combined effect of architectural choices across route-level optimization, parallel execution, caching, and rebalancing, rather than as an incidental artifact of a single run.

8.2 Relation to prior large-scale routing work

Accorsi and Vigo demonstrated that very large CVRP-class instances (up to one million customers) can be processed in minutes with carefully engineered heuristics [1]. Arnold et al. showed that performance improves dramatically when time and space complexity are controlled at the level of local-search execution and data handling [3]. These studies establish that very large routing is within reach of modern optimization.

The present work differs in emphasis. Its contribution is framed around operational last-mile planning rather than benchmark CVRP alone, with an architectural focus on preserving global

coherence across clustering, route construction, and boundary correction while remaining practical on commodity hardware. The work belongs to the same broad research direction, but targets a more deployment-oriented last-mile setting with richer operational constraints.

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]. While outperforming the released baseline in total kilometers is a meaningful result, different operational objectives may lead to different optimal solutions.

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 to demonstrate that the architecture continues to operate and scale 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, and the implementation of route-boundary rebalancing.

Fourth, the reported experiments correspond to single runs under a fixed configuration. 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.

Despite these limitations, the results consistently indicate that large-scale routing can be addressed effectively through system-level design, rather than relying solely on advances in standalone optimization algorithms.

8.4 Why the architecture works in practice

The observed performance improvements and scaling behavior arise from a combination of architectural choices rather than from a single optimization technique. Four mechanisms are particularly important.

Bounded route-level optimization. Individual route optimization is treated as a primitive with tightly bounded cost. Because each route can be optimized in fractions of a second, the global problem can be decomposed into many independent routing tasks without creating a new computational bottleneck. This shifts the scaling constraint from algorithmic complexity to task orchestration.

Constraint-aware clustering with global intent. Clustering is not purely geometric. It incorporates volume, capacity, and density signals, producing route candidates that are already operationally meaningful. This reduces the search space of the routing stage and avoids pathological decompositions that would otherwise require expensive correction later.

Boundary-restricted rebalancing. Parallel clustering introduces local inconsistencies near cluster borders. The system resolves these through a distributed rebalancing process restricted to boundary regions. By allowing only local exchanges between neighboring clusters, the system

restores feasibility and improves spatial compactness without requiring global recomputation. This mechanism is critical for preserving fleet-level coherence after parallel decomposition.

Caching and localized graph reuse. A significant portion of routing cost in practice comes from repeated graph construction and distance evaluation. The multi-level caching strategy eliminates redundant work across clustering, rebalancing, and routing stages. By localizing graph generation and reusing distance computations, the system ensures that runtime scales with new information rather than repeated computation.

Taken together, these mechanisms transform the routing problem from a monolithic optimization task into a coordinated system of bounded operations. The result is a planner whose performance is dominated by the number of independent routing units rather than by the combinatorial explosion of the full problem, explaining both the observed near-linear runtime behavior and the ability to operate on commodity hardware.

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 routing workloads to be solved in seconds or minutes under real operational constraints, directly impacting delivery time, fleet utilization, and total operational cost.

Under the evaluated conditions, the system reduces total measured route distance by 23.3% and route count by 13.6% relative to the released Amazon baseline, while achieving a mean depot-level improvement of 17.59%. More importantly, it processes up to one million stops in approximately 20 minutes on commodity hardware, exhibiting near-linear empirical runtime growth as input size increases.

This behavior suggests that routing performance is no longer fundamentally limited by input size, but by how computation is organized. By decomposing the problem into bounded, coordinated stages, the system avoids the combinatorial explosion typically associated with large-scale routing and enables predictable scaling across orders of magnitude.

As a result, increasing hardware capacity acts primarily as an accelerator of response time rather than as a requirement for feasibility. The same planning system can operate across a wide range of scales without relying on specialized infrastructure, making large-scale routing 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] Martin Vizzolini. *A Last-Mile Optimizer that Outperforms Amazon’s Routes on a Laptop*. 2024. URL: <https://medium.com/@martinvizzolini/a-last-mile-optimizer-that-outperforms-amazons-routes-on-a-laptop-24242f93eb74> (visited on 04/02/2026).
- [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).