# VISO: VISUAL SKILL DISTILLATION FOR LARGE-SCALE ROUTING OPTIMIZATION

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

## ABSTRACT

Decomposition enables large-scale routing optimization by allowing powerful solvers to focus on tractable subproblems. Yet selecting productive subproblems remains challenging, and existing approaches leave their selection experience largely implicit and difficult to reuse. We propose **ViSO**, a novel framework that *guides routing decomposition with transferable visual skills* for solving large-scale routing problems. ViSO distills solver feedback into transferable visual skills that direct optimization toward promising subproblems. A unified decomposition interface couples this guidance with off-the-shelf solvers across symmetric, asymmetric, and capacitated routing problems. By distilling optimization experience into reusable visual skills, ViSO enables decomposition strategies to generalize to unseen instances without updating model parameters. Across four routing families with up to 85,900 nodes, ViSO reduces mean TSP and CVRP gaps by 10.5% and 11.5% relative to LKH-3 and HGS, respectively, and achieves a mean ACVRP objective value 2.72 lower than PyVRP's. ViSO outperforms neural and LLM/VLM solvers (e.g., achieving a 26.5% improvement on TSP over ViTSP and a 94.8% improvement on CVRP over NCO-LLM), while its skill-distillation procedure is able to improve existing divide-and-conquer frameworks (e.g., DRAGON).

## 1 INTRODUCTION

Combinatorial optimization problems (COPs) underlie applications in transportation, logistics, and manufacturing, where routing problems such as the Traveling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), and their variants are often required to be solved at scale. Exact solvers such as Concorde (Applegate, 2006) guarantee optimality but do not scale well. Designing heuristic solvers such as Lin-Kernighan-Helsgaun (LKH-3) (Helsgaun, 2017) for TSP and Hybrid Genetic Search (HGS) (Vidal, 2022) for VRP requires extensive domain knowledge and calibration of parameters (Adenso-Diaz & Laguna, 2006), resulting in high development cost. Learning-based neural solvers replace hand-crafted rules with trained policies, either constructing solutions end-to-end (Kool et al., 2018; Kwon et al., 2020; Sun & Yang, 2023) or through learned decomposition (Li et al., 2021; Ye et al., 2024; Zheng et al., 2024). Generalization across instance scales and structures remains a significant challenge (Fang et al., 2024; Li & Zhang, 2026).

Large language models (LLMs) and vision-language models (VLMs) provide an alternative paradigm for solving COPs. For example, LLMs have been used as direct optimizers over textual problem descriptions or as designers of heuristics expressed in code (Yang et al., 2024; Liu et al., 2024), and recent agentic methods such as DRAGON decompose a large instance with LLM agents and reconstruct each subproblem from textual state (Chen et al., 2026). Nevertheless, text alone can be an inadequate representation of complex combinatorial structures, making reliable reasoning increasingly difficult at scale. By contrast, ViTSP (Yin et al., 2026) adopts a visual reasoning paradigm for large-scale COPs, leveraging a VLM to identify promising subproblems for decomposition and subsequently re-optimizing them with an off-the-shelf exact solver. Despite its promise, ViTSP exhibits three fundamental limitations that substantially constrain its practical applicability. (1) It is confined to the symmetric Euclidean TSP and cannot accommodate constraints essential to broader routing problems, such as vehicle capacities, time windows, and depot requirements. (2) Its assumption of symmetric travel costs further limits real-world deployment, where one-way streets, turn restrictions, and traffic conditions inherently induce direction-dependent travel distances and times (Son et al., 2026). (3) More fundamentally, its visual reasoning remains implicit



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

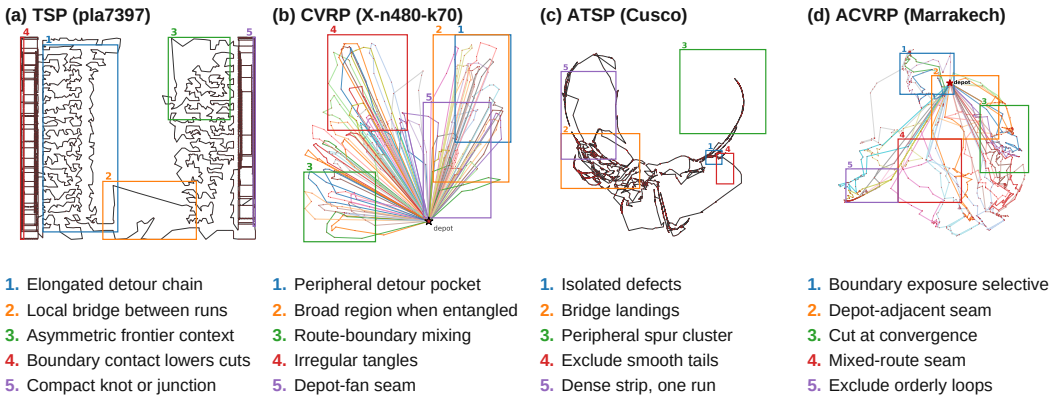



Figure 1: **Distilled visual skills guide ViSO toward effective subproblem regions across four problem families.** Numbered boxes show regions selected by the VLM based on family-specific visual skills, with each selection yielding a committed objective improvement.

and problem-specific: ViTSP neither characterizes the visual skills that enable effective decomposition nor provides a principled mechanism for extracting and transferring these skills across diverse routing problems, leaving the reasoning process difficult to interpret, reuse, or improve.

In this paper, we propose **ViSO** (**Vi**sual **S**kill Distillation for Large-Scale Routing **O**ptimization), a unified framework that distills solver feedback into reusable visual skills for large-scale routing problems. The central idea is to encode decomposition experience as explicit guidance for selecting productive subproblems. ViSO contrasts successful and unsuccessful decomposition attempts and consolidates experience across training instances into a compact skill library. In this way, each skill specifies a visual cue, a recommended action, and a failure mode to avoid, providing an inspectable representation of selection knowledge. During inference, these skills guide the VLM's region selection, with region-level citations linking selected subproblems to the skills invoked. Figure 1 illustrates skill-guided region selection across four routing problems, with each displayed selection yielding an objective improvement after re-optimization. The resulting bank enables decomposition experience to be reused on unseen instances without updating model parameters.

To support diverse TSP and VRP variants, ViSO integrates constraint-aware hypernode decomposition with compatible off-the-shelf solvers. For asymmetric instances specified by cost matrices, a multidimensional-scaling module constructs a two-dimensional layout for visual reasoning. We further assess the portability of the distillation procedure by adapting it to DRAGON, an LLM-based divide-and-conquer solution. Skills distilled from DRAGON's own optimization trajectories improve solution quality without modifying its core solver, both after an equal number of solved subproblems and under matched runtime budgets. Across four routing families with up to $85{,}900$ nodes, ViSO reduces mean TSP and CVRP gaps by approximately $10.5\%$ and $11.5\%$ relative to LKH-3 and HGS, respectively, Additionally, ViSO reduces mean TSP and CVRP gaps by $26.5\%$ and $94.8\%$ relative to the best-performing LLM/VLM-based solvers, respectively. Note that ViSO also provides competitive results on asymmetric COPs such as ATSP and ACVRP, achieving a mean ACVRP objective value $2.72$ below PyVRP's, whereas the evaluated LLM/VLM-based solvers fail to return feasible solutions. Our contributions are threefold:

1. **A unified vision-guided decomposition framework.** We introduce constraint-aware hypernode decomposition as a unified interface between pre-trained VLMs and exact or heuristic solvers, enabling visual subproblem selection across symmetric, asymmetric, and constrained COPs.
2. **Skill distillation for visual decomposition.** To the best of our knowledge, ViSO is the first routing optimization framework to systematically distill and reuse decomposition experience. By distilling decomposition trajectories into a bank of transferable skills for VLM-guided region selection, ViSO effectively improves decomposition quality on unseen instances and provides traceable citations linking skill-guided selections to the skills invoked.
3. **Large-scale evaluation and portability.** Across four COPs with up to $85{,}900$ nodes, ViSO outperforms all baselines (including LKH-3, HGS, and PyVRP) on TSP, CVRP, and ACVRP. Applying the distillation procedure to DRAGON's own trajectories also improves performance.

## 2 Related Work

**Exact, heuristic, and learning-based solvers.** Exact solvers such as Concorde and branch-cut-and-price frameworks guarantee optimality but scale poorly (Applegate, 2006; Pessoa et al., 2020). Heuristics such as LKH-3, HGS, and PyVRP offer strong routing performance (Helsgaun, 2017; Vidal, 2022; Wouda et al., 2024), although adapting them to new variants often requires domain knowledge and parameter calibration (Adenso-Diaz & Laguna, 2006). For very large instances, SISRs and FILO2 offer scalable alternatives (Christiaens & Vanden Berghe, 2020; Accorsi & Vigo, 2024), while OR-Tools emphasizes broad applicability (Furnon & Perron, 2024). Within this literature, neighborhood search, decomposition, and segment stability motivate selective re-optimization rather than rebuilding entire solutions (Shaw, 1998; Santini et al., 2023; Morabit et al., 2024).

Learning-based methods automate parts of this process. Neural solvers construct solutions end-to-end (Kool et al., 2018; Kwon et al., 2020; Luo et al., 2023; 2025a;b; Sun & Yang, 2023; Qiu et al., 2022; Fang et al., 2024), scale through heatmap composition or divide-and-conquer partitioning (Fu et al., 2021; Zong et al., 2022; Pan et al., 2023; Ye et al., 2024; Zheng et al., 2024), or learn subproblem selection and reconstruction (Li et al., 2021; Cheng et al., 2023; Li et al., 2025; Ouyang et al., 2026). L2R learns to reduce the candidate search space during solution construction (Zhou et al., 2026). Recent extensions address asymmetric costs (Meng et al., 2025; 2026; Son et al., 2026; Yi et al., 2026) and instance-level solver selection (Gao et al., 2024). Rather than replacing heuristics, NeuroLKH and DeepACO use trained networks to guide LKH search and ant colony optimization, respectively (Xin et al., 2021; Ye et al., 2023). These methods can perform well in distribution but remain vulnerable to shifts in scale, distribution, or constraints. ViSO instead guides decomposition with a pre-trained VLM, without task-specific model training.

**LLM/VLM-driven methods.** LLMs have been used as direct optimizers (Yang et al., 2024) and heuristic-code generators (Liu et al., 2024; 2026). In routing, they supply heuristic knowledge, generate training instances, or coordinate solver pipelines (Tran et al., 2025; Zhu et al., 2026; Zhang et al., 2026), while multimodal models have been tested on routing images at toy scales (Elhenawy et al., 2024). More directly related to ViSO, DRAGON decomposes and reconstructs large instances from textual states (Chen et al., 2026), whereas ViTSP uses a VLM to select image regions for exact re-optimization (Yin et al., 2026). However, ViTSP is limited to symmetric Euclidean TSP and reuses past decompositions as raw per-instance trajectories, leaving their transfer and interpretation unresolved. Building on advances in agentic memory and skill acquisition (Shinn et al., 2023; Wang et al., 2023; Zhao et al., 2024; Ni et al., 2026), ViSO provides an early attempt to distill the decomposition experience into reusable, explainable visual skills for symmetric, asymmetric, and constrained large-scale routing problems. Compared with ExpeL (Zhao et al., 2024), ViSO structures experience reuse around visual subproblem selection, coupling solver-measured improvements with image-conditioned skill sampling and traceable region-level decisions.

## 3 Method

ViSO improves a large routing solution through a *skill-guided* iterative decomposition loop (Figure 2). Starting from a feasible initial solution, ViSO renders the current solution as an image. An image-conditioned *visual scheduler* makes a single VLM pass over this image and turns a bank of distilled *skills* into an instance-specific sampling distribution, weighting each skill by how well its visual cue matches the instance. A VLM is then shown the rendered current solution together with $Q$ skills sampled from this distribution and proposes $Q$ rectangular regions that appear improvable. The VLM proposes each region together with its corresponding guiding skill, making the decomposition explainable. The nodes inside a proposed region and the fixed route segments surrounding them are contracted into a small subproblem. Next, an off-the-shelf exact or heuristic solver re-optimizes that subproblem. Its output is merged into the global solution only if it reduces total cost.

### 3.1 Preliminaries

We consider routing problems defined on $n$ nodes with a cost matrix $D \in \mathbb{R}^{n\times n}$, where $D_{ij} \geq 0$ is the cost of traveling from node $i$ to node $j$. An instance is *symmetric* when $D = D^\top$, as in the Euclidean TSP and the CVRP, and *asymmetric* when $D \neq D^\top$, as in the ATSP and the ACVRP. A

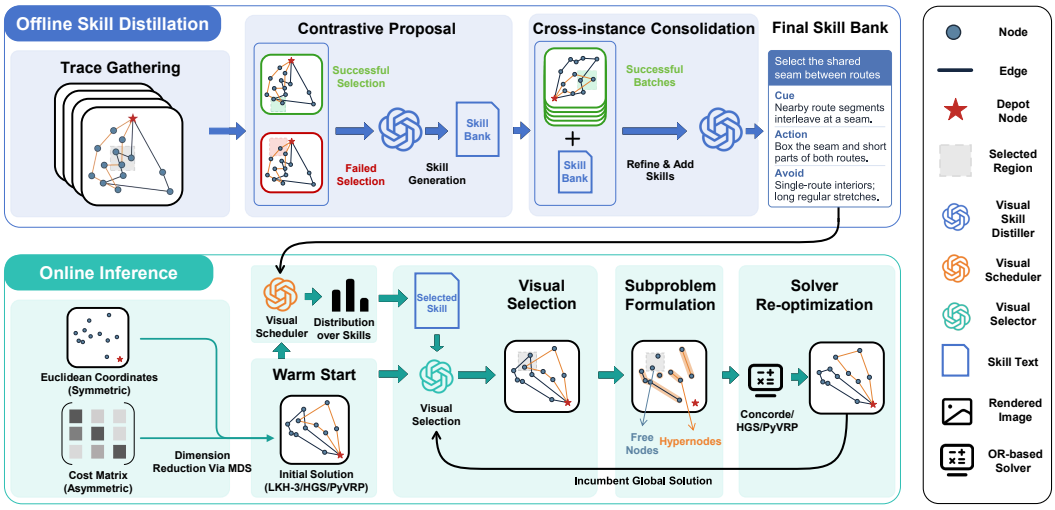



Figure 2: **The ViSO framework.** *Offline skill distillation* (top): skill-free trace gathering, contrastive success–failure proposal, and cross-instance consolidation distill a compact skill bank. *Online inference* (bottom): a warm-start solution is rendered (via MDS when the instance is non-geometric); the visual scheduler retrieves a distribution over skills; the VLM proposes box region(s), each citing one skill; the region is contracted into a hypernode subproblem, re-optimized by an off-the-shelf solver, and merged back only if it lowers total cost.

solution $\mathcal{S}$ is a cyclic tour $\Pi$ for TSP/ATSP or a set of depot-to-depot routes $\mathcal{R}$ for CVRP/ACVRP, visiting every customer exactly once. We minimize total travel cost $c(\mathcal{S})$ subject to feasibility; capacitated routes carry customer demands $q_i > 0$ with vehicle capacity $Q_{\text{cap}}$. Full formulations and additional variant constraints appear in Appendix A. We measure solution quality by the optimality gap $100\,(c - c^\star)/c^\star$ of a solution of cost $c$ against a reference $c^\star$, taken as a published optimum or best-known solution when available and a strong reference solver otherwise.

