RULER: REASONING FROM TRIALS TO TRANSFERABLE RULES FOR FINITE-BUDGET ONLINE ALGORITHM CONFIGURATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Online algorithm configuration adapts solver parameters to individual instances and evolving search states under a limited evaluation budget. Yet each trial offers more than a performance score: it reveals how a parameter edit affects solver behavior in a particular context. Turning these observations into transferable tuning knowledge requires reasoning about when an edit is applicable, what response it should induce, and whether new evidence supports that expectation. We introduce RULER—Rule-guided Updating through Local Experimental Reasoning, a framework that treats configuration trials as local experiments and distills their outcomes into reusable, falsifiable edit–response rules. RULER retrieves these rules as contextual hypotheses, reassesses their applicability to the current solver state. It then plans targeted experiments to test their predictions, refine promising edits, distinguish competing explanations, or explore alternatives when existing knowledge offers little guidance. This evidence–hypothesis–experiment loop enables RULER to improve the current configuration while accumulating tuning knowledge that can be reused across instances. Experiments across four problem–solver settings show that RULER reduces the reference gap by an average of 10.7% relative to the strongest baseline under matched evaluation budgets, maintains gains under matched wall-clock budgets and across planner scales, and transfers useful configuration knowledge to unseen instances.

1 INTRODUCTION

Metaheuristic algorithms (Talbi, 2021) are widely used for combinatorial optimization (Blum & Roli, 2003), yet their performance can be highly sensitive to hyperparameter configurations, with effective settings varying substantially across problem instances (Schede et al., 2025). To this end, *online algorithm configuration* adapts solver parameters during the optimization process according to the instance and the evolving search state (Reijnen et al., 2025). Such online adaptation requires continuously interpreting solver behavior, relating new observations to previous configuration trials, and deciding how the parameters should be adjusted accordingly, making configuration a highly contextual and sequential decision process (Nguyen et al., 2025). Accordingly, an effective online configurator should integrate heterogeneous search feedback, reason about the observed effects of previous parameter changes, and revise its decisions as new evidence accumulates.

Traditionally, sequential parameter configuration has been approached through model-based search, such as Bayesian optimization (BO) (Snoek et al., 2012), which uses past evaluations to estimate promising configurations, while dynamic configuration methods have used reinforcement-learning controllers to adapt parameter decisions from search-state feedback (Speck et al., 2021). More recently, Large language models (LLMs) (Wei et al., 2022; Dong et al., 2024), with their strong reasoning and in-context adaptation capabilities, provide a promising foundation for online algorithm configuration. For example, AutoEP (Xu et al., 2026) leverages online landscape features with pre-designed rules to guide LLM-based exploration and exploitation, while LLAMBO (Liu et al., 2024b) conditions on historical configuration–performance pairs to predict and generate promising configurations. Across these designs, LLMs function as history-conditioned reasoning mechanisms that synthesize evolving solver signals and previous outcomes into subsequent parameter decisions, demonstrating its potential to steer sequential parameter adaptation.

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

However, for tuning experience to transfer reliably, its reusable unit should extend beyond an isolated trial to capture the context-dependent relation between a parameter edit and the solver response it induces. Namely, the same edit can accelerate progress in one search regime yet reduce diversity or trigger stagnation in another (Eiben et al., 1999; Aleti & Moser, 2016; Doerr & Neumann, 2021), making raw trial histories alone, or context-agnostic adjustment rules, an incomplete basis for reuse. Reusable tuning knowledge should therefore specify when an edit is applicable, what behavioral and performance response it is expected to produce, and how the inferred relation should be revised as supporting or contradicting evidence accumulates.

To this end, we propose RULER—Rule-guided Updating through Local Experimental Reasoning, a framework that turns configuration trials into transferable tuning rules for finite-budget online algorithm configuration. At its core, RULER transforms online configuration from direct parameter proposal into an evidence-driven cycle of hypothesis formation, contextual validation, and purposeful experimentation. Specifically, RULER distills potentially transferable patterns from parameter edits and their observed solver responses into structured *edit–response hypotheses*. Each hypothesis specifies when a bounded edit may be applicable, what behavioral and performance response it is expected to induce, and which observations support or contradict that expectation. The resulting rules are maintained in persistent memory and retrieved to guide configuration decisions in new search states as contextual hypotheses. Rather than treating retrieved hypotheses as globally valid prescriptions, RULER reassesses them against the current solver behavior and local search regime before using them to guide further search. Under a limited evaluation budget, the framework must decide both *which* hypothesis to investigate and *what experimental purpose* the next trial should serve. An LLM plans targeted trials to test experience-based predictions, refine promising edits, distinguish between competing explanations, or explore alternatives where knowledge is limited, and generates configuration edits. The resulting observations are then used to revise the hypothesis memory, closing the evidence–hypothesis–experiment loop. Empirically, RULER improves optimization performance under matched evaluation budgets by combining the discovery of locally reusable tuning regularities with context-aware adaptation, thereby balancing robust knowledge transfer with effective downstream configuration on unseen instances across four problem–solver settings and multiple evaluation budgets. We summarize our contributions as follows:

