

A Speculative Verification and Expert Repair Framework for Dynamic Vehicle Routing

Qingyou Wang

w947366965@163.com

Hubei University of Automotive Technology

Research Article

Keywords: speculative execution, dynamic vehicle routing, online re-optimization, route-prefix caching, latency-bounded computation, feasibility recovery

Posted Date: October 15th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10382036/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

A Speculative Verification and Expert Repair Framework for Dynamic Vehicle Routing

Abstract

Online dispatch systems for dynamic vehicle routing with time windows must re-optimize partially committed routes within seconds whenever new orders or constraint conflicts arrive, and repeated incremental route evaluation dominates the online computation cost. This paper presents SRI-ECR-balanced, an online re-optimization framework that transfers the speculative-execution control pattern of modern AI inference systems—draft, verify, rollback, cache—to dynamic route repair. A balanced draft generator speculatively extends routes several customers at a time; a deterministic verifier accepts the longest feasible prefix; invalid suffixes are rolled back; conflict-specific repair experts restore rejected customers; and a route-prefix cache reuses cumulative distance, arrival-time, load, and slack states across decision epochs, in analogy to KV-cache reuse in language-model serving. On dynamic Solomon conflict-stress instances the framework maintains a feasible rate of 1.0 and a repair success rate of 1.0, while route-prefix caching delivers a $3.12\times$ incremental route-evaluation speedup ($3.08\times$ on a Homberger 200-customer slice) and adaptive early stopping cuts mean response time from 11.18 to 4.39 seconds—a $2.5\times$ latency reduction at unchanged feasibility. Under the same rolling-horizon event stream, SRI-ECR-balanced trades a bounded planned-distance gap of approximately 7.8% against OR-Tools and PyVRP for consistently faster online response and explicit repair diagnostics, a trade-off aligned with dispatch workloads in which timely feasible recovery outweighs marginal route improvement. The Homberger-200 mixed-stress slice serves all 200 customers with no rejected orders.

Keywords: speculative execution; dynamic vehicle routing; online re-optimization; route-prefix caching; latency-bounded computation; feasibility recovery

1 Introduction

Last-mile delivery systems increasingly operate in dynamic environments where orders arrive after dispatch, customer time windows are tight, and vehicle states evolve during execution. These conditions make the dynamic vehicle routing problem with time windows (DVRPTW) substantially different from a one-shot static routing problem. A static solver can spend substantial time improving a complete route plan before execution, whereas a dynamic dispatch system must repeatedly update partially committed routes while preserving feasibility and limiting unnecessary changes.

Classical VRPTW methods, including insertion heuristics, local search, large-neighborhood search, and highly optimized hybrid genetic search solvers, provide strong baselines for static

instances. However, dynamic re-optimization introduces additional requirements that are not captured by final distance alone. When a new order arrives or a conflict is detected, the system must answer several operational questions: can the current plan be repaired quickly, does the repair preserve capacity and time-window feasibility, how much of the previous route can be reused, and how much computation is justified under the current conflict severity? These questions motivate an evaluation focus on feasibility recovery, response time, route stability, and incremental evaluation efficiency, rather than only on static best-known-solution gaps. From a computing-systems perspective, each re-optimization epoch is a latency-bounded workload whose inner loop is dominated by repeated incremental route evaluation: every candidate insertion re-propagates cumulative distance, arrival time, load, and time-window slack along a route. Reducing this repeated computation, and stopping search once feasibility has been restored, are therefore first-class design goals rather than implementation details.

This paper proposes **SRI-ECR-balanced**, a speculative route inference framework for DVRPTW. SRI-ECR does not claim that routing is language modeling; rather, it transfers the control pattern of draft–verify–rollback–cache from modern inference systems to dynamic route re-optimization. In large-model inference, speculative decoding first drafts candidate tokens, verifies them, accepts correct prefixes, discards invalid suffixes, and reuses cached computation. SRI-ECR transfers this pattern to dynamic routing. A draft route generator proposes a short sequence of customers, a deterministic verifier checks feasibility and accepts the longest feasible prefix, invalid suffixes are rolled back, an expert-guided repair module restores rejected customers according to the detected conflict type, and a route-prefix cache reuses cumulative route evaluation states.

The final method uses a *balanced* draft strategy as its default setting. Aggressive drafting is useful for stress testing because it deliberately exposes the verifier and rollback modules to high-risk candidates, but it creates excessive failed speculation. Balanced drafting keeps multi-customer speculation while penalizing time-window and capacity risk, reducing unnecessary rollback and improving accepted prefix length. The method also uses an adaptive early-stop budget: repair receives enough effort to restore feasibility, but search stops once feasibility has been recovered and additional improvement is marginal. This design is better aligned with online routing, where timely feasible recovery is often more valuable than exhaustive local improvement.

SRI-ECR is not designed to replace highly optimized static VRP solvers in a one-shot optimality competition. Its target is dynamic conflict-stress re-optimization, where the primary challenge is to maintain feasible and responsive route plans under changing information. Accordingly, our experiments evaluate the framework along five dimensions: solution quality, dynamic response, constraint feasibility, computational efficiency, and module contribution.

The main contributions are:

- We formulate an inference-systems-inspired online routing framework, SRI-ECR-balanced, that organizes each re-optimization epoch as draft generation, deterministic verification, prefix acceptance, rollback repair, expert-guided refinement, and cache update—the routing analogue of speculative decoding with cache reuse in large-model serving.
- We design a route-prefix cache that stores cumulative distance, arrival-time, load, slack, and violation states and reuses them across verification and repair, together with an adaptive early-stop budget controller. On D-Solomon stress scenarios the cache yields a $3.12\times$ incremental route-evaluation speedup ($3.08\times$ on the Homberger-200 slice), and early stopping reduces

mean response time from 11.18 to 4.39 seconds—a $2.5\times$ latency reduction—at unchanged feasibility and repair success.

- We introduce a balanced speculative draft strategy with a feasible-prefix verifier and rollback mechanism. This design accepts the longest feasible route prefix while preventing invalid suffixes from contaminating later construction, raising prefix acceptance from 0.4400 to 0.6622 relative to aggressive speculation.
- We design an expert router that selects conflict-specific repair behavior for capacity, time-window, and mixed conflicts. Ablations show this conflict-conditioned dispatch is load-bearing: removing it collapses the feasible rate from 1.0 to 0.0 under stress.
- We provide a conflict-stress evaluation on dynamic Solomon instances, a time-budget comparison against OR-Tools and PyVRP, a replicated Solomon C/R/RC diagnostic, and a Homberger-200 scalability slice that serves all 200 customers with no rejected orders. The solver diagnostics are reported as a distance-response trade-off rather than a distance-dominance claim.

Table 1 summarizes the mechanism-level distinction between SRI-ECR-balanced and representative routing methods. The checkmarks indicate explicit online control components, not merely whether a method can satisfy constraints internally. The table compares explicit online re-optimization control mechanisms, not internal feasibility propagation or low-level solver engineering.

Table 1: Mechanism-level comparison with representative routing approaches.

Method	Draft	Prefix verify	Rollback repair	Expert routing	Prefix cache	Early stop
Regret insertion	–	–	–	–	–	–
ALNS/LNS	–	–	Partial	Operator score	–	Optional
OR-Tools	–	–	–	–	–	Time limit
PyVRP/HGS	–	–	–	–	–	Time limit
Rolling-horizon re-opt.	–	–	–	–	Partial	Optional
Speculative SRI-ECR	✓	✓	✓	✓	✓	✓

Overall, the paper positions SRI-ECR-balanced as a dynamic re-optimization framework: static solution quality should remain competitive enough to be meaningful, but the central evidence is whether the framework can recover feasibility quickly, control speculative risk, select effective repairs, and reuse computation under dynamic conflict stress.

2 Related Work

2.1 Vehicle Routing Problems with Time Windows

The vehicle routing problem with time windows (VRPTW) extends the classical vehicle routing problem [1] by requiring each customer to be visited within a prescribed service interval. This temporal constraint makes route construction substantially more difficult because the feasibility of visiting a customer depends not only on spatial distance and vehicle load, but also on arrival time, waiting time, service duration, and downstream time-window slack. Solomon’s seminal

benchmark and heuristic study established VRPTW as a standard testbed and showed that insertion-based heuristics can perform well across different spatial distributions and time-window tightness levels [2]. Exact and decomposition-based approaches, including early column-generation and optimization algorithms, further clarified the structure of VRPTW feasibility and pricing [3]. Survey work has also shown that both route construction/local search and metaheuristic design are essential for solving VRPTW variants at practical scale [4, 5].

Most VRPTW methods evaluate candidate routes through deterministic propagation of cumulative distance, arrival time, waiting time, load, and time-window feasibility. This deterministic structure is important for the present work: SRI-ECR does not replace feasibility checking with a black-box predictor. Instead, it preserves the established route-evaluation logic and places it inside a dynamic inference-style loop. The difference is that the verifier is used to accept the longest feasible prefix of a speculative route fragment, rather than merely checking a completed route after construction.

2.2 Dynamic Vehicle Routing and Rolling-Horizon Re-optimization

Dynamic vehicle routing problems arise when some information is revealed after the initial route plan is made. New customers may appear during execution, travel times may change, and vehicle states may become partially committed. Foundational stochastic dynamic routing models already showed that routing quality depends on both present routing decisions and future request uncertainty [6]. Pillac et al. classify dynamic routing problems from the perspective of information evolution and emphasize the degree of dynamism as a key factor in problem difficulty [7]. Later surveys further distinguish dynamic, stochastic, and prescriptive routing settings, highlighting the need to combine online decision making with anticipatory or robust optimization logic [8, 9]. In practical systems, dynamic routing is commonly handled through rolling-horizon re-optimization: the dispatcher periodically updates routes using newly released orders and current vehicle states.

