Interpreting Neural Combinatorial Optimization via Evolving Programmatic Bottlenecks

Haocheng Duan[1*], **Yuxin Guo**[1*], **Jieyi Bi**[2], **Anqi Xie**[1],
Sirui Li[3†], **Yining Ma**[4‡], **Cathy Wu**[4]

[1]Carnegie Mellon University, [2]Nanyang Technological University,
[3]Microsoft Research, [4]Massachusetts Institute of Technology

Abstract

While Neural Combinatorial Optimization (NCO) achieves strong performance, its interpretability remains an open question. In this paper, we introduce Evolving Programmatic Bottlenecks (EPB), the first post-hoc, surrogate-based interpretability method for NCO, which explains a black-box policy by distilling it into an interpretable behavior surrogate. A natural surrogate is a Concept Bottleneck Model (CBM), yet standard concepts are input attributes, whereas NCO is better explained by decision rules that are hard to predefine. EPB thus uses LLMs to evolve a bank of programmatic decision rules as concepts, each producing an action distribution, where a learned state-dependent router then mixes them to form the surrogate action. We optimize EPB through an iterative framework: Block I fixes the bank size and introduces the hybrid textual-numerical gradient descent to jointly optimize the textual programs and the numerical router; Block II adapts the bank size via failure-targeted expansion and redundancy pruning. Extensive experiments demonstrate that the EPB surrogate closely matches the teacher's behavior and solution quality. We then analyze the distilled rules, reveal several NCO decision patterns, and verify the rules' faithfulness to the surrogate's decisions. Moreover, the surrogate allows targeted program edits that improve its generalization beyond the teacher. Our work advances interpretable NCO and establishes EPB as a promising direction for interpreting complex learned decision-making policies. The code and the distilled surrogate are available at: `https://github.com/HaochengDuan/EPB`.

1 Introduction

Neural Combinatorial Optimization (NCO) has achieved remarkable success, with learned policies now matching strong classical heuristics on complex routing and scheduling [1–3]. However, this success hides a critical interpretability deficit. Current NCO frameworks function as black boxes, providing little insight into what decision strategies have been learned and why a particular action is favored by them. This hinders informed and trustworthy deployment in industry and prevents interpretability from catalyzing future NCO research in academia, as it has in computer vision [4–6].

In this paper, we study post-hoc, surrogate-based interpretability for NCO, which aims at distilling the black-box NCO policy into an interpretable student surrogate that mimics the teacher's decisions. The surrogate serves as both an interpretable behavioral proxy for analysis and a potential deployable alternative (e.g., AlphaDev [7]). Concept Bottleneck Models (CBMs) [8] offer a natural structure for such a surrogate [9], grounding it in human-understandable *concepts* that one can read, audit, and improve. Yet adapting CBMs to NCO is non-trivial. First, existing CBM concepts are typically atomic *attributes* or *features* (e.g., colors and textures in vision), whereas the more natural explanatory unit

*Equal contribution. †Work done outside employment. ‡Corresponding author: Yining Ma (yiningma@mit.edu).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

for NCO is a *decision rule* (e.g., a nearest-neighbor heuristic) that determines action choices. Second, while classical heuristics offer a starting point, such a rule library may be intractable to construct since the algorithmic design space spans combinatorially many decision rules and hyperparameters.

We introduce **Evolving Programmatic Bottlenecks (EPB)**. Rather than relying on a fixed, predefined concept library, EPB (Figure 1) uses large language models (LLMs) to automatically discover and evolve a bank of executable programs, each representing a decision rule as the concept. A state-dependent router then weights the programs' action distribution. This distills the teacher NCO model into a student surrogate, where both the decision rules (programs) and their contributions to the final prediction (weights) are directly inspectable. To our knowledge, EPB is the first framework to provide post-hoc, surrogate-based interpretations of NCO.

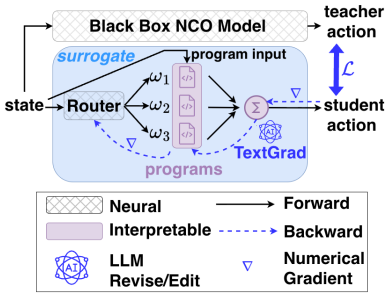


Figure 1: Framework of Evolving Programmatic Bottlenecks (EPB).

Learning this surrogate requires jointly evolving the program bank and training the neural router to combine the programs. This coupling raises two challenges: errors arise from the mixture and do not directly identify which programs need revision, and each revision changes how the router should weight the program. Moreover, as programs evolve, the bank may leave some teacher behaviors uncovered or accumulate redundant programs. EPB addresses them through two alternating optimization blocks. **Block I** fixes the bank size and jointly refines the programs and router through *Hybrid Textual-Numerical Gradient Descent*. Routing weights and deviations from the teacher guide feedback allocation to programs, which TextGrad [10] revises, while numerical gradients update the router. **Block II** adapts the bank through two operations: ADD creates new programs to address unresolved failures, while DROP removes redundant programs.

We apply EPB to two representative NCO models, POMO [2] and LEHD [3], across two routing tasks, TSP and CVRP. Results show that EPB surrogates remain close to the teachers in decision behavior and solution quality. Based on the surrogates, we analyze their decision patterns both at individual steps and in aggregate across instances, revealing, e.g., why they deviate from nearest neighbors and how their strategies shift as routes progress. We further provide counterfactual and program knockout tests that support the faithfulness of these rules to the surrogate's decisions. Notably, while matching their teachers in distribution, the POMO surrogates outperform in generalizing to larger instances, suggesting that the distilled programs may be less tied to the training size than the teacher's neural representations. Finally, because the surrogate contains white-box decision rules, we show that targeted program edits further improve its performance, such as out-of-distribution generalization.

Contributions: (i) we propose EPB, to our knowledge, the first framework to distill NCO policies into interpretable surrogates by using LLMs to evolve programs as concepts; (ii) we propose an iterative framework alternating between hybrid textual-numerical gradient descent and dynamic bank-size management; (iii) we validate the surrogate's fidelity in behavior and solution cost, and provide faithful explanations of individual and aggregated decision patterns; and (iv) we demonstrate its generalization to larger instances, and further gains beyond the teacher through program edits.

2 Related Work

Neural combinatorial optimization (NCO). Classical heuristics rely on hand-crafted decision rules that require problem-specific expertise. NCO instead learns policies from data, through constructive policies [1–3, 11–13] and diffusion-based solvers [14], with test-time scaling and search further enhancing performance [15–18]. However, the learned decision logic remains implicit in neural parameters and difficult to interpret at the level of individual choices. Early NCO interpretability work, such as probing [19], reveals what information NCO representations encode but does not yield executable decision rules. EPB pioneers the first behavioral surrogate study for NCO.

Surrogate-based Interpretability. Interpretability can be pursued by analyzing a model's internals, as in mechanistic interpretability [20, 21], by learning inherently interpretable models, such as programmatic policies in reinforcement learning (RL) [22, 23], or by fitting a post-hoc interpretable surrogate that mimics a trained model's behavior [24]. A surrogate can be inspected, verified, or deployed in place of the black box [25], though it explains behavior rather than mechanism. Common

surrogate families include trees [24–26], additive models [27], tree mixtures [28], and concept-based models [9, 29]. Conceptually close to EPB, PIRL [30] learns a single program that imitates a neural policy within a user-provided sketch for continuous control tasks. EPB instead uses LLMs to evolve a bank of programs as concepts, combined by a neural router to distill and interpret NCO policies.

Concept bottleneck models (CBMs). CBMs [8, 31, 32] link predictions to human-readable concepts, mostly in the form of input attributes. Later CBM extensions reduce annotation requirements [33–35] and broaden applications [36, 37]. Further work explores concept search [38], expert composition [39], and concept-based RL [40]. More recently, DiSciPLE [41] evolves programs with LLMs, but uses them as feature extractors rather than as state-to-action rules. Explaining NCO decisions, however, requires showing how each decision state gives rise to the action choices. EPB therefore uses decision rules as concepts, each implemented as a program that maps a state to an action distribution.

LLM-guided algorithm discovery. EPB relies on LLMs to generate and revise programs, building on their ability to write code [42–44], evolve programs and heuristics [45–47], optimize text from feedback [10], and refine outputs through reflection or search [48, 49]. Among them, FunSearch [45], EoH [46], AlphaEvolve [50], and ShinkaEvolve [51] evolve code to optimize task performance. EPB instead distills a trained NCO policy into interpretable programs to explain its behavior.

3 Preliminaries

Neural combinatorial optimization (NCO). A combinatorial optimization (CO) problem defines, for each problem instance $s \sim \mathcal{D}$, a feasible set $\mathcal{Y}(s)$ and an objective $J(y; s)$ over solutions $y = (a_{1:T})$, with the goal of finding $y^*(s) \in \arg\min_{y \in \mathcal{Y}(s)} J(y; s)$. We focus on routing problems, i.e., the Traveling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP). NCO learns to construct a solution $y = (a_1, \ldots, a_T)$ through sequential decisions (details in §A.1). Let $x_t = (s, a_{1:t-1})$ denote the decision state at step t, including the problem instance and the partial solution constructed so far. An NCO model defines an autoregressive policy $\pi_\theta(a_t \mid x_t)$ to select the next node to visit, until a feasible solution is constructed. As representative teachers, we evaluate POMO [2], which leverages symmetries via multi-start rollouts, and LEHD [3], a light-encoder/heavy-decoder model trained on partial solutions for large-scale generalization (§A.2).

Concept bottleneck models (CBM). A standard CBM [8] employs a two-stage pipeline, $y = f(g(x))$, where an encoder g first maps the input x to a set of human-interpretable concepts before a predictor f makes the final decision based solely on these concepts. The key idea is that the decision is bottlenecked by human-understandable concepts, making it more trustworthy. EPB borrows this idea with evolved programs as concepts: at each state s, g is a bank of LLM-evolved programs outputting action distributions $V_m(s)$, and f is their router-weighted mixture $\sum_m w_m(s)\, V_m(s)$.

4 Methodology

An NCO policy outputs action distributions, so its natural explanatory unit is a decision rule: an executable program that maps the current state to preferences over actions. Following the CBM idea, we construct a student surrogate $\pi_S(\cdot \mid s; \theta, \mathcal{B})$ including a bank $\mathcal{B}$ of human-readable programs evolved by LLMs, each producing an action distribution $V_m(s)$ and serving as a *programmatic bottleneck*, and a router with parameters θ that mixes these distributions into $\pi_S(\cdot \mid s)$ at each state. We first describe the surrogate architecture and then introduce how it is trained via a distillation loss so that $\pi_S(\cdot \mid s)$ matches the teacher's (original NCO model's) distribution $\pi_T(\cdot \mid s)$ at every state s.

4.1 The Student Surrogate with Programmatic Bottlenecks

At each decision state s, the surrogate produces an action distribution in three steps: (1) each program executes its code on s to produce an action distribution $V_m(s)$; (2) the router assigns each program a state-dependent weight $w_m(s)$; and (3) the surrogate outputs the weighted mixture $\sum_m w_m(s)\, V_m(s)$. We describe each component below and depict the architecture of the surrogate in Fig. 2.

The program bank. The bank $\mathcal{B} = \{(c_m, d_m)\}_{m=1}^{M}$ collects M programs, where c_m is a program that maps the decision state to per-action scores and d_m is a few-sentence description of the strategy it implements. The description guides revisions of the code (§B.1), while the code runs in the forward pass. The bank size M changes adaptively during training (§4.2.2).

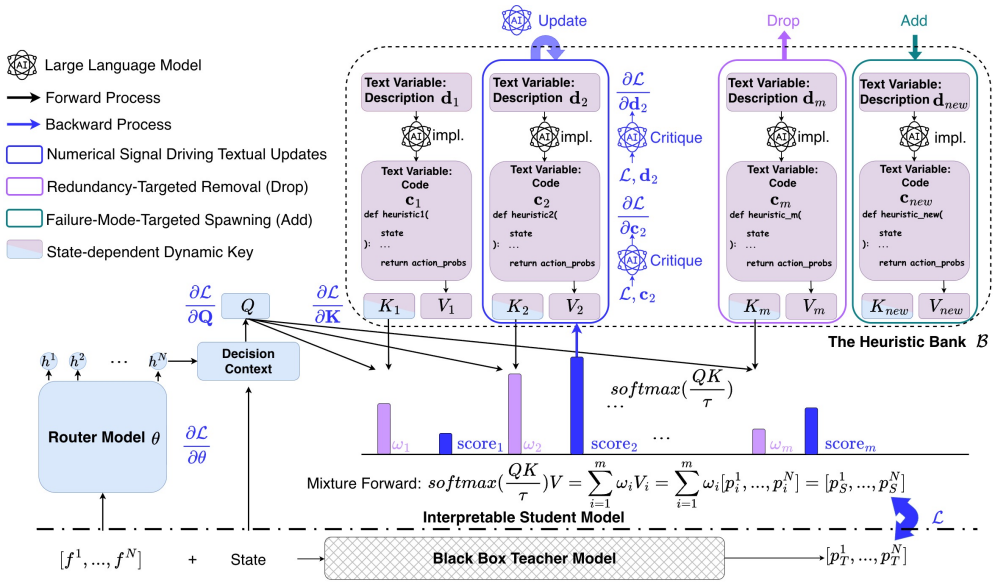


Figure 2: Overview of our Evolving Programmatic Bottlenecks (EPB) architecture.

Per-program distributions. Each program m produces a per-action distribution by executing its code c_m on the current state and applying a softmax with temperature τ_h:

$$V_m(s) \;=\; \mathrm{softmax}(\, c_m(s)/\tau_h \,)\,. \tag{1}$$

State-dependent dynamic routing. The router, an attention module parameterized by θ, determines how much each program contributes at each state. It assigns program m the weight,

$$w_m\left(s;\theta,\mathrm{sg}\left(\mathcal{B}\right)\right) \;=\; \underset{m}{\mathrm{softmax}}(Q\left(s;\theta\right)\cdot K_m\left(s;\theta,\mathrm{sg}\left(\mathcal{B}\right)\right)\,/\,\tau_r)\,, \tag{2}$$

where τ_r is a temperature. The query $Q(s;\theta) \in \mathbb{R}^D$ embeds the current state, following the teacher's context encoding (e.g., for TSP: global information, the current-state context, and the first-node context). The key K_m represents each program m, which is also state dependent as follows:

$$K_m\left(s;\theta,\mathrm{sg}(\mathcal{B})\right) \;\coloneqq\; \sum_n \mathrm{sg}\left(V_m(s,n)\right) h(s,n;\theta) \;\in\; \mathbb{R}^D, \tag{3}$$

where $h(s,n;\theta)$ is the embedding of candidate node n. The dot product $Q \cdot K_m$ thus scores how well program m's output fits the current state. The stop-gradient $\mathrm{sg}(\cdot)$ lets the router read the programs' outputs in the forward pass without passing gradients into them during backpropagation (§4.2.1).

We construct each key from the program's output $V_m(s)$, rather than learning a separate embedding for each program. First, the key adds no additional trainable parameters, so programs can be added or dropped (§4.2.2) flexibly without changing the router's architecture. Second, it represents each program in a simple, direct, yet effective way: a standard CBM concept is described by its activation on the input, whereas an EPB concept is a program, better described by what it recommends at the current state. Our ablation confirms that these keys achieve higher fidelity than freely learned static embeddings and fixed text embeddings with a learned projection (§B.3).

Mixture forward. The student's prediction is the routing-weighted mixture of the bank's outputs:

$$\pi_S(\cdot \mid s) \;=\; \sum_{m=1}^{M} w_m(s;\theta,\mathrm{sg}(V)) \cdot V_m(s). \tag{4}$$

4.2 Two-Block Optimization for EPB

Training EPB requires resolving three quantities: the programs in $\mathcal{B}$, the router parameters θ, and the bank size M. They live in different spaces, as θ is continuous, $\mathcal{B}$ is symbolic, and M is discrete,

Algorithm 1 Two-Block Training

Require: Initial bank $\mathcal{B}_0$, routing parameters θ_0, dataset $\mathcal{D}$, validation set $\mathcal{D}_{\text{val}}$

$\mathcal{B} \leftarrow \mathcal{B}_0$; $\theta \leftarrow \theta_0$
repeat
  // Inner block: hybrid optimization at fixed M
  repeat
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta, \mathcal{B})$ ▷ SGD on Continuous Routing Weights
    $\mathcal{B} \leftarrow \textsc{TextGradStep}(\mathcal{B}, \theta, \mathcal{D})$ ▷ TextGrad on Symbolic Bank
  until $\mathcal{L}_{\text{val}}$ plateaus
  // Outer block: bank size revision
  $\mathcal{B} \leftarrow \textsc{Add}(\mathcal{B}, \mathcal{D}, \mathcal{D}_{\text{val}})$ ▷ Alg. 2
  $\mathcal{B} \leftarrow \textsc{Drop}(\mathcal{B}, \mathcal{D}_{\text{val}})$ ▷ Alg. 3
until $\mathcal{B}$ unchanged by both ADD and DROP
return $(\theta, \mathcal{B})$

so no single optimizer handles all three. We therefore update them with different mechanisms in a nested two-block scheme (Algorithm 1). The inner **Block I** fixes M and jointly refines the programs $\mathcal{B}$ and router θ through Hybrid Textual-Numerical Gradient Descent (§4.2.1), while letting an LLM diagnose the programs' failures and rewrite them. The outer **Block II** adjusts M by adding programs for unresolved failure modes and dropping redundant ones (§4.2.2), after which we repeat Block I at the new size, continuing the loop until optimization convergence.

4.2.1 Block I: Hybrid Textual-Numerical Gradient Descent

Block I jointly trains the continuous routing parameters θ and the symbolic bank $\mathcal{B}$ under a shared distillation loss $\mathcal{L}(\theta, \mathcal{B})$: θ is updated by SGD, while $\mathcal{B}$ is updated through TextGrad proposed in [10]. TextGrad uses LLM calls to generate feedback from the loss and then revise the code and its natural language descriptions in $\mathcal{B}$ based on that feedback.