### 3.2 Rendering the Instance

ViSO obtains a feasible warm-start solution and keeps a single current solution in memory. To let a VLM reason about this solution, ViSO draws it as a two-dimensional image of the instance. The nodes are plotted at their positions and the current routes are overlaid as line segments, and the VLM is asked to mark improvable regions directly on this image. Euclidean instances (TSP, CVRP) obtain node positions from coordinates. However, as asymmetric instances (ATSP and ACVRP) are defined only by a cost matrix and have no native coordinates, we introduce a visualization module based on classical multidimensional scaling (MDS) (Torgerson, 1952), which derives a two-dimensional embedding from the symmetrized cost matrix, assigning each node $i$ a point $x_i \in \mathbb{R}^2$,

$$(x_1, \ldots, x_n) = \mathrm{MDS}(D). \tag{1}$$

ViSO then draws the current solution at these coordinates and maps a marked region back to the nodes it encloses. The detailed construction is given in Appendix B. The embedding of Eq. (1) is used only for VLM visualization. All cost evaluations, feasibility checks, and reconstruction use $D$.

### 3.3 Skill-Guided Region Selection

Let $\mathcal{B} = \{\sigma_1, \ldots, \sigma_M\}$ denote the skill bank, where each skill $\sigma_m$ is a short natural-language triple of a visual *cue* (what to look for), an *action* (what region to select), and a *pitfall* (what to avoid). At the start of an instance, an image-conditioned visual scheduler performs a single VLM pass over the initial rendered solution $X^0$ and maps the bank to an instance-specific sampling distribution,

$$p = (p_1, \ldots, p_M) = F_{\text{sched}}\big(X^0, \mathcal{B}\big), \qquad p_m > 0, \quad \textstyle\sum_{m=1}^{M} p_m = 1, \tag{2}$$

where $p_m$ reflects how strongly the cue of $\sigma_m$ matches the rendered instance. The distribution is computed once and held fixed for the remainder of the run. During each selection call, ViSO samples

$Q$ skill assignments from the scheduler distribution $\mathrm{Cat}(p)$. We use one sampled skill per region and $Q = 2$ in our experiments, and the VLM maps the current image $X$ and the sampled skills to $Q$ axis-aligned boxes together with per-box citations,

$$\{(C_q, \hat{\sigma}^{(q)})\}_{q=1}^{Q} = F_{\text{select}}\big(X;\, \sigma^{(1)}, \ldots, \sigma^{(Q)}\big), \tag{3}$$

where each box $C_q = (x_{\min}, x_{\max}, y_{\min}, y_{\max})$ is expressed in image coordinates and mapped back to the enclosed free-node set $W(C_q)$, and the citation $\hat{\sigma}^{(q)} \in \{\sigma^{(q)}, \mathsf{none}\}$ records whether the $q$-th box was selected under its assigned skill. Boxes exceeding the solver's node-count cap are handled by the zoom-in re-selection procedure (Algorithm 2 in Appendix G). Oversized regions use a cropped view with a finer grid under the same pinned skill; only admissible boxes are enqueued. The visual-selection interface follows Yin et al. (2026). The visual scheduler of Eq. (2), the injected skills, and the citation feedback of Eq. (3) are specific to ViSO. Both prompts are in Appendix H.

### 3.4 Two-stage parallel skill distillation

The bank $\mathcal{B}$ injected in Sec. 3.3 is distilled offline by a two-stage, hierarchically parallel procedure that converts raw solving experience into a compact set of reusable skills. A separate bank is distilled for every problem family and, within a family, for each size tier, since the visual cues that matter differ across variants and scales. Everything below therefore describes the construction of one bank. Traces are gathered by running the skill-free selector on the training instances. Every logged selection contributes an evidence tuple to a log

$$\mathcal{E} = \big\{(C_i, \Delta_i, t_i)\big\}_{i=1}^{|\mathcal{E}|}, \qquad \Delta_i = c(\mathcal{S}_i) - c(\mathcal{S}_i'), \tag{4}$$

recording the selected region, the change in global cost it produced, and its solver runtime. Let $I_\ell$ index the selections logged on training instance $\ell \in \{1, \ldots, L\}$. Then $\mathcal{E}_\ell^+ = \{i \in I_\ell : \Delta_i > 0\}$ and $\mathcal{E}_\ell^- = \{i \in I_\ell : \Delta_i \le 0\}$ denote its successes and failures.

**Stage A (contrastive proposal).** Stage A asks *what separates a productive region from an unproductive one on the same instance*. For each training instance $\ell$, the $r$-th strongest success is paired with the unused failure closest to it in region size,

$$\mathcal{P}_\ell = \big\{(e_r^+, e_r^-)\big\}_{r=1}^{\Lambda_\ell}, \qquad e_r^- = \arg\min_{e \in \mathcal{E}_\ell^- \setminus U_{\ell,r}} \big|\, |W(C(e))| - |W(C(e_r^+))| \,\big|, \tag{5}$$

where $C(e)$ is the box logged for selection $e$, $W(C(e))$ is its enclosed free-node set, and $U_{\ell,r} = \{e_1^-, \ldots, e_{r-1}^-\}$ contains the failures already paired on instance $\ell$ ($U_{\ell,1} = \varnothing$). A total of $0 \le \Lambda_\ell \le \min\{\Lambda, |\mathcal{E}_\ell^+|, |\mathcal{E}_\ell^-|\}$ pairs are formed per instance ($\Lambda = 15$). Every pair is judged by an independent VLM critic call that views the two rendered selections side by side and proposes up to three candidate skills, $\Sigma_{\ell r} = F_{\text{critic}}(e_r^+, e_r^-)$; all pairs are dispatched concurrently. The proposals are then consolidated by a two-level hierarchical merge, first within and then across instances,

$$\mathcal{B}_\ell = F_{\text{merge}}\big(\{\Sigma_{\ell r}\}_r\big), \qquad \mathcal{B}_A = F_{\text{merge}}\big(\{\mathcal{B}_\ell\}_{\ell=1}^{L}\big), \tag{6}$$

which clusters near-duplicate proposals under a single wording.

**Stage B (cross-instance consolidation).** Stage A alone may promote patterns idiosyncratic to a single instance, so Stage B audits the frozen bank $\mathcal{B}_A$ by comparing successes across different instances. The top improvements of distinct instances are assembled round-robin into groups $g_1, \ldots, g_G$ of $\gamma$ successes each ($\gamma = 5$, one instance per slot), and each group is reviewed by an independent VLM call that compares its images against $\mathcal{B}_A$ and emits editing operations,

$$O_g = F_{\text{verify}}\big(g, \mathcal{B}_A\big) \subseteq \big\{\textsc{add}(\sigma),\ \textsc{agree}(\sigma),\ \textsc{edit}(\sigma)\big\}, \tag{7}$$

adding a skill only when no existing one covers a recurring pattern, endorsing a skill whose cue demonstrably fires, or refining imprecise wording. The $G$ audits also run concurrently. A final merge reconciles the operations. The two stages play complementary roles: Stage A is *discriminative*, extracting cues that separate success from failure, while Stage B is confirmatory, recording cross-instance support and using independent successes to audit and refine the bank. A validation step discards any skill that references an instance identifier or an absolute coordinate, and directional wording is disallowed, so the bank remains instance-agnostic and invariant to rotation and reflection of the image. Because skills are phrased over visual cues rather than over instances, a bank distilled from one set of instances transfers to unseen ones, which we confirm empirically in Section 4. The Stage A critic prompt, the hierarchical merge prompts, and the Stage B consolidation prompt are presented in Appendix H, together with an example distilled skill.

### 3.5 HYPERNODE SUBPROBLEM CONSTRUCTION

A selected box $C$ identifies the *free* nodes $W = W(C) = \{w_1, \ldots, w_{|W|}\}$ inside it, which may be reordered. Routes containing no free node remain unchanged. On affected routes, nodes outside the box form consecutive runs. ViSO contracts each resulting segment $\kappa = (\kappa_1, \ldots, \kappa_p)$ into a single *hypernode* $h$ with $\mathrm{entry}(h) = \kappa_1$ and $\mathrm{exit}(h) = \kappa_p$, so that the run can be re-expanded after the solve. This hypernode construction adapts the segment-contraction schemes of Yin et al. (2026); Ouyang et al. (2026); the underlying idea of freezing stable segments during re-optimization is also used by Morabit et al. (2024). Each hypernode also aggregates the attributes of its run, including an internal cost $c(h) = \sum_{j=1}^{p-1} D_{\kappa_j,\kappa_{j+1}}$, a demand $q(h) = \sum_{i\in\kappa} q_i$, and, where the variant requires, a service duration and a feasible time-window interval (Appendix A). Writing $H = \{h_1, \ldots, h_{|H|}\}$ for the hypernodes, the subproblem is the reduced routing instance $\omega = (W, H)$ whose cost couples any two elements through the endpoints at which a route would pass between them,

$$\tilde{D}_{ij} = D_{\mathrm{exit}(i),\,\mathrm{entry}(j)} \quad (i \neq j), \qquad \mathrm{entry}(w) = \mathrm{exit}(w) = w \ \text{ for } w \in W. \tag{8}$$

The reduced matrix is generally asymmetric even when $D$ is symmetric. We set $\tilde{D}_{ii} = 0$; self-edges are unused. Its block expansion is given in Appendix A. For single-depot vehicle-routing variants, the depot is retained as an additional node $h_0$ with $\mathrm{entry}(h_0) = \mathrm{exit}(h_0) =$ depot. This appends a depot row and column to $\tilde{D}$ via Eq. (8). Closed routes start and end there. Segment-internal edges and visit order are preserved, but connections between segments may change, including outside the box. The pure tour case ($\omega = (W, H)$, no depot) applies to TSP and ATSP. For TSP and ATSP subproblems, we apply the Jonker–Volgenant transformation (Jonker & Volgenant, 1983; Cirasella et al., 2001) to $\tilde{D}$ and solve the resulting symmetric TSP instance with Concorde. VRP subproblems use the solver backends specified in Sec. 3.6. Appendix A provides the corresponding resource encodings, solver constraints, and feasibility arguments for the fixed-segment neighborhood.

### 3.6 RECONSTRUCTION AND ACCEPTANCE

Let $\mathcal{S}^*$ denote the incumbent global solution (the tour $\Pi$ for TSP and ATSP, the route set $\mathcal{R}$ for the VRP family) with cost $c(\mathcal{S}^*)$. Each subproblem $\omega = (W, H)$ is dispatched to the solver suited to its class, which returns a re-optimized ordering of the reduced instance,

$$\Pi_\omega = F_{\mathrm{solver}}\big(\tilde{D},\ \mathcal{K}_\omega\big), \tag{9}$$

where $\mathcal{K}_\omega$ collects the side constraints the subproblem inherits (none for TSP/ATSP, capacity for CVRP/ACVRP, and so on per Table 3). Tour subproblems use Concorde after transformation (Applegate, 2006); CVRP uses HGS (Vidal, 2022), and ACVRP and the additional VRP variants use PyVRP (Wouda et al., 2024). Expanding every hypernode of $\Pi_\omega$ back into its stored segment preserves the segment's internal edges and order; unaffected routes are retained. Infeasible solver outputs are discarded. The complete candidate $\mathcal{S}'$ is checked against the original instance, and its cost is recomputed including all internal segment costs. With $\mathrm{feasible}(\mathcal{S}')$ denoting the full constraint check, we accept iff

$$\mathcal{S}^* \leftarrow \mathcal{S}' \quad \text{iff} \quad \mathrm{feasible}(\mathcal{S}') \ \wedge\ c(\mathcal{S}') < c(\mathcal{S}^*). \tag{10}$$

Following Yin et al. (2026), region proposal and subproblem solving run asynchronously in parallel. A serialized verifier checks candidates against the latest incumbent before acceptance; process counts and the full algorithm appear in Appendix G (Algorithm 1).

## 4 EXPERIMENTS

### 4.1 EXPERIMENTAL SETUP

**Benchmarks.** We evaluate on four COPs. (i) **TSP**: 33 TSPLIB (Reinelt, 1991) instances, $n = 1{,}000$–$85{,}900$, gap to published optima. (ii) **ATSP**: 40 real-city asymmetric instances at $n \in \{1000, 2000, 5000, 10000\}$. Since no optimum is known, LKH-3 is the reference and gaps are measured against it. The instances are generated with the city-sampling procedure of RRNCO (Son et al., 2026). (iii) **CVRP**: the 100 instances of the CVRPLIB X-set (Uchoa et al., 2017), $n = 101$–$1{,}001$, gap to best-known solutions. (iv) **ACVRP**: 40 real-city asymmetric CVRP instances, gap to

**Table 1: Results on symmetric routing benchmarks.** Mean gap (%) to published optima on 33 TSPLIB instances (TSP) and to best-known solutions on 100 CVRPLIB X-set instances (CVRP); Imp. (%) indicates the relative improvement compared to the best-performing LLM/VLM-based solvers. #Feas. counts instances with feasible solutions. Lower indicates better performance. Best results are in bold.

| **TSP** — 33 TSPLIB instances | | | | |
|---|---|---|---|---|
| Method | #Feas. | All | Small ($n \leq 2400$) | Large ($n > 2400$) |
| *OR-based solvers* | | | | |
| LKH-3 | 33 | 0.323 | **0.048** | 0.746 |
| Concorde | 33 | 0.483 | 0.305 | 0.757 |
| OR-Tools | 32 | 8.72 | 5.83 | 13.54 |
| Farthest insertion | 33 | 14.80 | 14.49 | 15.27 |
| *Neural solvers* | | | | |
| DRHG | 29 | 3.61 | 2.97 | 4.83 |
| L2C-Insert | 33 | 10.16 | 4.58 | 18.74 |
| L2R | 32 | 8.61 | 8.60 | 8.62 |
| *LLM/VLM-based solvers* | | | | |
| NCO-LLM | 31 | 13.10 | 4.97 | 27.88 |
| DRAGON | 31 | 15.42 | 14.79 | 16.57 |
| ViTSP | 33 | 0.393 | 0.190 | 0.705 |
| **ViSO (ours)** | 33 | **0.289** | 0.156 | **0.494** |
| Imp. (%) | NA | 26.5% | 17.9% | 29.9% |

| **CVRP** — 100 CVRPLIB X-set instances | | | | |
|---|---|---|---|---|
| Method | #Feas. | All | Small ($n < 500$) | Large ($n > 500$) |
| *OR-based solvers* | | | | |
| HGS | 100 | 0.261 | **0.106** | 0.590 |
| SISRs | 100 | 0.251 | 0.212 | **0.336** |
| FILO2 | 100 | 0.268 | 0.222 | 0.368 |
| LKH-3 | 98 | 3.23 | 2.97 | 3.80 |
| OR-Tools | 93 | 5.89 | 5.59 | 6.73 |
| *Neural solvers* | | | | |
| DRHG | 100 | 20.73 | 21.33 | 19.44 |
| SIL | 100 | 24.13 | 23.50 | 25.49 |
| L2R | 100 | 8.22 | 7.93 | 8.85 |
| URS | 99 | 6.61 | 5.83 | 8.32 |
| *LLM/VLM-based solvers* | | | | |
| NCO-LLM | 100 | 4.46 | 3.43 | 6.68 |
| DRAGON | 100 | 20.74 | 20.79 | 20.61 |
| ViTSP | NA | NA | NA | NA |
| **ViSO (ours)** | 100 | **0.231** | 0.124 | 0.459 |
| Imp.(%) | NA | 94.8% | 96.4% | 93.1% |

PyVRP. For skill distillation, each problem family and size tier uses a roughly 70/30 training/held-out split. Skills are distilled only from training-split trajectories. Unless stated otherwise, Tables 1 and 2 report full-benchmark results; training and held-out results appear in Table 9 (Appendix E).