- **From configuration trials to transferable rules.** We represent reusable tuning knowledge as contextual, falsifiable edit–response rules that specify when a parameter edit may apply, what solver response it should produce, and what evidence supports or contradicts that expectation. This representation allows tuning experience to accumulate and transfer across instances without assuming that an effective edit applies universally.
- **Local experimental reasoning for online configuration.** We introduce RULER, an LLM-guided framework that assesses retrieved rules against the current solver state and plans targeted experiments under a limited evaluation budget. These experiments test rule predictions, refine promising edits, distinguish competing explanations, and explore alternatives, with their outcomes guiding configuration updates and future rule reuse.
- **Empirical effectiveness and transfer.** Across four problem–solver settings and multiple evaluation budgets, RULER achieves an average 10.7% relative gap reduction over the strongest baseline under matched evaluation budgets. Additional studies demonstrate improvements under matched wall-clock budgets on selected instances, robustness across planner scales and evaluation budgets, and effective cross-instance rule transfer.

2 PRELIMINARY

Given an input instance $\boldsymbol{x} \in \mathcal{X}$ and a fixed solver A with configuration space Θ, we aim to identify a configuration θ that maximizes the performance of A on $\boldsymbol{x}$. Evaluating A under configuration θ produces a phase-level behavioral summary $\boldsymbol{z}(\boldsymbol{x}, \theta)$ and a final performance value $r(\boldsymbol{x}, \theta)$, i.e., $(\boldsymbol{z}(\boldsymbol{x}, \theta), r(\boldsymbol{x}, \theta)) = A(\boldsymbol{x}, \theta)$, where $\boldsymbol{z}$ summarizes observable solver behaviors during the search, such as incumbent progress, diversity, local-search gain, repair success, or stagnation. The optimal configuration is defined as

$$\theta^*_{\boldsymbol{x}} = \arg\max_{\theta \in \Theta} r(\boldsymbol{x}, \theta). \tag{1}$$

Since $\theta^*_{\boldsymbol{x}}$ is generally unknown, we employ a policy π to iteratively propose configurations based on previous interactions. At step t, evaluating θ_t yields a trial $\boldsymbol{\tau}_t = (\theta_t, \boldsymbol{z}_t, r_t)$, where $(\boldsymbol{z}_t, r_t) =$

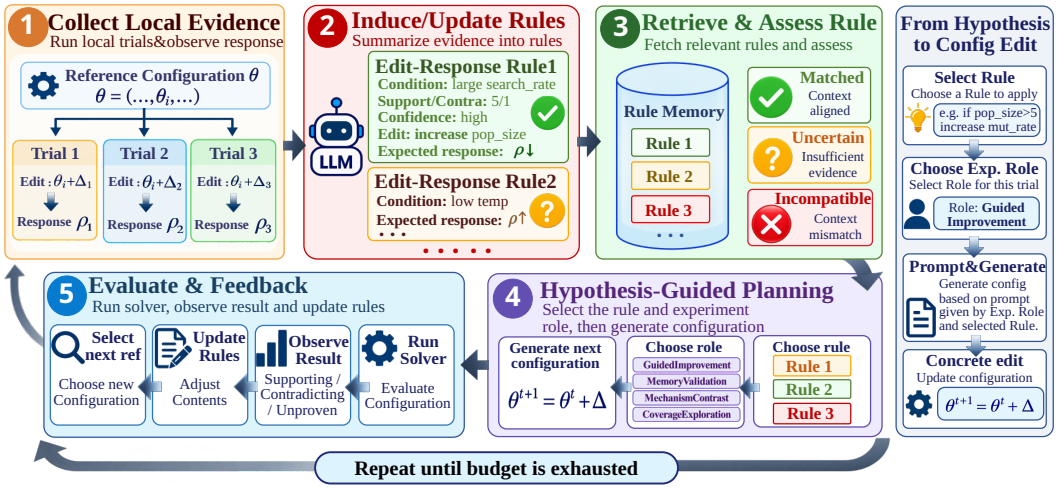

Figure 1: RULER converts local solver evaluations into edit–response rules, retrieves relevant rules from persistent memory, and assesses them in the current search context. During planning, it selects a relevant rule and an experimental role to generate a bounded configuration edit. The evaluation results are used to update the rule memory and guide the next round.

$A(\boldsymbol{x}, \theta_t)$. We denote the accumulated interaction history by $\mathcal{H}_t = (\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_t)$, and the next configuration is proposed as $\theta_{t+1} = \pi(\boldsymbol{x}, \mathcal{H}_t)$. Under a fixed evaluation budget B, the iterative search terminates at $T = \inf\{t : C_t \geq B\}$, where C_t denotes the cumulative evaluation cost up to step t. The final solution is selected as the best-performing configuration encountered within B: $\hat{\theta}_{\boldsymbol{x}} = \theta_{t^*}, t^* \in \arg\max_{0 \leq t \leq T} r_t$. Under a limited evaluation budget, an effective policy should use more than the final performance of each trial. It should also consider how the solver responds when the configuration is changed around the current search state. We therefore instantiate π with an LLM to identify useful edit–response patterns from the interaction history and use them to guide subsequent configuration updates.