Specifically, for a decision state s, we define the per-state distillation loss and its expectation as

$$\mathcal{L}(\theta, \mathcal{B}) = \mathbb{E}_{s\sim\mathcal{D}}\left[\ell_s(\theta, \mathcal{B})\right], \qquad \ell_s(\theta, \mathcal{B}) = \mathrm{KL}(\pi_T(\cdot \mid s) \,\|\, \pi_S(\cdot \mid s; \theta, \mathcal{B})). \tag{5}$$

Programs are code and cannot be updated by numerical gradients, but gradients with respect to their outputs can help guide textual feedback. Each program output $V_m(s)$ affects the loss through two paths: directly, as a component of the mixture in Eq. 4, and indirectly, through its routing key K_m in Eq. 3. To isolate the direct contribution, we use $\mathrm{sg}(V)$ in the router, so $\partial w_j / \partial V_m = 0$ for all j, m. The router still reads $V_m(s)$ in the forward pass, but no gradient flows from the routing weights back to the program outputs (§B.2). The resulting gradients with respect to the program outputs and router parameters are given below, with derivations in Appendix §B.4:

$$\nabla_{V_m(s)} \ell_s = w_m(s; \theta, \mathrm{sg}(V))\, r_s, \qquad \nabla_\theta \ell_s = \sum_{m=1}^{M} \langle r_s, V_m(s) \rangle \nabla_\theta w_m(s; \theta, \mathrm{sg}(V)), \tag{6}$$

where $r_s = \nabla_{\pi_S(\cdot|s)} \ell_s(\theta, \mathcal{B})$ is the gradient of the loss with respect to the student's output distribution. Thus, the distillation loss induces two coupled signals: an output-space sensitivity for attributing failures to program m, and a standard numerical gradient for updating the routing network θ.

Numerical signal driving textual updates. A standalone TextGrad backward call derives a textual gradient for a variable from that variable's own failures. In EPB, however, the M programs jointly produce each prediction through the mixture, so a failure belongs to the mixture rather than to any single program, and we must first decide which programs to revise. The program-output gradient in Eq. 6, i.e., $\nabla_{V_m(s)} \ell_s = w_m(s)\, r_s$, is a natural starting point but is insufficient on its own: since the mixture error r_s is shared across the bank, the gradient differs across programs only through the routing weight. It thus tells us how much a program contributes to the mixture, but not whether the program itself is wrong. We therefore keep the routing weight and multiply it by a program-specific error term, measuring how far the program's output deviates from the teacher's with $\mathrm{KL}\big(\pi_T(\cdot \mid s) \,\|\, V_m(s)\big)$. Since our teachers are near-deterministic and fidelity is measured by top-action agreement, we count this deviation only when the program's top action differs from the

teacher's, $n^\star(s) = \arg\max_{a \in \mathcal{A}(s)} \pi_T(a \mid s)$. This yields the feedback allocation score

$$\mathrm{score}_m(s) = \underbrace{w_m(s)}_{\text{contribution}} \cdot \underbrace{\mathbb{1}\left[\arg\max_{a \in \mathcal{A}(s)} V_m(s,a) \neq n^\star(s)\right]}_{\text{wrong top action}} \cdot \underbrace{\mathrm{KL}\big(\pi_T(\cdot \mid s) \,\|\, V_m(s)\big)}_{\text{deviation from teacher}}. \tag{7}$$

For each program, we rank states by $\mathrm{score}_m(s)$ and pass its top-K states, $\mathcal{F}_m^t = \mathrm{TopK}_s\ \mathrm{score}_m^t(s)$, to TextGrad, which converts these failures into natural-language update directions for rewriting c_m and d_m. Restricting feedback to the most relevant failures keeps the LLM calls manageable.

Textual outputs driving numerical updates. The coupling also runs in reverse. In Eq. 6, the gradient for θ depends on the program outputs through $\langle r_s, V_m(s)\rangle$, so a TextGrad rewrite that changes a program's output distribution also changes the subsequent SGD gradient that trains the router.

Closed-loop textual-numerical optimization. The two coupling directions close the training loop under the shared loss $\mathcal{L}$, yielding an approximate *block coordinate descent* process:

$$\begin{aligned} \theta^{t+1} &= \theta^t - \eta_t \nabla_\theta \mathcal{L}(\theta^t, \mathcal{B}^t), \quad \mathcal{F}_m^t = \mathrm{TopK}\,\mathrm{score}_m^t(s), \\ \Delta_m^{\mathrm{text},t} &= \text{TG-Backward}\big(c_m^t, d_m^t, \mathcal{F}_m^t\big), \quad (c_m^{t+1}, d_m^{t+1}) = \text{TG-Step}\big(c_m^t, d_m^t, \Delta_m^{\mathrm{text},t}\big). \end{aligned} \tag{8}$$

Here TG-Backward uses an LLM to turn the selected failures $\mathcal{F}_m^t$ into a critique $\Delta_m^{\mathrm{text},t}$ of (c_m^t, d_m^t), which TG-Step then applies as a rewrite. The continuous block θ is updated by SGD at every batch, while the symbolic block B is updated periodically. This asymmetric schedule reflects the different costs: SGD performs small local steps, whereas each TextGrad step rewrites executable programs and therefore corresponds to a larger, non-differentiable move in function space.

4.2.2 Block II: Bank Size Evolution via Add and Drop

During training, we adjust the bank size M by adding or removing programs. A static bank is ill-suited because optimization faces two competing pressures: previously unseen failure modes may emerge that no existing heuristic can resolve (requiring expansion), while refining different heuristics on similar teacher samples can make initially distinct programs behave similarly, leaving the bank larger than necessary. We thus introduce ADD, which spawns new specialists for uncovered failure modes, and DROP, which prunes redundant or inactive heuristics. Both are triggered upon a validation plateau, a regime in which the validation loss changes only marginally.

Failure-mode-targeted spawning (Add). Add searches for failure cases that suggest missing programs: decision steps where *every* program fails, which are unlikely to be fixed by re-routing alone. We pass the fault set to the LLM, which groups it into K hypothesized failure modes, each with a candidate program. We then refine candidates on the fault subset over several rounds. Each fault state is assigned to the candidate currently closest to the teacher. Each candidate receives TextGrad feedback from its assigned states, and we accept edits only if they improve agreement with the teacher on a held-out split. Surviving candidates are added to the bank (Algorithm 2, §B.5).

Redundancy-targeted removal (Drop). Drop uses greedy leave-one-out pruning. It evaluates each removal, drops the program with the smallest increase in validation loss if this increase is within a preset tolerance, and recomputes. Pruning stops when no removal meets the tolerance, the per-call drop limit is reached, or the bank reaches its minimum size. This keeps the portfolio compact by eliminating programs whose contribution is covered by the rest of the bank (Algorithm 3, §B.5).

5 Experiments

We evaluate EPB by distilling POMO (TSP-50/100, CVRP-50/100) and LEHD (TSP-100, CVRP-100). Detailed experimental settings, additional results and discussions such as LLM costs, sensitivity to random seeds, and the distilled programs are provided in Appendices §C and §D. Fig. 3 shows an example training curve for POMO-CVRP-50, where top-1 agreement measures teacher–student agreement given the same state. Hybrid gradient descent and bank size management jointly refine the program bank, improve the student surrogate's performance, and help escape plateaus.

5.1 Surrogate Performance

Table 1: Fidelity comparison of the surrogate to the teacher model. N: number of nodes.

		TSP						CVRP					
Teacher	N	GPT-6	Fable 5.1	Tree	NAM	MoËT	EPB	GPT-6	Fable 5.1	Tree	NAM	MoËT	EPB
Top-1 agreement (%) ↑													
POMO	50	85.9	84.7	85.6	87.2	83.5	**94.8**	68.6	66.7	70.8	69.7	71.6	**85.8**
POMO	100	84.2	84.6	84.7	86.5	83.4	**94.1**	67.8	66.2	69.2	68.6	71.0	**86.5**
LEHD	100	84.6	85.2	85.4	86.9	83.1	**94.7**	64.8	63.5	66.9	68.7	68.0	**85.1**
Greedy gap (%) ↓													
POMO	50	9.65	8.09	16.50	11.16	18.36	**3.02**	14.90	13.23	25.95	17.89	14.16	**2.88**
POMO	100	10.98	9.04	20.05	13.69	20.82	**3.79**	16.37	14.43	28.63	19.38	15.62	**3.94**
LEHD	100	11.47	9.53	20.08	13.75	22.34	**2.57**	15.89	14.09	21.90	18.20	15.42	**3.53**
Search gap (%) ↓													
POMO	50	2.54	2.45	4.59	2.61	7.08	**0.03**	10.35	9.27	8.60	9.88	7.52	**1.13**
POMO	100	4.58	3.90	8.31	4.98	10.89	**0.07**	12.85	11.27	12.06	12.37	9.12	**1.25**
LEHD	100	2.09	1.64	3.46	2.61	7.15	**0.34**	14.34	12.69	14.47	12.13	11.53	**2.16**

Table 1 reports the fidelity of the students to the teacher, measured by top-1 action agreement on the teacher's greedy trajectories, and solution quality, measured by cost gaps under greedy decoding and search (multi-start for POMO, random reconstruction for LEHD). We compare against two kinds of interpretable surrogates. The first kind is a single scoring program written and tuned by an LLM coding agent, using either GPT-6 or Fable 5.1. The second kind is based on a fixed set of hand-crafted features, and we compare with three methods in Table 1: Tree means a CART decision tree; NAM [27] sums the outputs of per-feature neural networks; and MoËT [28] combines decision-tree experts through a learned, state-dependent softmax gate. EPB achieves the highest top-1 agreement in all six settings, substantially outperforming the strongest baseline. It also attains the lowest cost gaps under both greedy and search decoding. Notably, the margin becomes larger on CVRP, where capacity-dependent decisions involve conditional logic that feature-based surrogates struggle to capture. Absolute costs (Table 13) and instance-level tour comparisons are depicted in §D.3.1 and §D.3.2.

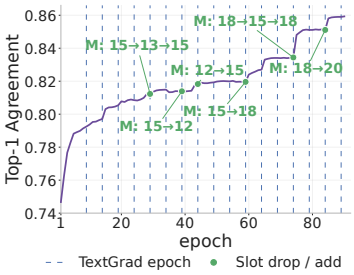


Figure 3: Training top-1 agreement.

5.2 Analysis on Distilled Programs

Table 2 summarizes the distilled surrogates. POMO-TSP programs shift priorities across construction phases, while LEHD-TSP programs increasingly emphasize tour closure alongside local geometry (§D.4.1). Both CVRP banks coordinate spatial preferences, demand, and depot returns based on remaining capacity. Next, we analyze their decision patterns both at individual steps and in aggregate across instances. The full gallery and sample code are provided in §D.4 and §D.5, respectively.

Table 2: High-level summary of distilled program banks learned by EPB.

Bank	**High-level behavior**
POMO TSP	**Priorities shift with tour progress.** Programs use progress or remaining-city counts to emphasize peripheral and isolated cities early, then start-aware closure and multi-step lookahead.
LEHD TSP	**Local geometry and tour progress jointly guide decisions.** Programs balance short travel with insertion and closure costs, using remaining-city geometry and directions relative to the start-to-current heading. Closure receives greater weight as fewer cities remain.
POMO CVRP	**Customer preferences adapt to available capacity.** Programs combine proximity, route savings, and directional coherence with how well customer demand fits the remaining capacity, including whether to continue serving customers or return to the depot.
LEHD CVRP	**Local geometry and vehicle load jointly guide decisions.** Programs combine short moves and route savings with demand fit and angular ordering around the depot, using load to regulate depot returns.

Interpreting individual decision patterns. EPB explains individual decision steps by linking action choices to human-readable programs. Figure 4 highlights the steps where the POMO-TSP-100 student does not choose the nearest unvisited city. We focus on these steps because the nearest neighbor (NN) rule matches approximately 83% of local decisions in the analyzed high-quality tours. At the highlighted steps, the highest-weight programs express distinct geometric preferences. H02 (*hull-sweep early*) rewards candidates farther from the centroid of unvisited cities while penalizing travel distance, favoring peripheral cities when the extra travel is justified. H04 (*isolated rescue*) gives extra preference to isolated cities that might otherwise require costly later detours. H01 (*far-from-start first*) balances short moves with a preference for cities farther from the starting city, leaving nearby cities available for tour closure. Together, these cases illustrate how the student balances immediate proximity with visiting peripheral or isolated cities and preparing for tour closure. In CVRP, the distilled rules explain why the surrogate bypasses the nearest customer to serve customers far from the depot while they are nearby or to reduce detours on the return trip (Figure 15). More examples are provided in §D.6.

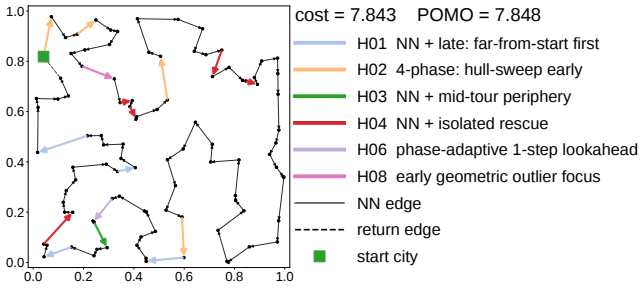


Figure 4: POMO-TSP-100 case study. The highlighted heuristic has the highest router weight at that step.

Interpreting aggregated decision patterns. We aggregate decision patterns across 200 instances to reveal how strategies shift across route stages (Figure 5). It shows three phases: routes open around spatial structure, grow through savings and demand fit, and close toward the depot. Early decisions organize routes around angular sectors or prioritize peripheral and isolated customers. Midway, Clarke-Wright savings rules favor customers far from the depot but close to the vehicle, while demand-fit rules favor customers that leave little capacity unused. Near completion, depot-aware rules favor customers along the return corridor, discourage detours, and determine when to return. This suggests that the student balances vehicle utilization against the cost of returning to the depot.

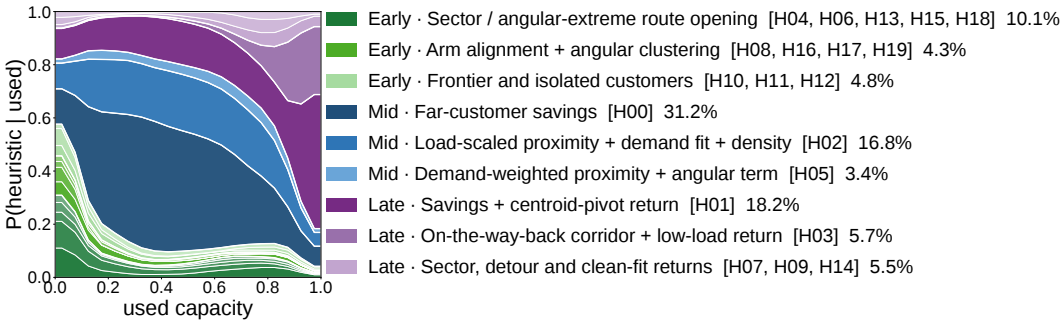


Figure 5: CVRP program activation across route progress (200 instances)

Faithfulness of the EPB surrogate. We further test whether EPB's explanations are causally faithful to its decisions using counterfactual and program-removal tests. Specifically, if a concept causally drives a decision, weakening it should weaken the student's preference. For example, if the student prefers a city because it is isolated, making the city less isolated should reduce that preference. We perturb states to weaken a target city's peripherality, isolation, or proximity to the homeward corridor, compare against matched controls, and track the effect from the programs that encode the concept, through the router, to the student's preference for the target. We also measure the teacher's response on the same perturbed states (Table 3) for reference. For all three concepts, the intervention reduces the relevant programs' contribution and the student's preference for the target, and the teacher's preference shifts in the same direction on average. These results support the causal role of the tested concepts in student decisions and indicate that the explanations partially reflect the teacher's behavior. Details are in §D.7. For a fourth concept, local demand density, the teacher also responds in the predicted direction on average, though more weakly than the student (§D.7.4).

Table 3: Counterfactual faithfulness tests. We report the mean effect g (control minus treatment, $g > 0$ is the predicted direction) with 95% instance-level bootstrap CIs, and the fraction of samples with $g > 0$ in parentheses. †Manipulation check only.

Channel	POMO-TSP hull	LEHD-TSP isolation	POMO-CVRP corridor
Carrier score (Eq. 22)†	+0.38 [0.34, 0.43] (100%)	+1.20 [1.18, 1.23] (100%)	+0.92 [0.90, 0.94] (100%)
Routing log-weight (Eq. 23)	+1.54 [1.08, 1.99] (73%)	+0.29 [0.20, 0.37] (72%)	+0.39 [0.32, 0.46] (61%)
Program contribution (Eq. 24)	+0.38 [0.33, 0.43] (92%)	+0.08 [0.06, 0.09] (74%)	+0.25 [0.24, 0.26] (91%)
Student log-odds (Eq. 25)	+1.09 [0.80, 1.38] (83%)	+0.54 [0.45, 0.63] (71%)	+3.14 [3.00, 3.28] (87%)
Teacher log-odds (Eq. 26)	+9.57 [7.69, 11.37] (84%)	+1.47 [1.24, 1.74] (76%)	+9.79 [9.29, 10.29] (83%)

Table 4: Out-of-distribution generalization results on unseen problem sizes.

	Greedy			Search		
Experiment	Teacher	EPB	Gap (%)	Teacher	EPB	Gap (%)
POMO-TSP-500	22.190	18.954	−14.58	20.479	17.762	−13.27
POMO-CVRP-500	50.103	43.683	−12.81	48.124	41.042	−14.72
LEHD-TSP-50	5.761	5.919	+2.74	5.728	5.729	+0.03
LEHD-TSP-500	16.807	17.633	+4.91	16.571	17.107	+3.24

Program-removal tests further show that removing the program with the highest routing weight usually changes the student's action, whereas removing a randomly chosen program rarely does, confirming that dominant programs causally drive the student's decisions (§D.8).