**Protocol.** Where applicable, every method receives the same per-instance runtime budget, and ViSO results are averaged over three runs. The main tables report solution quality and coverage; ViSO's measured per-instance total wall-clock times appear in Appendix D. Appendix C details initialization, timing exceptions, and the matched ViTSP configuration; Appendix H gives model and hardware settings.

**Baselines.** We compare three groups of solvers. *(1) Operations research (OR)-based solvers:* LKH-3 (Helsgaun, 2017), Concorde (Applegate, 2006), HGS (Vidal, 2022), SISRs (Christiaens & Vanden Berghe, 2020), FILO2 (Accorsi & Vigo, 2024), PyVRP (Wouda et al., 2024), OR-Tools (Furnon & Perron, 2024), and farthest insertion (Rosenkrantz et al., 1977). *(2) Neural solvers:* GLOP (Ye et al., 2024), DRHG (Li et al., 2025), L2C-Insert (Luo et al., 2025b), SIL (Luo et al., 2025a), L2R (Zhou et al., 2026), URS (Zhou et al., 2025), RRNCO (Son et al., 2026), RADAR (Yi et al., 2026), and UniCO (Pan et al., 2025) in its MatPOENet and MatDIFFNet variants. *(3) LLM/VLM-based solvers:* NCO-LLM (Tran et al., 2025), DRAGON (Chen et al., 2026), and ViTSP (Yin et al., 2026).

### 4.2 Main results

**ViSO achieves competitive solution quality across all four families.** As shown in Tables 1 and 2, gaps are averaged over the reported instances; lower is better. TSP gaps are relative to the published optima, and CVRP gaps to BKS. For **TSP**, ViSO attains a mean optimality gap of $0.289\%$. On the large tier ($n > 2400$), ViSO ($0.494\%$) is well clear of LKH-3 under the same runtime budget ($0.746\%$), and it demonstrates an advantage across the full set ($0.289\%$ vs. $0.323\%$, a $10.5\%$ relative reduction), while LKH-3 remains best on the small tier ($0.048\%$). Across the 100-instance **CVRP** X-set, ViSO attains a lower mean gap than HGS under the same runtime budget ($0.231\%$ vs. $0.261\%$, an $11.5\%$ relative reduction). The margin concentrates on the harder X>500 tier ($0.459\%$ vs. $0.590\%$), while the easy tier is a near-tie. Figure 3 shows how ViSO's margin over the primary solver evolves with instance size on the two symmetric families.

With LKH-3 as the **ATSP** reference, ViSO achieves a mean gap of $0.207\%$ and ties or beats it on 10 of 40 instances with the win rate rising with size (4 of 7 at $n$=10,000; per-instance results in Table 6, Appendix D). Its mean gap is substantially lower than GLOP's ($11.65\%$), the only neural ATSP method that reaches these sizes. ViSO beats the PyVRP reference on **ACVRP** by $2.72\%$ overall under the same runtime budget, with the margin growing with size ($-2.34\%$ at $n$=500, $-3.10\%$ at $n$=1000), whereas OR-Tools and LKH-3 trail the reference by more than $10\%$ and the zero-shot neural solvers (RADAR, RRNCO) exceed $100\%$ under their strongest decode.

**Table 2: Results on asymmetric routing benchmarks.** Mean gap (%) on 40 real-city instances, relative to LKH-3 (ATSP) and PyVRP (ACVRP) given the same runtime budget. Please note that all LLM/VLM-based baselines are unable to generate feasible solutions for asymmetric COPs. Lower indicates better perfromance; and #Feas. counts instances with feasible solutions. OOS denotes scales unsupported by the released checkpoints. Best results are in bold.

| **ATSP** — 40 real-city instances | | | | |
|---|---|---|---|---|
| Method | #Feas. | All | Small ($n \leq 2{,}000$) | Large ($n > 2{,}000$) |
| *OR-based solvers* | | | | |
| LKH-3 (reference) | 40 | **0.000** | **0.000** | **0.000** |
| OR-Tools | 40 | 19.99 | 13.81 | 29.26 |
| *Neural solvers* | | | | |
| GLOP | 40 | 11.65 | 11.61 | 11.70 |
| RADAR | 19 | 265.5 | 265.5 | OOS |
| UniCO-MatPOENet | 19 | 630.4 | 630.4 | OOS |
| UniCO-MatDIFFNet | 19 | 948.8 | 948.8 | OOS |
| *LLM/VLM-based solvers* | | | | |
| **ViSO (ours)** | 40 | 0.207 | 0.215 | 0.196 |

| **ACVRP** — 40 real-city instances | | | | |
|---|---|---|---|---|
| Method | #Feas. | All | Small ($n = 500$) | Large ($n = 1{,}000$) |
| *OR-based solvers* | | | | |
| PyVRP (reference) | 40 | 0.000 | 0.000 | 0.000 |
| LKH-3 | 40 | 11.47 | 6.13 | 16.81 |
| OR-Tools | 40 | 10.55 | 9.83 | 11.27 |
| *Neural solvers* | | | | |
| RRNCO | 40 | 104.9 | 66.9 | 143.0 |
| RADAR | 40 | 102.5 | 86.3 | 118.7 |
| *LLM/VLM-based solvers* | | | | |
| **ViSO (ours)** | 40 | **−2.717** | **−2.337** | **−3.098** |

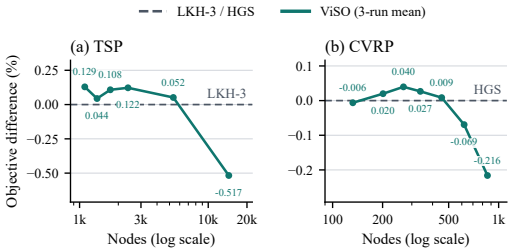



Figure 3: **Performance by instance size.** Size-binned objective difference (%), $100(c_{\text{ViSO}} - c_{\text{baseline}})/c_{\text{baseline}}$, averaged over 3 runs. Negative values favor ViSO over LKH-3 (TSP) or HGS (CVRP) under matched runtime budgets.

The ATSP and ACVRP reference solutions are obtained using LKH-3 and PyVRP, respectively, given the same runtime budget as ViSO. Training scales, decoding configurations, and coverage details are provided in Appendix C. Per-instance results for all 213 instances are listed in Appendix D. The distilled skills transfer to held-out instances in all four families, reducing mean gaps relative to the skill-free variant from $0.264\%$ to $0.202\%$ on TSP, $0.170\%$ to $0.162\%$ on ATSP, $0.241\%$ to $0.220\%$ on CVRP, and $-2.739\%$ to $-2.826\%$ on ACVRP (Table 9, Appendix E).

### 4.3 Ablation Studies

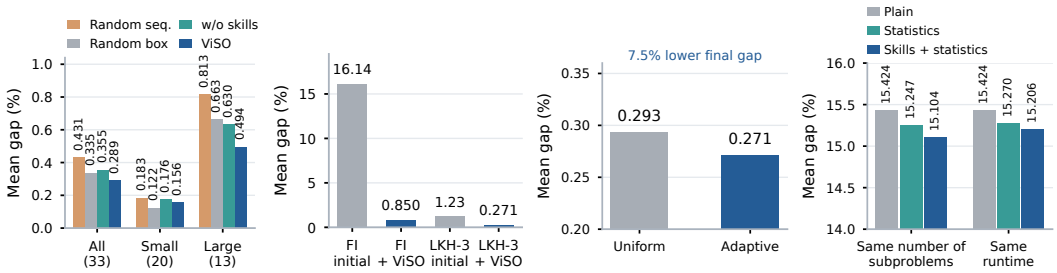
We conduct five ablation studies to assess component contributions, robustness to initial solution, and transferability to existing frameworks. Detailed results are provided in Appendix E and I.

**Effectiveness and transferability of distilled skills.** To isolate the benefit of skill guidance, we keep the same VLM and remove all skill inputs. Skills reduce the residual gaps on TSP, CVRP, and ATSP by $18.5\%$, $4.9\%$, and $15.2\%$, respectively, and increase the objective-cost advantage over PyVRP on ACVRP by $2.8\%$ relative to the skill-free variant. Gains also hold on held-out instances across all four families, supporting transfer beyond the distillation set (Table 9, Appendix E).

**Visual selection versus random region selection.** We assess whether skill-guided region selection yields greater improvements than random selection with the same per-instance runtime budgets. Across all 33 TSP instances, ViSO achieves a lower mean gap than both random box and random sequence selection (Figure 4(a)). Its advantage is most pronounced on the large tier, where skill guidance also improves upon the skill-free VLM selector. Detailed results are provided in Table 10.

**Robustness to initialization.** We assess ViSO's sensitivity to warm-start quality by replacing LKH-3 initialization with the computationally cheaper farthest-insertion (FI) heuristic. Both variants are evaluated on 10 TSP instances, with each run terminated after solving the same number of subproblems. With FI initialization, ViSO reduces the mean gap from $16.14\%$ to $0.850\%$ (Figure 4(b)), demonstrating its ability to substantially improve weak initial solutions. Experimental settings and detailed comparisons are provided in Table 11.

**Adaptive skill scheduling.** We evaluate whether conditioning skill probabilities on the initial instance image improves performance relative to uniform sampling. Figure 4(c) isolates the contribution of the visual scheduler under the same LKH-3 initialization, with both variants stopped after




(a) VLM vs. random region selection

(b) Robustness to a weak warm start

(c) Adaptive vs. uniform skill sampling

(d) Skill-enhanced DRAGON

Figure 4: **Ablation results.** Mean gaps (%); lower is better. (a) Overall, skill-guided ViSO attains a lower gap on 33 TSP instances than random boxes, random sequences, and skill-free VLM selection. (b) Starting from a much weaker farthest-insertion solution, ViSO still reduces the gap from $16.14\%$ to $0.850\%$. (c) Image-conditioned skill sampling lowers the final gap by $7.5\%$ relative to uniform sampling. (d) Distilled skills and node statistics improve DRAGON on 31 TSPLIB instances.

solving the same number of subproblems. Replacing image-conditioned sampling with a uniform distribution over the same skill bank increases the final gap from $0.271\%$ to $0.293\%$. Thus, adaptive scheduling yields a $7.5\%$ relative reduction in the remaining gap. The visual scheduler adds one initial VLM call, with comparable total token usage (1.06M versus 1.12M).

**Generalization to DRAGON.** Lastly, we also test whether the distillation procedure can improve subproblem selection beyond ViSO's visual interface. We apply ViSO's distillation procedure to DRAGON's training trajectories and use the resulting skills in its text-only selection interface, leaving the core destroy–repair solver unchanged. We add computed node statistics to make geometric and tour structure explicit for the text-only LLM. On 31 TSPLIB instances, we compare plain DRAGON, a statistics-only control (the LLM chooses which of 25 computed per-node statistics to display, e.g., edge crossings, relative edge length, detours, local density, and candidate 2-opt gain), and the full skill-plus-statistics interface (Figure 4(d)). The full interface improves over plain DRAGON both after solving the same number of subproblems and under the same runtime budget. With the same number of subproblems, skills provide an additional gain over statistics alone, including on held-out instances. These results support transferring the distillation procedure to other divide-and-conquer solvers; interface details and numerical comparisons appear in Appendix I.

## 5 Conclusion and Limitations

Effective subproblem selection is central to large-scale routing decomposition, yet prior selection experience often remains difficult to reuse. ViSO distills solver feedback into reusable visual skills and combines them with hypernode reduction and MDS-based rendering for large-scale routing optimization. Across more than 200 instances spanning four routing families with up to 85,900 nodes, ViSO improves aggregate solution quality over LKH-3 on TSP, HGS on CVRP, and PyVRP on ACVRP, while outperforming the evaluated neural and LLM-based baselines. Ablations show that skill guidance improves held-out performance across all four families. Applying the distillation procedure to DRAGON's own trajectories also improves performance without modifying its core solver. These findings support the reuse of decomposition experience as a practical approach to improving routing optimization without model parameter updates.

**Limitations.** ViSO relies on a hosted VLM, incurring API costs. Per-instance token usage and runtime, and one-time distillation costs, are reported in Appendix D and Appendix H. Empirical validation is limited to four routing families; reductions for six additional VRP variants remain untested (Appendix A). Additionally, the skill bank also remains fixed during inference. Future work will explore distilling the selector into a lightweight local VLM and updating skills online.

## REPRODUCIBILITY STATEMENT

The appendices provide implementation details, prompt templates, experimental configurations, and per-instance results. Outputs from the hosted VLM may vary across runs. We will release the code upon acceptance.

### AI USE STATEMENT

In this work, we used generative AI tools only to polish the manuscript for grammar, clarity, and readability and to assist with writing software code. We did not use generative AI tools to generate synthetic datasets; develop theoretical models or conceptual frameworks; formulate mathematical claims; provide key arguments for proofs or assist in writing proofs; propose or refine research hypotheses; design or provide feedback on research methodology or experiments; translate text; clean or reformat datasets; conduct qualitative or thematic data analysis; or interpret results. The authors reviewed all AI-assisted text and code and tested the code for correctness. We take full responsibility for the final content of this work, including the manuscript, code, claims, and results.

## REFERENCES

Luca Accorsi and Daniele Vigo. Routing one million customers in a handful of minutes. *Computers & Operations Research*, 164:106562, 2024.

Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, 54(1):99–114, 2006.

David L. Applegate. *The traveling salesman problem: a computational study*, volume 17. Princeton university press, 2006. URL https://books.google.com/books?hl=en&lr=&id=vhsJbqomDuIC&oi=fnd&pg=PP11&dq=The+traveling+salesman+problem+:+a+computational+study&ots=YLCJVzMU78&sig=-6F9EbUUIJl46yILKTkg6CxfIHg.

Shengkai Chen, Zhiguang Cao, Jianan Zhou, Yaoxin Wu, Senthilnath Jayavelu, Zhuoyi Lin, Xiaoli Li, and Shili Xiang. Dragon: Llm-driven decomposition and reconstruction agents for large-scale combinatorial optimization. *arXiv preprint arXiv:2601.06502*, 2026.

Hanni Cheng, Haosi Zheng, Ya Cong, Weihao Jiang, and Shiliang Pu. Select and optimize: Learning to solve large-scale tsp instances. In *International conference on artificial intelligence and statistics*, pp. 1219–1231. PMLR, 2023.

Jan Christiaens and Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54(2):417–433, 2020.

Jill Cirasella, David S Johnson, Lyle A McGeoch, and Weixiong Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Workshop on algorithm engineering and experimentation*, pp. 32–59. Springer, 2001.

Mohammed Elhenawy, Ahmad Abutahoun, Taqwa I Alhadidi, Ahmed Jaber, Huthaifa I Ashqar, Shadi Jaradat, Ahmed Abdelhay, Sebastien Glaser, and Andry Rakotonirainy. Visual reasoning and multi-agent approach in multimodal large language models (mllms): Solving tsp and mtsp combinatorial challenges. *Machine Learning and Knowledge Extraction*, 6(3):1894–1920, 2024.

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. Invit: A generalizable routing problem solver with invariant nested view transformer. *arXiv preprint arXiv:2402.02317*, 2024.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7474–7482, 2021.