3 Method

We present RULER, an evidence-grounded framework for hypothesis-guided online algorithm configuration that treats the effect of a configuration edit as contingent on the search state in which it is applied. RULER distills historical edit–response observations into persistent rules, each of which specifies an applicability condition, an expected solver response, and the evidence supporting or contradicting the relation. When retrieved for the current search state, a rule is instantiated as a contextual hypothesis and reassessed before it is used to guide further optimization. As illustrated in Figure 1, each round begins by evaluating bounded configuration edits from a common reference trial and extracting local evidence from the resulting changes in solver behavior and performance. An LLM then uses this evidence to induce or revise persistent rules, instantiate and assess retrieved rules as contextual hypotheses, and plan the next role-structured experiments. The resulting solver responses update both the interaction history and the rule memory, and the process repeats until the evaluation budget is exhausted.

3.1 Local Evidence and Edit–Response Rule Induction

While automated configuration often evaluates each trial primarily by its final performance, such sparse scalar feedback reveals little about how a specific edit changes solver behavior; RULER therefore treats edits evaluated from a common reference state as comparable local experiments.

Specifically, RULER maintains a reference trial $\boldsymbol{\tau}_{\text{ref}} = (\theta_{\text{ref}}, \boldsymbol{z}_{\text{ref}}, r_{\text{ref}})$, which identifies the current reference configuration and its observed solver behavior. Each $\boldsymbol{\tau}_{\text{ref}}$ is associated with a recoverable solver checkpoint, ignoring the notation for brevity, where all child trials in the current round are evaluated. Evaluating different edits from the same checkpoint controls the starting state and makes the resulting responses directly comparable. At each round, RULER evaluates a batch of bounded edits $\{\delta_i\}_{i=1}^{n_t}$ around θ_{ref}, subject to the parameter types and constraints in Θ, as illustrated in Figure 2. Each edit yields a child configuration θ_i, whose evaluation produces a child trial

$\boldsymbol{\tau}_i = (\theta_i, \boldsymbol{z}_i, r_i)$. For each child, RULER records the response observed after applying δ_i from the reference state as $\rho_i = (\boldsymbol{z}_i, \Delta r_i), \Delta r_i = r_i - r_{\text{ref}}$, where $\boldsymbol{z}_i$ summarizes the solver behavior during the evaluation and Δr_i measures performance change relative to the reference. The edit and its observed response form an edit–response observation $e_i = (\delta_i, \rho_i)$, which serves as the local evidence for rule induction. See Example D.1 for a concrete edit–response observation.

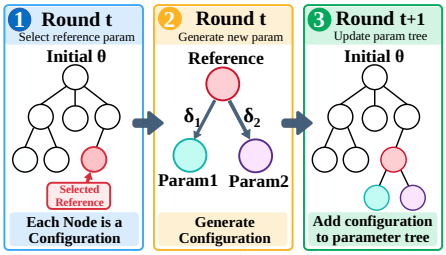

Figure 2: Construction of local trials around a reference configuration.

The local evidence set $\mathcal{E}_t = \{e_i\}_{i=1}^{n_t}$ collects the observations obtained around the current reference. Based on $\mathcal{E}_t$ and the accumulated interaction history, the LLM updates the evidence associated with existing rules and may induce new rules for response patterns not explained by the current memory. Each edit–response rule is represented as $R_j = \left(\phi_j, \mathcal{D}_j, \hat{\rho}_j, E_j^+, E_j^-\right)$, where ϕ_j specifies the conditions under which the rule may apply, $\mathcal{D}_j$ describes the associated family of bounded edits, $\hat{\rho}_j$ specifies the expected solver response, and E_j^+ and E_j^- reference supporting and contradicting observations, respectively. A rule may initially be supported by limited evidence and is revised as additional observations become available. The resulting rules are maintained in a persistent memory $\mathcal{M}_t = \{R_j\}_{j=1}^{m_t}$, which accumulates reusable edit–response relations across optimization instances. Example D.2 provides a concrete illustration of such a rule.

3.2 Contextual Hypothesis Assessment

Given the current reference trial, local evidence, and remaining budget $b_t = B - C_t$, RULER defines the local evidence context as $\mathcal{L}_t = (\boldsymbol{\tau}_{\text{ref}}, \mathcal{E}_t, b_t)$. Although accumulated evidence may justify retaining an edit–response rule in memory, it does not establish that the rule is applicable at the current search state. RULER therefore retrieves a candidate set $\mathcal{Q}_t = \text{Retrieve}(\mathcal{M}_t, \mathcal{L}_t)$ by jointly considering each rule's historical evidence and relevance to the current context. Inclusion in $\mathcal{Q}_t$ indicates that a rule warrants further assessment, rather than that it is already applicable or supported.