Trade-off between code complexity and solution quality. The complexity of the program affects interpretability. Simpler programs are easier to interpret but may sacrifice fidelity. To study this trade-off, we run EPB on TSP-100 with per-program limits on the number of distinct feature primitives, executable statements, and branches (§D.9). In the simplest setting, each program uses at most two primitives. We then prune each bank one program at a time and keep the size with the lowest search gap, which gives 8 programs for POMO and 10 for LEHD. Including shared primitive implementations, the total statement count of the bank falls from 537 to 51 for POMO and from 419 to 49 for LEHD. Despite this reduction, solution cost gaps to the teachers remain small, rising from 0.03% to 0.58% for POMO and from 0.17% to 0.33% for LEHD. They are also well below those of the LLM coding-agent baselines, which use a similar amount of code. This shows that EPB can retain near-teacher solution quality with simple program code, facilitating a more interpretable surrogate.

5.3 Generalization and Improvement

Generalization to larger instances. Table 4 reports zero-shot transfer to unseen sizes. EPB remains close to LEHD, which is designed to generalize across sizes. Notably, on both TSP and CVRP, the POMO surrogate significantly outperforms its teacher on larger instances, suggesting that distillation may yield more generalizable, less overfitted decision rules, possibly because EPB always combines relatively size-agnostic programs rather than relying on embeddings fitted to the training size.

Intervention through program edits. The readable programs of EPB also let us diagnose and fix failures by directly editing code. We test this on out-of-distribution diagonal-band instances (nodes

Table 5: Intervention results on EPB surrogates. Gaps (%) are relative to LKH for TSP and HGS for CVRP. Entries report greedy / search.

Policy	TSP-100 Diagonal band	CVRP-50 Diagonal band
Teacher	38.18 / 20.90	13.34 / 7.66
EPB	24.16 / 8.70	10.88 / 4.17
Edited Teacher	36.85 / 15.40	7.40 / 4.06
Edited EPB	**7.01 / 1.78**	**5.94 / 2.43**

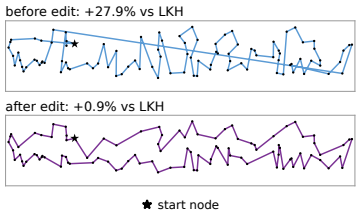


Figure 6: EPB before and after editing on a TSP-100 diagonal-band instance.

on a thin diagonal band). On TSP-100, all programs initially favor nearby nodes without reserving any for the return leg, leaving a long closing edge (Figure 6). We add distance penalties and three lane-keeping programs that go out along one side of the band and back along the other. As shown in Table 5, the edited EPB performs best. A black-box teacher, by contrast, cannot be edited directly. The most direct option is to add the same rule's score as a bias to its log-probabilities, which helps but less than editing EPB's programs. A similar edit also improves CVRP-50 (see Table 5 and §D.10).

5.4 Ablation Studies

Ablation on hybrid gradients. Figure 7 (a) shows that numerical gradients alone (blue circles) yield limited gains. TextGrad (purple triangles) accelerates early training, but both saturate at fixed bank size.

Ablation on bank size management. In Figure 7 (a), further activating bank size evolution (green squares) at epoch 25 produces a clear accuracy jump to break the plateau of fixed-size hybrid gradient descent.

Ablation on evolving banks. Table 6 further shows that EPB achieves lower greedy and search gaps than two frozen baselines, which train only the router over a fixed bank initialized either by *Expert* or by an *LLM* (both released on GitHub). Expert's larger gaps suggest that even a well-designed initial bank cannot match the teacher, so the programs should be evolved.

Ablation on dynamic routing. Figure 7(b) shows that state-dependent dynamic keys (purple circles, §4.1) outperform static keys (blue squares), with a clear gap already at epoch 1 despite identical initial program banks. A third variant, which builds keys from text embeddings of the program descriptions, performs similarly to static keys (§B.3), since both describe a program identically in every state, whereas dynamic keys reflect its behavior in each state.

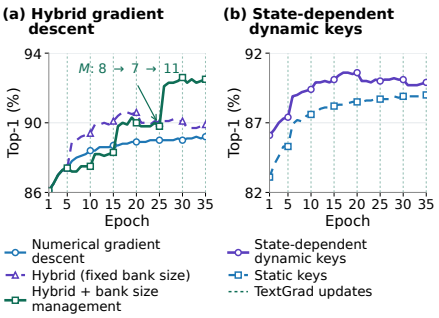


Figure 7: Ablation studies.

Table 6: Cost gap (%) to the teacher (↓) with frozen banks (Expert- or LLM-initialized) versus evolving programmatic banks (EPB).

		TSP			CVRP		
Teacher	N	Expert	LLM	Ours	Expert	LLM	Ours
Greedy gap (%) ↓							
POMO	50	13.76	16.26	**3.02**	21.90	22.55	**2.88**
POMO	100	16.02	12.45	**3.79**	23.27	12.38	**3.94**
LEHD	100	6.62	7.78	**2.57**	23.98	7.40	**3.53**
Search gap (%) ↓							
POMO	50	3.54	3.53	**0.03**	10.49	10.74	**1.13**
POMO	100	4.43	2.65	**0.07**	12.34	5.03	**1.25**
LEHD	100	0.98	1.08	**0.34**	13.42	5.23	**2.16**

6 Conclusion

We introduce Evolving Programmatic Bottlenecks (EPB), the first post-hoc interpretable surrogate framework that distills NCO policies into executable decision rules composed by a state-dependent router. Hybrid textual-numerical gradient descent refines the programs and router, while bank size management adapts the number of programs. Across POMO and LEHD on TSP and CVRP, EPB closely matches teacher behavior, and counterfactual and knockout tests support the causal faithfulness of its rules to the surrogate's decisions. The surrogate also outperforms POMO on larger instances, and targeted program edits improve its out-of-distribution performance beyond the teacher.

While EPB takes a first step toward surrogate-based interpretability for NCO, it has several limitations. First, the router remains a neural network whose internal computation is not interpretable. A programmatic replacement may help but must retain sufficient expressiveness as suggested by our capacity ablation (§D.11). Second, the failure-attribution score suits near-deterministic teachers, while stochastic teachers may require distribution-level measures. Third, EPB explains teacher behavior rather than internal mechanisms. Finally, discrete LLM rewrites can abruptly change a program's behavior, destabilizing refinement. Future work could stabilize refinement and extend EPB, and more broadly the general idea of *programmatic bottleneck models* to other sequential decision-making models, including more state-of-the-art NCO models, models with continuous action spaces, and other black-box solvers beyond neural networks, using interpretable surrogates to diagnose behavior patterns, guide policy improvement, and enable trustworthy deployment.

Acknowledgments and Disclosure of Funding

This research is supported by the National Research Foundation, Singapore under its AI Singapore AI Research Fundamental Research Collaborative (US-NSF Researcher Call) (AISG Award No: AISG3-RP-2025-036-USNSF). The authors would also like to thank Dr. Been Kim (Google DeepMind) and Prof. Bistra Dilkina (University of Southern California) for their helpful and insightful discussions during the early stages of this work.

References

[1] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

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

[3] 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.

[4] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[5] David Steinmann, Wolfgang Stammer, Felix Friedrich, and Kristian Kersting. Learning to intervene on concept bottlenecks. *arXiv preprint arXiv:2308.13453*, 2023.

[6] Shirley Wu, Mert Yuksekgonul, Linjun Zhang, and James Zou. Discover and cure: Concept-aware mitigation of spurious correlation. In *International Conference on Machine Learning*, pages 37765–37786. PMLR, 2023.

[7] Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

[8] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International conference on machine learning*, pages 5338–5348. PMLR, 2020.

[9] Shantanu Ghosh, Ke Yu, Forough Arabshahi, and Kayhan Batmanghelich. Dividing and conquering a blackbox to a mixture of interpretable models: route, interpret, repeat. *Proceedings of machine learning research*, 202:11360, 2023.

[10] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[11] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015.

[12] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[13] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

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

[15] Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. Simulation-guided beam search for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:8760–8772, 2022.

[16] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. In *International Conference on Learning Representations*, 2022.

[17] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In *Advances in Neural Information Processing Systems*, volume 34, pages 11096–11107, 2021.

[18] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[19] Zhiqin Zhang, Yining Ma, Zhiguang Cao, and Hoong Chuin Lau. Probing neural combinatorial optimization models. *arXiv preprint arXiv:2510.22131*, 2025.

[20] Atticus Geiger, Duligur Ibeling, Amir Zur, Maheep Chaudhary, Sonakshi Chauhan, Jing Huang, Aryaman Arora, Zhengxuan Wu, Noah Goodman, Christopher Potts, and Thomas Icard. Causal abstraction: A theoretical foundation for mechanistic interpretability. *arXiv preprint arXiv:2301.04709*, 2023.

[21] Eric J Michaud, Isaac Liao, Vedang Lad, Ziming Liu, Anish Mudide, Chloe Loughridge, Zifan Carl Guo, Tara Rezaei Kheirkhah, Mateja Vukelić, and Max Tegmark. Opening the ai black box: program synthesis via mechanistic interpretability. *arXiv preprint arXiv:2402.05110*, 2024.

[22] Quentin Delfosse, Hikaru Shindo, Devendra Dhami, and Kristian Kersting. Interpretable and explainable logical policies via neurally guided symbolic abstraction. In *Advances in Neural Information Processing Systems*, 2023.

[23] Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J. Lim. Learning to synthesize programs as interpretable and generalizable policies. In *Advances in Neural Information Processing Systems*, 2021.

[24] Mark Craven and Jude Shavlik. Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, 8, 1995.

[25] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

[26] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[27] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. *Advances in neural information processing systems*, 34:4699–4711, 2021.

[28] Marko Vasić, Andrija Petrović, Kaiyuan Wang, Mladen Nikolić, Rishabh Singh, and Sarfraz Khurshid. Moët: Mixture of expert trees and its application to verifiable reinforcement learning. *Neural Networks*, 151:34–47, 2022.

[29] Ricardo Moreira, Jacopo Bono, Mário Cardoso, Pedro Saleiro, Mário AT Figueiredo, and Pedro Bizarro. Diconstruct: Causal concept-based explanations through black-box distillation. *arXiv preprint arXiv:2401.08534*, 2024.

[30] Abhinav Verma, Vijay Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, 2018.

[31] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie J. Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International Conference on Machine Learning*, 2018.

[32] Amirata Ghorbani, James Wexler, James Y. Zou, and Been Kim. Towards automatic concept-based explanations. In *Advances in Neural Information Processing Systems*, 2019.

[33] Mert Yuksekgonul, Maggie Wang, and James Zou. Post-hoc concept bottleneck models. *arXiv preprint arXiv:2205.15480*, 2022.

[34] Tuomas Oikarinen, Subhro Das, Lam M Nguyen, and Tsui-Wei Weng. Label-free concept bottleneck models. *arXiv preprint arXiv:2304.06129*, 2023.

[35] Divyansh Srivastava, Ge Yan, and Tsui-Wei Weng. Vlg-cbm: Training concept bottleneck models with vision-language guidance. In *Advances in Neural Information Processing Systems*, 2024.

[36] Akshay Kulkarni, Ge Yan, Chung-En Sun, Tuomas Oikarinen, and Tsui-Wei Weng. Interpretable generative models through post-hoc concept bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8162–8171, 2025.

[37] Chung-En Sun, Tuomas Oikarinen, Berk Ustun, and Tsui-Wei Weng. Concept bottleneck large language models. In *International Conference on Learning Representations*, 2025.

[38] Jean Feng, Avni Kothari, Lucas Zier, Chandan Singh, and Yan Shuo Tan. Bayesian concept bottleneck models with llm priors. *Advances in Neural Information Processing Systems*, 38:93889–93920, 2026.

[39] Francesco De Santis, Gabriele Ciravegna, Giovanni De Felice, Arianna Casanova, Francesco Giannini, Michelangelo Diligenti, Johannes Schneider, Danilo Giordano, Mateo Espinosa Zarlenga, and Pietro Barbiero. Mixture of concept bottleneck experts. *arXiv preprint arXiv:2602.02886*, 2026.

[40] Zhuorui Ye, Stephanie Milani, Geoff Gordon, and Fei Fang. Licorice: Label-efficient concept-based interpretable reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pages 102754–102782, 2025.

[41] Utkarsh Mall, Cheng Perng Phoo, Mia Chiquier, Bharath Hariharan, Kavita Bala, and Carl Vondrick. Disciple: Learning interpretable programs for scientific visual discovery. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29258–29267. IEEE, 2025.

[42] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[43] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[44] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[45] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[46] 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.

[47] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

[48] 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.

[49] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[50] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[51] Robert Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, volume 2026, pages 74026–74078, 2026.

Interpreting Neural Combinatorial Optimization via Evolving Programmatic Bottlenecks (Appendix)

A Preliminary **15**

A.1 TSP and CVRP 15

A.2 Teacher/Student NCO model Structure 15

B Methodology **16**

B.1 Hierarchical Symbolic Optimization 16

B.2 On the Necessity of the Stop-gradient 17

B.3 Routing Key Design 17

B.4 Derivation of the Hybrid Gradient Decomposition 18

B.5 Add and Drop Algorithms 20

C Experimental Setup **21**

C.1 Hyperparameters 21

C.2 Prompt Setting 21

D Additional Results and Discussions **27**

D.1 Sensitivity to Random Seeds 27

D.2 LLM Cost 27

D.3 Fidelity 28

D.4 Heuristic Gallery[1] 29

D.5 Code Example 32

D.6 Interpreting each step - more examples 34

D.7 Trustworthy Concept Attributions (Counterfactual Tests) 35

D.8 Trustworthy Program attribution (Knockout Tests). 39

D.9 Trade-off Between Code Complexity and Solution Quality 39

D.10 Improvements through Program Edits 42

D.11 Overly shallow routing 43

E LLM Statement **43**

F Social Impact Statement **43**

G License **43**

A Preliminary

A.1 TSP and CVRP

TSP The Traveling Salesman Problem (TSP) is defined on a set of n nodes, where each node i is associated with a coordinate $x_i \in \mathbb{R}^2$. A problem instance is denoted by $s = \{x_1, \ldots, x_n\}$, typically sampled i.i.d. from a distribution over the plane (e.g., uniform in the unit square). A feasible solution to TSP is a permutation $\pi = (\pi_1, \ldots, \pi_n)$ of the nodes, representing a tour that visits each node exactly once and returns to the starting node. The total cost of a tour is defined as

$$L(\pi \mid s) = \sum_{t=1}^{n} d(x_{\pi_t}, x_{\pi_{t+1}}),$$

where $d(\cdot, \cdot)$ denotes the Euclidean distance between the two nodes and $\pi_{n+1} = \pi_1$. The objective of TSP is to find a permutation π^* that minimizes the total tour cost: $\pi^* = \arg\min_\pi L(\pi \mid s)$. In neural combinatorial optimization settings, solutions are constructed by sequentially selecting the next node to visit conditioned on the current state:

$$p(\pi \mid s) = \prod_{t=1}^{n} p(\pi_t \mid s, \pi_{1:t-1}),$$

where previously visited nodes are excluded from the candidate set to ensure valid tours.

CVRP The Capacitated Vehicle Routing Problem (CVRP) is defined on a set of n customer nodes and a depot. Each customer node i is associated with a location x_i in $\mathbb{R}^2$ and a demand $d_i > 0$, and the depot has location x_0. Node locations and customer demands are drawn from a fixed distribution. A feasible solution to CVRP consists of a set of routes, where each route starts and ends at the depot and visits a subset of customers. The total demand served in each route must not exceed the vehicle capacity C. The objective is to construct a set of routes that visits all customers while minimizing the total travel cost:

$$L(\pi \mid s) = \sum_{\text{routes}} \sum_{t=1}^{|\pi|} d(x_{\pi_t}, x_{\pi_{t+1}}),$$

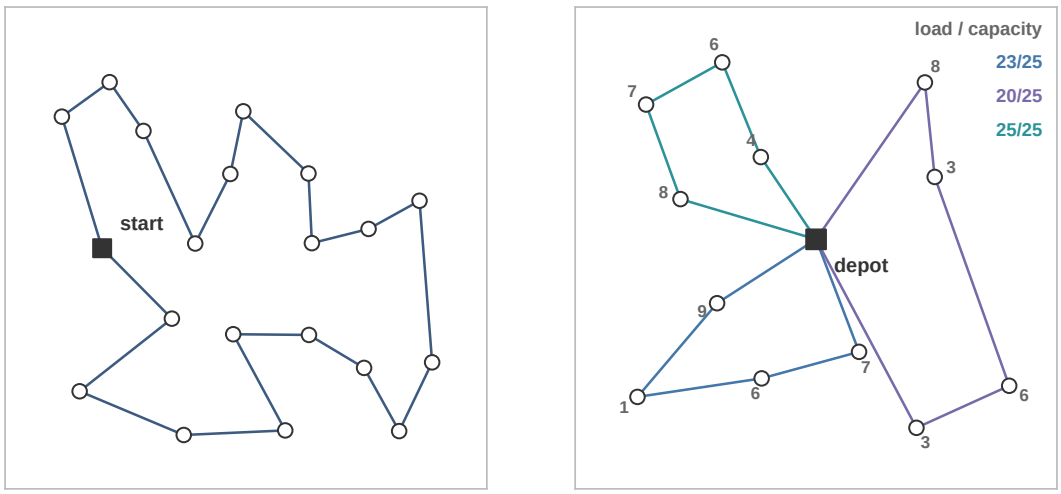
where each route is represented as a sequence π of visited nodes, and $\pi_{|\pi|+1}$ includes returning to the depot at the end of the route. In neural combinatorial optimization settings, solutions are generated by sequentially selecting the next node conditioned on the current partial route and the remaining vehicle capacity. Nodes who have already been served or whose demand exceeds the remaining capacity are masked to ensure feasibility.

A.2 Teacher/Student NCO model Structure

We use POMO [2] and LEHD [3] as representative black-box NCO teachers. Both models define an autoregressive policy over feasible future actions, but they differ in architecture, training procedure, and inference strategy.