Rolling-horizon methods provide a natural operational framework, but they also create several difficulties. First, re-optimization must be fast enough for online decision making. Second, new routes should not unnecessarily disrupt previously planned routes. Third, partial route commitments mean that an infeasible insertion can propagate conflicts to later customers. Public last-mile datasets, such as the Amazon Last Mile Routing Research Challenge data and the LaDe industrial delivery dataset, further show that real delivery systems contain route-level, stop-level, package-level, and courier-event information that goes beyond clean benchmark coordinates [10, 11]. Existing dynamic routing approaches often focus on when to trigger re-optimization, how to insert new requests, or how to adapt metaheuristics to changing information. SRI-ECR addresses a complementary question: how should each re-optimization step be structured so that candidate route fragments can be generated, verified, rolled back, repaired, and evaluated incrementally?

2.3 Metaheuristic and Learning-based Routing Solvers

Strong metaheuristics remain central to modern vehicle routing. Adaptive large neighborhood search (ALNS) provides a general destroy-and-repair framework that can solve several VRP variants by repeatedly removing and reinserting customers with adaptive operator selection [12]. Related ALNS work for pickup-and-delivery with time windows illustrates how operator choice and repair design can strongly affect feasibility recovery in time-constrained routing [13]. Hybrid genetic search

methods with diversity management have also achieved strong performance on time-constrained vehicle routing problems [14], and recent open-source HGS implementations and neighborhoods such as SWAP* have strengthened reproducible benchmarking for routing solvers [15]. Engineering solvers such as OR-Tools and PyVRP further provide practical implementations for constrained vehicle routing; PyVRP, in particular, supports rich VRP features including time windows and release times, and is built around high-performance hybrid search ideas [16–18].

Learning-based routing solvers have explored whether neural policies can learn constructive heuristics for routing problems. General methodological surveys emphasize that machine learning can support combinatorial optimization through construction, improvement, prediction, and search-control components [19]. Reinforcement learning approaches formulate route construction as sequential decision making [20, 21], while attention-based models learn flexible heuristics across routing variants such as TSP, VRP, orienteering, and prize-collecting TSP [22]. POMO improves neural combinatorial optimization by exploiting multiple optima and solution symmetries [23]. Neural large neighborhood search integrates learned repair policies into LNS-style search, showing that learned components can be useful when embedded inside a broader metaheuristic structure [24]. More recent work such as RouteFinder moves toward foundation models for many VRP variants by representing variants as combinations of attributes [25]. These methods are promising because they reduce manual heuristic design and can amortize experience across instances. However, learned routing policies still face challenges in hard constraint satisfaction, out-of-distribution dynamic settings, scalability, and interpretability. SRI-ECR therefore uses learning-inspired system design rather than relying on a fully neural route generator: the draft can be heuristic, the verifier remains deterministic, and repair decisions are made by conflict-aware experts.

2.4 AI-Inference-Inspired Optimization Mechanisms

Recent AI inference systems have introduced mechanisms that are not optimization algorithms in the classical operations-research sense, but that offer useful design patterns for dynamic decision systems. Speculative decoding accelerates autoregressive inference by using a lightweight draft model to propose several future tokens, then verifying them with a target model and accepting the valid prefix [26]. KV-cache and paged-cache mechanisms reuse previous attention states to avoid recomputing the whole context during generation; PagedAttention and vLLM show how cache management can substantially improve serving efficiency for large language models [27]. Mixture-of-experts and routing mechanisms further suggest that different inputs or conflicts can be dispatched to specialized components rather than handled by a single universal procedure. Recent LLM-based program-search and automatic heuristic-design studies also demonstrate a broader trend: language-model or inference-inspired systems can be combined with deterministic evaluators to search over algorithmic objects or heuristics [28, 29].

SRI-ECR transfers these ideas at the level of mechanism rather than model architecture. Customers are analogous to tokens in a generated route sequence; speculative route fragments are analogous to draft tokens; the deterministic feasibility-cost verifier plays the role of a validator; accepted route prefixes are retained while invalid suffixes are rolled back; and route-prefix states play a role similar to cached inference states. This analogy is not meant to claim that routing is language modeling. Rather, it provides a structured way to design online route re-optimization around generation, verification, recovery, and reuse.

2.5 Research Gap

The existing literature provides strong foundations for static VRPTW solving, dynamic rolling-horizon re-optimization, metaheuristic repair, and neural route construction. However, these streams leave a gap for dynamic conflict-stress routing. Existing VRP methods mainly focus on static solution quality or general dynamic re-optimization, but few studies formulate dynamic routing as an inference-like process with speculative generation, deterministic verification, rollback recovery, expert-guided repair, and route-prefix caching. In particular, computational reuse across decision epochs—the routing analogue of cache reuse in inference serving—has received little systematic treatment, even though rolling-horizon systems repeatedly re-evaluate largely unchanged route prefixes at every epoch.

This gap matters because dynamic delivery systems must repeatedly react to new orders and constraint conflicts under tight response-time requirements. A method that only optimizes final route distance may still be unsuitable if it frequently produces infeasible intermediate plans, disrupts existing routes, or recomputes unchanged route prefixes from scratch. SRI-ECR addresses this gap by integrating five mechanisms into one re-optimization loop: balanced speculative route drafting, feasible-prefix verification, rollback repair, conflict-aware expert routing, and route-prefix caching. The resulting framework is evaluated not only by distance, but also by feasibility, repair success, response time, rollback behavior, prefix acceptance, and cache speedup.

3 Problem Formulation

This section defines the dynamic vehicle routing problem with time windows (DVRPTW) considered in this paper. The formulation is designed to support both classical route-cost evaluation and the dynamic re-optimization metrics used by SRI-ECR.

3.1 Sets and Indices

Let $G = (V, A)$ be a complete directed graph. The node set is $V = \{0\} \cup N$, where node 0 denotes the depot and $N = \{1, \dots, n\}$ denotes the customer set. Let $K = \{1, \dots, m\}$ be the set of available vehicles. For customers $i, j \in V$, arc $(i, j) \in A$ has travel distance d_{ij} and travel time τ_{ij} . In Euclidean benchmark instances, d_{ij} and τ_{ij} may be identical up to scaling.

Time is discretized by rolling decision epochs $\mathcal{T} = \{t_0, t_1, \dots, t_H\}$. At each epoch $t \in \mathcal{T}$, only customers whose release time has passed are available for dispatch. We denote the set of released but not yet served customers by

$$N_t = \{i \in N : r_i \leq t, i \text{ has not been served}\}. \quad (1)$$

3.2 Parameters

Each customer $i \in N$ has demand q_i , service time s_i , time window $[e_i, l_i]$, and release time r_i . Each vehicle $k \in K$ has capacity Q_k , maximum route duration H_k , current location $p_k(t)$, current load $L_k(t)$, and current time $a_k(t)$ at decision epoch t . The depot has time window $[e_0, l_0]$ and zero demand.

For dynamic re-optimization, let R^{t-} denote the route plan immediately before epoch t , and let R^t denote the updated route plan after re-optimization. The route-change penalty compares edges or customer positions between R^{t-} and R^t to discourage unnecessary disruption.

3.3 Decision Variables

For a static route plan at epoch t , define binary routing variables

$$x_{ijk}^t = \begin{cases} 1, & \text{if vehicle } k \text{ travels directly from } i \text{ to } j \text{ in } R^t, \\ 0, & \text{otherwise,} \end{cases} \quad (2)$$

for all $i, j \in V, i \neq j$, and $k \in K$.

Let $y_{ik}^t \in \{0, 1\}$ indicate whether customer i is served by vehicle k at epoch t , and let $z_i^t \in \{0, 1\}$ indicate whether customer i is rejected or deferred at epoch t . Let T_i^t be the service start time at customer i , and let U_{ik}^t be the cumulative load of vehicle k after visiting customer i .

3.4 Objective Function

The dynamic objective combines distance, time-window violation, vehicle usage, rejection, and route-change cost:

$$\begin{aligned} \min C(R^t) = & \alpha C_{\text{dist}}(R^t) + \beta C_{\text{late}}(R^t) + \gamma C_{\text{veh}}(R^t) \\ & + \delta C_{\text{reject}}(R^t) + \eta C_{\text{change}}(R^t, R^{t-}). \end{aligned} \quad (3)$$

The terms are defined as

$$C_{\text{dist}}(R^t) = \sum_{k \in K} \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} d_{ij} x_{ijk}^t, \quad (4)$$

$$C_{\text{late}}(R^t) = \sum_{i \in N_t} \max(0, T_i^t - l_i), \quad (5)$$

$$C_{\text{veh}}(R^t) = \sum_{k \in K} \mathbb{I} \left[\sum_{j \in N_t} x_{0jk}^t > 0 \right], \quad (6)$$

$$C_{\text{reject}}(R^t) = \sum_{i \in N_t} z_i^t. \quad (7)$$

The route-change cost can be measured by changed edges:

$$C_{\text{change}}(R^t, R^{t-}) = |E(R^{t-}) \setminus E(R^t)|, \quad (8)$$

where $E(R)$ is the set of directed customer-to-customer and depot-to-customer edges in route plan R . Other stability measures, such as position changes or customer reassignment, can be substituted without changing the SRI-ECR framework.

3.5 Routing Constraints

Each released customer is either served once or rejected/deferred:

$$\sum_{k \in K} y_{ik}^t + z_i^t = 1, \quad \forall i \in N_t. \quad (9)$$

Customer assignment is linked to route arcs:

$$\sum_{\substack{j \in V \\ j \neq i}} x_{ijk}^t = y_{ik}^t, \quad \sum_{\substack{j \in V \\ j \neq i}} x_{jik}^t = y_{ik}^t, \quad \forall i \in N_t, k \in K. \quad (10)$$

Each used vehicle leaves and returns to the depot at most once:

$$\sum_{j \in N_t} x_{0jk}^t \leq 1, \quad \sum_{i \in N_t} x_{i0k}^t \leq 1, \quad \forall k \in K. \quad (11)$$

Flow conservation holds for every vehicle and customer:

$$\sum_{\substack{j \in V \\ j \neq i}} x_{ijk}^t - \sum_{\substack{j \in V \\ j \neq i}} x_{jik}^t = 0, \quad \forall i \in N_t, k \in K. \quad (12)$$

3.6 Capacity and Time-Window Constraints

For each vehicle, cumulative load must not exceed capacity:

$$0 \leq U_{ik}^t \leq Q_k, \quad \forall i \in N_t, k \in K. \quad (13)$$

Load propagation can be written with a large constant M :

$$U_{jk}^t \geq U_{ik}^t + q_j - M(1 - x_{ijk}^t), \quad \forall i, j \in N_t, i \neq j, k \in K. \quad (14)$$

Service start times obey travel and service propagation:

$$T_j^t \geq T_i^t + s_i + \tau_{ij} - M(1 - x_{ijk}^t), \quad \forall i, j \in V, i \neq j, k \in K. \quad (15)$$

Hard time windows require

$$e_i \leq T_i^t \leq l_i, \quad \forall i \in N_t \text{ with } z_i^t = 0. \quad (16)$$

When soft time windows are used in stress tests, violations are allowed but penalized through C_{late} and reported separately as time-window violation metrics.