For each retrieved rule $R_j \in \mathcal{Q}_t$, RULER instantiates its persistent edit–response relation as a contextual hypothesis $h_{t,j}$: if the current context satisfies the rule condition, an edit covered by the rule is expected to induce its predicted response. An LLM-based matcher then assesses whether this hypothesis is suitable for testing: $m_{t,j} = \text{Match}(R_j, \mathcal{L}_t, \mathcal{H}_t)$, $m_{t,j} \in \{\texttt{matched}, \texttt{uncertain}, \texttt{incompatible}\}$. A hypothesis is `matched` when its condition and edit pattern are compatible with the current context, `incompatible` when the context conflicts with its condition, and `uncertain` when the available information is insufficient for either determination.

Importantly, contextual matching determines whether a hypothesis should participate in subsequent experiment planning, but does not itself provide supporting or contradicting evidence. Only the solver response observed after executing the corresponding edit can support, contradict, or leave the hypothesis unproven. RULER thus separates historical credibility, contextual applicability, and empirical validation. See Example D.3 and Example D.4 for an illustration of memory filtering, contextual retrieval, and target-side rule validation.

3.3 Hypothesis-Guided Experiment Planning

Given the retrieved rules, their contextual assessments, and the remaining evaluation budget, RULER factorizes next-round planning into three decisions: selecting an associated rule when applicable, assigning an experimental role, and generating a bounded configuration edit conditioned on that role. This structure avoids asking the LLM to search the full configuration space directly and instead organizes each evaluation around an explicit experimental purpose.

RULER considers four roles. Specifically, MemoryValidation empirically tests a retrieved rule that is contextually relevant but lacks sufficient target-side evidence. GuidedImprovement refines a promising direction supported by existing rule evidence or measured responses on the current instance. MechanismContrast evaluates a qualitatively different edit to distinguish competing explanations of the observed solver behavior. CoverageExploration probes insufficiently tested regions or mechanism families when the current rules provide limited guidance. Accordingly,

a matched hypothesis with supporting target-side evidence may guide GUIDEDIMPROVEMENT; a matched or uncertain hypothesis that remains unvalidated is routed to MEMORYVALIDATION; competing hypotheses motivate MECHANISMCONTRAST; and the absence of a suitable hypothesis motivates COVERAGEEXPLORATION. Incompatible hypotheses are excluded from rule-guided planning. See Example D.5 for an illustration of how four roles work.

Guarding is implemented as a controller policy rather than a separate experimental role. When an evaluation contradicts the tested hypothesis, RULER deactivates the rule for the current context and records the observation as contradicting evidence. This affects whether the rule remains trusted, but the quality of the evaluated configuration is assessed independently from its performance. For each batch position, the LLM selects an associated rule $R_{j_i} \in \mathcal{Q}_t \cup \varnothing$, assigns a role a_i, and proposes a bounded edit δ_i. With maximum parallel width P, the resulting batch is

$$\mathcal{B}_{t+1} = \{(\delta_i, a_i, R_{j_i})\}_{i=1}^{P_t}, \qquad P_t \le P, \tag{2}$$

where P_t is determined by the remaining evaluation budget. Each edit is instantiated as a valid child configuration of the current reference and evaluated from the common reference checkpoint. The resulting trials are appended to the interaction history. For rule-associated experiments, the measured response is classified as supporting, contradicting, or unproven with respect to the tested hypothesis; all observations are retained as local evidence for inducing or revising edit–response rules. Let $i^* = \arg\max_{1 \le i \le P_t} r_i$ denote the best-performing child. If $r_{i^*} > r_{\mathrm{ref}}$, RULER updates the reference to $\boldsymbol{\tau}_{i^*}$; otherwise, the current reference is retained. The remaining children are preserved as comparative evidence. These rounds continue until the evaluation budget is exhausted. RULER then performs an evaluation-free consolidation step over the complete interaction history to revise provisional rules, induce previously unexplained relations, and merge or audit existing rules before updating the persistent memory. Example D.6 provides an end-to-end illustration of ruler and we provide additional methodological details in Appendix A.

4 EXPERIMENT

Evaluation benchmarks. We evaluate RULER on two classical combinatorial optimization problems: the Traveling Salesman Problem (TSP) (Dantzig et al., 1954) and the Capacitated Vehicle Routing Problem (CVRP) (Dantzig & Ramser, 1959). TSP seeks a minimum-cost tour that visits each city exactly once, whereas CVRP seeks a set of minimum-cost vehicle routes that serve all customers subject to vehicle-capacity constraints. We evaluate RULER with four configurable metaheuristic solvers across the two problems. For TSP, we use a genetic algorithm (GA) (Holland, 1992) with 2-opt local search (Croes, 1958), and simulated annealing (SA) (Kirkpatrick et al., 1983). For CVRP, we use particle swarm optimization (PSO) (Venter & Sobieszczanski-Sobieski, 2003) with iterated local search (ILS) (Lourenço et al., 2003), and SA with incumbent ILS. Specific dataset instances we used are listed in Appendix E.1. Implementation details for these solvers are provided in Appendix E.2. These settings cover both population-based and trajectory-based search with different mechanisms. We evaluate TSP on TSPLIB95 (Reinelt, 1991), using 63 instances for GA+2opt and 15 for SA, and CVRP on CVRPLIB (Uchoa et al., 2017), using 11 instances for each solver setting.