POMO POMO is an attention-based reinforcement learning solver for combinatorial optimization. Its key idea is to harness the fact that many routing solutions have multiple equivalent solutions (evaluated by the relevant downstream metric). For example, a TSP solution cycle can be represented by starting from any city. POMO uses this symmetry by generating multiple rollout trajectories from different starting nodes for each problem instance during training and inference. This encourages the model to explore multiple equivalent optima rather than anchoring to a single starting point.

LEHD LEHD was proposed to improve generalization to large-scale problem instances. Earlier constructive NCO models often use a heavy-encoder/light-decoder structure, where node embeddings are computed once by the encoder and then reused throughout sequential decoding. This static embedding strategy can encourage the model to learn scale-dependent features, which may hurt performance when generalizing to larger instances. LEHD instead uses a lightweight encoder and a heavy decoder that dynamically captures relationships among the current partial solution and the available nodes at each construction step. LEHD is further trained with partial-solution augmentation.



(a) TSP: one closed tour through all *n* nodes

(b) CVRP: capacity-feasible routes from the depot

Figure 8: **(a)** TSP: a closed tour visiting each node once. **(b)** CVRP: routes start and end at the depot (square), each with total demand at most C. Numbers indicate customer demands.

Given an optimal tour, it samples consecutive subsequences with random lengths and directions, and treats each subsequence as an augmented training example. The model then learns to reconstruct each partial solution step by step. At inference time, LEHD constructs full solutions greedily and can further improve them with Random Re-Construct, which repeatedly reconstructs sampled solution segments and accepts a replacement only if it improves the objective.

Student Model Similar to Concept Bottleneck Models, our student router is designed to follow the architecture of the corresponding teacher model while exposing an interpretable intermediate decision layer. For both TSP and CVRP, the goal is to obtain node embeddings that serve as the input representation for the router network. For POMO-based teachers, we train a POMO encoder to compute these embeddings. For LEHD-based teachers, we use the LEHD encoder together with few decoder layers to obtain the node embeddings.

B Methodology

B.1 Hierarchical Symbolic Optimization

A heuristic can fail for two structurally different reasons. Its implementation may have a local bug, or its underlying strategy may be wrong for the failure mode it covers. This motivates a hierarchical strategy that updates the natural language description d_m and the code implementation c_m of each heuristic in a self-consistent way. A dispatcher selects which heuristics enter this update phase based on accumulated failure attribution scores from Section 4.2.1. Phase 1 attempts local code refinement, and Phase 2 escalates to strategy-level revision only when Phase 1 cannot resolve the failure. The TextGrad framework gives us two distinct channels per bank entry: backward calls on the code c_m, and backward calls on the natural-language description d_m. We do not run the two channels simultaneously. Instead, we organize them into a two-phase hierarchical procedure: **Phase 1** acts only on c_m, attempting local code refinement, and **Phase 2** acts only on d_m (with c_m subsequently re-implemented from the revised description), invoked only when Phase 1 cannot resolve the failure.

For each selected heuristic, Phase 1 attempts up to R rounds of TextGrad on c_m. After each update, the candidate c_m is evaluated on a held-out validation set. If the validation loss with the teacher distribution improves over the pre-substitution bank's by at least δ, the candidate is retained and d_m is regenerated from the new code to reflect the updated implementation.

If no update results from Phase 1, this indicates the failure is strategy-level rather than implementation-level. Phase 2 then updates d_m via TextGrad on description-level gradients, and c_m is re-implemented

from the revised d_m. The new code is committed under the same validation criterion. If both phases fail to commit a new heuristic, the description is reverted and the heuristic is left unchanged. In this way, the bank maintains internally aligned (c_m, d_m) pairs regardless of the sequence of changes.

TextGrad backward output formats

TextGrad's backward LLM call produces a structured natural-language diagnostic in one of two formats, depending on the channel.

Code channel (backward on c_m, four fields):

```
LOGIC:     [what the code currently computes for ranking
            these top candidates]
TEACHER:   [what features of the teacher's pick the code
            should recognize]
FLAW:      [why the code's score for the teacher's pick
            fell behind the wrong argmax]
DIRECTION: [what change would push the teacher's pick
            above the wrong argmax]
```

Description channel (backward on d_m, three fields):

```
MISSING:    [what geometric reasoning the description
             omits or under-emphasizes]
MISLEADING: [anything in the description that points
             the implementation in the wrong direction]
DIRECTION:  [how the description should be revised]
```

B.2 On the Necessity of the Stop-gradient

The stop-gradient in Eq. 3 is a design choice rather than a requirement. The router parameters θ are continuous and updated by SGD, while the heuristic outputs $V_m(s)$ are produced by executable code and revised by TextGrad; $V_m(s)$ is never updated by gradients regardless of whether the path is stopped. Removing the stop-gradient would only add a second term to $\nabla_{V_m(s)}\ell_s$, flowing through the routing key K_m. Since this gradient is used not as an update but as an attribution signal for selecting which heuristic receives textual feedback (Eq. 7), stopping the routing path keeps the signal clean: it isolates $V_m(s)$'s direct effect on the mixture from its indirect effect on routing, so that the gradients on θ and on $V_m(s)$ no longer share a path and update their respective channels independently. The resulting gradient, $\nabla_{V_m(s)}\ell_s = w_m(s; \theta, \mathrm{sg}(V))r_s$, scales the shared mixture-level error r_s by the program's routing weight. Since this gradient alone cannot determine whether a program is wrong, the attribution score in Eq. 7 additionally incorporates each program's top-action disagreement and KL divergence from the teacher to allocate textual feedback.

B.3 Routing Key Design

Table 7: Distillation KL divergence for different routing keys with a fixed heuristic bank.

	Training epoch						
Key design	1	10	20	30	40	50	60
State-dependent Dynamic Keys	**0.0252**	**0.0189**	**0.0176**	**0.0171**	**0.0167**	**0.0166**	**0.0167**
Static Keys	0.0359	0.0242	0.0219	0.0203	0.0197	0.0192	0.0191
Text Embedding Keys	0.0353	0.0231	0.0212	0.0203	0.0195	0.0193	0.0191

Setup. We compare EPB's dynamic keys with two static alternatives while keeping the heuristic bank fixed and training only the router. EPB constructs each key as the expectation of state-dependent node embeddings under the corresponding heuristic's action distribution. The *static* variant assigns each heuristic a freely learnable embedding $e_m \in \mathbb{R}^D$ that is independent of the current state. The *text* variant encodes each heuristic's natural-language description d_m using the frozen OpenAI `text-embedding-3-large` model and projects the embedding to the shared key dimension:

$$z_m = E_{\text{text}}(d_m) \in \mathbb{R}^{3072}, \qquad K_m^{\text{text}} = W z_m, \qquad W \in \mathbb{R}^{D \times 3072}.$$

The text embeddings remain fixed, while the projection is learned during router training. We report the distillation KL divergence to the teacher at selected training epochs.

Results. EPB's dynamic keys achieve lower KL divergence at every reported epoch (Table 7). At epoch 60, both static alternatives reach 0.0191, compared with 0.0167 for EPB. The lowest reported value for EPB is 0.0166 at epoch 50. Text-derived keys closely track freely learned static keys and provide no improvement at the final reported epoch. Both static alternatives allow state-dependent routing through the learned query, but their keys do not reflect what each heuristic recommends in the current state.

Figure 9 compares dynamic and static keys under two training schemes. Numerical training updates only the router while keeping the heuristic bank fixed. Hybrid training also revises the programs through TextGrad. Dynamic keys yield higher top-1 agreement under both schemes, while hybrid training further improves agreement for either key design. The combination of hybrid training and dynamic keys achieves the highest agreement. These results support incorporating each heuristic's current action distribution into its routing key, both with a fixed bank and during program evolution.

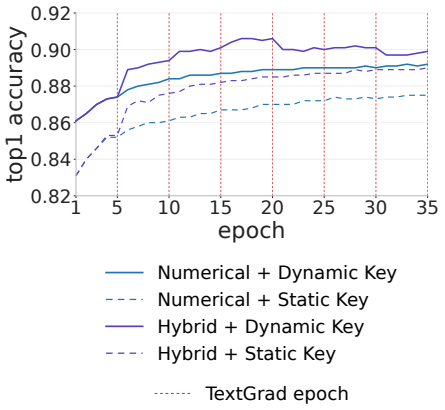


Figure 9: Top-1 agreement with the teacher under numerical and hybrid training. Red vertical lines mark epochs that perform TextGrad.

B.4 Derivation of the Hybrid Gradient Decomposition

We derive the gradient decomposition used in §4.2.1. For a fixed decision state s, the student policy is the routing-weighted mixture

$$\pi_S(\cdot \mid s) = \sum_{m=1}^{M} w_m(s; \theta, \mathrm{sg}(V))\, V_m(s). \tag{9}$$

For readability, write

$$\bar{V} = \mathrm{sg}(V), \qquad w_m = w_m(s; \theta, \bar{V}), \qquad V_m = V_m(s).$$

Then Eq. 9 becomes

$$\pi_S(\cdot \mid s) = \sum_{m=1}^{M} w_m V_m. \tag{10}$$

Consider small perturbations

$$V_m \mapsto V_m + \delta V_m, \qquad \theta \mapsto \theta + \delta\theta.$$

The corresponding first-order perturbation of the student mixture is

$$\begin{aligned} \delta\pi_S(\cdot \mid s) &= \sum_{m=1}^{M} \Big[(w_m + \delta w_m)(V_m + \delta V_m) - w_m V_m\Big] \\ &= \sum_{m=1}^{M} (w_m\, \delta V_m + V_m\, \delta w_m) + O(\|\delta\|^2), \end{aligned} \tag{11}$$

where the second-order term (i.e. $\sum_m \delta w_m\, \delta V_m$) is omitted.

The routing weights depend on θ and on the detached heuristic outputs $\bar{V} = \mathrm{sg}(V)$, as in Eq. 2. Hence their first-order perturbation is

$$\delta w_m = \left\langle \nabla_\theta w_m(s; \theta, \bar{V}), \delta\theta \right\rangle + \left\langle \nabla_{\bar{V}} w_m(s; \theta, \bar{V}), \delta\bar{V} \right\rangle. \tag{12}$$

Since $\bar{V} = \mathrm{sg}(V)$, perturbations of V do not propagate through the router:

$$\delta\bar{V} = 0, \qquad \frac{\partial w_j(s;\theta,\mathrm{sg}(V))}{\partial V_m(s)} = 0 \qquad \forall j, m \in [M]. \tag{13}$$

Therefore,

$$\delta w_m = \langle \nabla_\theta w_m(s;\theta,\mathrm{sg}(V)), \delta\theta \rangle . \tag{14}$$

Substituting Eq. 14 into Eq. 11 gives

$$\delta\pi_S(\cdot \mid s) = \sum_{m=1}^{M} w_m(s;\theta,\mathrm{sg}(V))\, \delta V_m(s) + \sum_{m=1}^{M} V_m(s)\, \langle \nabla_\theta w_m(s;\theta,\mathrm{sg}(V)), \delta\theta \rangle . \tag{15}$$

Now define the mixture-level gradient

$$r_s = \nabla_{\pi_S(\cdot|s)} \ell_s(\theta, \mathcal{B}). \tag{16}$$

By the first-order Taylor expansion of the loss around the current student output,

$$\delta\ell_s \approx \langle r_s, \delta\pi_S(\cdot \mid s) \rangle . \tag{17}$$

Combining Eq. 15 and Eq. 17, we obtain

$$\begin{aligned} \delta\ell_s &\approx \sum_{m=1}^{M} \langle r_s, w_m(s;\theta,\mathrm{sg}(V))\, \delta V_m(s) \rangle \\ &\quad + \sum_{m=1}^{M} \langle r_s, V_m(s) \rangle\, \langle \nabla_\theta w_m(s;\theta,\mathrm{sg}(V)), \delta\theta \rangle \\ &= \sum_{m=1}^{M} \langle w_m(s;\theta,\mathrm{sg}(V))\, r_s, \delta V_m(s) \rangle \\ &\quad + \left\langle \sum_{m=1}^{M} \langle r_s, V_m(s) \rangle\, \nabla_\theta w_m(s;\theta,\mathrm{sg}(V)), \delta\theta \right\rangle . \end{aligned} \tag{18}$$

Since a first-order perturbation can also be written as

$$\delta\ell_s \approx \sum_{m=1}^{M} \left\langle \nabla_{V_m(s)} \ell_s, \delta V_m(s) \right\rangle + \langle \nabla_\theta \ell_s, \delta\theta \rangle ,$$

comparison with Eq. 18 yields

$$\nabla_{V_m(s)} \ell_s = w_m(s;\theta,\mathrm{sg}(V))\, r_s, \tag{19}$$

and

$$\nabla_\theta \ell_s = \sum_{m=1}^{M} \langle r_s, V_m(s) \rangle\, \nabla_\theta w_m(s;\theta,\mathrm{sg}(V)). \tag{20}$$

The first branch, $\nabla_{V_m(s)} \ell_s = w_m(s) r_s$, is the direct mixture-path sensitivity used for textual failure attribution. The second branch, $\nabla_\theta \ell_s$, is the standard numerical gradient used to update the routing network. Because the router observes $\mathrm{sg}(V)$, the indirect path from heuristic outputs to the loss through the router,

$$V_m \to w \to \pi_S \to \ell_s,$$

is removed. Thus the heuristic-output sensitivity contains only the direct path

$$V_m \to \pi_S \to \ell_s.$$

B.5 Add and Drop Algorithms

In this appendix, we present the algorithmic details of slot add and drop in Algorithms 2 and 3.

Algorithm 2 Failure-Mode-Targeted Bank Expansion (Add)

Require: Existing bank $\mathcal{B}$, dataset $\mathcal{D}$, validation set $\mathcal{D}_{\text{val}}$, teacher, LLM, threshold ρ_{admit}
Ensure: Expanded bank $\mathcal{B}^+$

$\mathcal{F} \leftarrow$ decision steps on which every $h \in \mathcal{B}$ performs poorly
$\mathcal{C} \leftarrow$ LLM-PROPOSE$(\mathcal{F})$ ▷ LLM attributes failures into K buckets

// Pretrain candidates on $\mathcal{F}$
for round $t = 1, \ldots, T$ **do**
  for each $s \in \mathcal{F}$ **do** ▷ assign each fault case to the best current candidate
    assign s to $h \in \mathcal{C}$ whose output is closest to the teacher's
  end for
  for each $h \in \mathcal{C}$ **do** ▷ refine each candidate on its assigned fault cases
    $\nabla_h \leftarrow$ TEXTGRAD$(h,$ samples assigned to $h)$
    $h' \leftarrow$ LLM-EDIT(h, ∇_h)
    replace $h \leftarrow h'$ **if** h' improves agreement on a held-out split
  end for
end for

// Greedy admission by marginal coverage gain
$\mathcal{B}^+ \leftarrow \mathcal{B}$
while $\mathcal{C} \neq \emptyset$ **do**
  $h^\star \leftarrow \arg\max_{h \in \mathcal{C}} \left[C(\mathcal{B}^+ \cup \{h\}; \mathcal{D}_{\text{val}}) - C(\mathcal{B}^+; \mathcal{D}_{\text{val}}) \right]$
  $\Delta^\star \leftarrow C(\mathcal{B}^+ \cup \{h^\star\}; \mathcal{D}_{\text{val}}) - C(\mathcal{B}^+; \mathcal{D}_{\text{val}})$
  if $\Delta^\star < \rho_{\text{admit}}$ **then**
    break
  end if
  $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \cup \{h^\star\}; \quad \mathcal{C} \leftarrow \mathcal{C} \setminus \{h^\star\}$
end while
return $\mathcal{B}^+$

Algorithm 3 Redundant-Target Removing (Drop)

Require: Current bank $\mathcal{B}$, validation set $\mathcal{D}_{\text{val}}$, student model, tolerance ϵ, max drops per call $D_{\max}$, minimum bank size $M_{\min}$
Ensure: Pruned bank $\mathcal{B}^-$

$\mathcal{B}^- \leftarrow \mathcal{B}; \quad d \leftarrow 0$
$L_0 \leftarrow \mathcal{L}_{\text{val}}(\mathcal{B}^-; \mathcal{D}_{\text{val}})$ ▷ baseline validation loss

while $d < D_{\max}$ **and** $|\mathcal{B}^-| > M_{\min}$ **do**
  for each $h \in \mathcal{B}^-$ **do**
    $L_h \leftarrow \mathcal{L}_{\text{val}}(\mathcal{B}^- \setminus \{h\}; \mathcal{D}_{\text{val}})$
    $\delta_h \leftarrow L_h - L_0$ ▷ degradation from removing h
  end for
  $h^\star \leftarrow \arg\min_{h \in \mathcal{B}^-} \delta_h$
  if $\delta_{h^\star} > \epsilon$ **then**
    break ▷ no safe removal remains
  end if
  $\mathcal{B}^- \leftarrow \mathcal{B}^- \setminus \{h^\star\}; \quad L_0 \leftarrow L_{h^\star}; \quad d \leftarrow d + 1$
end while
return $\mathcal{B}^-$

C Experimental Setup

All experiments were conducted on a workstation equipped with an NVIDIA RTX A6000 GPU. Unless otherwise specified, we use Claude Sonnet 4.6 and GPT-5.4 as the LLMs for heuristic generation and TextGrad rewriting. For LEHD-TSP, we instead adopt GPT-5 as the LLM optimizer. On a single NVIDIA RTX A6000, training EPB takes roughly 10 to 72 hours per setting.

C.1 Hyperparameters

Table 8: Key hyperparameters of the selected student checkpoints. The TextGrad interval is the number of batches between TextGrad backward passes.

Parameter	**TSP-50**	**TSP-100**	**CVRP-50**	**LEHD-TSP-100**
Encoder layers	3	4	6	3
Batch size	4,096	2,048	16,384	1,024
Learning rate	6×10^{-4}	3×10^{-4}	4×10^{-4}	1×10^{-4}
Epochs	144	80	50	50
TextGrad interval	100	200	200	200

Table 8 summarizes the key hyperparameters of the selected student checkpoints. All student models use an embedding dimension of 128 and 8 attention heads. The LEHD-TSP student uses a three-layer light encoder and a three-layer heavy decoder.