Each vehicle must also satisfy a route-duration bound:

$$T_{0,k}^{\text{return},t} - a_k(t) \leq H_k, \quad \forall k \in K. \quad (17)$$

3.7 Dynamic Order Release and Rolling-Horizon Re-optimization

The release-time constraint ensures that an order cannot be scheduled before it becomes known:

$$y_{ik}^t = 0, \quad \forall i \in N, r_i > t, k \in K. \quad (18)$$

At each decision epoch t , the solver receives $(N_t, R^{t-}, p_k(t), L_k(t), a_k(t))$ and returns an updated route plan R^t . Routes already executed before t are fixed; only the unserved suffixes and newly released customers are re-optimized. This rolling-horizon setting makes response time and route stability part of the evaluation, even when the mathematical objective in Eq. (3) is evaluated at a single epoch.

The SRI-ECR algorithm operates on this formulation by constructing route fragments rather than solving the full mixed-integer model directly. The deterministic verifier evaluates whether a candidate fragment satisfies Eqs. (13)–(18), accepts the longest feasible prefix, and passes any rejected suffix to rollback and repair. Thus, the formulation defines both the feasibility semantics and the metrics used in the experimental evaluation.

4 Methodology

This section presents SRI-ECR-balanced, the fixed configuration of the SRI-ECR speculative route inference framework for dynamic vehicle routing with time windows (DVRPTW). The central design goal is not to replace high-performance static VRPTW solvers on final distance, but to support fast, feasible, and stable route re-optimization when new orders arrive and existing routes must be updated under time-window and capacity constraints. SRI-ECR-balanced views dynamic route construction as an inference-like process: a draft generator proposes a short route continuation, a deterministic verifier accepts the longest feasible prefix, invalid suffixes are rolled back, conflict-aware experts repair the rejected customers, and a route-prefix cache reuses previous evaluation states.

4.1 Problem Setting

Let $G = (V, E)$ be a complete routing graph with depot 0 and customer set $N = V \setminus \{0\}$. Each customer $i \in N$ is associated with location (x_i, y_i) , demand q_i , service time s_i , time window $[e_i, l_i]$, and release time r_i . A fleet of at most K vehicles with capacity Q starts from and returns to the depot. In the dynamic setting, only customers with $r_i \leq t$ are available at decision time t . The system periodically re-optimizes routes over a rolling horizon while preserving feasibility and limiting unnecessary changes to previously planned routes.

Given a route $\pi = (i_1, \dots, i_m)$, route evaluation computes cumulative travel time, service start time, waiting time, load, time-window violation, route-duration violation, and return-to-depot time. A route is feasible if it satisfies capacity, customer time windows, and depot return constraints. A solution is feasible if all released customers are served exactly once, no route violates constraints, and the number of used vehicles does not exceed K .

The optimization objective combines routing cost and dynamic penalties:

$$\min C = \alpha C_{\text{dist}} + \beta C_{\text{late}} + \gamma C_{\text{vehicle}} + \delta C_{\text{reject}} + \eta C_{\text{change}}, \quad (19)$$

where C_{dist} is total distance, C_{late} is total lateness, C_{vehicle} penalizes vehicle usage, C_{reject} penalizes unserved or rejected orders, and C_{change} measures route instability relative to the previous plan.

4.2 Overview of SRI-ECR

SRI-ECR-balanced consists of five interacting modules:

1. **Balanced speculative draft generation:** proposes short customer sequences with controlled risk.
2. **Feasible-prefix verification:** deterministically accepts the longest feasible prefix of each draft.
3. **Prefix rollback and repair:** rolls back invalid suffixes and reinserts them into feasible positions.
4. **Expert-guided repair:** routes capacity, time-window, and batch insertion conflicts to specialized repair policies.
5. **Route-prefix cache:** reuses incremental route evaluation states to reduce repeated computation.

Unless otherwise specified, the main method uses the balanced draft strategy with adaptive early stop. Aggressive draft generation is used only in stress tests to deliberately expose the verifier, rollback, and repair mechanisms to higher-risk candidate routes.

4.3 Balanced Speculative Draft Generation

The draft generator proposes short route fragments of length k . In conservative construction, draft routes are produced by a feasibility-aware insertion heuristic. In aggressive construction, candidates are chosen primarily by proximity and may violate capacity or time-window constraints, which is useful for stress testing but too risky as a default policy.

The main method therefore uses a balanced draft score that trades off distance and constraint risk:

$$\text{score}(i \mid \pi) = d(\text{last}(\pi), i) + \lambda_{\text{tw}} \max(0, a_i - l_i) + \lambda_{\text{cap}} \max(0, L_\pi + q_i - Q), \quad (20)$$

where a_i is the estimated service start time after appending i , L_π is the load of the partial route, and $\lambda_{\text{tw}}, \lambda_{\text{cap}}$ control time-window and capacity risk. The balanced mode keeps the speculative behavior needed for multi-customer drafts, but avoids the excessive rollback caused by purely aggressive drafting.

4.4 Feasible-Prefix Verification

Given a draft route $\hat{\pi} = (i_1, \dots, i_k)$, the verifier evaluates prefixes (i_1) , (i_1, i_2) , $\dots$, $(i_1, \dots, i_k)$ and returns the longest feasible prefix:

$$m^* = \max\{m \leq k : (i_1, \dots, i_m) \text{ is feasible}\}. \quad (21)$$

The prefix $(i_1, \dots, i_{m^*})$ is accepted and cached. The suffix $(i_{m^*+1}, \dots, i_k)$ is rejected and passed to the rollback and repair module.

The verifier records both final feasibility and interception statistics, including prevented capacity violations, prevented time-window violations, rejected suffix length, failed speculation rate, and accepted prefix length. These metrics are important because the verifier’s value is not only that the final solution is feasible, but also that it prevents invalid draft suffixes from contaminating subsequent route construction.

4.5 Prefix Rollback and Repair

SRI-ECR uses prefix rollback rather than all-or-nothing rejection. If the first m^* draft customers are feasible, they remain in the route; only the invalid suffix is rolled back. This mirrors speculative decoding in AI inference, where correct prefixes are accepted and incorrect suffixes are discarded.

Rolled-back customers are reinserted by a repair procedure. For each pending customer, the repair module searches feasible insertion positions across existing routes and, if necessary, opens a new route when vehicle capacity allows. The repair process is evaluated by repair success rate, repair rejected orders, repair time, and the number of insertion evaluations.

4.6 Expert-Guided Repair

The expert router is implemented as a three-stage module: conflict diagnosis, expert policy selection, and repair action generation. Table 2 summarizes the rule set used in the experiments. The router is intentionally rule-based rather than learned, because the goal of the current paper is to evaluate the inference-style control architecture under reproducible feasibility semantics. Capacity and duration violations are detected directly from route states, while time-window risk is detected by positive lateness or negative slack. The batch threshold is 12 pending customers. In the repair-budget notation, the first value is the maximum number of routes scanned and the second value is the maximum number of insertion positions scanned per route; ‘unbounded’ means that no such cap is applied within the wall-clock limit. If multiple signals are active, the router follows the priority order in the table.

Table 2: Conflict diagnosis, expert policy selection, and repair action generation in SRI-ECR-balanced.

Priority	Conflict signal	Identification rule and threshold	Repair action generation	Budget/fallback
1	Capacity overflow	Route load plus candidate demand exceeds Q	Rank routes by residual capacity; insert high-demand customers first	Increase insertion budget; fallback to new route
2	Time-window risk	Arrival exceeds l_i or slack is negative	Prioritize early due dates and minimum lateness insertion	Increase route-position scans; fallback to regret repair
3	Route-duration violation	Predicted depot return exceeds maximum route duration	Shorten overloaded route by relocation or route split	Open new route if fleet capacity remains
4	Many pending customers	Pending set has at least 12 customers	Batch regret insertion with nearest-route filtering	Use batch budget cap; fallback to one-by-one repair
5	Multi-route conflict	Source and target routes are both affected by rollback or relocation	Relocate/swap repair between affected routes	Limit to affected routes; fallback to global insertion
6	No dominant violation	No higher-priority signal is active	Standard regret-style insertion	Default budget; reject only if no feasible insertion exists

The expert decision changes the pending-customer ordering, route ordering, insertion-position priority, search budget, and fallback behavior. For example, capacity repair prioritizes high-demand customers and routes with larger residual capacity, while time-window repair prioritizes earlier due

times and scans more insertion positions before falling back to regret repair. Under the adaptive early-stop strategy, insertion search stops after the first feasible repair for the current customer once feasibility has been restored, whereas fixed high budget continues scanning all available route-position candidates. This makes the router a functional diagnosis-policy-action component rather than a label attached to a common repair operator.

4.7 Route-Prefix Cache

Dynamic re-optimization repeatedly evaluates partially unchanged route prefixes. SRI-ECR caches route-prefix states, including cumulative distance, arrival time, load, slack, and violation status. When a route is extended or locally modified, cached prefixes are reused instead of recomputing the entire route from scratch.