Vincent Furnon and Laurent Perron. Or-tools routing library, 2024. URL https://developers.google.com/optimization/routing/.

Chengrui Gao, Haopu Shang, Ke Xue, and Chao Qian. Neural solver selection for combinatorial optimization. *arXiv preprint arXiv:2410.09693*, 2024.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12(966-980):1, 2017.

Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in neural information processing systems*, 33:21188–21198, 2020.

Eduardo Lalla-Ruiz and Martijn Mes. Mathematical formulations and improvements for the multi-depot open vehicle routing problem: E. lalla-ruiz, m. mes. *Optimization Letters*, 15(1):271–286, 2021.

Ke Li, Fei Liu, Zhenkun Wang, and Qingfu Zhang. Destroy and repair using hyper-graphs for routing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18341–18349, 2025.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in neural information processing systems*, 34:26198–26211, 2021.

Xiayang Li and Shihua Zhang. Learning-based tsp-solvers tend to be overly greedy. *Computers & Operations Research*, pp. 107467, 2026.

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.

Fei Liu, Yilu Liu, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37090–37098, 2026.

Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36:8845–8864, 2023.

Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *International Conference on Learning Representations*, volume 2025, pp. 14754–14780, 2025a.

Fu Luo, Xi Lin, Mengyuan Zhong, Fei Liu, Zhenkun Wang, Jianyong Sun, and Qingfu Zhang. Learning to insert for constructive neural vehicle routing solver (2025). *arXiv preprint arXiv:2505.13904*, 2025b.

Dian Meng, Zhiguang Cao, Yaoxin Wu, Yaqing Hou, Hongwei Ge, and Qiang Zhang. Eformer: an effective edge-based transformer for vehicle routing problems. *arXiv preprint arXiv:2506.16428*, 2025.

Dian Meng, Zhiguang Cao, Jie Gao, Yaoxin Wu, and Yaqing Hou. Uniteformer: unifying node and edge modalities in transformers for vehicle routing problems. *Advances in Neural Information Processing Systems*, 38:93830–93854, 2026.

Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Learning to repeatedly solve routing problems. *Networks*, 83(3):503–526, 2024.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Wenbin Ouyang, Sirui Li, Yining Ma, and Cathy Wu. Learning to segment for vehicle routing problems. In *International Conference on Learning Representations*, volume 2026, pp. 149846–149882, 2026.

Wenzheng Pan, Hao Xiong, Jiale Ma, Wentao Zhao, Yang Li, and Junchi Yan. Unico: On unified combinatorial optimization via problem reduction to matrix-encoded general tsp. In *International Conference on Learning Representations*, volume 2025, pp. 23403–23443, 2025.

Xuanhao Pan, Yan Jin, Yuandong Ding, Mingxiao Feng, Li Zhao, Lei Song, and Jiang Bian. H-tsp: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9345–9353, 2023.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523, 2020.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 35:25531–25546, 2022.

Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4): 376–384, 1991.

Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

Alberto Santini, Michael Schneider, Thibaut Vidal, and Daniele Vigo. Decomposition strategies for vehicle routing heuristics. *INFORMS Journal on Computing*, 35(3):543–559, 2023.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pp. 417–431. Springer, 1998.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Jiwoo Son, Zhikai Zhao, Federico Berto, Chuanbo Hua, Zhiguang Cao, Changhyun Kwon, and Jinkyoo Park. Rrnco: Towards real-world routing with neural combinatorial optimization. In *International Conference on Learning Representations*, volume 2026, pp. 91794–91818, 2026.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

Warren S Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

Cong Dao Tran, Quan Nguyen-Tri, Huynh Thi Thanh Binh, and Hoang Thanh-Tung. Large language models powered neural solvers for generalized vehicle routing problems. In *Towards agentic AI for science: hypothesis generation, comprehension, quantification, and validation*, 2025.

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

Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Niels A Wouda, Leon Lan, and Wouter Kool. Pyvrp: A high-performance vrp solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34:7472–7483, 2021.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. Deepaco: Neural-enhanced ant systems for combinatorial optimization. *Advances in neural information processing systems*, 36: 43706–43728, 2023.

Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 20284–20292, 2024.

Hang Yi, Ziwei Huang, Yining Ma, and Zhiguang Cao. Radar: Learning to route with asymmetry-aware distance representations. *arXiv preprint arXiv:2603.03388*, 2026.

Zhuoli Yin, Yi Ding, Reem Khir, and Hua Cai. Vitsp: A vision language models guided framework for solving large-scale traveling salesman problems. In *International Conference on Learning Representations*, volume 2026, pp. 132572–132599, 2026.

Ni Zhang, Zhiguang Cao, Jianan Zhou, Cong Zhang, and Yew-Soon Ong. An agentic framework with llms for solving complex vehicle routing problems. In *International Conference on Learning Representations*, volume 2026, pp. 114166–114206, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Zhi Zheng, Changliang Zhou, Xialiang Tong, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 37:6081–6125, 2024.

Changliang Zhou, Canhong Yu, Shunyu Yao, Xi Lin, Zhenkun Wang, Yu Zhou, and Qingfu Zhang. Urs: A unified neural routing solver for cross-problem zero-shot generalization. *arXiv preprint arXiv:2509.23413*, 2025.

Changliang Zhou, Xi Lin, Zhenkun Wang, and Qingfu Zhang. Learning to reduce search space for generalizable neural routing solver. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6841–6852, 2026.

Jianghan Zhu, Yaoxin Wu, Zhuoyi Lin, Zhengyuan Zhang, Haiyan Yin, Zhiguang Cao, Senthilnath Jayavelu, and Xiaoli Li. Bridging synthetic and real routing problems via llm-guided instance generation and progressive adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 36591–36599, 2026.

Zefang Zong, Hansen Wang, Jingwei Wang, Meng Zheng, and Yong Li. Rbg: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4648–4658, 2022.

# A PER-VARIANT HYPERNODE FORMULATIONS

Table 3: Variant-specific hypernode attributes and their solver constraints. Empirically evaluated problems are highlighted in **bold**.

| Variant | Hypernode carries | Solver constraint |
|---|---|---|
| **TSP** | (none) | exact tour (Concorde) |
| **ATSP** | (none) | exact tour (Concorde) |
| **CVRP** | segment demand | vehicle capacity (HGS) |
| **ACVRP** | segment demand | capacity, asymmetric $D$ (PyVRP) |
| CVRPTW | demand, effective TW, virtual service | capacity, time windows (PyVRP) |
| MDCVRP | demand, internal length | depot fleet, optional duration cap (PyVRP) |
| OVRP | demand | open route, zero return edge (PyVRP) |
| VRPB | demand, delivery/pickup type | capacity, backhaul order check (PyVRP) |
| VRPL | demand, segment length | route-length / duration cap (PyVRP) |
| VRPMB | demand, homogeneous type | prefix-load capacity (PyVRP) |

Table 3 summarizes the attributes stored by each hypernode and the constraints enforced by its solver. The four families evaluated in our experiments are shown in bold. The remaining rows describe extensions of the same construction.

**Routing formulations.** For TSP and ATSP, a solution is a cyclic permutation $\Pi = (\pi_1, \ldots, \pi_n)$ of all $n$ nodes, with cost

$$c(\Pi) = \sum_{i=1}^{n-1} D_{\pi_i, \pi_{i+1}} + D_{\pi_n, \pi_1}. \tag{11}$$

For the single-depot capacitated variants, a solution is a set of routes $\mathcal{R} = \{R^1, \ldots, R^m\}$, each starting and ending at the depot, that visit every customer exactly once. With customer demands $q_i > 0$ and vehicle capacity $Q_{\text{cap}}$, the objective is

$$\min_{\mathcal{R}} \; c(\mathcal{R}) = \sum_{R \in \mathcal{R}} c(R) \qquad \text{s.t.} \qquad \sum_{i \in R} q_i \le Q_{\text{cap}} \quad \text{for every route } R \in \mathcal{R}, \tag{12}$$

where demands are summed over customers. Additional variants modify depot, return, or resource constraints.

The formulations below build on the fixed-segment aggregation framework and feasibility arguments of Ouyang et al. (2026). We give the resource encodings used here; in particular, the time-window encoding uses one virtual client even when internal waiting is unavoidable. The claims concern fixed-order segments under each variant's stated assumptions, not equivalence to the unrestricted routing problem.

The variants differ in segment construction, carried resources, and subproblem constraints, while sharing the selection, visualization, and orchestration framework. Throughout, $\kappa = (\kappa_1, \ldots, \kappa_p)$ denotes a fixed-order directed segment contracted into hypernode $h$. Its entry and exit nodes are $\kappa_1$ and $\kappa_p$, and its internal cost is $c(h) = \sum_{j=1}^{p-1} D_{\kappa_j, \kappa_{j+1}}$. For VRPB and VRPMB, fixed runs are split at changes in customer type. Only edges within each resulting homogeneous segment remain frozen.

**Reduced cost matrix.** Ordering the elements of $\omega$ as $H$ followed by $W$, Eq. (8) has the block expansion

$$\tilde{D} = \begin{pmatrix} \tilde{D}^{HH} & \tilde{D}^{HW} \\ \tilde{D}^{WH} & \tilde{D}^{WW} \end{pmatrix},$$
$$\tilde{D}^{HH}_{kl} = D_{\text{exit}(h_k), \text{entry}(h_l)}, \quad \tilde{D}^{HW}_{kj} = D_{\text{exit}(h_k), w_j}, \tag{13}$$
$$\tilde{D}^{WH}_{ik} = D_{w_i, \text{entry}(h_k)}, \qquad \tilde{D}^{WW}_{ij} = D_{w_i, w_j}.$$

The $HH$ and $WW$ formulas apply off the diagonal; $\tilde{D}^{HH}_{kk} = \tilde{D}^{WW}_{ii} = 0$. Depot rows and columns are appended using the same endpoint rule; MDCVRP retains one node per involved depot.

**Cost and fleet accounting.** Let $r$ be a reduced customer route, and let $E(r)$ contain its connecting edges, including departure from and return to the depot. With $c(w) = 0$ for free nodes and zero internal cost at depots, the expanded route distance is

$$C(r) = \sum_{(u,v)\in E(r)} \tilde{D}_{uv} + \sum_{h\in r\cap H} c(h). \tag{14}$$

For OVRP the return-edge cost is zero. For a fixed subproblem, internal costs are constant. They may be omitted from the reduced distance objective, but must be restored when evaluating expanded solutions and route-length constraints. For the six additional VRP variants, vehicles assigned to unaffected nonempty routes are reserved. With a finite fleet $K$, the subproblem may therefore use at most $K_{\text{rem}} = K - |\mathcal{R}_{\text{unaff}}|$ vehicles. MDCVRP applies this accounting separately at each depot. A nonempty subproblem with no remaining vehicle is rejected. Expanded solutions are checked for exact customer coverage, capacity, fleet limits, and the variant-specific constraints before acceptance. OVRP, VRPL, and MDCVRP use nonnegative integer-valued distance matrices; fractional external matrices are rejected rather than silently truncated.

**CVRP and ACVRP.** The hypernode aggregates demand $q(h) = \sum_{i\in\kappa} q_i$, and a reconstructed route $r$ satisfies the capacity constraint iff $\sum_{u\in r} q(u) \le Q_{\text{cap}}$. The reduced instance is $\{h_0\} \cup W \cup H$, where $h_0$ is the zero-demand depot node defined in Sec. 3.5. Every reconstructed route begins and ends at the depot. A run adjacent to the depot in the current solution is contracted like any other segment. Its depot attachment remains free and is re-optimized by the sub-solver. CVRP and ACVRP pass $\tilde{D}$ directly to HGS and PyVRP, respectively.

**CVRPTW.** Write $[e_j, \ell_j]$ and $s_j$ for the service-start window and service time at $\kappa_j$, and $t_j = D_{\kappa_j,\kappa_{j+1}}$ for travel to the next node ($1 \le j < p$). Assume a fixed segment order, static travel times, and permitted waiting. Define

$$\begin{aligned} A_1 &= 0, & A_j &= \sum_{i=1}^{j-1}(s_i + t_i), \\ B_j &= \max_{1\le i\le j}(e_i - A_i), & U &= \min_{1\le j\le p}(\ell_j - A_j), \\ B &= B_p, & S &= A_p + s_p. \end{aligned} \tag{15}$$

Forward propagation gives the service start time $A_j + \max(a, B_j)$ at node $\kappa_j$ when the segment entry arrival is $a$. If $A_j + B_j > \ell_j$ for any prefix $j$, the segment is infeasible. Otherwise, arrival $a$ at its entry is feasible iff $a \le U$, and the earliest departure is $F_h(a) = \max(a, B) + S$. A single PyVRP client reproduces this behavior with

$$[e(h), \ell(h)] = [\min(B, U), U], \qquad s(h) = S + \max(0, B - U). \tag{16}$$

This formulation also covers $B > U$, when internal waiting is unavoidable. The virtual service-start time need not equal the service start time at $\kappa_1$. To verify the encoding, if $B \le U$, the virtual client departs at $\max(a, B) + S$. If $B > U$, every feasible arrival satisfies $a \le U$, so it departs at $U + [S + (B - U)] = B + S = F_h(a)$. In both cases its arrival is feasible iff $a \le U$, provided the prefix feasibility conditions above hold. Demand is aggregated as for CVRP. The implementation sets travel duration equal to $D$ and uses a distance-only objective. Before contraction, each primitive input is scaled by $10^3$: travel times are rounded, and window bounds and service times are truncated to integers. The encoding is exact for this integer model; after expansion, the original floating-point schedule is checked again.

**MDCVRP.** Let $\mathcal{D}_\omega$ contain the depots serving the affected incumbent routes. For each $d \in \mathcal{D}_\omega$, the reduced instance retains one depot node and, when vehicles remain, one vehicle type with that depot as both its start and end location. If its fleet limit is $K_d$ and unaffected routes use $U_d$ vehicles, at most $K_d - U_d$ new routes may use that depot. Free nodes and hypernodes may be reassigned among these involved depots; a hypernode's source-depot label is metadata, not a fixed assignment. Depots outside $\mathcal{D}_\omega$ are not searched. The implementation supports uniform vehicle capacity, zero customer service time, and an optional uniform route-duration cap. With travel duration equal to distance, hypernode service duration is $c(h)$, so the cap applies to the full $C(r)$ in Eq. (14), including internal edges. Inputs with heterogeneous depot limits or nonzero customer service times are not supported.

**OVRP.** Routes need not return to the depot. We model this by setting the return-edge cost to zero, $\tilde{D}_{u,\text{depot}} \leftarrow 0$, in the reconstructed subproblem. This is the standard zero-return-cost representation of open routes (Lalla-Ruiz & Mes, 2021).

**VRPB.** Each customer is either a linehaul (delivery) or a backhaul (pickup). Frozen runs are split at type changes, and each homogeneous hypernode aggregates its type's demand. Every route must visit all linehauls before any backhaul; either class may be empty. For $m = |W| + |H|$ reduced clients and maximum unpenalized connection cost $d_{\max}$, backhaul-to-linehaul arcs receive cost $M = 2md_{\max} + 1$. With nonnegative integer distances, any complete order-respecting solution uses at most $2m$ connecting edges and costs at most $2md_{\max}$, excluding constant internal costs. Thus a prohibited arc is more expensive than every such solution. Because the subsolver is heuristic, the implementation explicitly rejects any returned ordering violation and also enforces capacity. Type-change boundary edges may be reconnected.