C.2 Prompt Setting

C.2.1 Pipeline overview

The training loop has two LLM-driven phases. **(I) Proposal/init.** For each new heuristic slot, the LLM is asked to (a) propose a strategy in natural language, then (b) implement that strategy as a Python module. (II) TextGrad refinement. In TextGrad epochs, every n batches (Table 8), failure cases (where the heuristic's argmax disagrees with the LEHD teacher's) drive a backward pass that produces text gradients on the code and on the description; an optimizer prompt then synthesises those gradients into an updated module. A single shared context block (`_SPARSE_TEACHER_CONTEXT`) is embedded into every system prompt so all calls share the same problem framing.

C.2.2 Shared context

Shared context (_SPARSE_TEACHER_CONTEXT)

```
**Subgraph decoder formulation (LEHD-style).**
At each decoding step the heuristic receives ONLY the unvisited subgraph
plus the start (first_loc) and current end (current_loc) of the partial tour.
Visited nodes are NOT shown -- the input subgraph IS the remaining problem.
The output is a score over the K candidate next-nodes, where K shrinks across
the rollout from N-1 down to 1.  Your code MUST work for ANY K.

Progress within the tour is implicit: it is exposed ONLY through K
(= unselected_locs.size(1)).  No global N, step index, or step_norm is
    provided
-- this matches what the LEHD teacher itself sees.  If your strategy needs a
"how-far-along" signal, derive it from the geometry of unselected_locs (e.g.
spread, mean distance to first_loc) or from K itself, and AVOID hard-coded
thresholds tied to a particular N.

**Critical: the teacher distribution is near one-hot.**
The teacher policy is a near-deterministic LEHD model:
  - top-1 candidate receives ~95% probability
  - top-2 candidate receives ~4.5% probability
  - the remaining K-2 candidates share less than 0.5% combined
```

```
Two implications for heuristic design:

1. **Output sharp, decisive scores.** The winning candidate's score should be
   substantially higher than the runner-up. After softmax with the training
   temperature, your output should look near-one-hot in clear-cut situations
   -- not a smoothly graded distribution. A decisive winner gap matters more
   than carefully shaping the long tail.

2. **Top-1 is what matters; top-2 is a soft backup; the rest is irrelevant.**
   Don't waste complexity on shaping the bottom of the distribution. A simple
   "score = -distance + lookahead_bonus" can outperform an 8-signal weighted
   mixture if the simpler one consistently picks the right top-1.

Heuristics are evaluated on:
  (a) whether their argmax matches the teacher's argmax  (primary)
  (b) whether their top-2 contains the teacher's argmax  (secondary)
The full distribution shape does not matter. Avoid over-engineering for it.
```

C.2.3 Phase I — initial proposal

Propose — `_make_idea_prompt` **(system)**

Propose strategy (system)

```
You are an expert in the 2D symmetric Euclidean Travelling Salesman Problem.
Training is on {problem_size}-city instances, but the heuristic must
    generalise to any
subgraph size K in [1, {problem_size - 1}] because it is called repeatedly
    during decoding
on a shrinking unvisited set.

{_SPARSE_TEACHER_CONTEXT}

Describe ONE strategy for choosing the next city at each decoding step (1-3
    sentences).
Focus on a strategy that produces a CLEAR top-1 winner from the candidates,
    not one
that ranks many of them similarly. Top-1 hit rate is the primary metric.

The strategy must rely only on (unselected_locs, first_loc, current_loc).
There is NO global "visited mask" or full-instance node list available --
    visited nodes
are not in the input.  Progress within the tour is exposed ONLY via K =
unselected_locs.size(1).  No step index or step_norm is provided.

Diversity guideline:
Avoid proposing a strategy that is functionally identical to an existing one.
- You may reuse existing features as secondary signals, as long as the
    overall
  combination or decision logic is meaningfully new.
- Simpler strategies with one or two strong signals are preferred over
    complex
  multi-signal blends.

Use only TSP reasoning; do not mention VRP concepts.
Do NOT write code or mathematical formulas.
```

Implement — `_make_system_prompt` **(system)**

Implement strategy (system)

```
You are writing heuristic modules for the 2D symmetric Euclidean Travelling
    Salesman Problem (TSP).
Training is on {problem_size}-city instances, but at inference time K (the
    number of unvisited
candidate cities at each decoding step) varies from {problem_size - 1} down
    to 1.  Your code MUST
work for ANY K.

{_SPARSE_TEACHER_CONTEXT}

Your task is to write a Python module containing exactly one function:

  def heuristic(unselected_locs, first_loc, current_loc) -> torch.Tensor:
      """Score each unvisited candidate.

      Args:
        unselected_locs:  torch.Tensor [B, K, 2]  candidate (x,y) coords in
    [0,1]^2
        first_loc:        torch.Tensor [B, 2]    start node coords
        current_loc:      torch.Tensor [B, 2]    current end-of-tour coords

      Returns:
        torch.Tensor [B, K]  finite scores; higher = more preferred next-node.

      """

Rules:
- Imports allowed: torch, torch.nn.functional as F, math. Nothing else.
- All tensor operations must be batched (first dim = B).  K may differ across
     calls.
- No Python `for` loops over batch dim or candidate dim.  No `while` loops.
  No itertools.  All ops must be vectorised torch ops.
- No file I/O, network access, environment access, printing, or side effects.
- Do not modify input tensors in-place.
- Coordinates are in [0,1]^2.  Embed any constants directly in code.
- No PARAMS dict required.

**Critical: K=1 edge case.** Your code WILL be called with K=1 at the last
    decoding
step.  Common K=1 pitfalls that produce NaN/inf and cause your heuristic to
    be
discarded:
  - `tensor.std(dim=K_dim)` defaults to `unbiased=True` which needs >=2
    samples;
    at K=1 it returns NaN.  Use `tensor.std(dim=K_dim, unbiased=False)` or
    just
    skip dispersion calcs when K==1.
  - `tensor.var(dim=K_dim)` has the same default-unbiased issue.  Use
    `var(dim=K_dim, unbiased=False)`.
  - Dividing by `K - 1`, `(K * (K - 1))`, or `count - 1` blows up at K=1.
    Use
    `max(K - 1, 1)` or add a `+ 1e-8` denominator guard.
  - Pairwise distance matrices [B, K, K] degenerate to [B, 1, 1] with the
    only
    entry being self-distance 0; downstream divisions or normalisations on
    this
    can NaN.
  - When K=1, the answer is forced (one candidate); your code can early-
    return
    `torch.zeros(B, 1, ...)` if any computation would be degenerate.
```

```
Return a torch.Tensor of shape [B, K] with finite scores.
Return ONLY the complete Python module, no explanation, no markdown fences.
```

Implement — user message + module template

Implement strategy (user) + HEURISTIC_MODULE_TEMPLATE

```
Implement the following heuristic strategy as a Python module.

Strategy: {description}

Use this structure:

import torch
import torch.nn.functional as F
import math


def heuristic(unselected_locs, first_loc, current_loc):
    """Score each unvisited candidate.

    Args:
        unselected_locs: torch.Tensor [B, K, 2]
        first_loc:       torch.Tensor [B, 2]
        current_loc:     torch.Tensor [B, 2]

    Returns:
        torch.Tensor [B, K] of finite scores; higher = preferred next-node.

    Note: progress within the tour is exposed only implicitly via K =
    unselected_locs.size(1).  No global N or step index is provided.
    """
    # TODO: Replace with strategy implementation.
    raise NotImplementedError
```

C.2.4 Phase II — TextGrad refinement loop

Backward (code) — `_make_backward_prompt` (system)

Backward on code (system)

```
You are the BACKWARD PASS of a TextGrad optimizer for TSP next-node selection
     heuristics.
Training was on {problem_size}-city instances; K (candidate count) varies at
    inference.

{_SPARSE_TEACHER_CONTEXT}

The heuristic uses signature
  heuristic(unselected_locs [B,K,2], first_loc [B,2], current_loc [B,2]) -> [
    B,K]

You will see a heuristic function and one decision step where it picked a
    different
top-1 from the teacher. Produce a TEXT GRADIENT: a concise diagnosis of why
    the code's
score for the teacher's top-1 candidate fell behind its (wrong) argmax.

This is NOT a code fix. Output only a diagnosis in this exact format:
LOGIC: [what the code currently computes for ranking these top candidates]
```

```
TEACHER: [the teacher chose candidate X with ~95% mass -- what about X would
    the code recognize?]
FLAW: [why the code's score for X fell below the score for the (wrong) argmax
    ]
DIRECTION: [what change would push X's score above the wrong argmax -- focus
    on
            making the winning gap larger or fixing the tiebreaker, NOT on
    shaping
            the full distribution]

Be concrete about geometry (distances, angles, tour-closing cost, cluster
    structure).
2-5 sentences total. No code. Note that K varies -- your reasoning must hold
    for any K.
```

Backward (code) — per-failure user message

Backward on code (per-failure user message)

```
Heuristic code:
{code}

Failure case [KL(teacher||heuristic)={kl_val:.4f}]:{rank_line}
  Step: {int(step)}/{int(step_max)} ({100*int(step)//int(step_max)}% complete)

  Unvisited candidates remaining (K): {n_unv}
  Current position: {cur_pos}
  Tour-start position: {first_pos}
  5 nearest unvisited (geometry only): {[f"pos={x['pos']} d={x['dist_to_cur
    ']}" for x in nu]}
  Heuristic top-2: {h3}
  Teacher   top-2: {t3}

Generate the text gradient for this failure case.
Focus especially on why the heuristic's top-ranked candidate differs from the
     teacher's,
described purely in geometric terms (positions, distances, angles).
    Candidates have NO
stable identifier -- your reasoning must be over geometry, not node ids.
```

Backward (description) — `_make_backward_desc_prompt` **(system)**

Backward on description (system)

```
You are the BACKWARD PASS of a TextGrad optimizer for TSP heuristic
    descriptions.
Training was on {problem_size}-city instances; the description must remain
    valid for any K.

{_SPARSE_TEACHER_CONTEXT}

Given a heuristic's natural language description and a decision step where
    the heuristic
picked the wrong top-1, produce a TEXT GRADIENT: what does the description
    fail to capture
that would help the implementation pick the right top-1 next time?

Format:
MISSING: [what geometric reasoning the description omits or under-emphasises]
```

```
MISLEADING: [anything in the description that points the implementation in
    the wrong direction]
DIRECTION: [how the description should be revised -- focus on the top-1
    ranking decision,
           not on shaping a full distribution]

2-4 sentences. No code. Focus on strategy-level geometric reasoning that
    affects top-1.
```

Backward (description) — per-failure user message

Backward on description (per-failure user message)

```
Heuristic description: "{description}"

Heuristic code:
{code}

Failure case [KL={kl_val:.4f}]:
  Step: {int(step)}/{int(step_max)} ({100*int(step)//int(step_max)}% complete)

  Current pos: {sample_info.get('cur_pos', '?')}
  Heuristic top-2 (geometry only): {h3}
  Teacher   top-2 (geometry only): {t3}

Generate the text gradient for the DESCRIPTION (not the code).
Reason in geometric terms only -- candidates have no stable identifier.
```

Optimizer step — `_make_optimizer_prompt` (system)

Optimizer step (system)

```
You are part of an optimization system that improves Python heuristic
    functions for TSP
next-node selection. Training was on {problem_size}-city instances; K (
    candidate count)
varies from {problem_size - 1} down to 1 at inference time.

{_SPARSE_TEACHER_CONTEXT}

Heuristic signature is fixed:
  heuristic(unselected_locs [B,K,2], first_loc [B,2], current_loc [B,2]) -> [
    B,K]

You will receive the current heuristic code and text gradients (diagnoses of
    failure cases).
Your task: synthesize the gradients and produce an improved version of the
    code that picks
the teacher's top-1 candidate more often (and at least keeps it in the top-2
    when it can't win).

- Address the most common and severe failure patterns across all gradients
- Momentum gradients indicate persistent failure modes -- treat them as high
    priority
- **Sharpen the winning gap.** If the existing code outputs scores that are
    too close
  together at the top, increase the multiplier on the dominant signal so the
    winner
  pulls ahead more decisively after softmax.
- Keep the function signature unchanged.
```

```
- Imports: torch, torch.nn.functional as F, math only.
- Hard constraint: NO Python ‘for‘ loops over batch dim or candidate dim. NO ‘
    while‘ loops.
  All ops must be vectorised torch ops.
- Code must work for ANY K (do not assume K = {problem_size - 1}).
- **K=1 numerical safety**: ‘tensor.std/var(dim, unbiased=True)‘ returns NaN
    at K=1.
  Always pass ‘unbiased=False‘ when reducing along the K dimension.  Avoid
    dividing
  by ‘K-1‘ or ‘K*(K-1)‘ without guarding (‘max(K-1, 1)‘ or ‘+ 1e-8‘).  Early-
    return
  ‘torch.zeros(B, 1, ...)‘ when K==1 if any computation would be degenerate.
- Embed constants directly in code. No markdown fences.
- Prefer simplicity. A 5-line scorer that picks top-1 right is better than a
    50-line
  multi-signal blend that only gets top-3 right.

Put the improved Python module between {new_variable_start_tag} and {
    new_variable_end_tag}.
```

C.2.5 Teacher identity in prompts

We test whether EPB relies on explicit teacher identity by removing the POMO/LEHD names and all background information about these models from the LLM prompts. In these blind runs, the LLM receives only the optimization task, code interface, and action feedback from an unnamed teacher.

Table 9: Solution-cost gaps (%) to the teacher with original and blind prompts. Lower is better.

	Greedy		**Search**	
Setting	Original	Blind	Original	Blind
POMO CVRP-50	+2.88	+3.32	+1.13	+1.12
LEHD TSP-100	+2.57	+2.51	+0.34	+0.33

The blind runs produce similar solution-cost gaps. These results suggest that explicit teacher identity is not required for the solution quality achieved in these settings. The LLM can still draw on its general knowledge of routing heuristics.

D Additional Results and Discussions

D.1 Sensitivity to Random Seeds

We run EPB with DeepSeek-V4-Flash on POMO-TSP-100 using three random seeds (42, 43, and 44). Convergence speed varies with bank initialization, but all three runs reach similar final validation top-1 agreement (Figure 10). The best greedy gaps range from 3.56% to 5.08%, while multi-start gaps remain below 0.21% for all seeds (Table 10). These results support consistent final performance across seeds in this setting.

Table 10: Solution-cost gaps (%) to the teacher across seeds.

Seed	**Greedy**	**Search**
42	+3.56	+0.036
43	+4.66	+0.138
44	+5.08	+0.204

D.2 LLM Cost

One-time LLM usage. EPB uses the LLM only during offline distillation and requires no LLM calls at deployment. Table 11 reports the usage of the three settings with the highest LLM usage.

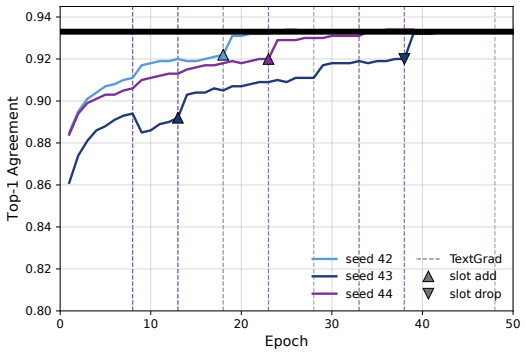


Figure 10: EPB convergence across seeds on POMO-TSP-100 with DeepSeek-V4-Flash.

Table 11: LLM usage for one complete distillation run.

Setting	LLM calls	Tokens
POMO CVRP-100	8,071	29.66M
LEHD TSP-100	5,471	30.33M
LEHD CVRP-100	9,704	44.24M

Cost breakdown. TextGrad accounts for 90.6% of all calls, of which 96% are paired code and description backward passes. Input tokens account for 79% of all tokens, and 93% of these come from repeated prefixes. Thus, approximately 73% of all tokens are potentially cacheable. In the runs reported in Table 12, prompt caching served 70–79% of the input tokens.

Larger gradient batches and less frequent TextGrad and description updates may further reduce calls, but we have not tested these changes. To our knowledge, this is the first work to distill sequential NCO policies into banks of executable programs evolved by an LLM. We prioritize establishing feasibility and fidelity and have not yet systematically optimized distillation cost.

Open models and parameter count.

To test lower-cost distillation and examine the effect of LLM size, we compare DeepSeek-V4-Flash and DeepSeek-V4-Pro on POMO-TSP-100. Flash has 284B total parameters and 13B activated parameters. Pro has 1.6T total parameters and 49B activated parameters. Both runs follow the same training and evaluation protocol. Both models achieve solution quality close to the main results (Table 12), showing that the smaller Flash model also supports effective distillation in this setting.

Table 12: Distilling POMO-TSP-100 with open models. Gaps are relative to the teacher.

LLM	Greedy (%)	Search (%)
Main results	+3.79	+0.072
DeepSeek-V4-Flash	+3.56	+0.036
DeepSeek-V4-Pro	+2.71	−0.071

Comparison with program-search methods. EoH [46] uses approximately 1,000 LLM queries for TSP and FSSP and 2,000 for online bin packing. FunSearch generates approximately 10^6 program samples. EPB uses roughly 5,000–10,000 calls, broadly comparable in scale to EoH. These methods optimize task objectives, whereas EPB fits a teacher's sequential decisions. Since the objectives differ and neither method reports token usage, this comparison concerns only orders of magnitude.

D.3 Fidelity

D.3.1 Absolute Solution Cost

Table 13 reports the absolute mean solution costs of the teachers and all interpretable surrogate methods on TSP and CVRP, complementing the relative gaps in Table 1.

Table 13: Absolute solution cost of the teacher and of every method in Table 1. Ref.: teacher cost (reference, not a competitor). N: number of nodes. All methods in a row use the same test instances: 1,000 per setting, except 1,500 for POMO TSP-50 and 256 for LEHD CVRP-100.