For each route, the cache is indexed by route identifier and prefix length. When an insertion, deletion, or swap modifies a route at position p , all cached states before p remain valid, while cached states from p onward are invalidated and recomputed incrementally. If a customer is moved between routes, only the affected suffixes of the source and target routes are invalidated. Unchanged routes keep their full prefix cache. The current implementation uses a single-level route-prefix cache; multi-level route-block caching is left for future work.

The cache module reports cache hit rate, cache lookups, incremental evaluation time, estimated full recomputation time, and speedup ratio:

$$\text{Speedup} = \frac{T_{\text{full recomputation}}}{T_{\text{incremental cache}}}. \quad (22)$$

4.8 Adaptive Early-Stop Budget

The original adaptive budget allocated generous repair effort under conflict stress, which preserved feasibility but was slower than a fixed high budget. The final SRI-ECR-balanced configuration therefore uses adaptive early stop: the search begins with enough repair effort to recover feasibility, but stops once feasibility is restored and a feasible insertion is found for the current pending customer. This separates feasibility recovery from optional distance improvement and makes the dynamic response more suitable for online routing.

In experiments, adaptive early stop is compared with fixed medium and fixed high budgets, as well as earlier adaptive variants. The final default is SRI-ECR-balanced with adaptive early stop.

5 Experiments

The experiments evaluate SRI-ECR-balanced from five perspectives: solution quality, dynamic response, constraint feasibility, computational efficiency, and module contribution. Since the proposed method targets dynamic re-optimization rather than static optimality alone, the main evaluation focuses on whether SRI-ECR-balanced can quickly restore feasible routes under dynamic and conflict-intensive scenarios.

5.1 Datasets and Dynamic Instances

We use Solomon 100-customer VRPTW instances and Homberger large-scale VRPTW instances as the base benchmarks. Dynamic instances are generated by assigning release times to customers and evaluating rolling re-optimization under different dynamic levels. Conflict stress variants are generated by tightening time windows, increasing capacity pressure, and combining both sources of difficulty.

The main stress scenarios are:

- **Normal:** original dynamic setting without additional pressure;
- **TW-tight:** time windows are compressed to increase temporal conflicts;
- **CAP-tight:** capacity is reduced and demands are increased;
- **Mixed-stress:** both time-window and capacity pressure are applied.

For scalability evidence, we include a Homberger-200 mixed-stress slice with balanced draft length $k = 5$ and a 60-second wall-clock budget.

5.2 Baselines and Variants

The comparison includes constructive heuristics, engineering solvers, and SRI-ECR variants:

- **Nearest Neighbor, Savings, Regret Insertion:** classical constructive baselines;
- **OR-Tools and PyVRP/HGS:** engineering and strong heuristic baselines for VRPTW;
- **w/o verifier, w/o rollback, w/o cache:** hard ablations for core modules;
- **w/o expert router and random repair:** ablations for conflict-aware repair;
- **fixed medium/high budgets:** budget baselines;
- **adaptive variants:** current adaptive, early stop, budget cap, and feasibility-then-improvement variants.

AM/POMO-style neural construction methods are discussed in the related-work section but are not used as primary empirical baselines. The present evaluation includes order release times, rolling-horizon re-optimization, conflict-stress feasibility recovery, rollback, repair, and cache reuse. Existing AM/POMO implementations would require substantial adaptation to support these dynamic and constraint-recovery semantics. We therefore focus the empirical comparison on constructive heuristics, engineering solvers, and module ablations that can be driven by the same dynamic event stream.

For OR-Tools and PyVRP, fairness is enforced at the dynamic-event level. All methods receive the same released-order stream, rolling period, depot, vehicle capacity, time-window constraints, and distance matrix. At each decision epoch, OR-Tools and PyVRP solve the currently active released subproblem from scratch under the same per-epoch wall-clock budget reported in the tables; they

do not receive SRI-ECR’s route-prefix cache, expert router, or rollback state. SRI-ECR-balanced inherits its previous route state and cache because reuse is part of the proposed online mechanism. Served, rejected, feasible-rate, distance, and response-time metrics are computed by the same evaluation code for all methods. If a method fails to place a released order within the tested budget, the order is counted as rejected or unserved under the same rule.

Unless otherwise specified, the final method is SRI-ECR-balanced with adaptive early stop. Aggressive draft is used only for stress testing and draft-mode analysis.

5.3 Metrics

We report standard VRPTW metrics and dynamic-routing-specific metrics:

- **Solution quality:** number of vehicles, cumulative planned distance, objective value;
- **Feasibility:** feasible rate, capacity violations, time-window violations, rejected orders;
- **Dynamic response:** response time, route stability, route change cost;
- **Speculation:** rollback rate, accepted prefix length, prefix acceptance rate;
- **Repair:** repair success rate, repair evaluations, repair time;
- **Cache:** cache hit rate and cache speedup ratio.

In the dynamic tables, distance denotes cumulative planned route distance over rolling decision epochs under the common evaluator, not observed vehicle mileage from a deployed fleet. Feas. denotes final feasible rate; Repair denotes repair success rate; Resp. (s) denotes average response time in seconds; Eval. denotes the number of route insertion or evaluation attempts during repair; and Speedup denotes incremental route-evaluation speedup from route-prefix caching. Speedup is not an end-to-end runtime speedup.

5.4 Supplementary Statistical Diagnostics

For compactness, the main result tables report mean values over the corresponding scenarios or slices. Table 3 reports mean $\pm$ standard deviation for the original compact scenario-slice comparisons and paired diagnostic tests for the main distance, response-time, repair, and budget differences. These small-slice diagnostics have limited statistical power and are not used as the primary basis for the paper’s claims. To address this limitation, Section 5.12 adds a larger replicated C/R/RC Solomon diagnostic with 60 matched dynamic runs. Single-slice results, such as the Amazon real-stop slice and the Homberger-200 scalability slice, are not treated as significance tests.

5.5 Parameter Settings

Table 4 reports the fixed SRI-ECR-balanced parameters used in the main experiments. These settings are kept fixed across the reported D-Solomon stress tests unless a variant explicitly changes the corresponding component. Draft-length and budget variants are used as sensitivity checks rather than post-hoc tuning.

Table 3: Supplementary statistical diagnostics for key comparisons. Values are mean $\pm$ standard deviation over matched scenario slices.

Comparison	Metric	SRI-ECR / full	Baseline	t p	Wilcoxon p
Ordinary dynamic, $n = 3$	Distance	3980.840 $\pm$ 866.233	OR-Tools: 1247.055	3343.818 $\pm$ 0.1041	0.2500
Ordinary dynamic, $n = 3$	Resp.	0.0362 $\pm$ 0.0517	OR-Tools: 0.0004	3.0026 $\pm$ 0.0001	0.2500
Conflict solver, $n = 3$	Distance	4914.608 $\pm$ 119.448	OR-Tools: 60.588	4557.311 $\pm$ 0.0102	0.2500
Conflict solver, $n = 3$	Resp.	0.0010 $\pm$ 0.0001	OR-Tools: 0.0003	3.0022 $\pm$ < 0.0001	0.2500
Expert ablation, $n = 4$	Repair	1.000 $\pm$ 0.000	w/o expert: 0.103	0.343 $\pm$ 0.0010	0.1250
Budget ablation, $n = 3$	Resp.	early stop: 4.387 $\pm$ 1.137	fixed high: 1.592	11.177 $\pm$ 0.0166	0.2500

Table 4: Main SRI-ECR-balanced parameter settings.

Component	Parameter	Value used in main experiments
Draft generation	draft mode / length / beam width	balanced / $k = 5 / 3$
Draft risk weights	$\lambda_{tw}, \lambda_{cap}$	5.0, 500.0
Draft precheck	precheck constraints	false; verifier handles feasibility
Stress scenarios	TW/CAP/demand factors	TW-tight: 0.5/1.0/1.0; CAP-tight: 1.0/0.8/1.2; mixed: 0.5/0.7/1.3
Adaptive budget	strategy / early stop	early-stop / true
Repair budgets	low / medium / cap / high	(1,4) / (3,12) / (4,16) / unbounded
Large-instance control	max refine customers / 2-opt route size	120 / 40
Local refinement	2-opt passes	1
Expert router	batch threshold / time-risk threshold	12 pending customers / positive lateness or negative slack
Repair budget columns	route limit / position limit	Number of routes scanned / insertion positions scanned per route
Solver-baseline budget	OR-Tools/PyVRP epoch time	3 seconds
Homerger slices	online time budget	60 seconds for 200 customers; 120 seconds for 400 customers

5.6 Overall Conflict-Stress Performance

Table 5 summarizes the main balanced SRI-ECR results on D-Solomon stress scenarios. The complete SRI-ECR-balanced framework achieves a feasible rate of 1.0 and a repair success rate of 1.0 across the four stress settings. Removing the expert router causes feasibility to collapse to 0.0 and reduces repair success to 0.3434, while removing the cache preserves feasibility but removes the measured speedup.

Table 5: Main D-Solomon stress results under balanced draft ($k = 5$).

Variant	Feas.	Repair	Resp. (s)	Eval.	Speedup
SRI-ECR-balanced	1.000	1.000	4.225	754.25	3.124
w/o expert router	0.000	0.343	0.780	116.50	2.311
w/o cache	1.000	1.000	4.973	754.25	1.000
fixed high budget	1.000	1.000	10.826	3278.75	3.204

These results support the main claim that feasibility under conflict stress requires more than a generic repair procedure. The expert router is responsible for selecting conflict-appropriate repair behavior, while the cache reduces repeated route evaluation.

5.7 Balanced Versus Aggressive Draft

Figure 1 compares balanced and aggressive draft generation. Aggressive drafting is useful for stress testing because it creates many invalid suffixes, but it also increases rollback. Balanced drafting preserves feasibility while reducing rollback from 0.8717 to 0.6805 and increasing prefix acceptance from 0.4400 to 0.6622. It also reduces average response time from 17.65 seconds to 10.21 seconds in the draft-mode comparison.