**VRPL.** Each hypernode carries its internal length $c(h)$. The route-length constraint is $C(r) \le L$, with $C(r)$ from Eq. (14) and $L$ the route limit. Connecting-edge duration equals $\tilde{D}_{uv}$, hypernode service duration equals $c(h)$, and free-node service duration is zero, so PyVRP's route-duration cap accounts for the complete expanded length, including depot edges.

**VRPMB.** Each customer is either a delivery or a pickup, but the two types may interleave along a route. Let $q^+(u)$ and $q^-(u)$ denote delivery and pickup demand. Fixed runs are split at every type change, so every reduced client is homogeneous: $q^+(u)q^-(u) = 0$, with demand summed within its segment. For a reduced route $r = (u_1, \dots, u_k)$, the vehicle departs with all deliveries and must satisfy

$$\begin{aligned} L_0 &= \sum_{j=1}^{k} q^+(u_j), \quad L_j = L_{j-1} - q^+(u_j) + q^-(u_j), \quad j = 1, \dots, k, \\ 0 &\le L_j \le Q_{\text{cap}}, \qquad j = 0, \dots, k. \end{aligned} \tag{17}$$

Load is monotone within each homogeneous hypernode, so checking its endpoints also checks every internal node. Checking delivery and pickup totals separately is insufficient. The additional cuts release cross-type boundary edges for reconnection, enlarging the neighborhood relative to an unsplit frozen run; internal edges and order within each resulting segment remain fixed. Expanded routes are checked at every original customer.

## B MDS VISUALIZATION DETAILS

Given an asymmetric cost matrix $D$, we compute the rendering coordinates from Sec. 3.2 using classical multidimensional scaling (Torgerson, 1952), augmented with two robustness steps. We first symmetrize, $\bar{D} = \frac{1}{2}(D + D^\top)$, and cap every entry at the 95th percentile of the off-diagonal values, $\bar{D}_{ij} \leftarrow \min\{\bar{D}_{ij},\, P_{95}(\bar{D})\}$, so that a few very large costs (e.g. pairs connected only through long detours) do not compress the remaining points into a single cluster. From the capped matrix we form the squared-cost matrix $\bar{D}^{(2)}$ with entries $\bar{D}^2_{ij}$ and double-center it,

$$B = -\tfrac{1}{2}\,\mathbf{J}\,\bar{D}^{(2)}\,\mathbf{J}, \qquad \mathbf{J} = \mathbf{I} - \tfrac{1}{n}\mathbf{1}\mathbf{1}^\top. \tag{18}$$

The eigendecomposition $B = U\Lambda U^\top$ yields the planar embedding $X = U_2[\Lambda_2]_+^{1/2} \in \mathbb{R}^{n\times 2}$ from the two largest eigenvalues, with negative eigenvalues clipped to zero. If the computation fails, produces non-finite coordinates, or produces coordinates that are numerically all zero, we use a circular layout ordered by node index. The coordinates are rescaled to a fixed drawing extent. The embedding is computed once per instance, concurrently with the warm start, before optimization begins. As stated in the main text, the embedding is used only to render the instance and to map a selected box back to the nodes it encloses; all cost evaluations, feasibility checks, and reconstructions use the true asymmetric $D$.

## C BENCHMARK COMPOSITION AND TIME ALIGNMENT

**Benchmarks.** *TSP*: 33 TSPLIB instances, $n = 1{,}000$–$85{,}900$, assigned by size to per-instance optimization budgets of 100s (16 instances), 200s (4), 500s (6), 1,000s (5), and 3,000s (2). *ATSP*: 40 self-generated real-city instances at $n \in \{1000\,(\times 19),\ 2000\,(\times 5),\ 5000\,(\times 9),\ 10000\,(\times 7)\}$. *CVRP*: the full CVRPLIB X-set, split into X<500 (68 instances) and X>500 (32). *ACVRP*: 40 self-generated real-city instances. For skill distillation, each family and size tier is partitioned into training and held-out sets using a roughly 70/30 split. Only training trajectories are used to construct the corresponding skill bank. ViSO results are the mean over three independent runs throughout. The TSP no-skill ablation uses a fixed set of three runs per instance. Gaps are averaged within each instance and then across instances; relative gains are computed before rounding.

**Initialization and comparison configuration.** TSP and ATSP initialize with ten LKH-3 runs, CVRP with 30 s of HGS, and ACVRP with 30 s of PyVRP. The optimization budgets in Table 4 are applied after initialization. All ViSO model calls use gpt-5.4 (Appendix H). For the ViTSP comparison, both methods use $P{=}4$ parallel VLM producers, matching their selector parallelism.

**Time alignment.** The nominal per-instance wall-clock target is $T_{\text{matched}} = T_{\text{init}} + T_{\text{budget}}$, the warm-start time plus the tier budget above; Table 4 summarizes the policy for every family and tier. Except where noted below, we align each baseline to $T_{\text{matched}}$ according to how it uses computation time. Solvers with a native time limit (Concorde, OR-Tools, LKH-3, HGS, PyVRP) receive it as their limit; iterative neural methods (GLOP revisions, L2C-Insert and NCO-LLM refinement, DRHG repair) are given a per-instance wall-clock cap equal to it; methods without a controllable wall-clock parameter use fixed configurations rather than matched time limits. This applies to the fixed-configuration L2R evaluations, the fixed-decode URS model, and the fixed-decode neural baselines used for the asymmetric families. On CVRP, ViSO receives a 100s (X<500) or 200s (X>500) optimization budget on top of the HGS warm start, and the HGS-only reference receives the same nominal total allowance (130s / 230s), with results averaged over three seeds per instance. On ACVRP, the PyVRP reference is run for the same total wall-clock as the corresponding ViSO run.

**Coverage and decoding of asymmetric neural baselines.** The released RADAR and UniCO ATSP checkpoints were trained at $n = 100$ and support only the $n = 1{,}000$ tier of our benchmark. We therefore evaluate them zero-shot on the 19 instances at that scale; the $n = 2{,}000$, $5{,}000$, and $10{,}000$ instances are unsupported by the released artifacts.

RADAR, UniCO, and RRNCO use fixed decoding procedures without a controllable wall-clock limit and therefore cannot be strictly time-matched. We use their strongest available decoding configurations. RADAR uses best-of-$K$ seeds, and RRNCO uses POMO decoding with 8-fold augmentation. These fixed-decode runtimes may be shorter or longer than the matched budget. For reference, greedy decoding produces ACVRP gaps of $+134.2\%$ for RADAR and $+112.0\%$ for RRNCO.

**Solver-specific exceptions.** On CVRP, LKH-3 fails to return feasible solutions on two instances (one per size tier), and OR-Tools on seven large-tier instances; their aggregate gaps exclude those failures. For URS, we report the stronger CVRP-specialist checkpoint, which fails on one instance with a CUDA out-of-memory error.

Table 4: Per-instance optimization budget in seconds, applied on top of initialization (TSP/ATSP: ten LKH-3 runs; CVRP: 30 s of HGS; ACVRP: 30 s of PyVRP).

| Family | Instance size $n$ | #Inst. | Budget (s) |
|---|---|---|---|
| TSP | 1,000–1,889 | 16 | 100 |
| | 2,103–2,392 | 4 | 200 |
| | 3,038–7,397 | 6 | 500 |
| | 11,849–18,512 | 5 | 1,000 |
| | 33,810–85,900 | 2 | 3,000 |
| ATSP | 1,000 | 19 | 100 |
| | 2,000 | 5 | 200 |
| | 5,000 | 9 | 500 |
| | 10,000 | 7 | 1,000 |
| CVRP | 101–491 | 68 | 100 |
| | 502–1,001 | 32 | 200 |
| ACVRP | 500 / 1,000 | 40 | 200 |

## D PER-INSTANCE RESULTS

Tables 5–8 report results for every instance, averaged over three runs/seeds. ViSO gaps are measured against published optima for TSP, LKH-3 for ATSP, best-known solutions for CVRP, and the PyVRP reference run with the same runtime budget for ACVRP. The tables also report the share of VLM-selected subproblems whose re-optimization improved the incumbent (Impr.), per-run token usage in millions (input+output), and total wall-clock time in seconds, including initialization. Held-out test instances are marked with †.

Table 5: Per-instance TSP results (mean of 3 runs): ViSO gap (%) vs. the published optimum, share of VLM-selected subproblems whose re-optimization improved the incumbent (Impr., %), per-run token usage (millions, input+output), and total wall-clock including initialization (s). Test-split instances marked †.

| Instance | Gap | Impr. | Tok(M) | Time | Instance | Gap | Impr. | Tok(M) | Time |
|---|---|---|---|---|---|---|---|---|---|
| dsj1000 | 0.003 | 6.7 | 0.15 | 102 | u2152 | 0.136 | 16.5 | 0.32 | 211 |
| pr1002 | 0.047 | 15.0 | 0.09 | 101 | u2319† | 0.061 | 5.7 | 0.31 | 216 |
| u1060† | 0.021 | 7.3 | 0.17 | 106 | pr2392 | 0.258 | 20.8 | 0.27 | 212 |
| vm1084 | 0.048 | 6.2 | 0.11 | 101 | pcb3038 | 0.063 | 13.5 | 1.23 | 508 |
| pcb1173 | 0.306 | 23.9 | 0.10 | 103 | fl3795 | 0.417 | 2.8 | 1.27 | 530 |
| d1291† | 0.409 | 18.9 | 0.09 | 107 | fnl4461† | 0.045 | 17.5 | 1.16 | 546 |
| rl1304 | 0.138 | 4.5 | 0.12 | 102 | rl5915 | 1.465 | 22.3 | 0.57 | 565 |
| rl1323† | 0.111 | 11.1 | 0.11 | 101 | rl5934 | 0.626 | 22.4 | 0.77 | 558 |
| nrw1379 | 0.004 | 22.3 | 0.12 | 106 | pla7397† | 0.334 | 15.5 | 0.99 | 602 |
| fl1400† | 0.184 | 2.1 | 0.13 | 107 | rl11849 | 0.342 | 43.4 | 1.84 | 1,340 |
| u1432 | 0.023 | 13.5 | 0.12 | 106 | usa13509† | 0.329 | 40.2 | 2.04 | 1,406 |
| fl1577 | 0.075 | 7.8 | 0.11 | 110 | brd14051 | 0.330 | 49.2 | 1.64 | 1,441 |
| d1655 | 0.319 | 20.9 | 0.09 | 107 | d15112 | 0.274 | 61.7 | 1.57 | 1,609 |
| vm1748 | 0.119 | 23.1 | 0.19 | 106 | d18512† | 0.512 | 54.6 | 1.53 | 1,799 |
| u1817 | 0.382 | 31.4 | 0.09 | 107 | pla33810 | 0.593 | 45.1 | 5.39 | 4,936 |
| rl1889† | 0.019 | 9.1 | 0.13 | 108 | pla85900 | 1.091 | 55.9 | 1.48 | 24,327 |
| d2103 | 0.464 | 12.2 | 0.20 | 186 | | | | | |

Table 6: Per-instance ATSP results (mean of 3 runs): ViSO gap (%) vs. LKH-3 (negative = ViSO beats LKH-3), share of VLM-selected subproblems whose re-optimization improved the incumbent (Impr., %), per-run token usage (millions), and total wall-clock including initialization (s). City names abbreviated from the instance id. Held-out test cities marked †.

| City | $n$ | Gap | Impr. | Tok(M) | Time | City | $n$ | Gap | Impr. | Tok(M) | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Addis_Ababa | 1,000 | +0.431 | 42.4 | 0.24 | 120 | Cusco_Peru | 2,000 | -0.084 | 26.7 | 0.38 | 254 |
| Asuncion_Paraguay† | 1,000 | +0.647 | 0.0 | 0.22 | 112 | Kyoto | 2,000 | -0.004 | 24.0 | 0.46 | 250 |
| Athens† | 1,000 | +0.080 | 19.6 | 0.23 | 113 | Melbourne† | 2,000 | +0.189 | 31.9 | 0.45 | 257 |
| Bangkok | 1,000 | +0.148 | 14.9 | 0.26 | 118 | Reykjavik_Iceland | 2,000 | +0.635 | 5.1 | 0.42 | 247 |
| Chicago | 1,000 | -0.228 | 17.5 | 0.25 | 120 | Barcelona | 5,000 | +0.975 | 0.6 | 1.49 | 734 |
| Fez_Morocco† | 1,000 | +0.069 | 22.2 | 0.21 | 116 | Buenos_Aires | 5,000 | +0.374 | 4.4 | 1.37 | 600 |
| Hanoi | 1,000 | +0.212 | 10.6 | 0.22 | 116 | Cape_Town | 5,000 | +0.496 | 6.4 | 1.49 | 768 |
| Hobart_Australia | 1,000 | +0.186 | 14.9 | 0.25 | 119 | Colombo_Sri_Lanka | 5,000 | +0.162 | 27.4 | 1.27 | 758 |

| City | $n$ | Gap | Impr. | Tok(M) | Time | City | $n$ | Gap | Impr. | Tok(M) | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Lagos | 1,000 | +0.072 | 21.8 | 0.23 | 119 | Jakarta† | 5,000 | -0.553 | 17.7 | 1.75 | 756 |
| London | 1,000 | +0.028 | 19.4 | 0.28 | 118 | Marrakech_Morocco† | 5,000 | -0.335 | 31.2 | 1.49 | 775 |
| Nairobi | 1,000 | +0.467 | 14.4 | 0.22 | 116 | Mexico_City | 5,000 | -0.126 | 23.4 | 1.36 | 752 |
| Osaka | 1,000 | +0.488 | 31.1 | 0.25 | 122 | Rome | 5,000 | +0.664 | 2.4 | 1.54 | 744 |
| Paris | 1,000 | +0.095 | 12.4 | 0.26 | 114 | Sao_Paulo† | 5,000 | +0.409 | 8.5 | 1.49 | 740 |
| San_Francisco | 1,000 | +0.330 | 30.0 | 0.23 | 116 | Bukhara_Uzbekistan† | 10,000 | +0.682 | 18.9 | 2.33 | 2,030 |
| San_Juan_Puerto_Rico† | 1,000 | +0.120 | 9.1 | 0.23 | 121 | Cairo | 10,000 | -1.178 | 33.3 | 3.12 | 2,056 |
| Santiago | 1,000 | +0.296 | 21.8 | 0.23 | 113 | Mumbai | 10,000 | -0.551 | 19.4 | 2.85 | 2,228 |
| Seoul | 1,000 | +0.263 | 36.5 | 0.22 | 119 | New_York_City | 10,000 | +1.561 | 3.2 | 2.37 | 3,249 |
| Stockholm† | 1,000 | +0.621 | 9.1 | 0.24 | 115 | Quebec_City† | 10,000 | -0.052 | 20.4 | 2.96 | 2,085 |
| Taipei† | 1,000 | +0.071 | 27.4 | 0.25 | 116 | Sydney | 10,000 | +0.665 | 10.8 | 2.99 | 2,085 |
| Amsterdam | 2,000 | +0.028 | 17.5 | 0.43 | 245 | Tokyo | 10,000 | -0.057 | 22.4 | 2.53 | 2,064 |

Table 7: Per-instance CVRP results (mean of 3 seeds), full CVRPLIB X-set: ViSO gap (%) vs. best-known solutions, share of VLM-selected subproblems whose re-optimization improved the incumbent (Impr., %), per-run token usage (millions, input+output), and total wall-clock including initialization (s). BKS-based termination is disabled in the reported skill-guided results. Impr. pools improved and completed VLM subproblem counts across the three seeds; token usage and time are per-run means. Held-out test instances are marked †.