		TSP							CVRP						
Teacher	N	Ref.	GPT-6	Fable	Tree	NAM	MoËT	EPB	Ref.	GPT-6	Fable	Tree	NAM	MoËT	EPB
Greedy decoding ↓															
POMO	50	5.71	6.26	6.17	6.65	6.35	6.76	**5.88**	10.75	12.35	12.17	13.54	12.67	12.27	**11.06**
POMO	100	7.83	8.69	8.54	9.40	8.91	9.46	**8.13**	16.19	18.84	18.52	20.82	19.33	18.72	**16.83**
LEHD	100	7.81	8.71	8.56	9.38	8.89	9.56	**8.01**	16.28	18.87	18.58	19.85	19.25	18.79	**16.86**
Search decoding ↓															
POMO	50	5.70	5.84	5.84	5.96	5.84	6.10	**5.70**	10.47	11.56	11.45	11.38	11.51	11.26	**10.59**
POMO	100	7.80	8.15	8.10	8.44	8.18	8.64	**7.80**	15.88	17.92	17.67	17.80	17.85	17.33	**16.08**
LEHD	100	7.77	7.93	7.90	8.04	7.97	8.33	**7.80**	15.97	18.26	17.99	18.28	17.91	17.81	**16.31**

D.3.2 Tour Comparison

We compare teacher and EPB tours under search decoding to reduce the impact of error accumulation in greedy decoding. Figures 11–12 show examples across teacher models.

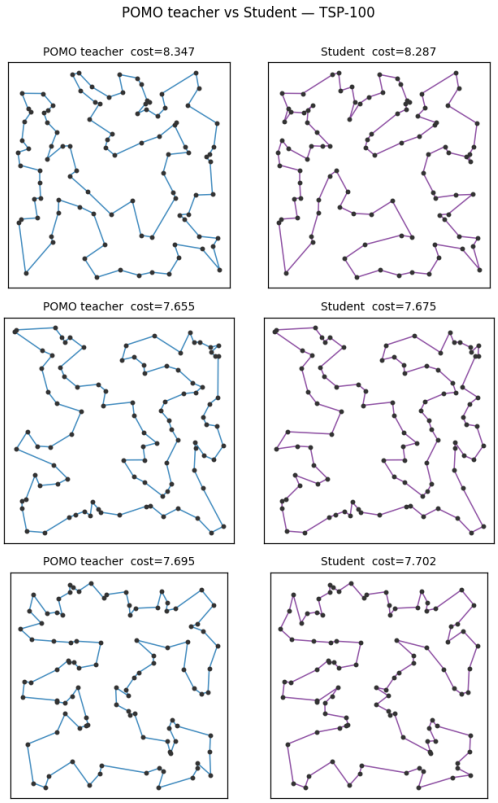


Figure 11: POMO-TSP-100 Tour Comparison

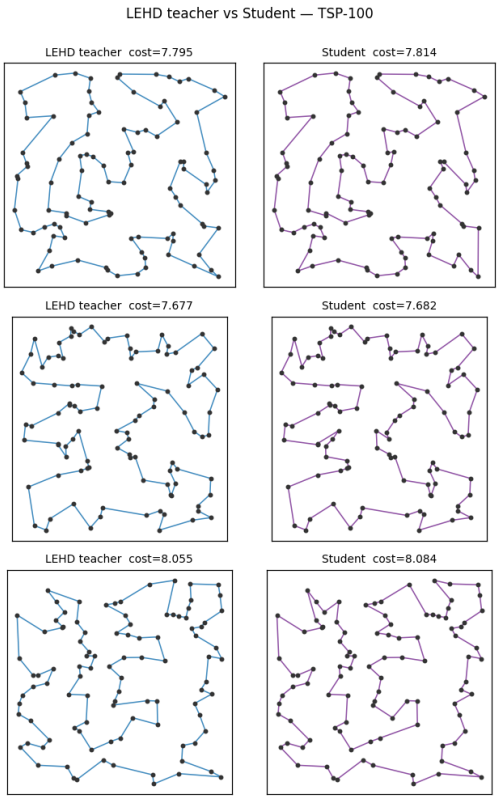


Figure 12: LEHD-TSP-100 Tour Comparison

D.4 Heuristic Gallery[2]

This appendix presents three example heuristic (program) banks distilled from POMO for TSP-100 (Table 14), LEHD for TSP-100 (Table 15), and POMO-CVRP-50 (Table 16)

[2]To conserve space, this appendix reports condensed natural-language summaries in place of the heuristics' source description/code. Each summary is produced by prompting a large language model to refine the description.

Table 14: TSP-100 heuristic bank distilled from the POMO teacher.

Heuristic ID	Description
H00	Combines proximity with an early isolation bonus and a late pull towards the start, then uses closing-cost estimates through two or three remaining cities near completion.
H01	Favours nearby cities throughout, adds a far-from-start bonus late in the tour, then penalises the distance through a candidate and back to the start while further strengthening proximity.
H02	Rewards peripheral cities and outward steps early, strengthens proximity as the tour progresses, and finishes with a nearest-onward connection cost, a renewed periphery bonus and a pull towards the start.
H03	Adds mid-tour periphery and far-from-start bonuses to proximity, briefly favours cities nearer the start later, then restores a strong far-from-start preference near completion.
H04	Favours nearby cities with a large gap between their nearest and second-nearest unvisited neighbours. With eight or fewer cities remaining, it estimates up to six greedy onward steps and a return to the start, with extra weight on proximity.
H05	Combines proximity, periphery and closing-cost estimates through three cities. It emphasises peripheral and far-from-start cities when few remain, with separate scores for the last one or two cities.
H06	Combines proximity, cheap onward connections and a far-from-start bonus, then increasingly weights closing-cost estimates through two or three cities, with a separate term for the last two cities.
H07	Adds a heading-alignment bonus among nearby candidates early and mid-tour, then increasingly penalises the distance through a candidate and back to the start. This closing term is disabled for the last five cities.
H08	Combines proximity with a centroid-based geometric bonus, then favours shorter distances through a candidate and back to the start. With ten or fewer cities remaining, it uses greedy completion estimates with extra weight on the immediate step.
H09	Rewards proximity, dense angular sectors and peripheral position early, then gives more weight to isolation and short closing paths. Near completion, it discourages visiting cities close to the start.
H10	Balances immediate distance against the distance to each candidate's nearest unvisited neighbour, increasingly rewarding isolated cities as the tour progresses.
H11	Favours nearby cities, with a mid-tour bonus for moving towards the remaining cities' centroid. The directional bonus grows with centroid distance, while a far-from-start bonus becomes stronger late.
H12	Combines strong proximity with early guidance towards the remaining cities' centroid. Its start-distance term changes from a penalty to a bonus, while the cost of continuing through the candidate's nearest unvisited neighbour and back to the start gains weight.
H13	Combines normalised proximity with early bonuses for forward motion, sparse neighbourhoods and closeness to the remaining cities' centroid, then gradually shifts towards minimising the distance through a candidate and back to the start.

Table 15: TSP-100 heuristic bank distilled from the LEHD teacher.

Heuristic ID	Description
H00	Favours short steps, with additional bonuses for the two nearest cities. It penalises isolation and increasingly favours cities near the start as fewer cities remain.
H01	Combines the immediate step with the cheapest onward connection and a weighted return from that same onward city to the start. The return term gains weight as fewer cities remain.
H02	Scores the extra distance of visiting a candidate before returning to the start, then adds the cheapest further insertion. The insertion term gains weight as fewer cities remain.
H03	Combines proximity and an onward-step closure cost with bonuses for isolated cities and cities far from the start, while postponing the city closest to the start as closure becomes more important.
H04	Combines proximity with the distance through each candidate's nearest remaining neighbour and back to the start. A small alignment bonus favours steps directed towards the start.
H05	Combines short steps with a tangential sweep around the remaining cities' centroid, penalises inward movement, and adds bonuses for peripheral and isolated cities. Return-to-start terms guide closure.
H06	Balances proximity against isolation and the gap between the two nearest onward connections, with additional bonuses for peripheral cities and steps along the heading.
H07	Favours short steps while discouraging early use of the city closest to the start. A preference for cities beyond the median distance from the start strengthens as fewer cities remain.
H08	Favours a sideways step towards the denser side of the heading, restricting choices to cities within a local distance allowance of the nearest city.
H09	Favours directions near the centre of the largest empty forward angular sector, penalises backward directions, and adds a small bonus for longer steps within that sector.

Continued on next page

Table 15: TSP-100 heuristic bank distilled from the LEHD teacher (continued).

Heuristic ID	Description
H10	Favours short steps within a forward cone, but turns towards a backward region when forward options are absent or sparse and the backward region is dense or offers sufficient progress towards the start.

Table 16: CVRP-50 heuristic bank distilled from the POMO teacher.

Heuristic ID	Description
H00	Far-customer savings: a dominant savings term favours customers far from the depot but close to the vehicle; it returns mainly when all feasible customers are closer to the depot than to the vehicle.
H01	Centroid pivot: savings plus inverse distance, with depot return driven by whether the depot is closer than the current node to the remaining customers' centroid (ratio 0.75).
H02	Local demand density: rewards unvisited demand within radius 0.12 of a candidate, scaled by inverse squared distance. The depot opens only when no customer fits and capacity is at most 0.15.
H03	Homeward corridor: penalises the perpendicular distance to the current-to-depot line and rewards customers on that segment. Depot return gains weight below 0.25 capacity.
H04	Ordered sweep: visits customers in angular order around the depot in the direction set by the first customer, favouring the next in order and the angular extreme of the unvisited set.
H05	Capacity-pressure blend: demand-weighted proximity with an angular term favouring customers just ahead counter-clockwise around the depot while underway; plain proximity with at most three customers left. The depot penalty grows with feasible customers.
H06	Gap pivot: near the depot with more than 0.7 capacity it turns towards the largest angular gap between feasible customers; at the depot it favours high demands far from the first arm.
H07	Angular continuity: keeps the route in the current direction from the depot, sharper once underway. Returns when the remaining demand lies in opposite or distant 30° sectors.
H08	Angular proximity to the current position as seen from the depot (Gaussian, 0.5 rad), with a bonus for leaving a quarter of the capacity. Below 0.3 capacity it favours near-depot customers.
H09	Detour: penalises the extra distance of going home via the customer rather than directly. Depot return gains weight below 0.5 capacity or with at most two feasible customers.
H10	Frontier and isolation: rewards customers far from the remaining customers' centroid and with few unvisited neighbours within radius 0.15; below 0.2 capacity it favours customers near the depot.
H11	Outward continuation: rewards customers far from the remaining customers' centroid, and those nearer to the vehicle than to the depot with high savings; below 0.3 capacity it favours exact fits and near-depot customers.
H12	Frontier with perfect-fit trigger: favours nearby customers far from the remaining customers' centroid, with a strong bonus for demands matching the remaining capacity within 0.03; the depot is allowed only when no such fit exists and capacity is low.
H13	Sparse-sector pivot: when the current 60° sector holds fewer than two feasible customers and the opposite sector at least two, it favours the opposite sector or the depot.
H14	Clean-fit chaining: rewards leftovers that match another unvisited demand and customers in dense $\pm 30^\circ$ clusters; with at most three remaining it serves the farthest first.
H15	Bimodal sector test: returns to the depot when the vehicle is in the smaller of two dominant 45° demand sectors more than 90° apart; starts from isolated, low-demand customers.
H16	Departure specialist: at the depot with a full vehicle it scores arm alignment, angular density and average savings instead of proximity; elsewhere it is the nearest-customer rule.
H17	Three-regime switch: first-arm alignment and cluster density at departure, savings and current-arm alignment underway, nearest low-demand customers at or below 0.15 capacity.
H18	Early arm lock: within the first three stops it penalises leaving the first-customer arm exponentially in the angle; at the depot it favours the angular boundary of the unvisited set.
H19	Arm switching at low capacity: below 0.2 capacity it favours near-depot customers more than 60° from the current arm unless an exact fit exists; otherwise proximity, fit and angular clustering.

D.4.1 The surrogates emphasize tour progress and geometry differently

Figure 13 compares programs associated with non-NN decisions in the two surrogates. The POMO surrogate emphasizes phase-conditioned rules, shifting from early peripheral and isolated-city visits to late start-aware closure and multi-step completion. The LEHD surrogate emphasizes proximity, insertion and closure costs, directions relative to the start-to-current heading, and remaining-city geometry. Several LEHD programs strengthen closure preferences as fewer cities remain. Both

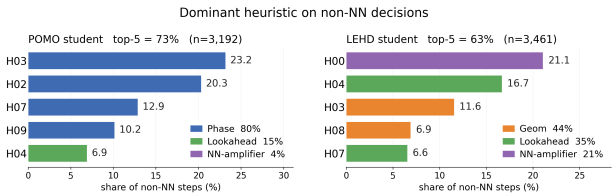


Figure 13: Dominant programs in the POMO and LEHD surrogates. Colors indicate categories.

combine geometry with progress-dependent adjustments. These differences may partly reflect the input constraints: POMO programs receive tour progress, whereas LEHD uses remaining-city counts.

D.5 Code Example

In this appendix, we show two example codes learned by our student model.

Listing 1: POMO TSP-100 H10 Code

```python
import torch

def heuristic(locs, current_node, mask, visited, step_norm, first_node):
    """
    POMO TSP-100 H10: Postponement-regret heuristic.

    Scores each feasible candidate by combining immediate proximity with
    a progress-scaled isolation bonus. As the tour progresses, the heuristic
    increasingly favors candidates whose nearest remaining neighbor is far away,
    i.e., nodes that may become expensive detours if postponed.
    """
    B, N, _ = locs.shape
    device = locs.device
    batch_idx = torch.arange(B, device=device)

    # Current city location: [B, 2]
    cur_loc = locs[batch_idx, current_node]

    # Immediate distance from current city to each candidate: [B, N]
    dist_to_cur = torch.norm(locs - cur_loc.unsqueeze(1), dim=-1)

    # Pairwise distances between all cities: [B, N, N]
    pairwise_dist = torch.norm(
        locs.unsqueeze(2) - locs.unsqueeze(1),
        dim=-1
    )

    # For each candidate i, find distance to its nearest other unvisited city.
    eye = torch.eye(N, device=device, dtype=torch.bool).unsqueeze(0)
    invalid_neighbor = (~mask).unsqueeze(1) | eye
    masked_pairwise = pairwise_dist.masked_fill(invalid_neighbor, 1e9)
    nearest_unvisited_dist = masked_pairwise.min(dim=-1).values

    # Progress-scaled postponement-regret weight.
    alpha = (1.5 + 2.0 * step_norm).unsqueeze(-1)

    # Higher is better: nearby nodes are good, but isolated nodes become
    # increasingly urgent as the tour progresses.
    logits = -dist_to_cur + alpha * nearest_unvisited_dist

    # Stand-alone safety: remove infeasible / already visited nodes.
    logits = logits.masked_fill(~mask, -1e9)

    return logits
```

Listing 2: LEHD TSP-100 H1 Code

```python
import math
import torch

def heuristic(unselected_locs, first_loc, current_loc):
    """
    LEHD TSP-100 H01: Coupled two-step closure lookahead.

    Scores each remaining city by combining the immediate current-to-candidate
    distance with the cheapest follow-up move from that candidate, where the
    follow-up node is also encouraged to be close to the tour start. This gives
    a local geometric correction rule that favors candidates forming a good
    two-step chain toward closure.
    """
    B, K, _ = unselected_locs.shape
    device = unselected_locs.device
    dtype = unselected_locs.dtype

    # K=1: forced choice.
    if K == 1:
        return torch.zeros(B, 1, device=device, dtype=dtype)

    # Distance from current city to each candidate: d(current, i)
    dist_current = torch.norm(
        unselected_locs - current_loc.unsqueeze(1),
        dim=-1
    ) # [B, K]

    # Distance from each candidate to the start city: d(i, start)
    dist_start = torch.norm(
        unselected_locs - first_loc.unsqueeze(1),
        dim=-1
    ) # [B, K]

    # Pairwise distances among remaining candidates: d(i, j)
    pairwise_dist = torch.cdist(unselected_locs, unselected_locs, p=2)
    eye = torch.eye(K, device=device, dtype=torch.bool).unsqueeze(0)
    pairwise_dist = pairwise_dist.masked_fill(eye, float("inf"))

    # K-aware closure weight: stronger when fewer nodes remain.
    sqrtK = math.sqrt(float(K))
    lam = min(1.0, 1.6 / sqrtK)
    w_step = 1.0 + 0.3 / sqrtK

    # Exact two-node tail behavior.
    if K == 2:
        lam = 1.0
        w_step = 1.0

    lam = torch.tensor(lam, device=device, dtype=dtype)
    w_step = torch.tensor(w_step, device=device, dtype=dtype)
    scale = torch.tensor(22.0 + 6.0 / sqrtK, device=device, dtype=dtype)

    # For each candidate i, choose the best follow-up j:
    # min_j [ d(i,j) + lambda * d(j,start) ], j != i.
    followup_cost = pairwise_dist + lam * dist_start.unsqueeze(1)
    best_followup = followup_cost.min(dim=2).values # [B, K]

    # Lower cost is better, so return negative cost as score.
    total_cost = w_step * dist_current + best_followup
    scores = -scale * total_cost

    return scores
```

D.6 Interpreting each step - more examples

LEHD-TSP Figure 14 links the selected LEHD student decisions to the highest-router-weight heuristic at each step. H09 (*boundary via widest gap*) favors candidates toward the widest empty sector facing away from the start. This matches the three cyan edges, which visit boundary cities before closer cities. H06 (*isolated + hull, keep heading*) rewards cities far from their nearest remaining neighbor. On the pink edge near $(0.7, 0.4)$, the student selects the most isolated of the 39 remaining cities over an NN candidate at almost the same distance from the current city, consistent with H06's preferences for isolated cities.