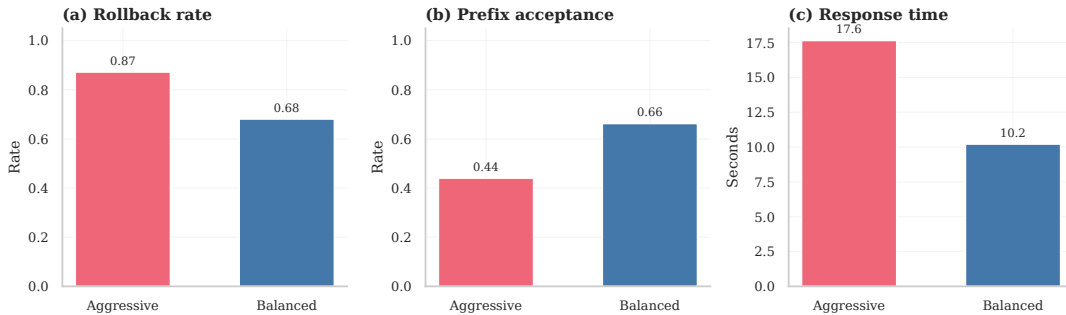


Figure 1: Balanced draft reduces unnecessary rollback and improves prefix acceptance compared with aggressive draft. Panels show (a) rollback rate, (b) prefix acceptance, and (c) response time.

This result motivates using balanced draft as the default SRI-ECR-balanced policy, while keeping aggressive draft as a controlled stress-test setting.

5.8 Expert-Guided Repair

The expert router is evaluated by comparing SRI-ECR-balanced against w/o expert router and random repair. On D-Solomon stress scenarios, full SRI-ECR-balanced achieves a repair success rate of 1.0 and feasible rate of 1.0. In contrast, removing the expert router reduces repair success to 0.3434 and feasible rate to 0.0. In the broader aggressive-stress setting, w/o expert router and random repair both achieve only 0.1994 repair success.

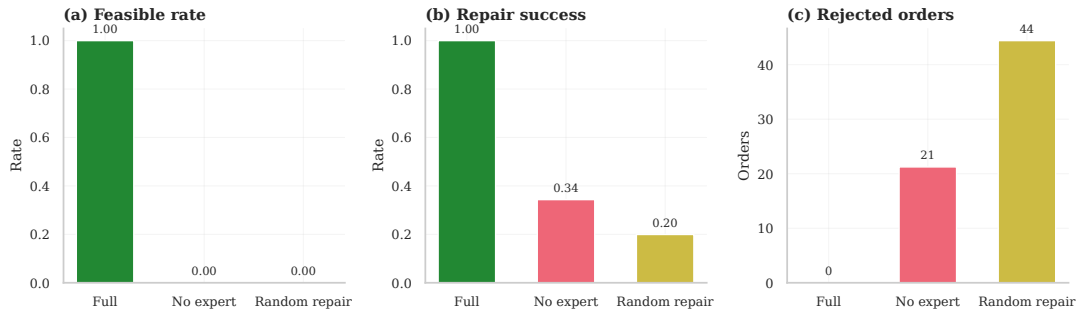


Figure 2: Expert-guided repair is necessary for restoring feasibility under conflict stress. Panels show (a) feasible rate, (b) repair success, and (c) rejected orders.

This confirms that the framework does not merely need a repair operator; it needs conflict-aware routing to choose the appropriate repair behavior for capacity, time-window, and mixed conflicts.

5.9 Adaptive Early-Stop Budget

The adaptive budget study compares fixed medium, fixed high, and several adaptive variants. Fixed medium is fast but fails under conflict stress, with feasible rate 0.0. Fixed high preserves feasibility but requires 11.18 seconds on average. The final adaptive early-stop strategy preserves feasibility and repair success while reducing response time to 4.39 seconds.

Table 6: Adaptive budget comparison under balanced draft ($k = 5$).

Variant	Feas.	Repair	Resp. (s)	Eval.
fixed medium	0.000	0.885	3.268	273.0
fixed high	1.000	1.000	11.177	3417.7
adaptive current	1.000	1.000	10.728	3417.7
adaptive early stop	1.000	1.000	4.387	860.0
budget cap	0.667	0.928	3.998	592.3
feasibility-then-improvement	0.667	0.946	3.447	358.7

The results show that the final adaptive mechanism should be interpreted as a feasibility-preserving budget controller with early stopping, not merely as a larger search budget.

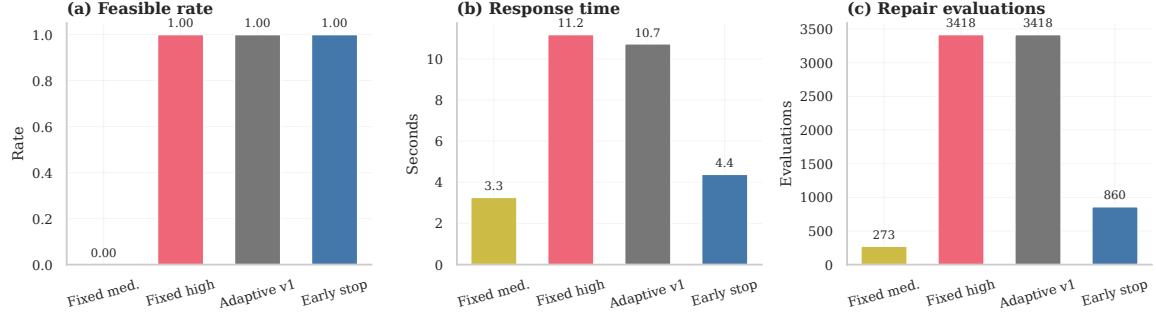


Figure 3: Adaptive early stop keeps feasibility while reducing response time and repair evaluations relative to fixed high budget. Panels show (a) feasible rate, (b) response time, and (c) repair evaluations.

5.10 Route-Prefix Cache

The route-prefix cache consistently improves incremental evaluation. On D-Solomon stress scenarios under balanced draft, SRI-ECR-balanced obtains a cache speedup ratio of 3.12, while the w/o cache variant has speedup 1.0 by construction. On the Homberger-200 mixed-stress slice, the cache speedup ratio remains 3.08 for the full method and 1.0 for w/o cache.

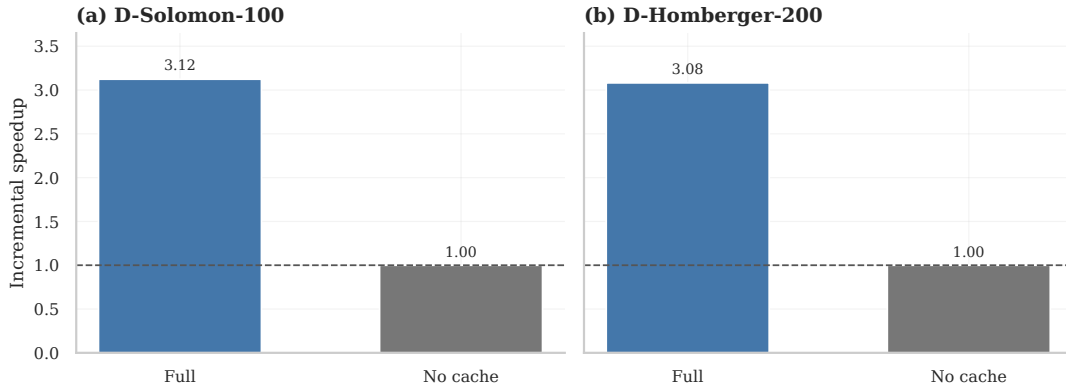


Figure 4: Route-prefix cache provides consistent incremental route-evaluation speedup over full recomputation on D-Solomon-100 and D-Homberger-200.

This supports the claim that repeated prefix evaluation is a major computational bottleneck in dynamic re-optimization and that caching route-prefix states is effective.

5.11 Homberger-200 Mixed-Stress Slice

To test whether the framework scales beyond Solomon-100, we evaluate a Homberger-200 mixed-stress slice with balanced draft, draft length $k = 5$, and a 60-second time budget. The instance provides at most 50 vehicles under the stressed capacity setting; full SRI-ECR-balanced uses 40 of them, serves all 200 customers with no rejected orders, and finishes in 59.61 seconds. Thus the feasible result is not obtained by allowing unlimited vehicles, although the vehicle count remains a cost-quality limitation rather than a dominance claim. Removing the expert router leaves only 135

customers served and 65 rejected orders. Removing the cache still serves all customers but loses the cache speedup and takes 63.06 seconds.

Table 7: Homberger-200 mixed-stress slice under a 60-second budget. The stress instance allows at most 50 vehicles.

Variant	Feas.	Served	Rejected	Resp. (s)	Speedup
SRI-ECR-balanced	1	200	0	59.61	3.08
w/o expert router	0	135	65	5.72	2.49
w/o cache	1	200	0	63.06	1.00
fixed high budget	0	159	41	61.67	2.38

This slice does not claim full large-scale dominance, but it verifies that the main feasibility, expert-repair, and cache effects persist on a 200-customer stress instance.

5.12 Additional Robustness and Boundary Analysis

The preceding experiments establish the main conflict-stress evidence. We further conduct three supplementary experiments to clarify the operating regime of SRI-ECR-balanced rather than to change the main claims.

Dynamic solver baselines. Table 8 compares SRI-ECR-balanced with regret insertion, OR-Tools, and PyVRP on ordinary dynamic D-Solomon settings using the same release-time event stream and a 3-second per-epoch budget for solver-based methods. OR-Tools and PyVRP achieve shorter routes than SRI-ECR-balanced, but their average response times are close to the full 3-second budget. SRI-ECR-balanced is much faster than these solver baselines, but it is not positioned as a replacement for strong solvers in ordinary dynamic distance minimization. Its intended regime remains dynamic conflict-stress re-optimization, where feasibility recovery and online response are central.

Table 8: Additional dynamic solver baselines on D-Solomon c101 low/medium/high settings with a 10-minute rolling period. Distance is cumulative planned route distance over decision epochs.