Baselines. We compare RULER against four categories of configuration methods. *1) Sampling-based methods* include Random-HPO (Bergstra & Bengio, 2012), Latin hypercube sampling (LHS) (McKay et al., 2000), and Sobol-QMC (Sobol, 1967). *2) Model-based methods* include TPE (Bergstra et al., 2011), SMBO (Hutter et al., 2011), and SMAC3 (Lindauer et al., 2022). *3) Evolutionary methods* include Differential Evolution (DE) (Storn & Price, 1997) and NSGA-II (Deb et al., 2002). *4) LLM-based methods* include Point-LLM and AutoEP-Rep., the latter following AutoEP (Xu et al., 2026). We also report results using the default hyperparameter configuration as a baseline. Since AutoEP does not release a complete runnable Python evaluation pipeline or the full prompt set, we implement AUTOEP-REP. from the published method and separately compare RULER with the originally reported AutoEP results in Section 4.2.

Evaluation metrics. We report the percentage gap to the Best-Known Solution (BKS), which measures the relative difference between the objective value of the solution found by a method and the corresponding benchmark reference objective. For TSP, we use the proven optima provided by TSPLIB95 (Reinelt, 1991); for CVRP, we use the proven optimum when available and otherwise the best-known solution reported by CVRPLIB (Uchoa et al., 2017). Let f_{method} denote the objective

Table 1: Overall performance across four problem–solver settings. We report mean reference gap (%) and end-to-end runtime (sec), with lower values indicating better performance. Bold entries denote the best gap and underlined entries the strongest non-RULER baseline. The bottom rows show RULER's relative gap reduction over the fixed configuration and strongest baseline.

Method		**TSP GA+2opt**		**TSP-SA**		**CVRP PSO+ILS**		**CVRP SA+ILS**		**Average**
Class	**Method**	Gap ↓	Time ↓	Gap ↓	Time ↓	Gap ↓	Time ↓	Gap ↓	Time ↓	Gap ↓
Control	Fixed config	0.914	19.0	6.544	1.0	3.274	6.7	2.060	11.2	3.198
Sampling	Random-HPO	0.767	18.4	4.578	1.1	2.375	8.3	1.827	11.7	2.387
	LHS	0.704	18.4	4.478	1.2	2.989	8.7	1.770	12.1	2.485
	Sobol-QMC	<u>0.683</u>	18.1	4.719	1.2	2.471	8.3	2.001	11.0	2.469
Model-based	TPE	0.801	18.6	4.800	1.2	3.052	8.5	2.251	11.6	2.726
	SMBO	0.792	18.6	4.593	1.4	<u>2.340</u>	9.6	<u>1.702</u>	12.4	2.357
	SMAC3	0.712	26.0	<u>4.367</u>	8.9	2.932	14.4	1.705	19.5	2.429
Evolutionary	DE	0.751	18.9	4.876	1.1	2.440	8.1	2.096	12.4	2.541
	NSGA-II	0.766	18.8	4.592	1.2	2.874	8.7	1.863	10.8	2.524
LLM-based	Point-LLM	0.901	23.5	6.544	27.6	2.860	38.7	1.975	34.5	3.070
	AutoEP-Rep.	0.696	36.2	4.422	26.1	2.403	39.8	1.779	37.8	<u>2.325</u>
Ours	**RULER**	**0.620**	27.6	**4.027**	11.5	**1.951**	43.4	**1.546**	18.8	**2.036**
Ours vs. fixed configuration		**+32.2%**	–	**+38.5%**	–	**+40.4%**	–	**+25.0%**	–	**+34.0%**
Ours vs. strongest baseline		**+9.2%**	–	**+7.8%**	–	**+16.6%**	–	**+9.2%**	–	**+10.7%**

value obtained by the evaluated method and f_{ref} the corresponding benchmark reference value. For minimization, $\text{Gap}(\%) = 100(f_{\text{method}} - f_{\text{ref}})/f_{\text{ref}}$, where lower values indicate better solutions. To quantify the improvement of RULER over a baseline, we also report the relative reduction in reference gap, $\text{Gain}(\%) = 100(\text{Gap}_{\text{base}} - \text{Gap}_{\text{RULER}})/\,\text{Gap}_{\text{base}}$. Runtime and planner latency are reported in the corresponding efficiency studies.

Implementation details. The main protocol consists of eight online configuration rounds, with RULER and baselines evaluating three candidate configurations per round. RULER and LLM-based method uses Qwen3-4B-AWQ (Yang et al., 2025) as its local planner served locally through vLLM (Kwon et al., 2023). Full implementation details, including evaluation budgets, configuration spaces, evaluation protocols, and prompt template are provided in Appendix E.2 and Appendix E.3.