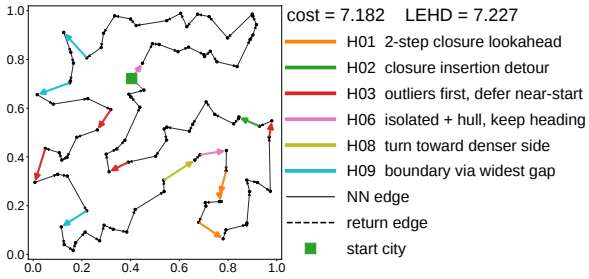


Figure 14: LEHD-TSP-100 Case study.

POMO-CVRP Figure 15 links selected POMO student decisions to geometric preferences. Given the large number of heuristics, we focus on three for discussion. H00 (*serve far customers while nearby*) favors customers whose distance from the depot most exceeds their distance from the vehicle. On the blue edges of routes 3 and 5, the selected customers have the largest such difference among all feasible customers, so the vehicle reaches far customers while they are cheap to serve. H05 (*capacity-pressure blend*) favors, through its angular term, customers just ahead in the counter-clockwise direction around the depot. On the pink edges of route 1, the selected customers lie ahead of the vehicle in this direction, whereas the nearest feasible customers (NN) lie behind it, matching the program's preference for sweeping without stepping back. H09 (*least detour on the way home*) favors customers that add little detour on the way back to the depot. On the orange edges of routes 1 and 3, the selected customers add far less detour than the NN candidates, consistent with serving customers on the return trip.

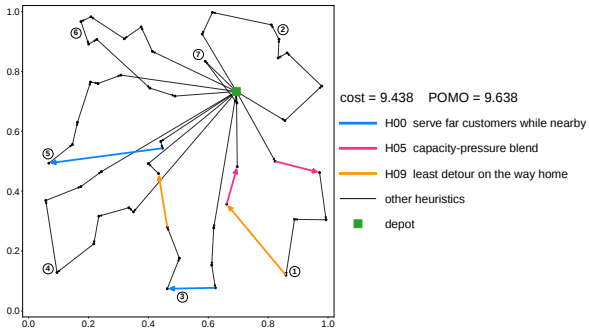


Figure 15: POMO-CVRP-50 Case study.

LEHD-CVRP Figure 16 links selected LEHD student decisions to geometric preferences. Given the large number of heuristics, we focus on three for discussion. H15 (*boundary customers first*) favors customers bordering the widest angular gap among unvisited customers around the depot. The orange edges, for example, in routes 4 and 8 are consistent with prioritizing the boundary of the remaining customer set. H19 (*serve on the way home*) favors depot toward customers when at most a quarter of the vehicle capacity remains. On the pink edges of routes 5 and 6, the selected customers lie almost directly along the current-to-depot direction, whereas the NN candidates are less aligned, consistent with serving customers along the return direction. H03 (*go outward first*) balances proximity to the current customer with a preference for customers farther from the depot. On the blue edges, it selects customers farther from the depot than the NN alternatives, matching the program's preference for outward continuation.

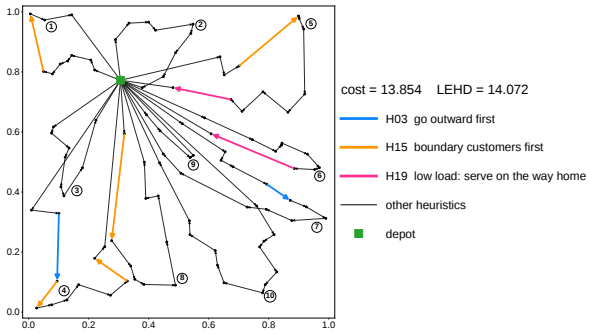


Figure 16: LEHD-CVRP-100 Case study.

D.7 Trustworthy Concept Attributions (Counterfactual Tests)

We examine whether the concepts represented by the programs influence the student's predictions. We test hull peripherality in POMO-TSP, isolation in LEHD-TSP, and the homeward corridor in POMO-CVRP through matched interventions, tracking changes in program scores, routing weights, and action probabilities. We apply the same interventions to the teacher to compare the policies' responses.

Intervention setup. For each base state B, we construct a treatment state T and a matched control state P. The student's choice in B is the target a, and a reference action r is selected using the rule for each test. We analyze base states where the teacher and student choose the same action and the student's preference for a over r is attributed mainly to programs implementing the tested concept. We compare the same a and r across the three states, keeping the reference node in place.

The treatment perturbs the problem instance to weaken the tested concept's support for a. The control matches the perturbation magnitude while preserving or strengthening that support. We compare the student's probability ratio for a versus r, which we expect to be lower under treatment than control.

Test Metrics. Let $\mathcal{M}$ denote the set of programs implementing the tested concept. For state s, program m produces raw action scores $z_m(\cdot \mid s)$ and an action distribution $V_m(\cdot \mid s)$ through masked softmax. The router assigns each program a weight $w_m(s)$. The student distribution is

$$\pi_S(a \mid s) = \sum_m w_m(s) V_m(a \mid s). \tag{21}$$

The carrier score sums the selected programs' score differences between a and r:

$$E_{\mathcal{M}}(s) = \sum_{m \in \mathcal{M}} \left[z_m(a \mid s) - z_m(r \mid s) \right]. \tag{22}$$

This is a manipulation check and is not counted as independent evidence.

The routing log-weight compares the total routing weights of the selected programs and the rest of the bank:

$$R_{\mathcal{M}}(s) = \log \sum_{m \in \mathcal{M}} w_m(s) - \log \sum_{m \notin \mathcal{M}} w_m(s). \tag{23}$$

The program contribution to the student's probability margin between a and r is

$$C_{\mathcal{M}}(s) = \sum_{m \in \mathcal{M}} w_m(s) \left[V_m(a \mid s) - V_m(r \mid s) \right]. \tag{24}$$

Summing over the entire bank gives $\pi_S(a \mid s) - \pi_S(r \mid s)$ exactly. $C_{\mathcal{M}}$ measures the contribution of the selected programs as a whole.

Student log-odds and teacher log-odds measure each policy's relative preference for a over r:

$$L_S(s) = \log \left[\pi_S(a \mid s) + \varepsilon \right] - \log \left[\pi_S(r \mid s) + \varepsilon \right], \tag{25}$$

$$L_T(s) = \log \left[\pi_T(a \mid s) + \varepsilon \right] - \log \left[\pi_T(r \mid s) + \varepsilon \right]. \tag{26}$$

Here, π_T is the teacher's action distribution and ε is a small constant for numerical stability. For each metric f, the matched effect is

$$g_f = f(P) - f(T). \tag{27}$$

Each test is oriented so that $g_f > 0$ is the predicted direction.

D.7.1 POMO-TSP: hull peripherality

We examine hull peripherality in the POMO-TSP-100 student. During early tour construction, H02 scores candidates as

$$\begin{aligned} z_2(i) &= 1.5\,\mathrm{hb}(i) + 0.5\,\mathrm{cont}(i) - 4\,d_{\mathrm{cur}}(i), \\ \mathrm{hb}(i) &= \frac{\|x_i - \mu\|}{\max_{j \in U} \|x_j - \mu\|}, \\ \mathrm{cont}(i) &= \left\langle \frac{x_i - x_c}{\|x_i - x_c\|}, \hat{u}_{\mathrm{out}} \right\rangle. \end{aligned} \tag{28}$$

Here, U is the set of unvisited cities and μ is their centroid. The positions of candidate i and the current city are x_i and x_c, respectively, with $d_{\text{cur}}(i) = \|x_i - x_c\|$. The unit vector $\hat{u}_{\text{out}}$ points from μ to x_c. The hb term rewards normalized distance from the centroid, while cont rewards alignment of the move from x_c to x_i with this outward direction. Together, they express the peripherality concept. The distance penalty favors nearby cities. We choose the candidate with the second-highest student probability in the base state as the reference r.

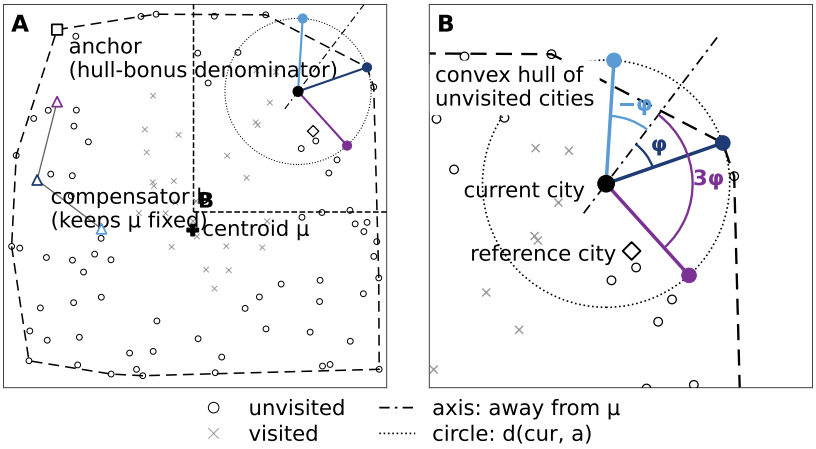


Figure 17: Hull peripherality test on POMO-TSP. (A) Full instance. (B) Treatment reduces peripherality, while control preserves it.

As shown in Figure 17, both the treatment and control move the target along a circle centered at the current city, preserving $d_{\text{cur}}(a)$. The treatment rotates the target toward the centroid, reducing both $\text{hb}(a)$ and $\text{cont}(a)$. The control reflects the target across the line through the current city and the centroid, preserving both terms. The target moves by the same distance in both states.

To keep the centroid fixed, we move another unvisited city b by an equal and opposite displacement in each state. We also preserve the maximum distance to the centroid used to normalize hb and match the change in the target's distance to the start city across treatment and control.

Table 3 reports positive mean matched effects for the routing balance, program contribution, and student log-odds. On average, the treatment reduces the program's support for a and the student's preference for a over r relative to the control. The teacher responds in the same direction.

D.7.2 LEHD-TSP: isolation

We examine isolation in the LEHD-TSP-100 student. For candidate i, let $d_1(i)$ denote its distance to the nearest other unvisited city. The programs in $\mathcal{M}$ contain the following isolation terms:

$$\begin{aligned} \text{H03}: &\quad 2.0\,\kappa\,\frac{d_1(i) - \min_k d_1(k)}{\max_k d_1(k) - \min_k d_1(k)}, \\ \text{H05}: &\quad 2.0\,g\,\frac{d_1(i) - \text{mean}_k\, d_1(k)}{\text{sd}_k\, d_1(k)}, \\ \text{H06}: &\quad 2.1\,\frac{d_1(i)}{\text{mean}_k\, d_1(k)}. \end{aligned} \tag{29}$$

The normalization statistics are recomputed over the candidates in each state. The positive gates κ and g depend on the tour stage and remain fixed across B, T, and P. Each term favors candidates farther from their nearest neighbour. H03 also uses distance to the start city, and H06 includes a partner-gap term. We choose the candidate nearest to the current city as the reference r and consider states where $a \neq r$.

As shown in Figure 18, the treatment rotates the target a about the current city toward its nearest unvisited neighbour q, reducing $d_1(a)$. The control rotates it by the same angle in the opposite direction, increasing $d_1(a)$. Both preserve $d_{\text{cur}}(a)$ and move the target by the same distance.

Only the target moves. We require $q \neq r$ and preserve q as the target's nearest neighbour in both states. The reference and its two nearest neighbours remain fixed, and the target stays outside that neighbourhood. This preserves the reference's first and second nearest-neighbour distances.

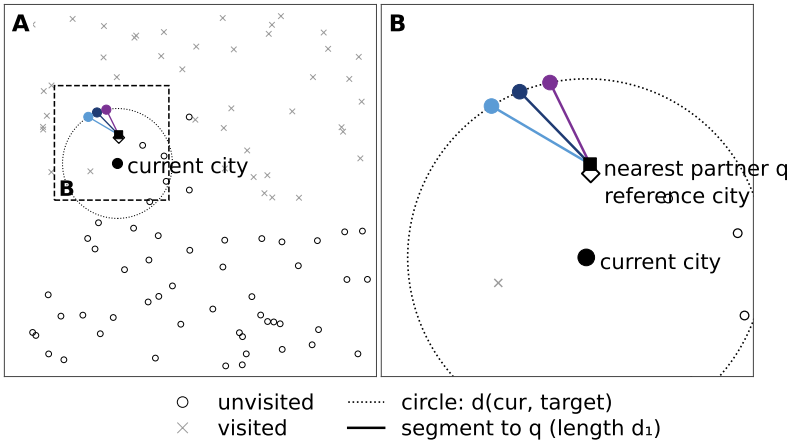


Figure 18: Isolation test on LEHD-TSP. Treatment moves a toward q, and control moves it away. Both match displacement and preserve distance to the current city.

Since the control increases isolation, we report both changes relative to the base state:

$$\Delta_{\text{down}} = f(B) - f(T), \qquad \Delta_{\text{up}} = f(P) - f(B). \tag{30}$$

Their sum is the matched effect $g_f = f(P) - f(T)$.

Table 17 reports positive mean effects for both components across all five metrics. On average, the student favors a less relative to r when isolation decreases and more when it increases. The teacher shows the same pattern.

Table 17: Isolation intervention on LEHD-TSP. Entries give mean effects with 95% instance-bootstrap confidence intervals and the percentage of states with positive effects in parentheses.

Metric	Down	Up	Total
Carrier score†	0.59 [0.58, 0.60] (100%)	0.61 [0.60, 0.63] (100%)	1.20 [1.18, 1.23] (100%)
Routing log-weight	0.16 [0.10, 0.22] (74%)	0.13 [0.08, 0.19] (70%)	0.29 [0.20, 0.37] (72%)
Program contribution	0.05 [0.04, 0.06] (75%)	0.03 [0.02, 0.03] (70%)	0.08 [0.06, 0.09] (74%)
Student log-odds	0.36 [0.29, 0.43] (71%)	0.18 [0.15, 0.22] (66%)	0.54 [0.45, 0.63] (71%)
Teacher log-odds	0.83 [0.69, 1.02] (77%)	0.64 [0.55, 0.74] (74%)	1.47 [1.24, 1.74] (76%)

†Manipulation check only.

D.7.3 POMO-CVRP: homeward corridor

We examine the homeward corridor concept in the POMO-CVRP-50 student. Let c denote the current customer, o the depot, L the remaining capacity, and α_i the angle between the directions from c to o and from c to candidate i. We measure the detour through i and its perpendicular distance to the line through c and o:

$$\begin{aligned} D(i) &= d(c,i) + d(i,o) - d(c,o), \\ \rho_\perp(i) &= d(c,i)\,|\sin\alpha_i|, \end{aligned} \tag{31}$$

where d denotes Euclidean distance.

The programs in $\mathcal{M} = \{\text{H03}, \text{H09}\}$ contain the following scoring terms:

$$\begin{aligned} \text{H03}: \quad & -3\rho_\perp(i) - 1.5d(c,i) \\ & + \mathbb{1}\big[0 \le d(c,i)\cos\alpha_i \le d(c,o)\big] + \text{demand terms}, \\ \text{H09}: \quad & -6D(i) - 2d(c,i) + \text{demand terms}. \end{aligned} \tag{32}$$

Here, the demand terms depend only on the demand dem_i of customer i and on L. H03 penalizes distance from the return line and rewards customers whose projection lies between c and o. H09 penalizes detours on the return to the depot. We examine states with low remaining capacity and choose the nearest feasible customer as the reference r.

As shown in Figure 19, the treatment rotates the target away from the direction to the depot, increasing both $D(a)$ and $\rho_{\perp}(a)$. The control reflects the target across the line through c and o, preserving both quantities and its distance to the depot. Treatment and control move the target by the same distance and preserve $d(c, a)$.

Only the target moves. Demands, remaining capacity, the visited set, and feasible actions remain unchanged. The target's projection stays on the segment from c to o, keeping the indicator term in H03 fixed. The reference remains the nearest feasible customer because all distances to the current customer are preserved.

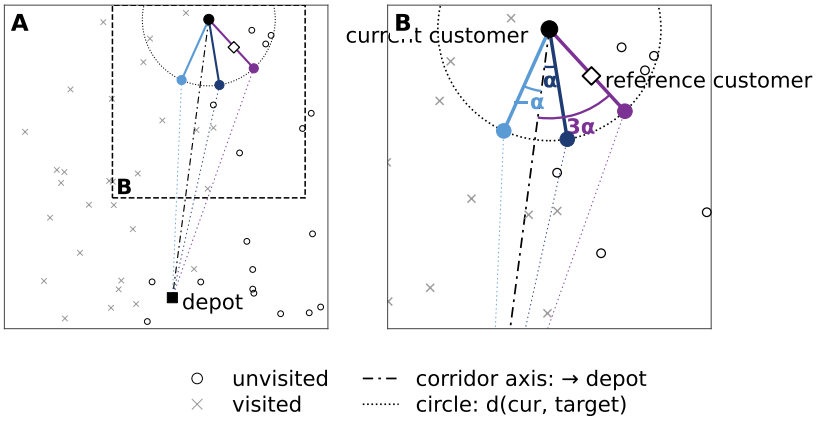


Figure 19: Homeward corridor intervention on POMO-CVRP. Treatment and control match target displacement and preserve distance to the current customer. Demands, remaining capacity, and feasible actions stay fixed.

We also fix the action distributions of all programs outside $\mathcal{M}$ to their base-state values and recompute routing weights to check whether the student's response persists.

Table 18: Homeward corridor intervention on POMO-CVRP. Mean student log-odds effects with 95% instance-bootstrap CIs. Parentheses give the fraction of states with positive effects.

Setting	**Student log-odds effect**
Standard intervention	3.14 [3.00, 3.28] (87%)
Other program distributions fixed	2.65 [2.53, 2.77] (88%)

Routing log-weight, program contribution, and student and teacher log-odds all show positive mean effects (Table 3). The mean student log-odds effect remains positive when all program distributions outside $\mathcal{M}$ are fixed to their base-state values and routing weights are recomputed (Table 18).