| Instance | Gap | Impr. | Tok(M) | Time | Instance | Gap | Impr. | Tok(M) | Time |
|---|---|---|---|---|---|---|---|---|---|
| X-n101-k25 | 0.000 | 0.0 | 0.06 | 130 | X-n336-k84 | 0.411 | 24.5 | 0.10 | 131 |
| X-n106-k14† | 0.000 | 4.3 | 0.11 | 130 | X-n344-k43 | 0.212 | 5.1 | 0.12 | 130 |
| X-n110-k13† | 0.000 | 0.0 | 0.06 | 131 | X-n351-k40 | 0.214 | 24.4 | 0.09 | 130 |
| X-n115-k10† | 0.000 | 0.0 | 0.06 | 130 | X-n359-k29 | 0.404 | 21.2 | 0.10 | 130 |
| X-n120-k6 | 0.000 | 0.0 | 0.06 | 130 | X-n367-k17 | 0.000 | 5.1 | 0.10 | 130 |
| X-n125-k30† | 0.000 | 1.8 | 0.05 | 130 | X-n376-k94† | 0.007 | 8.5 | 0.10 | 130 |
| X-n129-k18† | 0.000 | 0.0 | 0.06 | 131 | X-n384-k52 | 0.457 | 16.8 | 0.11 | 130 |
| X-n134-k13 | 0.000 | 0.0 | 0.06 | 130 | X-n393-k38† | 0.071 | 7.6 | 0.10 | 130 |
| X-n139-k10† | 0.000 | 0.0 | 0.06 | 130 | X-n401-k29 | 0.292 | 19.0 | 0.11 | 130 |
| X-n143-k7 | 0.000 | 0.0 | 0.06 | 130 | X-n411-k19 | 0.068 | 11.9 | 0.10 | 130 |
| X-n148-k46 | 0.000 | 0.0 | 0.06 | 131 | X-n420-k130 | 0.063 | 13.2 | 0.12 | 130 |
| X-n153-k22 | 0.024 | 0.0 | 0.13 | 130 | X-n429-k61† | 0.192 | 20.3 | 0.09 | 130 |
| X-n157-k13 | 0.000 | 0.0 | 0.06 | 131 | X-n439-k37 | 0.056 | 8.0 | 0.11 | 130 |
| X-n162-k11† | 0.000 | 0.0 | 0.06 | 130 | X-n449-k29 | 0.861 | 25.5 | 0.08 | 130 |
| X-n167-k10† | 0.000 | 0.0 | 0.06 | 131 | X-n459-k26 | 0.249 | 18.9 | 0.10 | 130 |
| X-n172-k51 | 0.000 | 0.0 | 0.06 | 130 | X-n469-k138 | 0.397 | 21.8 | 0.08 | 130 |
| X-n176-k26 | 0.000 | 2.5 | 0.09 | 130 | X-n480-k70 | 0.240 | 31.6 | 0.10 | 130 |
| X-n181-k23 | 0.021 | 0.0 | 0.09 | 131 | X-n491-k59† | 0.386 | 26.4 | 0.09 | 130 |
| X-n186-k15 | 0.000 | 0.0 | 0.06 | 130 | X-n502-k39† | 0.144 | 16.7 | 0.24 | 230 |
| X-n190-k8 | 0.135 | 14.8 | 0.11 | 130 | X-n513-k21 | 0.000 | 3.1 | 0.28 | 230 |
| X-n195-k51 | 0.023 | 0.0 | 0.07 | 130 | X-n524-k153 | 0.228 | 11.5 | 0.29 | 231 |
| X-n200-k36 | 0.026 | 1.6 | 0.07 | 131 | X-n536-k96† | 0.481 | 17.0 | 0.23 | 230 |
| X-n204-k19 | 0.000 | 1.2 | 0.12 | 131 | X-n548-k50† | 0.174 | 17.4 | 0.34 | 230 |
| X-n209-k16 | 0.049 | 3.2 | 0.12 | 130 | X-n561-k42 | 0.075 | 8.8 | 0.37 | 230 |
| X-n214-k11 | 0.123 | 4.1 | 0.11 | 130 | X-n573-k30 | 0.603 | 21.5 | 0.22 | 230 |
| X-n219-k73 | 0.006 | 0.0 | 0.09 | 131 | X-n586-k159† | 0.261 | 19.4 | 0.35 | 231 |
| X-n223-k34 | 0.131 | 2.2 | 0.11 | 131 | X-n599-k92† | 0.295 | 20.1 | 0.39 | 231 |
| X-n228-k23† | 0.063 | 0.6 | 0.11 | 130 | X-n613-k62 | 0.412 | 16.0 | 0.40 | 231 |
| X-n233-k16† | 0.054 | 0.0 | 0.08 | 130 | X-n627-k43 | 0.628 | 22.0 | 0.27 | 230 |
| X-n237-k14 | 0.020 | 0.6 | 0.12 | 130 | X-n641-k35 | 0.447 | 24.0 | 0.29 | 230 |
| X-n242-k48 | 0.085 | 6.1 | 0.14 | 130 | X-n655-k131 | 0.064 | 22.7 | 0.15 | 230 |
| X-n247-k50† | 0.124 | 3.3 | 0.11 | 130 | X-n670-k130 | 0.293 | 29.1 | 0.19 | 231 |
| X-n251-k28 | 0.181 | 5.4 | 0.12 | 130 | X-n685-k75 | 0.329 | 17.6 | 0.31 | 230 |
| X-n256-k16 | 0.218 | 0.0 | 0.12 | 130 | X-n701-k44 | 0.676 | 22.0 | 0.39 | 230 |
| X-n261-k13† | 0.216 | 3.6 | 0.13 | 130 | X-n716-k35 | 0.759 | 26.4 | 0.20 | 230 |
| X-n266-k58† | 0.306 | 20.8 | 0.10 | 130 | X-n733-k159 | 0.177 | 22.2 | 0.31 | 230 |
| X-n270-k35 | 0.052 | 1.2 | 0.12 | 130 | X-n749-k98 | 0.678 | 25.2 | 0.25 | 230 |
| X-n275-k28† | 0.031 | 0.0 | 0.08 | 131 | X-n766-k71 | 0.563 | 24.0 | 0.26 | 230 |
| X-n280-k17 | 0.150 | 12.1 | 0.12 | 130 | X-n783-k48† | 0.752 | 22.5 | 0.37 | 230 |
| X-n284-k15 | 0.162 | 6.2 | 0.11 | 130 | X-n801-k40 | 0.392 | 22.6 | 0.30 | 230 |
| X-n289-k60 | 0.333 | 15.1 | 0.10 | 130 | X-n819-k171 | 0.441 | 24.9 | 0.22 | 230 |
| X-n294-k50 | 0.123 | 4.9 | 0.11 | 130 | X-n837-k142† | 0.537 | 30.6 | 0.26 | 230 |
| X-n298-k31 | 0.123 | 1.1 | 0.11 | 130 | X-n856-k95 | 0.213 | 23.4 | 0.23 | 230 |
| X-n303-k21† | 0.090 | 12.7 | 0.11 | 130 | X-n876-k59† | 0.698 | 25.7 | 0.21 | 231 |
| X-n308-k13 | 0.182 | 8.1 | 0.11 | 130 | X-n895-k37 | 0.864 | 23.7 | 0.23 | 231 |
| X-n313-k71 | 0.250 | 19.5 | 0.10 | 130 | X-n916-k207 | 0.333 | 25.9 | 0.20 | 231 |
| X-n317-k53 | 0.009 | 3.5 | 0.10 | 131 | X-n936-k151 | 0.467 | 28.8 | 0.21 | 230 |
| X-n322-k28† | 0.174 | 3.7 | 0.12 | 130 | X-n957-k87† | 0.294 | 23.6 | 0.30 | 231 |
| X-n327-k20 | 0.306 | 10.7 | 0.11 | 130 | X-n979-k58 | 1.158 | 35.0 | 0.19 | 231 |
| X-n331-k15 | 0.031 | 11.1 | 0.12 | 130 | X-n1001-k43† | 1.250 | 24.8 | 0.31 | 231 |

Table 8: Per-instance ACVRP results (mean of 3 seeds): ViSO gap (%) vs. the PyVRP reference at equal wall-clock (negative = ViSO better), share of VLM-selected subproblems whose re-optimization improved the incumbent (Impr., %), per-run token usage (millions, input+output), and total wall-clock including initialization (s). Test-split instances marked †.

| City | $n$ | Gap | Impr. | Tok(M) | Time | City | $n$ | Gap | Impr. | Tok(M) | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Amsterdam | 500 | -2.099 | 17.9 | 0.38 | 233 | Addis_Ababa | 1000 | -3.506 | 26.4 | 0.39 | 233 |
| Asuncion_Paraguay† | 500 | -3.176 | 21.4 | 0.48 | 231 | Athens† | 1000 | -3.590 | 25.2 | 0.31 | 233 |
| Bangkok | 500 | -3.042 | 20.3 | 0.33 | 231 | Barcelona | 1000 | -2.720 | 29.5 | 0.29 | 235 |
| Buenos_Aires | 500 | -2.069 | 17.0 | 0.52 | 233 | Bukhara_Uzbekistan† | 1000 | -4.417 | 27.0 | 0.35 | 234 |
| Cairo | 500 | -3.125 | 16.2 | 0.53 | 233 | Colombo_Sri_Lanka | 1000 | -2.871 | 26.7 | 0.40 | 235 |
| Cape_Town | 500 | -2.757 | 17.1 | 0.49 | 233 | Cusco_Peru | 1000 | -2.504 | 28.2 | 0.29 | 235 |
| Chicago | 500 | -2.754 | 18.0 | 0.55 | 231 | Hanoi | 1000 | -2.626 | 31.4 | 0.21 | 233 |
| Fez_Morocco† | 500 | -2.915 | 21.7 | 0.47 | 231 | Hobart_Australia | 1000 | -3.015 | 28.1 | 0.24 | 235 |
| Jakarta† | 500 | -2.397 | 20.4 | 0.44 | 231 | Marrakech_Morocco† | 1000 | -2.262 | 24.7 | 0.34 | 235 |
| Kyoto | 500 | -1.979 | 20.4 | 0.41 | 233 | Melbourne† | 1000 | -3.249 | 25.7 | 0.32 | 233 |
| Lagos | 500 | -1.929 | 24.9 | 0.42 | 231 | Mexico_City | 1000 | -3.675 | 27.3 | 0.35 | 235 |
| London | 500 | -2.249 | 17.5 | 0.48 | 231 | Mumbai | 1000 | -3.158 | 25.7 | 0.33 | 234 |
| New_York_City | 500 | -1.317 | 18.8 | 0.35 | 233 | Nairobi | 1000 | -2.578 | 28.3 | 0.34 | 233 |
| Quebec_City† | 500 | -2.149 | 19.1 | 0.52 | 233 | Osaka | 1000 | -2.398 | 25.5 | 0.31 | 233 |
| Rome | 500 | -3.736 | 15.9 | 0.43 | 234 | Paris | 1000 | -3.259 | 26.4 | 0.36 | 233 |
| San_Juan_Puerto_Rico† | 500 | -1.061 | 18.1 | 0.43 | 231 | Reykjavik_Iceland | 1000 | -3.754 | 31.0 | 0.30 | 235 |
| Sao_Paulo† | 500 | -2.125 | 16.9 | 0.50 | 233 | San_Francisco | 1000 | -2.518 | 24.4 | 0.44 | 233 |
| Seoul | 500 | -2.343 | 20.3 | 0.46 | 231 | Santiago | 1000 | -3.283 | 26.2 | 0.40 | 233 |
| Sydney | 500 | -1.637 | 20.4 | 0.40 | 233 | Stockholm† | 1000 | -2.974 | 25.3 | 0.35 | 233 |
| Tokyo | 500 | -1.876 | 15.7 | 0.53 | 233 | Taipei† | 1000 | -3.602 | 30.3 | 0.31 | 233 |

## E DETAILED ABLATION RESULTS

The tables provide numerical results for Ablations 1–4 in Section 4; the DRAGON transfer results are in Appendix I. Table 9 reports the skill-free and skill-guided gaps across all four problem families. They also report training-split results, relative gains, improvement counts, and token usage.

Table 9: Ablation of the distilled skill bank. We compare the mean gap (%) obtained with the skill-free visual selector and with skill-guided ViSO, together with the relative gain from skill guidance and the training/held-out breakdown.

| | All instances | | | Train | | Test | |
|---|---|---|---|---|---|---|---|
| Family | w/o skills | ViSO | Rel. gain | w/o skills | ViSO | w/o skills | ViSO |
| TSP | 0.355 | 0.289 | 18.5% | 0.395 | 0.327 | 0.264 | 0.202 |
| ATSP | 0.244 | 0.207 | 15.2% | 0.277 | 0.227 | 0.170 | 0.162 |
| CVRP | 0.243 | 0.231 | 4.9% | 0.244 | 0.235 | 0.241 | 0.220 |
| ACVRP | −2.643 | −2.717 | +2.8% | −2.602 | −2.671 | −2.739 | −2.826 |

Table 10: Region-selection ablation on TSP, with every selector given the same per-tier runtime budget. Results are mean final gaps (%) over 33 instances and three runs per instance.

| Selector | All (33) | Small (20) | Large (13) |
|---|---|---|---|
| Random sequence | 0.431 | 0.183 | 0.813 |
| Random box | 0.335 | **0.122** | 0.663 |
| ViSO without skills | 0.355 | 0.176 | 0.630 |
| **ViSO** | **0.289** | 0.156 | **0.494** |

Table 11: Initialization and skill-sampling ablation, with every variant stopped after solving the same number of subproblems. Results are means over 10 TSP instances, with three runs per instance. Impr. is the number of committed improvements per run, and Tok (M) is mean total VLM token usage in millions.

| Warm start | Skill sampling | Start gap | Final gap | Impr. | Tok (M) |
|---|---|---|---|---|---|
| LKH-3 | scheduler | 1.23 | **0.271** | 26.7 | 1.06 |
| LKH-3 | uniform | 1.23 | 0.293 | 26.5 | 1.12 |
| Farthest insertion | scheduler | 16.14 | 0.850 | 41.3 | 1.14 |

The initialization and skill-sampling ablations use the same ten instances: `u1817`, `u2152`, `pr2392`, `pcb3038`, `fl3795`, `fnl4461`, `rl5934`, `pla7397`, `rl11849`, and `brd14051`. Replacing LKH-3 with farthest insertion raises the starting gap from $1.23\%$ to $16.14\%$. ViSO reduces the latter to $0.850\%$, a decrease of $15.29$ percentage points, or $94.7\%$ of the initial gap, showing that its improvements are not limited to near-optimal warm starts.

## F SKILL VS. NO-SKILL CONVERGENCE AND BOX SELECTION

The convergence curves and box-selection heatmaps supplement the skill ablation in Sec. 4. Both use twelve instances (three per family) and the three runs used for the paper's final reported results. In the heatmaps, skill-guided selection repeatedly targets a few high-value areas, while skill-free selection is more diffuse.