4.1 Overall Performance and Target-Side Validation

Overall comparison. Table 1 reports the overall performance across the four solver settings. RULER achieves the lowest mean reference gap in all cases, with an average 10.7% relative gap reduction over the strongest baselines. The consistent gains across GA-, SA-, and PSO-based solvers show that RULER delivers broad improvements across diverse metaheuristic families. In terms of end-to-end runtime, RULER is also competitive with LLM-based baselines in three of the four settings. The exception is CVRP PSO+ILS, which exposes the largest configuration space with ten tunable hyperparameters spanning swarm dynamics, perturbation, local search, and ILS; these richer parameter interactions require more detailed rule summarization and proposal reasoning, resulting in longer model outputs and higher inference cost.

Instance-level consistency. Table 2 compares RULER with a representative strong comparator for each solver setting. W/T/L denotes the number of instances where RULER achieves a lower, equal, or higher gap than the comparator, respectively. RULER achieves a lower mean gap in all four settings, while wins and ties together outnumber losses in every setting. This shows that the performance gains are broadly consistent across individual instances.

Table 2: Instance-level consistency against strong comparators (W/T/L: win/tie/loss).

Setting	Comparator	Mean gap (%) ↓		W/T/L	Gain
		RULER	Comp.		
TSP GA+2opt	Sobol-QMC	**0.620**	0.683	22/27/14	**9.2%**
TSP-SA	AutoEP-Rep.	**4.027**	4.422	8/2/5	**8.9%**
CVRP PSO+ILS	SMAC3	**1.951**	2.932	4/5/2	**33.5%**
CVRP-SA+ILS	LHS	**1.546**	1.770	5/5/1	**12.7%**

Table 3: Cross-instance transfer: evaluation outcomes of retrieved hypotheses on later targets.

Setting	Hypotheses		Evaluation outcome		
	Retrieved	Evaluated	Supporting	Unproven	Contradicting
TSP GA+2opt	74	74	39	17	18
TSP-SA	32	32	13	15	4
CVRP PSO+ILS	177	61	21	12	28
CVRP-SA+ILS	18	9	1	8	0

Table 4: Comparison with published AutoEP results.

TSP GA+2opt — same five instances as AutoEP												
Method	eil51		rd100		kroA150		rat575		dsj1000		Mean	
	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)
AutoEP Reported	**0.000**	5.800	0.010	10.100	0.010	31.900	**0.080**	170.200	3.580	312.800	0.736	106.160
RULER-8	**0.000**	**0.107**	0.430	**0.113**	0.207	**0.123**	2.554	**0.923**	3.451	**4.372**	1.328	**1.128**
RULER-16	**0.000**	0.187	**0.000**	0.207	**0.000**	0.250	1.225	2.437	**2.119**	10.040	**0.669**	2.625
CVRP PSO+ILS — scale-matched official CVRPLIB instances												
Method	$N = 20$		$N = 50$		$N = 100$		$N = 200$		$N = 500$		Mean	
	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)	Gap (%)	Time (min)
AutoEP Reported	0.060	5.800	0.090	10.700	0.830	31.200	2.480	58.600	4.250	168.500	1.542	54.960
RULER-8	**0.000**	**3.007**	**0.000**	**3.345**	6.848	**3.238**	2.475	**3.365**	2.475	**3.623**	2.360	**3.315**
RULER-8+ILS	**0.000**	3.020	**0.000**	3.385	**0.000**	3.302	**1.581**	3.472	**2.315**	3.738	**0.779**	3.383

Cross-instance transfer. Table 3 examines whether rules learned from previous instances remain useful on subsequent targets. Across all four solver settings, we observe supporting target-side evidence for retrieved hypotheses, demonstrating that RULER can reuse configuration knowledge across instances for both TSP and CVRP. The target-side evaluations further allow RULER to assess which retrieved hypotheses remain useful in the current instance.

4.2 Comparison with Published AutoEP Results

Comparison protocol and implementation details. Because the released AutoEP materials do not include the complete prompt set or a directly reusable Python pipeline, we reproduce the published method as closely as possible and report the original AutoEP results separately. For fair comparison, we disable vLLM when comparing against AutoEP. We use the same five TSP instances reported by AutoEP; for CVRP, where only problem sizes are available, we use scale-matched official CVRPLIB instances. RULER-8 denotes the standard eight-round setting, RULER-16 doubles the rounds, and RULER-8+ILS adds a 5k-iteration fresh-start ILS refinement.

Better solution quality with substantially lower runtime. Across the TSP and scale-matched CVRP comparisons in Table 4, the corresponding RULER variants achieve lower mean reference gaps and substantially lower reported runtimes than AutoEP, despite using a compact 4B local planner instead of AutoEP's 30B model. Cross-instance rule transfer further helps on current instance.