D.7.4 POMO-CVRP: local demand density

H02 rewards candidates surrounded by unvisited demand, scaled by their distance to the current node:

$$z^{\mathrm{dens}}(i) = \frac{D_{0.12}(i)}{d(c, i)^2}. \tag{33}$$

Here, $D_{0.12}(i)$ is the unvisited demand within radius 0.12 of candidate i, including i itself, and c is the current customer. We use $\mathcal{M} = \{\mathrm{H02}\}$ and use the nearest feasible customer as the reference r.

The treatment transfers just enough demand from customers near the target to customers outside the neighborhoods of both the target and the reference to make H02 prefer the reference. The control gives the same receivers the same amount from customers outside both neighborhoods, preserving the local demand of the target and the reference as well as the total unvisited demand. Customer locations, remaining capacity, the visited set, and feasible actions remain unchanged.

Holding H02's density term at its base-state values while recomputing the other programs and the router eliminates the student effect. This supports the density term's role in the student's response.

Table 19 shows positive mean effects for routing log-weight, program contribution, and student log-odds. The teacher responds in the same direction but more weakly than the student. Its mean log-odds effect is positive in each of three independent batches, yet only 56% of states show a positive effect, compared with 97% for the student, and its probability margin changes by 0.09, compared with 0.30 for the student. Its top choice changes under treatment in 10% of states, compared with 43% for the student.

Table 19: Local demand density intervention on POMO-CVRP. Mean matched effects with 95% instance-bootstrap CIs. Parentheses give the fraction of states with $g > 0$.

Metric	Effect
Carrier score†	5.65 [5.05, 6.36] (100%)
Routing log-weight	2.68 [2.46, 2.91] (93%)
Program contribution	0.43 [0.41, 0.44] (100%)
Student log-odds	1.38 [1.27, 1.50] (97%)
Teacher log-odds	0.97 [0.57, 1.45] (56%)

†Manipulation check only.

D.8 Trustworthy Program attribution (Knockout Tests).

We remove each program in states where it dominates and renormalize the router weights. Mean next-node flip rates across programs are 65.2–98.1%, versus 0.2–0.6% when removing a uniformly sampled other program in the same states (Table 20). Removing the dominant program yields a higher flip rate than the control for all but H01 on POMO-CVRP, whose output is nearly uniform over customers and has little effect on the next-node choice.

Table 20: Dominance knockout. Program m has the highest router weight in the evaluated states. Flip rates are means across programs with 95% instance-bootstrap intervals.

Setting	Remove m (F_m)	Remove random (F_{rand})	# programs ($F_m > F_{\text{rand}}$) / total
POMO TSP	94.3% [93.8, 94.8]	0.6% [0.4, 0.8]	14/14
LEHD TSP	98.1% [97.6, 98.5]	0.2% [0.1, 0.4]	11/11
POMO CVRP	65.2% [64.6, 65.7]	0.5% [0.5, 0.6]	19/20

D.9 Trade-off Between Code Complexity and Solution Quality

Setup. We repeat distillation on TSP-100 with limits on program complexity. Constrained programs access the state through a fixed set of primitives: distance to the current node, distance to the start node, nearest-neighbour distance, local density, heading cosine, distance to the centroid of unvisited nodes, and the number of unvisited nodes. POMO also includes tour progress. These primitives are implemented in a shared handwritten library with 33 statements for POMO and 21 for LEHD. Unconstrained programs compute their features directly from the raw state.

We check each program's syntax tree during initialization, revision, and addition. The *simple* setting allows at most 2 distinct primitives, 15 statements, and 1 branch. The *medium* setting allows 3 primitives, 30 statements, and 2 branches. Branches include `if`, `where`, and `masked_fill`. Both settings forbid loops, sorting, and access to raw inputs outside primitive calls. The *unconstrained* setting uses the default pipeline without these restrictions.

All distillation settings use the same LLM (`deepseek-v4-flash`). We evaluate solution quality along the pruning path of each learned bank, removing programs one at a time using validation top-1 agreement. Table 21 compares coding-agent policies with EPB banks at the sizes with the lowest evaluated search gaps.

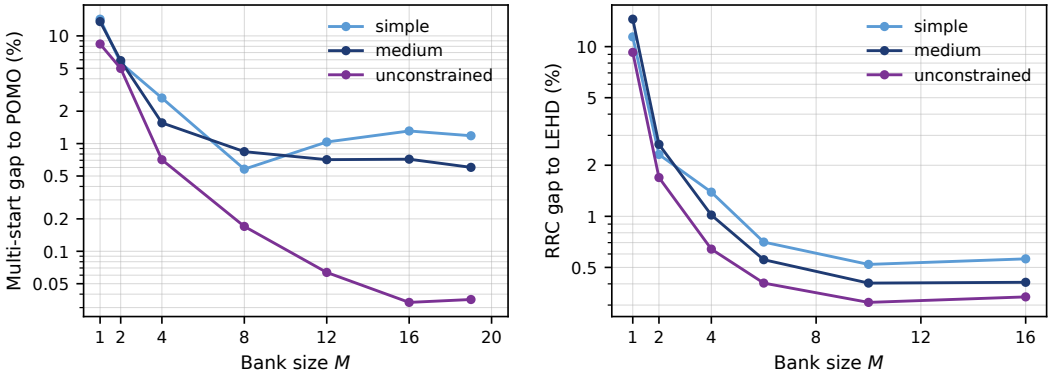
Metrics. We remove comments, blank lines, and docstrings and regenerate the source from its syntax tree before measuring complexity. *Statements* counts statement nodes, including nested

statements, but excludes imports, the function signature, and the final return of the score vector. *Halstead volume* is

$$V = (N_1 + N_2)\log_2(n_1 + n_2),$$

where n_1, n_2 count distinct operators and operands, and N_1, N_2 count their occurrences. We compute it with `radon` 6.0.1. *Coefficients* counts numeric weights, thresholds, and exponents, excluding indices, dimension arguments, numerical-stability constants, and mask fill values.

All complexity measurements in Table 21 include primitive implementations. Per-program measurements expand each called primitive and its helper functions before averaging across programs. Whole-bank measurements count each used primitive and helper once. For coding-agent policies, per-program and whole-bank measurements coincide and are reported only in the whole-bank columns.



(a) POMO-TSP-100 (b) LEHD-TSP-100

Figure 20: Search gap to the teacher along the pruning path of each bank, for bank sizes $M \leq 20$. Programs are removed one at a time using validation top-1 agreement.

Table 21: Program complexity and search gaps to the teacher on TSP-100. Complexity includes primitives and is reported as per-program means and whole-bank totals.

			Per program			Whole bank		
Policy	M	**Gap (%)**	Stmt.	Vol.	Coef.	Stmt.	Vol.	Coef.
POMO-TSP-100								
Agent: GPT-6	1	4.58				52	1,262	29
Agent: Fable 5.1	1	3.90				40	919	17
EPB bank: simple	8	0.579	11.4	116	2.9	51	573	20
EPB bank: medium	29	0.455	14.5	138	4.5	154	2,147	85
EPB bank: unconstrained	16	0.034	33.6	655	13.0	537	10,485	208
LEHD-TSP-100								
Agent: GPT-6	1	2.09				45	1,072	28
Agent: Fable 5.1	1	1.64				38	831	16
EPB bank: simple	10	0.333	8.8	129	2.5	49	609	23
EPB bank: medium	28	0.255	14.5	242	6.5	206	4,089	152
EPB bank: unconstrained	10	0.172	41.9	943	19.1	419	9,426	191

Complexity and solution quality. Table 21 reports complexity at both the program and bank levels. Including shared primitive implementations, the simple banks contain 51 statements for POMO and 49 for LEHD, compared with 537 and 419 for the unconstrained banks. Their Halstead volumes are also substantially lower, decreasing from 10,485 to 573 for POMO and from 9,426 to 609 for LEHD. Despite these reductions in program code, the simple banks retain search gaps of 0.579% and 0.333% to their respective teachers. These results show that combining compact programs can preserve near-teacher solution quality with substantially less program code.

Bank size and complexity trade-off. Figure 20 shows that, within a given program complexity setting, improvements in solution quality tend to saturate as the bank grows, with limited benefits from further expansion. For the simple and medium settings, this may partly reflect the fixed primitive vocabulary: additional programs combine the same features and may overlap with existing decision rules, limiting the benefits of increasing their number. Meanwhile, relaxing the constraints from simple to medium and unconstrained further reduces the search gaps, but also substantially increases whole-bank code complexity (Table 21). From simple to unconstrained, the absolute gap reductions are approximately 0.55 percentage points for POMO and 0.16 for LEHD, accompanied by increases in statement counts, Halstead volume, and coefficient counts. Bank configuration therefore requires jointly considering program count and per-program complexity, balancing additional gains in solution quality against the resulting code complexity.

Comparison with coding-agent policies. Table 21 compares standalone coding-agent policies with EPB program banks. Including primitive implementations, the simple EPB banks contain 51 and 49 statements, comparable in scale to the agents' 40 to 52 statements for POMO and 38 to 45 for LEHD. Their search gaps are 0.579% and 0.333%, compared with 3.90% and 1.64% for the better coding-agent policy in each setting. Thus, the simple banks achieve lower solution costs with a similar amount of program code. This comparison concerns program code, as EPB additionally uses a learned router.

Listing 3: POMO TSP Simple Tier Code: 2 primitives, 3 statements, Halstead volume 37, 2 coefficients.

```python
import torch
from tsp_feats import feat

def heuristic(locs, current_node, mask, visited, step_norm, first_node):
    # Uses 2 concept(s): dist_cur, heading_cos
    d = feat.dist_cur(locs, current_node)
    h = feat.heading_cos(locs, current_node, visited)

    # Combines proximity with a heading-alignment bonus.
    logits = -6.0 * d + 1.5 * h
    return logits
```

Listing 4: POMO TSP Medium Tier Code: 3 primitives, 4 statements, Halstead volume 66, 3 coefficients.

```python
import torch
from tsp_feats import feat

def heuristic(locs, current_node, mask, visited, step_norm, first_node):
    # 3 concepts: dist_cur (greedy), dist_start (closing leg), nn_dist (isolation)
    d_cur = feat.dist_cur(locs, current_node)
    d_start = feat.dist_start(locs, first_node)
    nn = feat.nn_dist(locs, mask)

    # Favors nearby candidates and proximity to the start,
    # with a small isolation bonus.
    logits = -3.0 * d_cur - 1.5 * d_start + 0.2 * nn
    return logits
```

Figure 21: Example programs from the POMO-TSP-100 banks. `dist_cur` and `dist_start` give distances to the current and start nodes. `heading_cos` measures heading alignment, and `nn_dist` gives the distance to the nearest other unvisited node. Complexity values in the listing captions refer to the displayed program bodies and exclude primitive implementations.

D.10 Improvements through Program Edits

We edit the student on two instance families outside the training distribution: TSP-100 and CVRP-50 instances on a thin diagonal band. Both the teacher and the student perform poorly on these families. The interventions change only program code, and the router and all network weights remain frozen.

Each intervention introduces a closed-form rule that scores candidate nodes without search or route planning. An instance-level shape check restricts the rule to its target family, so its contribution is zero on the uniform instances used for training. The rule modifies program scores, while the router continues to combine the programs and choose the next node.

The rules were designed on a separate set of instances. All results below are on 256 held-out instances per family that played no part in their design. Gaps are the mean excess over LKH for TSP and HGS for CVRP. To test whether the same rule also helps the teacher, we add its score as a bias to teacher's (the POMO model) log-probabilities, $\arg\max_j[\log p_T(j) + \gamma z_j]$, where z is the standardised rule score, and report the best γ.

D.10.1 TSP-100, diagonal band

On a thin band, an efficient tour travels along one side and returns along the other. All 14 programs of the student favour candidates close to the current node, and none reserves nodes for the return leg. The student consequently sweeps both sides on the way out, crosses the band frequently, and leaves scattered nodes to collect before closing the tour with a long edge back to the start.

The edit encourages the student to follow one side outward and the other back, without skipping nodes on the return leg. We add three lane-keeping programs and modify the original 14 programs where they compute the distance from the current node to each candidate. Immediately after that computation, we add penalties to the distances of candidates that break the two-lane pattern, and the rest of each program is unchanged. The penalties discourage choosing nodes on the opposite side during the outward leg, moving against the direction of travel, and skipping nodes on the return leg. They are assigned to the 7 nearest neighbor, 3 phase-based, and 4 closing programs according to their roles. We also add the lane-keeping rule's score to the final scores of all 17 programs. Before addition, the rule score is standardised and scaled to one standard deviation of the receiving program's scores.

For TSP-100, the edit reduces EPB's greedy gap from 24.16% to 7.01% and its search gap from 8.70% to 1.78% (Table 5). Edited EPB outperforms both the teacher and the teacher with the same rule in both settings. After the edit, the number of nodes left for the return lane rises from 15.4 to 40.6, close to LKH's 42.1. The median closing-edge length falls from 21.2 to 3.2 nearest-neighbour spacings, compared with 1.6 for LKH. Figure 6 illustrates this change on one held-out instance.

D.10.2 CVRP-50, diagonal band

On diagonal-band instances, the customers lie on a thin band along the diagonal of the square. Good solutions cut the band into consecutive stretches and serve each stretch with one route. EPB's routes instead drive out to the far parts of the band and take only part of the customers there, so later routes cover the same stretches again: 4.79 routes per instance reach the outer half of the band, against 4.25 for HGS. The 20 programs score customers by distance, angle and demand fit, and none of them divides the band into stretches, giving the router little support for this structure.

We introduce three rules. At the depot, the start rule favours the customers farthest out along the band on one side of the depot, as many as fit in one vehicle, unless going out along one side of the band and back along the other is shorter. While customers of the current stretch lie farther out, the route rule favours keeping to the vehicle's side of the band and moving outward; otherwise it favours the outermost remaining customer. The return rule penalises returning to the depot when this would force one more route and a cheap continuation exists. Each score is standardised, scaled to 8, 1 and 2 times the standard deviation of each program's own scores, and added to the final scores of all 20 programs. For instances that are not bands, the rules are zero, and the programs' outputs are unchanged.

The edit changes EPB's routes as intended. The routes reaching the outer half of the band fall from 4.79 to 4.40 per instance (HGS 4.25), and the excess of twice the farthest depot distance summed over routes falls from 7.2% to 2.0% of the HGS cost. EPB's greedy gap falls from 10.88% to 5.94%, and its search gap from 4.17% to 2.43% (Table 5). Both gaps are below those of the teacher and the teacher with the same rules.

D.11 Overly shallow routing

With the program bank fixed, the one-layer router yields higher held-out KL and larger greedy and multi-start gaps than every deeper configuration. At $D = 128$, increasing depth from one to two layers reduces the greedy gap from 16.33% to 10.03% and the multi-start gap from 3.70% to 2.10%. The narrower four-layer router also outperforms the one-layer model despite having fewer parameters. These results support retaining a sufficiently expressive neural router for state-dependent program composition, while further increases in model size do not consistently improve performance.

Table 22: Router capacity with a frozen program bank ($M = 14$). Cost gaps (%) are relative to the POMO teacher (↓).

Configuration	Parameters	Greedy gap	MS gap
$L = 4,\ D = 64$	213,504	**8.73**	**1.63**
$L = 1,\ D = 128$	249,984	16.33	3.70
$L = 2,\ D = 128$	448,256	10.03	2.10
$L = 3,\ D = 128$	646,528	11.57	2.34
$L = 4,\ D = 128$	844,800	11.15	2.15
$L = 6,\ D = 128$	1,241,344	9.56	1.72
$L = 4,\ D = 256$	3,360,768	10.83	2.05

E LLM Statement

LLMs are a core component of EPB, generating and revising programs as detailed in §4 and §C.2. They were also used to assist with writing, drawing figures, and routine coding. The authors reviewed and verified all AI-assisted content and take full responsibility for the paper.

F Social Impact Statement

EPB may improve the transparency and auditability of neural routing policies by exposing executable decision rules and enabling targeted modifications. These capabilities may help researchers inspect model behavior, diagnose failures, and evaluate policy changes.

This study focuses on synthetic routing benchmarks and does not involve human subjects, sensitive personal data, or real-world deployment. We do not foresee direct negative societal impacts within this scope. Nevertheless, the surrogate approximates the teacher's behavior, and its explanations do not establish the teacher's internal reasoning. Practical deployment should therefore independently validate the surrogate's fidelity, solution quality, and robustness under the intended conditions.

G License

Our experiments use publicly available checkpoints and code assets from prior neural combinatorial optimization models, including POMO [2] and LEHD [3], as teacher policies and/or baselines. For POMO, we use the publicly released repository associated with the NeurIPS 2020 paper. We did not find an explicit license file or named software license in the public repository, and therefore use the checkpoint only for academic research comparison and policy distillation. We do not redistribute the original or modified POMO checkpoints. For LEHD, we use the publicly released repository associated with the NeurIPS 2023 paper. The repository states that the code is copyrighted by CIAM Group and can only be used for non-commercial purposes; commercial use requires contacting the authors. We use the LEHD checkpoint only for non-commercial academic research, follow the stated usage restriction, and do not redistribute the original or modified LEHD checkpoints.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Main contribution and scope are accurately reflected in Abstract and §1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of the work are discussed in §6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The theory proof is displayed in §4.2.1 and §B.4.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We reported it in §5, §C and §D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code repository is linked on the first page.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We reported it in §C, with additional implementation details available in the publicly released code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We reported it in Table 3 , §D.1, §D.7, and §D.8, with additional implementation details available in the publicly released code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We reported this in §C and §D.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The study uses standard benchmark datasets and computational experiments, does not involve human subjects or sensitive data, and complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss potential societal benefits, the scope of the study, and considerations for practical deployment in Appendix F.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?.

    Answer: [Yes]

    Justification: We use publicly available checkpoints from prior NCO models as teacher policies and/or baselines. We cite the original papers and repositories that released these checkpoints, report their available license and usage information in §G, and use them only for research comparison and policy distillation.

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We release the EPB implementation and distilled program banks, together with setup, training, and evaluation instructions, in the public repository linked on the first page.

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects. All experiments are conducted using standard benchmark combinatorial optimization problems and existing model checkpoints.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subject research. Therefore, IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: We have declared in §E.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.