Method	Feasible rate	Avg. planned dist.	Avg. response (s)
Regret insertion	1.000	3731.630	0.0072
OR-Tools	1.000	3343.818	3.0026
PyVRP	1.000	3343.438	3.0198
SRI-ECR-balanced	1.000	3980.840	0.0362

Conflict-stress solver baselines. Because SRI-ECR-balanced is positioned for conflict-stress re-optimization, we also run OR-Tools and PyVRP under the TW-tight, CAP-tight, and mixed-stress D-Solomon settings using the same 3-second per-epoch solver budget. Table 9 shows the same trade-off as in ordinary dynamic settings: OR-Tools and PyVRP produce shorter routes, while SRI-ECR-balanced responds substantially faster and still serves all orders. The distance gap is material: SRI-ECR-balanced is about 7.8% longer than OR-Tools/PyVRP in this conflict-stress

comparison. This result should not be read as distance dominance over strong solvers. Instead, it clarifies that SRI-ECR-balanced trades route distance for a lightweight online repair process with explicit verifier, rollback, and cache diagnostics.

Table 9: Conflict-stress dynamic solver baselines on D-Solomon c101 high-dynamic settings. Results average TW-tight, CAP-tight, and mixed-stress cases with a 10-minute rolling period. Distance is cumulative planned route distance over decision epochs.

Method	Feas.	Served	Rejected	Avg. planned dist.	Resp. (s)
OR-Tools	1.000	100.0	0.0	4557.311	3.0022
PyVRP	1.000	100.0	0.0	4557.311	3.0182
SRI-ECR-balanced	1.000	100.0	0.0	4914.608	0.0010

Per-epoch time-budget curve. The preceding conflict-stress solver baseline uses a 3-second per-epoch budget. To check whether this choice hides a stronger low-budget solver regime, we additionally run OR-Tools, PyVRP, and SRI-ECR-balanced under per-epoch budgets of 0.01, 0.05, 0.1, 0.5, 1, and 3 seconds across the normal, TW-tight, CAP-tight, and mixed-stress D-Solomon c101 high-dynamic settings. Figure 5 shows that OR-Tools and PyVRP remain route-quality leaders even under short budgets. At 0.01 seconds, PyVRP obtains an average planned distance of 4541.54 with a 0.0106-second response time, while SRI-ECR-balanced obtains 5155.29 with a 0.0007-second response time. At 3 seconds, OR-Tools and PyVRP converge to 4540.43, while SRI-ECR-balanced remains at 4884.89. Thus, the stronger solver baselines are not defeated by shortening the budget; the empirical result is a distance-response trade-off.

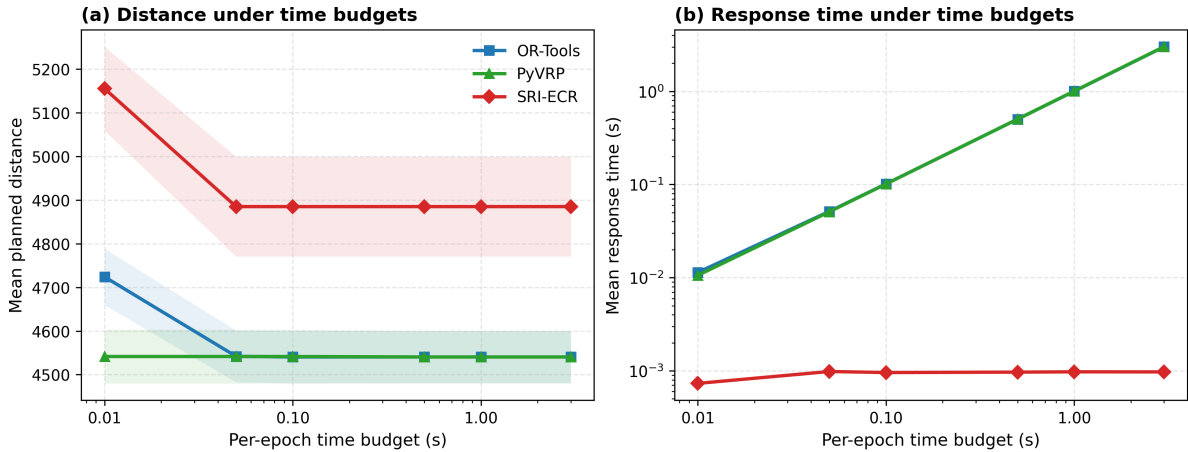


Figure 5: Per-epoch time-budget curve on D-Solomon c101 high-dynamic settings averaged over normal, TW-tight, CAP-tight, and mixed-stress scenarios. Strong solvers remain better in planned distance, while SRI-ECR-balanced stays in a lower response-time regime.

Replicated Solomon C/R/RC diagnostics. To reduce the risk that the solver comparison depends only on c101, we further run a replicated dynamic diagnostic on Solomon C/R/RC instances: c101, c102, r101, r102, rc101, and rc102, with five release seeds per instance and both normal and

mixed-stress settings. This yields 60 matched dynamic runs for each method under a 0.5-second solver budget. Table 10 reports mean $\pm$ standard deviation. All methods serve all customers in this diagnostic, but they occupy different operating regimes. OR-Tools and PyVRP obtain substantially shorter planned distances, regret insertion is the fastest simple baseline, and SRI-ECR-balanced is much faster than OR-Tools/PyVRP but has a higher planned distance.

Table 10: Replicated Solomon C/R/RC dynamic diagnostic over six instances, five release seeds, and two scenarios. Values are mean $\pm$ standard deviation over 30 matched runs per scenario.

Method	Scenario	Feas.	Planned dist.	Resp. (s)	Served
Regret	Normal	1.000	4317.539 $\pm$ 507.194	0.0014 $\pm$ 0.0007	100.0
Regret	Mixed-stress	1.000	4455.050 $\pm$ 582.993	0.0015 $\pm$ 0.0007	100.0
OR-Tools	Normal	1.000	3933.670 $\pm$ 578.593	0.5025 $\pm$ 0.0006	100.0
OR-Tools	Mixed-stress	1.000	4107.994 $\pm$ 576.894	0.5024 $\pm$ 0.0005	100.0
PyVRP	Normal	1.000	3933.440 $\pm$ 575.264	0.5028 $\pm$ 0.0003	100.0
PyVRP	Mixed-stress	1.000	4107.505 $\pm$ 576.372	0.5028 $\pm$ 0.0005	100.0
SRI-ECR-balanced	Normal	1.000	4401.420 $\pm$ 525.106	0.0028 $\pm$ 0.0015	100.0
SRI-ECR-balanced	Mixed-stress	1.000	4511.859 $\pm$ 571.050	0.0030 $\pm$ 0.0016	100.0

Paired tests on the 60 matched runs confirm the direction of this trade-off. Against OR-Tools, SRI-ECR-balanced is longer in planned distance by 435.81 on average ($p < 10^{-6}$ for both paired t and Wilcoxon tests) but faster in response time by 0.4996 seconds on average ($p < 10^{-6}$). Against PyVRP, the corresponding distance penalty is 436.17 and the response-time reduction is 0.4999 seconds. Compared with regret insertion, SRI-ECR-balanced is also slightly longer and slower in this ordinary replicated diagnostic; this reinforces that the contribution is not generic speed or distance dominance, but explicit conflict diagnosis, feasible-prefix verification, rollback repair, expert-guided recovery, and cache reuse under the proposed online-control framework.

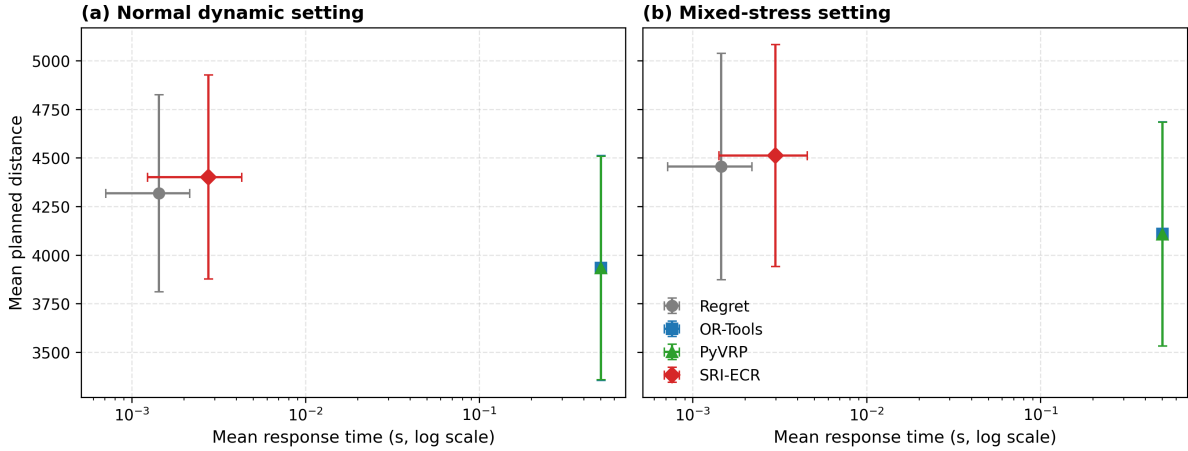


Figure 6: Distance-response trade-off on the replicated Solomon C/R/RC diagnostic. Error bars show standard deviation over 30 matched runs per scenario. Strong solvers produce shorter routes, while SRI-ECR-balanced occupies a lower-response-time regime than OR-Tools/PyVRP.

This trade-off can be expressed by a simple operational cost

$$J_m = D_m + \rho T_m, \quad (23)$$

where D_m is cumulative planned route distance, T_m is average online response time, and ρ is the operational penalty per second of dispatch delay. In the conflict-stress solver comparison, SRI-ECR-balanced has a distance penalty of 357.297 relative to OR-Tools but saves 3.0012 seconds of response time. Therefore, SRI-ECR-balanced has lower expected cost whenever $\rho > 119.051$ distance-cost units per second. This is a break-even sensitivity calculation rather than an empirically calibrated dispatch-cost model. It does not claim that SRI-ECR dominates strong solvers; it quantifies the response-time value that would be required for the faster online repair regime to be preferable.