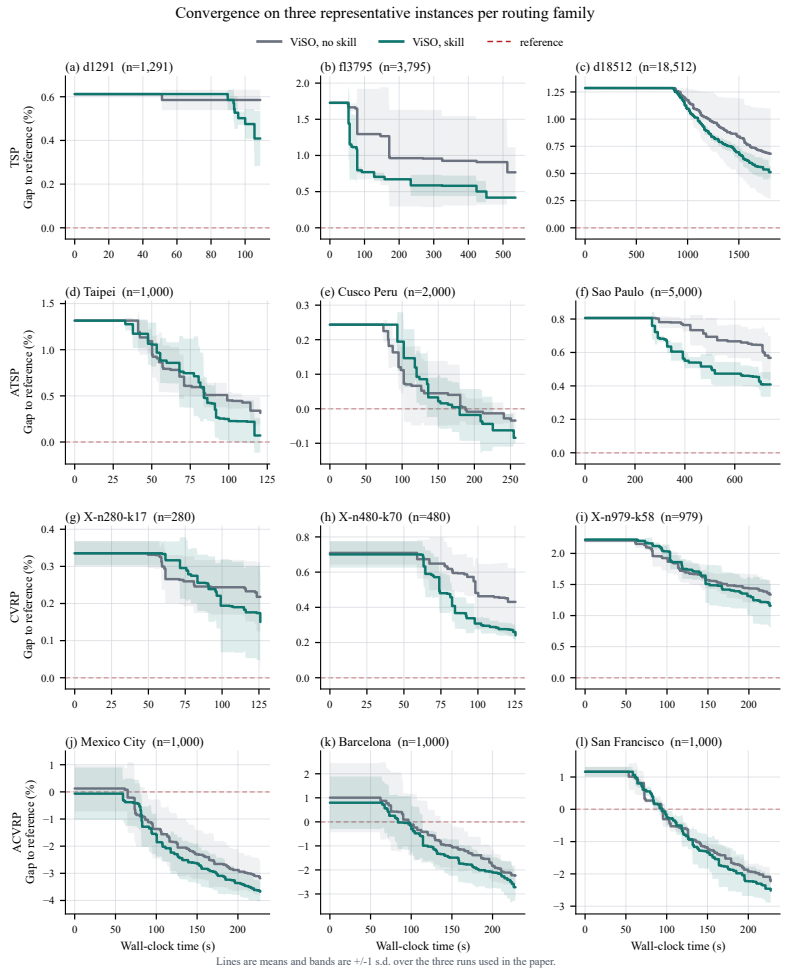



Figure 5: **Skill vs. no-skill convergence.** Gap to reference versus wall-clock time for skill-guided and skill-free ViSO on twelve instances, each method averaged over the three runs used for the paper's final reported result. Skills reach a lower objective earlier.

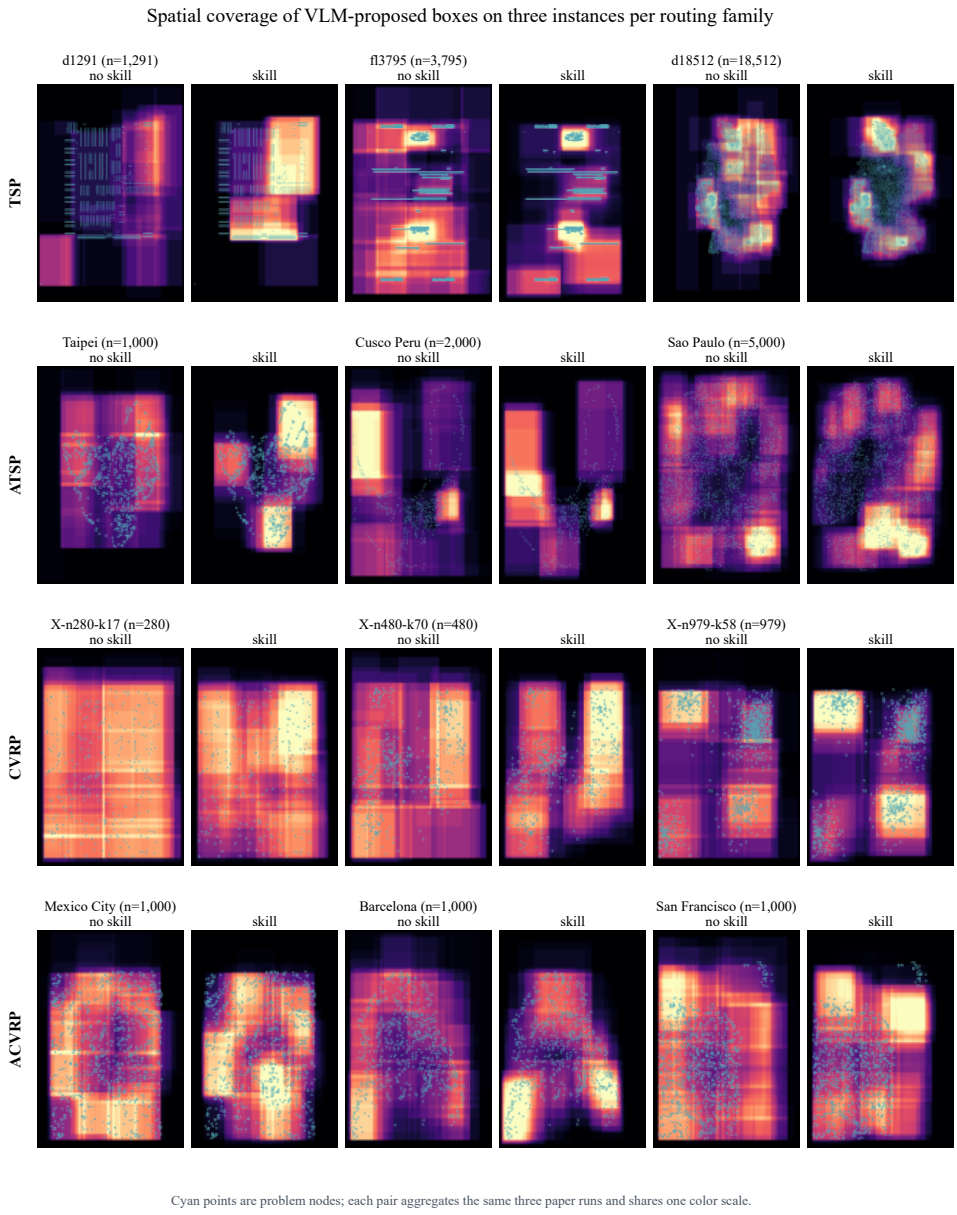



Figure 6: **Skill vs. no-skill box selection.** Spatial heatmaps of where the skill-guided and skill-free selectors place their chosen regions, on the same twelve instances. Skills concentrate selection on a few high-value regions, while skill-free selection is more diffuse.

## G ASYNCHRONOUS ORCHESTRATION DETAILS

ViSO runs as communicating processes over shared state: the incumbent solution $\mathcal{S}^*$ and its cost, a pending-subproblem queue, a priority queue of already-improving subproblems, and the evidence log. *Producers* ($P$=4 VLM selectors) repeatedly render the latest incumbent, draw $Q$=2 skills from the scheduler distribution, request regions from the VLM, and enqueue the resulting hypernode subproblems. *Solver consumers* (at most sixteen concurrent) pull subproblems and re-optimize them with the class-appropriate solver of Sec. 3.6; a subproblem whose re-optimization improves on the solution it was cut from is promoted to the priority queue. Because the incumbent may have changed while a subproblem was solving, a *verifier* re-evaluates every candidate against the latest incumbent before the hill-climbing commit of Eq. (10). Commits are serialized under a lock to prevent concurrent improvements from corrupting the solution. The warm start is LKH-3 for TSP/ATSP, HGS for CVRP, and PyVRP for ACVRP; for asymmetric instances the MDS embedding of Appendix B is computed concurrently with the warm start. A run terminates when the per-instance wall-clock budget of Appendix C expires. Non-improvement-based early stopping is disabled in all four families. Both skill-guided and skill-free CVRP runs also disable BKS-based stopping and continue after initialization until the time limit. Algorithm 1 summarizes the full procedure.

**Algorithm 1** ViSO: asynchronous skill-guided decomposition

**Require:** instance with cost matrix $D$, skill bank $\mathcal{B}$, budget $T$, regions per call $Q$, producers $P$

launch $\mathcal{S}^* \leftarrow$ INITIALIZE($D$) and, if $D$ is non-geometric, its MDS embedding concurrently; await both
$X^0 \leftarrow$ RENDER($\mathcal{S}^*$) {native coordinates or the completed MDS embedding (Sec. 3.2)}
$p \leftarrow F_{\text{sched}}(X^0, \mathcal{B})$ {one VLM pass, Eq. (2)}
launch $P$ producers, up to 16 solver consumers, and one verifier
**Producer** (each of $P$, in parallel):
**while** budget $T$ not exhausted **do**
  $X \leftarrow$ RENDER($\mathcal{S}^*$); draw $\sigma^{(1)}, \ldots, \sigma^{(Q)} \sim \text{Cat}(p)$
  $\{(C_q, \hat{\sigma}^{(q)})\}_{q=1}^{Q} \leftarrow F_{\text{select}}(X; \sigma^{(1)}, \ldots, \sigma^{(Q)})$ {Eq. (3); zoom-in for oversized boxes}
  for each $C_q$: build $\omega = (W, H)$ by hypernode contraction (Sec. 3.5); push $\omega$ onto queue $\Omega$
**end while**
**Solver consumer** (each, in parallel):
**while** budget $T$ not exhausted **do**
  pop $\omega$ from $\Omega$; $\Pi_\omega \leftarrow F_{\text{solver}}(\tilde{D}, \mathcal{K}_\omega)$ {Eq. (9)}
  **if** $\Pi_\omega$ improves on the solution it was cut from **then** push $(\omega, \Pi_\omega)$ onto priority queue $\Omega'$
**end while**
**Verifier** (serialized):
**while** budget $T$ not exhausted **do**
  pop $(\omega, \Pi_\omega)$ from $\Omega'$; expand hypernodes to obtain candidate $\mathcal{S}'$ against the freshest $\mathcal{S}^*$
  **if** $\text{feasible}(\mathcal{S}') \wedge c(\mathcal{S}') < c(\mathcal{S}^*)$ **then** $\mathcal{S}^* \leftarrow \mathcal{S}'$ {Eq. (10)}
**end while**
**return** $\mathcal{S}^*$

**Zoom-in re-selection.** Algorithm 2 describes refinement of boxes whose free-node count exceeds the solver-dependent cap $N_{\max}$ in Appendix H. Re-selection uses a cropped view with grid resolution $\lceil (y_{\max} - y_{\min})/100 \rceil$. The assigned skill remains pinned throughout, and retry tokens are charged to the producing call. Only the final admissible box and its recorded citation are enqueued. The procedure retains the incoming citation for a pinned-skill zoom and otherwise records the citation of the resulting selection, which may be none.

**Algorithm 2** Zoom-in refinement of oversized boxes

**Require:** image $X$, box $C$, incoming citation $\hat{\sigma}$, pinned skill $\sigma$, free-node set $W(C)$, solver cap $N_{\max}$

**while** $|W(C)| > N_{\max}$ **do**
  $X_C \leftarrow$ crop $X$ to $C$ and render a finer grid
  $(C, \hat{\sigma}) \leftarrow \textsc{ReselectRegion}(X_C; \sigma, \hat{\sigma})$ {pinned skill; recorded citation}
  recompute $W(C)$ from the current solution
**end while**
**return** $C$ with citation $\hat{\sigma}$

## H PROMPTS, HARDWARE, HYPERPARAMETERS, AND TOKEN USAGE

**Models.** We use gpt-5.4, served via an Azure endpoint, for all language-model calls: the box-selection VLM, the visual scheduler pass of Eq. (2), and the distillation critic, merge, and consolidation calls of Sec. 3.4.

**Prompt structure.** Every VLM-facing prompt includes an image legend describing the node dots, route segments in visiting order, and reference grid. For asymmetric instances, the legend notes that node positions approximate the costs through MDS and provide a visual guide rather than exact geometry. The *selection* prompt appends one skill block per requested box (the Name/Cue/Action/Avoid fields of Sec. 3.3), instructs the VLM to cite the complete skill name it followed (or `none`), and fixes an output format of one coordinate block per box; a fast-response variant omits the free-text reason line to reduce completion tokens and latency. The *visual scheduler* prompt shows the full bank and asks for a JSON probability distribution over the skills, requiring every skill to receive strictly positive mass summing to one. The *distillation* prompts implement Sec. 3.4: the pair critic sees one successful and one failed selection image with their statistics; the merge prompts are text-only and cluster near-duplicates while tracking support; the consolidation prompt shows $\gamma$ successful selections from distinct instances next to the frozen bank and emits ADD/AGREE/EDIT operations. All skill text is capped at 60 words, and absolute direction words are forbidden and validated post hoc, enforcing the rotation/reflection invariance of the bank. The two inference-time templates and the five distillation templates are rendered below. Placeholders in braces are filled per instance and per call, and the domain wording (tour/route, city/customer) is switched per problem family. In the distillation templates, `{skill_rules}` expands to the shared formatting rules stated above (the 60-word cap, the Name/Cue/Action/Avoid fields, and the ban on absolute direction words and instance identifiers).

**Example distilled skill.** The bridge selection in Figure 1(a) cites the following large-tier TSP skill. All skills in our banks use this four-field format.

```
Name:   Straddle a local bridge between runs
Cue:    A compact bridge links nearby parallel or layered runs, with
        bends or crossings concentrated around the link.
Action: Box the bridge and short segments of each attached run so
        pairings on both sides can change.
Avoid:  Boxes covering only one run or long regular stretches beyond
        the bridge.
```

**Visual scheduler prompt (Eq. (2)).**

```
You are designing a sampling distribution for the box-selection assistant
of a VLM-guided {TSP|ATSP|CVRP|ACVRP} routing optimizer.

You will see ONE image -- the initial {tour|vehicle routes} for the
instance just after the warm start. On every subsequent box-selection
call, a separate VLM will receive a similar image and decide which
sub-rectangle to optimize next. To help that downstream VLM, we will
hand it skills drawn from the distribution YOU produce now.
```

```
IMAGE LEGEND (exactly as the downstream VLM sees it):
- Red dots: {city|customer} locations. {ATSP/ACVRP: node positions are
  an approximate 2D MDS embedding of the asymmetric costs -- a hint,
  not exact geometry.}
- {Black|Colored} lines: current {tour edges|vehicle routes}.
- Grid lines: spatial reference grid overlaid on the map.
- Full-instance bounds: x in [{x_min}, {x_max}], y in [{y_min}, {y_max}].
A productive sub-region is one whose local re-optimization would
{shorten the tour|reduce total route cost}.

Your job is to assign a sampling probability to EACH skill in the bank.
There is no "no-skill" option -- every box-selection call will be given
one of the skills below.

CANDIDATES -- each Skill is shown with Name + Cue + Action + Avoid.
{skill 1 ... skill M}

HOW TO WEIGH THE CANDIDATES
- Visual match: inspect the image carefully. For each skill, judge how
  often its Cue is likely to fire on THIS instance.
- Exploration vs exploitation: every skill MUST receive a strictly
  positive probability. The distribution must not collapse onto a
  single dominant skill.
- The full distribution MUST sum to 1.0.

OUTPUT FORMAT -- emit exactly one JSON object, nothing else. Keys are
the skill names exactly as listed above. Values are floats in (0, 1].
```