4.3 Planner Model Parameter Scale Robustness

Figure 3 compares six Qwen planners from 1.7B to 32B under the same state representation, rule memory, prompts, and evaluation budget. Although solution quality varies somewhat with model scale, the overall gaps remain close, and larger planners remain nearly consistent. This indicates that RULER shows limited sensitivity to planner size and can make effective configuration decisions without relying on a high-capacity LLM. At the same time, inference latency increases substantially with model size. Taken together, these results motivate the use of a compact 4B planner in our main experiments, as it provides a favorable balance between quality and efficiency.

4.4 Analysis

Performance under Same Wall-Clock Budgets. To visualize how solution quality evolves under comparable end-to-end wall-clock budgets, Figure 4 plots the optimization trajectories on three large TSP GA+2opt instances. RULER's runtime includes local-model inference and memory updates, while baseline trajectories are truncated to the same elapsed time. Since the planner introduces a relatively small per-call overhead compared with the growing solver cost on larger instances, its relative runtime impact becomes less significant as problem size increases. Despite this additional planning overhead, RULER reaches the lowest final gap on all three instances.

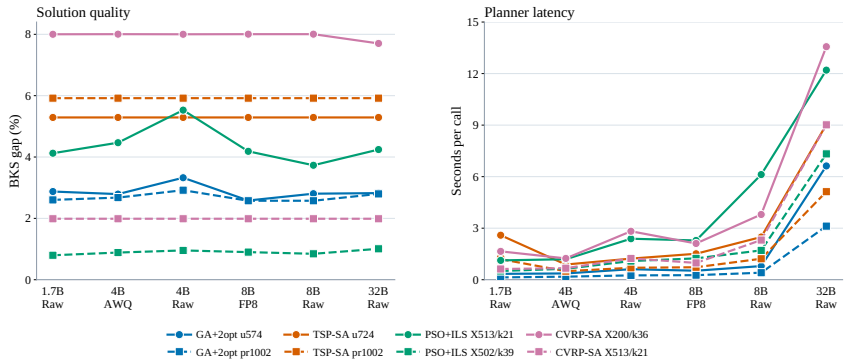

Figure 3: RULER solution quality and inference latency under different planner scales.

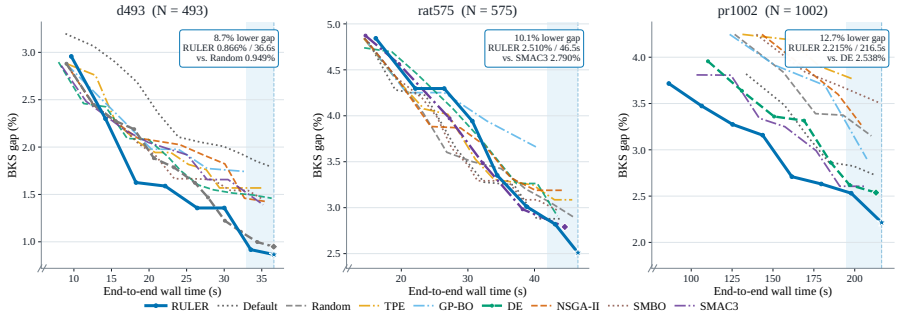

Figure 4: Solution-quality trajectories under matched wall-clock budgets.

Table 5: Component ablation of RULER. Lower values are better, and the relative degradation from the full method is reported in parentheses.

Problem	Solver	Target	Full	Component Ablation			Point-LLM
				w/o Rule Induction & Transfer	w/o Contextual Assessment	w/o Role Planning	
TSP	GA+2-opt	d493	**0.87**	1.32 (+52.5%)	1.71 (+97.7%)	1.58 (+82.5%)	1.54 (+78.2%)
		u724	**2.33**	2.82 (+20.8%)	3.43 (+47.3%)	2.52 (+7.9%)	2.82 (+21.1%)
CVRP	PSO+ILS	X-n101-k25	**6.69**	7.46 (+11.6%)	7.33 (+9.6%)	13.70 (+104.8%)	7.52 (+12.4%)
		X-n200-k36	**3.18**	8.42 (+164.5%)	7.88 (+147.5%)	3.41 (+7.0%)	4.02 (+26.3%)

Component Ablation. Table 5 ablates three components of RULER: *Rule Induction and Transfer*, *Contextual Hypothesis Assessment*, and *Hypothesis-Guided Experiment Planning*. We use Point-LLM as an LLM-only baseline. Removing *Rule Induction and Transfer* limits RULER's ability to reuse edit–response knowledge across instances, reducing the benefit of prior experience on new targets. Removing *Contextual Hypothesis Assessment* makes it harder to determine whether transferred rules fit the current context, making their reuse less reliable. Removing *Hypothesis-Guided Experiment Planning* leaves the LLM to jointly decide what to test and how to modify the configuration over a much broader decision space, making reliable decisions more difficult under limited budget.

Multiple-Budget Evaluation. Appendix C evaluates RULER under different round ranging from 2 to 16, demonstrating that its performance remains robust across different evaluation budgets.