Robustness under stochastic travel-time perturbation. To reduce the risk that the stress tests are overly deterministic, we perturb the travel-time matrix of the D-Solomon mixed-stress instance by independent uniform multiplicative noise in the range $\pm 20\%$. Table 11 shows that full SRI-ECR-balanced still serves all 100 customers with no rejected orders and repair success rate 1.0. Removing the expert router leaves 20 orders rejected and reduces repair success to 0.3103, indicating that conflict-aware repair remains important when travel times are stochastic.

Table 11: Robustness under $\pm 20\%$ multiplicative travel-time perturbation on D-Solomon mixed stress.

Variant	Feas.	Served	Rejected	Repair	Resp. (s)
SRI-ECR-balanced	1	100	0	1.0000	1.1754
w/o expert router	0	80	20	0.3103	0.1844
w/o cache	1	100	0	1.0000	1.3389

Amazon Last Mile real-stop slice. To check whether the implementation can operate on real last-mile stop locations, we extract the first complete route from the Amazon Last Mile Routing Research Challenge route data [10]. We use only the depot and 80 drop-off coordinates, convert latitude/longitude into local Euclidean coordinates, and create a simple artificial dynamic release process with a 10-minute rolling period. This is a coordinates-only real-stop-location slice, not a full reproduction of Amazon operations: package compatibility, driver behavior, traffic history, realized travel times, and the original route sequence constraints are not modeled. Table 12 shows that all methods serve all 80 stops. OR-Tools and PyVRP produce shorter routes under the 3-second solver budget, while SRI-ECR-balanced executes in milliseconds and retains a $2.32\times$ incremental cache speedup. This result is used only to show that the framework can run on real stop coordinates.

Homberger-400 boundary. We also run a Homberger-400 slice to identify the boundary of the current fixed online repair configuration. Under a 120-second budget, full SRI-ECR-balanced does not recover full feasibility: it serves 312/400 customers in the normal setting and 302/400 customers in the mixed-stress setting. This result should not be interpreted as positive scalability evidence. Instead, it shows that the current large-instance repair policy is insufficient at 400 customers under the tested online budget. The main scalability evidence in this paper remains the Homberger-200 slice, while larger instances require stronger decomposition or repair policies.

Table 12: Amazon Last Mile coordinates-only real-stop dynamic slice with 80 drop-off stops. Distance is cumulative planned route distance over artificial release epochs.

Method	Feas.	Served	Planned dist.	Resp. (s)	Speedup
Regret insertion	1	80	5793.361	0.0051	1.00
OR-Tools	1	80	5731.545	3.0038	1.00
PyVRP	1	80	5730.841	3.0174	1.00
SRI-ECR-balanced	1	80	6739.837	0.0043	2.32

6 Discussion

6.1 Why SRI-ECR Targets Dynamic Conflict-Stress Re-optimization

The main contribution of SRI-ECR-balanced should be interpreted in the context of dynamic conflict-stress re-optimization, not as a claim of universal dominance on static VRPTW benchmarks. Highly optimized static solvers such as HGS-based methods, PyVRP, and OR-Tools are designed to search intensively for high-quality complete route plans under fixed input. SRI-ECR addresses a different operational setting: routes must be updated after new orders are released, infeasible insertions must be detected before they damage the plan, and the system must preserve enough route stability for online execution.

This distinction explains why the evaluation emphasizes feasible rate, repair success, rollback behavior, prefix acceptance, response time, rejected orders, and cache speedup in addition to distance. In a dynamic dispatch system, a slightly shorter route is not useful if the solver frequently leaves orders unrepaired, violates hard constraints, or requires excessive re-optimization time. The conflict-stress experiments therefore test whether the framework can restore feasibility and reuse prior computation when time windows and capacity constraints become difficult. Under this interpretation, the central result is that SRI-ECR-balanced maintains feasible rate 1.0 and repair success rate 1.0 on D-Solomon stress scenarios while also showing consistent cache-based evaluation speedup.

This positioning also implies an important limitation: SRI-ECR-balanced is not the best choice when the only objective is minimizing planned distance and a solver can spend several seconds at every decision epoch. In both ordinary dynamic and conflict-stress solver baselines, OR-Tools and PyVRP produce shorter planned routes. In the conflict-stress comparison, SRI-ECR-balanced is approximately 7.8% longer than OR-Tools/PyVRP. The time-budget curve strengthens this conclusion rather than removing it: even at 0.01 seconds per epoch, PyVRP remains better in planned distance on the tested c101 stress settings, although SRI-ECR-balanced responds faster. The replicated C/R/RC diagnostic similarly shows a stable distance penalty relative to OR-Tools/PyVRP and a stable response-time advantage. The method should therefore be evaluated as a response-time and feasibility-recovery trade-off, not as a distance-quality replacement for strong solvers.

6.2 Effect of Balanced Speculation

Aggressive draft generation is useful for testing the verifier and rollback modules because it intentionally allows high-risk route fragments to enter the verification stage. However, it is not the

most suitable default policy for online routing. The results show that aggressive drafting produces a rollback rate of 0.8717 and a prefix acceptance rate of 0.4400. Balanced drafting reduces rollback to 0.6805 and raises prefix acceptance to 0.6622, while also reducing average response time from 17.65 seconds to 10.21 seconds in the draft-mode comparison.

This pattern supports the design choice that SRI-ECR should speculate, but not speculate blindly. If the draft generator is too conservative, the framework degenerates into one-step insertion and the verifier has little opportunity to contribute. If it is too aggressive, the system spends excessive time rolling back and repairing invalid suffixes. Balanced speculation occupies the useful middle ground: it still proposes multi-customer route fragments, but it penalizes obvious time-window and capacity risks before verification. This makes the default method more appropriate for actual dynamic dispatch, while aggressive drafting remains valuable as a stress-test setting.

6.3 Importance of Verifier, Rollback, and Expert-Guided Repair

The verifier and rollback mechanism are essential because speculative route fragments can contain both feasible and infeasible parts. An all-or-nothing rejection strategy would waste useful feasible prefixes, whereas accepting an unchecked draft could propagate violations into later route construction. SRI-ECR therefore accepts the longest feasible prefix and rolls back only the invalid suffix. This design mirrors the prefix-acceptance logic of speculative inference while preserving deterministic routing feasibility checks.

The expert router is particularly important in conflict-stress settings. The ablation results show that full SRI-ECR-balanced reaches feasible rate 1.0 and repair success rate 1.0, while removing the expert router reduces feasible rate to 0.0 and repair success to 0.3434 in the balanced D-Solomon stress setting. In the broader aggressive-stress setting, both w/o expert router and random repair achieve only 0.1994 repair success. This means the framework does not merely need “some” repair operator. It needs the repair process to be conditioned on the type of conflict. Capacity pressure, time-window conflicts, and mixed conflicts require different customer orderings, route priorities, and insertion search behavior.

These results also clarify the role of the deterministic verifier. The verifier is not only a final feasibility checker. It prevents invalid suffixes from contaminating subsequent route construction and creates structured repair tasks for the expert router. The combination of verifier, rollback, and expert-guided repair is therefore the core robustness mechanism of SRI-ECR-balanced.

6.4 Efficiency Contribution of Route-Prefix Cache and Adaptive Early Stop

The route-prefix cache contributes by reducing repeated incremental route evaluation. Dynamic re-optimization often preserves substantial route prefixes from the previous plan. Recomputing distance, arrival time, load, slack, and violation states from the depot for every candidate insertion wastes computation. SRI-ECR caches these prefix states and reuses them during verification and repair. The measured cache speedup is $3.12\times$ on D-Solomon stress scenarios and $3.08\times$ on the Homberger-200 mixed-stress slice.

This speedup should be interpreted carefully. It is an incremental route-evaluation speedup, not a claim that the entire end-to-end solver is three times faster. End-to-end response time also includes draft generation, repair search, local refinement, Python overhead, and bookkeeping. Nevertheless, cache speedup is meaningful because route evaluation is repeatedly invoked during insertion and

repair. The w/o cache variant preserves feasibility but loses the speedup, confirming that the cache is an efficiency module rather than a feasibility shortcut.

Adaptive early stop addresses a different efficiency problem. A fixed high repair budget can preserve feasibility, but it may spend unnecessary effort after a feasible repair has already been found. In the adaptive budget comparison, fixed high budget obtains feasible rate 1.0 and repair success 1.0 with average response time 11.18 seconds. Adaptive early stop keeps the same feasible rate and repair success while reducing response time to 4.39 seconds and repair evaluations to 860.0. This supports the online-routing principle that feasibility recovery should be prioritized first, and optional distance improvement should stop when marginal gains are unlikely to justify the response-time cost.

6.5 Threats to Validity

Several threats to validity should be made explicit. First, the dynamic and conflict-stress scenarios are benchmark-derived. They provide controlled, reproducible tests of time-window, capacity, and mixed conflicts, but they do not fully represent real last-mile operations with driver preferences, customer cancellations, package compatibility, service priorities, and traffic disruptions. The stochastic travel-time perturbation experiment and the Amazon Last Mile coordinates-only real-stop slice partially address this concern, but neither is a substitute for a full real-world deployment with operational constraints.

Second, parameter sensitivity remains a threat. Draft length, time-window risk weights, repair budgets, and early-stop thresholds affect the balance between response time and solution quality. We fix these settings in the main experiments to avoid post-hoc tuning, but broader parameter sweeps would be needed before operational deployment. Similarly, the cost-benefit threshold ρ is not calibrated from company data; it is a break-even sensitivity value showing when response time would be valuable enough to offset longer planned routes.

Third, the empirical baseline set does not include a direct learning-based solver such as AM or POMO. These models are important neural routing baselines, but standard implementations target static TSP/CVRP-like construction and do not directly support release times, rolling-horizon re-optimization, time-window conflict recovery, rollback repair, expert routing, or route-prefix caching. A feasibility-adapted POMO comparison on small dynamic VRPTW instances would be valuable, but it would require a separate adaptation and training protocol. We therefore treat the absence of learned-solver baselines as a validity threat rather than claiming superiority over neural routing methods.