**Skill-injected box-selection prompt (Eq. (3)).**

```
You are given {Q} candidate skill(s) for THIS image (distilled from
past runs). Apply one skill to select one box, and the other skill to
select another box. Follow a skill's cue and action to choose where
to place its box.

Candidate skill 1:
  Name: {name} | Cue: {cue} | Action: {action} | Avoid: {avoid}
Candidate skill 2:
  Name: {name} | Cue: {cue} | Action: {action} | Avoid: {avoid}

Instructions:
- Follow each skill's cue and
  action to select its box, and cite that skill for that box.
- You must emit exactly {Q} box(es). For EACH box, cite the skill name
  you used to select it. You can cite "none" if no box can be selected
  using the provided skills. Do not force-fit, but prefer a provided
  skill whenever it plausibly applies.
- When citing a skill, write the COMPLETE skill NAME exactly as given
  above. Do NOT write "Skill 1" or "Skill 2" -- those are slot labels,
  not citations.

Output format -- emit ONE block per box:

Box {i}:
Relevant skill(s): <skill name or "none">
Reason: <one sentence>          [omitted in fast-response mode]
<coordinates> x_min=..., x_max=..., y_min=..., y_max=... </coordinates>
```

**Stage A critic prompt (Eq. (5), one call per success/failure pair).**

```
You are a critic distilling reusable {TSP|ATSP|CVRP|ACVRP} subproblem-selection
skills. Below are TWO images from the SAME instance: a SUCCESSFUL
selection (the local solver improved the {tour|routes}) and a FAILED
selection (no improvement).

{image legend}

IMAGE 1 -- SUCCESSFUL SELECTION
  Instance size: {succ_size}   Nodes inside box: {succ_nodes}
  Area covered: {succ_area}% of map
  Absolute improvement: {succ_gain}

IMAGE 2 -- FAILED SELECTION (no improvement)
  Instance size: {fail_size}   Nodes inside box: {fail_nodes}
  Area covered: {fail_area}% of map
  Absolute improvement: 0

Propose 1 to 3 reusable SKILLS that explain why the success worked
and/or why the failure did not.

{skill_rules}

OUTPUT FORMAT (JSON only -- no prose):
{"skills": [{"name": "...", "cue": "...", "action": "...",
             "avoid": "..."}]}
```

**Stage A within-instance merge prompt (first level of Eq. (6)).**

```
You are consolidating skill patches from a single {TSP|ATSP|CVRP|ACVRP} instance.

Below are {n_patches} independently-proposed skill patches, each from
one success/failure pair of the same instance. Some patches likely
describe the same pattern with different wording; some are unique.

Your job:
1. CLUSTER near-duplicates (same pattern, different wording).
2. PICK the clearest, most general wording for each cluster.
3. RESOLVE conflicts (same name, divergent content) -- rename if needed.
4. RETAIN each distinct skill and its observed support.
5. For each merged skill, count how many of the {n_patches} input
   patches supported it (n_patches_in_instance).

{skill_rules}

INPUT PATCHES:
{patches_block}

OUTPUT FORMAT (JSON only):
{"skills": [{"name": "...", "cue": "...", "action": "...",
             "avoid": "...", "n_patches_in_instance": <int>}]}
```

**Stage A across-instance merge prompt (second level of Eq. (6)).**

```
You are consolidating skill summaries across multiple {TSP|ATSP|CVRP|ACVRP}
instances. Below are summaries from {n_instances} distinct training
instances. Each summary contains skills with per-instance support
counts.

Your job:
```

```
1. CLUSTER skills that describe the same pattern across instances,
   even if wording differs.
2. PICK the clearest, most general wording for each cluster.
3. RESOLVE conflicts (same name, divergent content) by rewording.
4. RETAIN each distinct skill and its cross-instance support record.
5. For each merged skill, record which instances contributed patches
   and how many patches each contributed.

{skill_rules}

INSTANCE SUMMARIES:
{summaries_block}

OUTPUT FORMAT (JSON only):
{"skills": [{"name": "...", "cue": "...", "action": "...",
             "avoid": "...", "patches_per_instance": {...},
             "n_instances": <int>}]}
```

**Stage B consolidation prompt (Eq. (7), one call per group).**

```
You are a critic looking at {gamma} SUCCESSFUL subproblem selections
from {gamma} DIFFERENT {TSP|ATSP|CVRP|ACVRP} instances. All achieved meaningful
improvement.

{image legend}
{images_block}

A REFERENCE BANK of skills has already been distilled from
contrastive analysis. Compare these successful selections against
the reference bank.

For each pattern you see, decide ONE of:
- AGREE with an existing skill if at least one of the images directly
  matches its cue and action.
- EDIT an existing skill if its wording is inaccurate, incomplete, or
  misleading given what these images actually show. Provide the
  refined full skill (<=60 words).
- ADD a new skill ONLY if no existing skill captures a generalizable
  pattern visible across these cross-instance successes.

Multiple ops are OK. Zero ops is OK. Skip patterns already perfectly
covered by the bank.

{skill_rules}

REFERENCE BANK:
{bank_block}

OUTPUT FORMAT (JSON only):
{"add":   [{"name": "...", "cue": "...", "action": "...",
            "avoid": "..."}],
 "agree": ["<existing_skill_name>", ...],
 "edit":  [{"name": "<existing_skill_name>", "cue": "...",
            "action": "...", "avoid": "..."}]}
```

**Stage B merge prompt (final reconciliation).**

```
You are consolidating Stage B ops from {n_groups} cross-instance
success groups against the reference bank produced by Stage A.

Each group independently proposed ADD / AGREE / EDIT ops. Your job:
```

```
1. CLUSTER all ADDs across groups -- same pattern under different
   names -> one skill with combined support.
2. AGGREGATE all AGREE ops by skill name -- count distinct groups
   that AGREEd with each existing skill.
3. RECONCILE EDIT proposals -- if multiple groups proposed
   conflicting edits to the same skill, pick the clearest and most
   general wording.
4. RETAIN Stage A originals and Stage B additions.

{skill_rules}

REFERENCE BANK (Stage A result):
{bank_block}

STAGE B GROUP OPS:
{ops_block}

OUTPUT FORMAT (JSON only):
{"skills": [{"name": "...", "cue": "...", "action": "...",
             "avoid": "...", "stage_b_ops": {...},
             "n_groups": <int>}]}
```

**Hyperparameters.** Regions per selection call $Q{=}2$; VLM producers $P{=}4$; at most sixteen concurrent solver consumers; a proposed box must enclose at most $N_{\max}{=}1000$ nodes for TSP/ATSP or $N_{\max}{=}500$ for CVRP/ACVRP; boxes exceeding this upper limit trigger the zoom-in re-selection of Sec. 3.3; the Concorde subproblem call uses a 10s time limit; distillation uses up to $\Lambda{=}15$ success/failure pairs per instance and consolidation groups of $\gamma{=}5$.

**Hardware.** The original experiments used a 16-core CPU and an NVIDIA A5000 GPU with 24 GB of memory.

**Skill-distillation overhead.** Distillation is a one-time offline procedure whose banks are reused across all test instances of a family. Table 12 reports its measured token usage. Across both size tiers, family totals range from $0.816$M to $1.508$M prompt tokens and from $0.090$M to $0.155$M completion tokens. Complete distillation across the four families consumed 4.42M prompt tokens and 0.47M completion tokens. The parallel procedure in Sec. 3.4 dispatches the per-pair critic calls of Stage A and the per-group calls of Stage B concurrently. It takes 31.6 minutes of wall-clock time for 296.8 minutes of accumulated API time, a ratio of $9.4\times$. Token usage is dominated by the Stage A.1 contrastive critic (one per success/failure pair); the text-only merges (A.2, A.3, B.3) are negligible.

Table 12: One-time skill-distillation token usage per family and size tier (VLM = gpt-5.4). "API" is the summed latency of all VLM calls; "Wall" is elapsed time under the parallel dispatch of Sec. 3.4.

| Family | Tier | Calls | Prompt tok. | Compl. tok. | Wall / API (min) |
|---|---|---|---|---|---|
| TSP | small | 189 | 459,384 | 50,042 | 3.9 / 38.5 |
| | large | 149 | 356,191 | 39,613 | 3.0 / 29.3 |
| ATSP | small | 239 | 585,286 | 63,449 | 3.6 / 34.4 |
| | large | 144 | 348,340 | 39,375 | 2.6 / 20.7 |
| CVRP | small | 353 | 875,966 | 92,179 | 5.6 / 56.1 |
| | large | 116 | 283,249 | 31,543 | 2.4 / 19.3 |
| ACVRP | small | 478 | 1,175,267 | 119,412 | 8.2 / 74.1 |
| | large | 138 | 332,891 | 35,361 | 2.4 / 24.3 |
| **All** | — | 1,806 | 4,416,574 | 470,974 | 31.6 / 296.8 |

# I SKILL DISTILLATION AND STATISTICS-GUIDED SELECTION IN DRAGON

Because geometric and tour structure can be difficult for an LLM to infer from text-only instance representations, we provide computed node statistics, such as crossings and detours, in a compact text table. An additional LLM call chooses the ranking statistic, node-filtering conditions, and

statistics to display before DRAGON selects nodes to repair; the full interface conditions this query on a sampled skill.

**Distillation from DRAGON trajectories.** We apply the two-stage distillation procedure of Sec. 3.4 to historical plain-DRAGON trajectories from the training split. The critic receives the instance coordinates, the complete tour at each selection, the selected node IDs, and the observed improvement. These inputs replace the rendered images used in ViSO. Skills retain the Name/Cue/Action/Avoid format, with cues and actions expressed in terms of coordinates and tour order. We pair successful and failed selections, merge the proposed skills within and across instances, and consolidate the bank using successful selections from distinct training instances. Banks are distilled separately for $n < 3000$ and $n \geq 3000$ and remain fixed during evaluation. For the large tier, consolidation groups contain at most three successful examples and at most 380,000 characters of instance data; small-tier groups contain at most five examples.

**Numerical representation of the current tour.** Let $R_t$ be DRAGON's current tour at destroy step $t$, $\mathbf{X}_D$ the node-coordinate matrix, and $H_t$ the previous node selections. We compute a feature vector $z_t(v)$ for each node $v$ and an instance-level summary $g_t$ from $(\mathbf{X}_D, R_t, H_t)$. The summary includes the instance size, density variation, and the prevalence of crossings, long edges, and detours. The node-feature catalogue contains 25 statistics. Local statistics describe incident-edge crossings, relative edge length, short- and long-window detours, turning, smoothness, density, separate tour passes through a neighbourhood, spatially close but tour-distant neighbours, boundary proximity, one-sided neighbourhoods, density transitions, neighbour counts, nearest-neighbour distance, candidate 2-opt gain, and selection recency. Six statistics describe a 16-node tour window starting at the node: spatial extent, boundary-edge length, endpoint proximity, neighbourhood containment, other tour nodes within its bounding box, and mean defect. Two further statistics give spatial-cluster size and the number of tour passes through that cluster.

The remaining statistic is a composite defect score. For feature $k$, let $\rho_{t,k}(v)$ be its within-instance percentile rank, with tied values assigned their average rank. The score is

$$z_{t,\mathrm{defect}}(v) = \frac{1}{5} \sum_{k \in \{\mathrm{gain2,cross,detour,long,turn}\}} \rho_{t,k}(v). \tag{19}$$

The implementation uses $\rho_{t,k}(v) = 100(r_{t,k}(v) - 0.5)/n$, where $r_{t,k}(v)$ is the one-based average rank among the $n$ nodes. Length-based statistics use local scale normalization. For example, let $s_v$ be the median distance to $v$'s eight nearest neighbours. Any zero scale is replaced by the median of the positive scales, floored at $10^{-9}$; if no positive scale exists, it is replaced by 1. Thus $s_v > 0$. Define $\ell_{uv} = 2\|\mathbf{X}_{D,u} - \mathbf{X}_{D,v}\|_2/(s_u + s_v)$, where $\mathbf{X}_{D,v}$ is the coordinate vector of node $v$. Then $z_{t,\mathrm{long}}(v) = \max\{\ell_{\mathrm{prev}_t(v),v}, \ell_{v,\mathrm{next}_t(v)}\}$. The statistics are computed by code; the LLM chooses which of them to request.

**Skill scheduling and query selection.** One scheduler call at the beginning of a run maps the initial summary $g_0$ and the bank $\mathcal{B}_D$ to a fixed distribution $p_D$ with positive mass on every skill. Each destroy step samples $\sigma_t \sim \mathrm{Cat}(p_D)$. An additional LLM call then receives $\sigma_t$, $g_t$, and the catalogue of feature names and definitions. It does not receive the full coordinates or tour sequence at this stage. The returned JSON query has the form

$$q_t = (r_t, o_t, \mathcal{F}_t, \mathcal{C}_t) = F_{\mathrm{query}}(\sigma_t, g_t, \mathcal{K}), \tag{20}$$

where $\mathcal{K}$ is the shared feature catalogue, $r_t$ is one ranking feature, $o_t \in \{\mathrm{asc}, \mathrm{desc}\}$ is its ordering, $\mathcal{F}_t$ contains at most two threshold conditions, and $\mathcal{C}_t$ contains two to six distinct feature columns. No skill-specific mapping to statistics is specified by hand.

**Query execution and node selection.** Let $U_t$ contain the current tour nodes with valid values of the requested ranking feature. Each filter is a triple $(k, \circ, b)$, with $\circ \in \{<, \leq, >, \geq, =\}$. The filtered set and its ranked shortlist are

$$A_t = \{v \in U_t : \ z_{t,k}(v) \circ b \text{ for every } (k, \circ, b) \in \mathcal{F}_t\}, \qquad L_t = \mathrm{Top}_{40}(A_t; z_{t,r_t}, o_t). \tag{21}$$

Here $\mathrm{Top}_{40}$ returns at most 40 nodes in the requested order. If the filters leave fewer than ten nodes, they are dropped before ranking. An empty eligible set falls back to the defect feature. An

invalid query uses descending defect ranking, no filters, and the columns defect, candidate 2-opt gain, crossings, and tour-distant neighbours. When selection recency is displayed, the table may append up to 20 additional nodes selected in the preceding two calls. Window statistics also include the window endpoint and its successor.

Each table row contains the node ID, its position in the tour, its two tour neighbours, and the requested feature values. We append this table and the sampled skill to DRAGON's original destroy prompt, which still contains the complete coordinates and tour. The original selection call returns up to 20 node IDs in DRAGON's `<sub>` format. The shortlist supplies evidence; it does not impose a hard restriction on the returned node IDs. DRAGON's OR-Tools repair call, with a five-second limit per subproblem, and its simulated-annealing acceptance rule are unchanged. The added query and scheduler calls are included in the recorded API usage.

**Statistics-only control and evaluation.** The statistics-only arm uses the same feature computation, query parser, table builder, and destroy call. Its query asks for statistics useful for shortening the tour without providing a skill. The selection prompt contains the resulting table but no skill, and no skill-scheduler call is made. Plain DRAGON receives neither addition.

The comparison covers 31 TSPLIB instances with three seeds per instance and the same train/test split as the distillation experiment. The historical runs on `pla33810` and `pla85900` completed no subproblems and are excluded.

**Transfer results.** After solving the same number of subproblems, the mean gap falls from $15.424\%$ for plain DRAGON to $15.247\%$ with statistics alone and to $15.104\%$ with the full interface. The statistics-only control provides a $1.15\%$ relative improvement over plain DRAGON, while distilled skills add a further $0.93\%$ improvement over the numeric interface. Under the same runtime budget, the full interface improves over plain DRAGON by $1.42\%$ overall. With the same number of subproblems on the held-out test split, it improves over plain DRAGON by $1.62\%$ (9 of 10 instances) and reduces the mean gap by a further $0.34\%$ relative to the statistics-only control.