Fourth, the original compact statistical diagnostics have limited power because several early comparisons are based on matched scenario slices rather than many independent random seeds. The replicated Solomon C/R/RC diagnostic partially mitigates this concern by adding 60 matched dynamic runs over six base instances, five release seeds, and two scenarios. These replicated tests confirm the direction of the solver trade-off: SRI-ECR-balanced is significantly longer in planned distance than OR-Tools/PyVRP but significantly faster in response time. They still should not be read as broad operational proof, because the release seeds and stress settings are benchmark-derived rather than independent real deployments.

Fifth, implementation effects may influence wall-clock comparisons. The prototype is written in Python, while OR-Tools and PyVRP call optimized solver components. We therefore avoid claiming universal runtime dominance and report response time, repair evaluations, and cache

speedup separately. The reported cache speedup is specifically an incremental route-evaluation speedup, not an end-to-end speedup of the whole solver.

Sixth, the Homberger-400 boundary shows that the current online repair design does not yet scale to all larger instances under the tested 120-second budget. This limits the scalability claim to the Homberger-200 slice and motivates future work on decomposition, candidate pruning, and stronger large-instance repair.

6.6 Limitations and Future Work

The current study has several limitations. First, the large-scale evidence is intentionally limited to a Homberger-200 mixed-stress scalability slice. This slice shows that the mechanism remains valid beyond Solomon-100: full SRI-ECR-balanced serves all 200 customers with no rejected orders and retains a $3.08\times$ cache speedup. However, it is not a full benchmark ranking across Homberger 400–1000 instances. A Homberger-400 boundary experiment further shows the current limit of the fixed online repair configuration: under a 120-second budget, full SRI-ECR-balanced serves 312/400 customers in the normal setting and 302/400 customers in the mixed-stress setting.

The Homberger-400 boundary is mainly caused by the repair search space and the current non-hierarchical insertion policy. As the number of customers increases, the number of candidate insertion positions grows quickly across routes, and the rule-based repair module spends more budget searching feasible positions for rolled-back customers. The route-prefix cache reduces repeated evaluation cost, but it does not reduce the combinatorial number of candidate insertions. Therefore, the bottleneck is not only route evaluation, but large-instance repair search. A practical next step is to add a large-instance repair layer before attempting Homberger 600–1000: cluster-first decomposition to restrict candidate routes, route-level candidate pruning based on nearest depots and slack, route-block caching instead of only prefix caching, parallel batch verification of insertion positions, and an ALNS/HGS handoff when the pending set exceeds the online repair budget. These changes are algorithmically compatible with SRI-ECR because they modify the repair search space without removing deterministic verification or expert routing.

Second, the current draft generator and expert router are heuristic and rule-based, and the paper does not include an empirical comparison with learning-based routing solvers such as AM/POMO. This is a deliberate scope choice for interpretability and reproducibility, because adapting neural construction policies to dynamic VRPTW with release times and feasibility-recovery semantics would require a separate training and feasibility-adaptation pipeline. Future work should evaluate learned draft generation, contextual-bandit expert selection, or feasibility-adapted POMO-style policies under the same rolling-horizon event stream. Such extensions should preserve the deterministic verifier, because constraint feasibility is a core strength of the framework.

Third, the experiments use benchmark-derived dynamic and stress scenarios. These scenarios are useful because they are controlled and reproducible, but real last-mile systems include additional factors such as stochastic travel times, driver preferences, package compatibility, service priority, and customer cancellations. As partial robustness checks, we perturb D-Solomon mixed-stress travel times with $\pm 20\%$ multiplicative noise and run a small Amazon Last Mile coordinates-only real-stop slice with 80 drop-off locations and artificial releases. The perturbation test preserves the main feasibility-recovery pattern, and the Amazon slice verifies that the implementation can operate on real stop coordinates. However, these checks do not constitute a full real-world validation. Applying SRI-ECR to operational delivery data would require integrating richer constraints into the

verifier and repair experts.

Finally, route-prefix caching is evaluated as an algorithmic mechanism inside the current implementation. A production system could further improve cache management by using more explicit invalidation policies, shared route-block storage, or parallel batch verification. These extensions align naturally with the inference-systems motivation of SRI-ECR, but they are outside the scope of the present paper.

7 Conclusion

This paper presented SRI-ECR-balanced, an online re-optimization framework that transfers the speculative-execution control patterns of modern inference systems to dynamic vehicle routing with time windows. The framework organizes each latency-bounded re-optimization epoch as an inference-style loop: balanced draft generation proposes candidate route fragments, deterministic verification accepts the longest feasible prefix, rollback removes invalid suffixes, expert-guided repair restores rejected customers, and route-prefix caching reuses incremental route evaluation states across epochs.

The experimental results support the view that SRI-ECR-balanced is a dynamic conflict-reoptimization framework rather than a one-shot static VRP optimizer. On D-Solomon conflict-stress scenarios, the full method maintains feasible rate 1.0 and repair success rate 1.0. Balanced speculation reduces rollback and improves prefix acceptance compared with aggressive speculation. Expert-guided repair is essential: removing the expert router collapses feasibility under stress. Route-prefix caching provides a $3.12\times$ incremental evaluation speedup on D-Solomon and a $3.08\times$ speedup on the Homberger-200 slice. Adaptive early stop preserves feasibility while reducing response time relative to a fixed high repair budget.

Additional defensive experiments clarify this operating regime. OR-Tools and PyVRP can produce shorter routes in ordinary dynamic and conflict-stress settings, and a time-budget curve shows that this distance advantage remains visible even under short per-epoch budgets. A replicated Solomon C/R/RC diagnostic over 60 matched dynamic runs confirms the same pattern: SRI-ECR-balanced is significantly longer in planned distance than OR-Tools/PyVRP, but significantly faster in response time. Stochastic travel-time perturbation preserves the main feasibility-recovery pattern; a small Amazon Last Mile coordinates-only real-stop slice shows that the framework can execute on real stop coordinates; and the Homberger-400 boundary experiment shows that the current fixed online repair configuration is not yet sufficient for full feasibility at 400 customers. A simple break-even cost calculation further shows how the faster response regime could become preferable when dispatch delay carries a sufficiently high operational penalty, although this threshold is not calibrated from operational cost data. These findings suggest that dynamic routing algorithms should be evaluated not only by final planned distance, but also by feasibility recovery, response time, repair success, route stability, computational reuse, and operating boundaries. Future work will extend the scalability study through cluster-first decomposition, candidate pruning, route-block caching, and learned draft or expert-routing policies while retaining deterministic feasibility verification.

Declarations

Funding. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflict of interest. The author declares that there is no conflict of interest.

Data availability. Data will be made available on request.

References

- [1] George B. Dantzig and John H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. doi: 10.1287/mnsc.6.1.80.
- [2] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987. doi: 10.1287/opre.35.2.254.
- [3] Martin Desrochers, Jacques Desrosiers, and Marius M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992. doi: 10.1287/opre.40.2.342.
- [4] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005. doi: 10.1287/trsc.1030.0056.
- [5] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005. doi: 10.1287/trsc.1030.0057.
- [6] Dimitris J. Bertsimas and Garrett Van Ryzin. A stochastic and dynamic vehicle routing problem in the euclidean plane. *Operations Research*, 39(4):601–615, 1991. doi: 10.1287/opre.39.4.601.
- [7] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013. doi: 10.1016/j.ejor.2012.08.015.
- [8] Ulrike Ritzinger, Jakob Puchinger, and Richard F. Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016. doi: 10.1080/00207543.2015.1043403.
- [9] Niklas Soeffker, Marlin W. Ulmer, and Dirk C. Mattfeld. Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. *European Journal of Operational Research*, 298(3):801–820, 2022. doi: 10.1016/j.ejor.2021.07.014.
- [10] Daniel Merchán, Jatin Arora, Julian Pachon, Karthik Konduri, Matthias Winkenbach, Steven Parks, and Joseph Nosal. 2021 amazon last mile routing research challenge: Data set. *Transportation Science*, 58(1):8–11, 2022. doi: 10.1287/trsc.2022.1173.

- [11] Lixia Wu, Haomin Wen, Haoyuan Hu, Xiaowei Mao, Yutong Xia, Ergang Shan, Jianbin Zheng, Junhong Lou, Yuxuan Liang, Liuqing Yang, Roger Zimmermann, Youfang Lin, and Huaiyu Wan. LaDe: The first comprehensive last-mile delivery dataset from industry. *arXiv preprint arXiv:2306.10675*, 2023. doi: 10.48550/arXiv.2306.10675.
- [12] David Pisinger and Stefan Røpke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007. doi: 10.1016/j.cor.2005.09.012.
- [13] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006. doi: 10.1287/trsc.1050.0135.
- [14] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013. doi: 10.1016/j.cor.2012.07.018.
- [15] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022. doi: 10.1016/j.cor.2021.105643.
- [16] Google for Developers. Vehicle routing problem with time windows. <https://developers.google.com/optimization/routing/vrptw>, 2026. Accessed 2026-05-28.
- [17] RoutingLab. Pyvrp documentation. <https://pyvrp.org/>, 2026. Accessed 2026-05-28.
- [18] Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: A high-performance VRP solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024. doi: 10.1287/ijoc.2023.0055.
- [19] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. doi: 10.1016/j.ejor.2020.07.063.
- [20] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations Workshop*, 2017.
- [21] MohammadReza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, 2018.
- [22] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ByxBFsRqYm>.
- [23] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198, 2020.

- [24] André Hottung and Kevin Tierney. Neural large neighborhood search for routing problems. *Artificial Intelligence*, 313:103786, 2022. doi: 10.1016/j.artint.2022.103786.
- [25] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. Routefinder: Towards foundation models for vehicle routing problems. *arXiv preprint arXiv:2406.15007*, 2024. doi: 10.48550/arXiv.2406.15007.
- [26] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286, 2023. URL <https://proceedings.mlr.press/v202/leviathan23a.html>.
- [27] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. doi: 10.1145/3600006.3613165.
- [28] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.
- [29] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024. doi: 10.48550/arXiv.2401.02051.