4.5 Case Study: When to Reuse Retrieved Rules

To illustrate how rule reuse affects an intermediate configuration decision, Figure 5 presents two rounds from different cases of online configuration optimization. On `rd400`, RULER evaluates `P-0344` even though another retrieved rule has a higher retrieval score. Its positive target-side response provides supporting evidence for the current activation, so the rule is retained for reuse. On `d493`, by contrast, the highest-scoring retrieved rule `P-0582` yields contradicting target-side evidence. RULER therefore rejects that activation and proceeds with the alternative `P-0344`. Together, these cases illustrate the distinction between retrieval and reuse: retrieval identifies plausible rules to test, whereas measured target-side evidence determines whether a particular activation is accepted. A more detailed decision-level trace is provided in Appendix B.

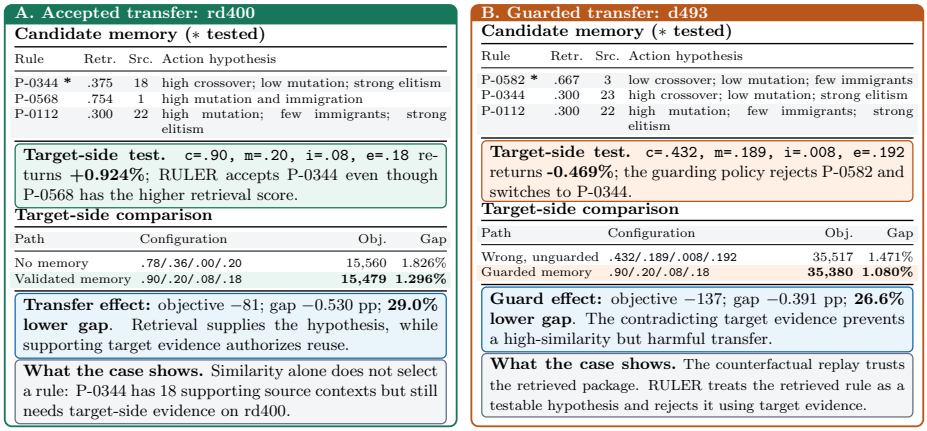

Figure 5: Case study of accepted and rejected rule reuse on two TSP GA+2opt targets.

5 Related Work

LLM-Guided Algorithm Configuration and Design. Algorithm configuration is commonly formulated as a sequential decision problem, where parameters are adapted to the evolving search state. GLEET (Ma et al., 2024), for example, learns a reinforcement learning policy for dynamic control during evolutionary search. More recent methods place LLMs directly in the optimization loop: AgentHPO (Liu et al., 2025), AutoEP (Xu et al., 2026), and LMABO (Ngo et al., 2026) use task information, search history, landscape features, or optimization state to guide subsequent parameter or acquisition decisions. Other work, including EoH (Liu et al., 2024a), ReEvo (Ye et al., 2024), LLaMEA (Van Stein & Bäck, 2024), and EoH-S (Liu et al., 2026), extends this idea to heuristic and algorithm design. RULER differs by learning from the *effects of executed edits*: instead of using past trials only as context for the next proposal, it extracts persistent local knowledge about how specific configuration changes affect solver behavior and performance.

Experience Memory for Optimization and LLM Agents. Agents increasingly reuse experience across trajectories. ExpeL (Zhao et al., 2024) extracts reusable natural-language insights from past interactions, Meta Agent Search (Hu et al., 2025) archives discovered agent designs, and Reasoning BO (Yang et al., 2026) maintains explicit knowledge during black-box optimization. RULER instead stores localized *edit–response hypotheses*: each hypothesis records when a configuration edit may apply and what response it is expected to induce. These hypotheses are retrieved as reusable knowledge, but are reassessed against the current local evidence before guiding new decisions.

Hypothesis-Guided Experimental Optimization. Prior work also uses hypotheses or external knowledge to decide which expensive evaluations to perform (Ren et al., 2023; Seng et al., 2025). HypBO (Cissé et al., 2024) incorporates expert hypotheses into Bayesian optimization, BORA (Cissé et al., 2025) uses LLM-derived contextual knowledge, and POPPER (Huang et al., 2025) sequentially tests free-form hypotheses through agent-designed experiments. In contrast, RULER induces hypotheses directly from observed configuration interventions and keeps them inside the online optimization loop. Retrieved hypotheses are reassessed to determine the role of subsequent evaluations—for example, discovering new response patterns, verifying or contrasting hypotheses, refining supported directions, or guarding against known failures. Each new observation then updates the hypothesis memory, allowing the evaluation budget to improve both the incumbent configuration and the reusable knowledge that informs later decisions.

6 Conclusion

We introduced RULER, a hypothesis-guided framework for short-budget online algorithm configuration. RULER converts observed edit–response relations into reusable rules, assesses them as contextual hypotheses, and plans evaluations through structured experimental roles. Across TSP and CVRP with four metaheuristic solvers, RULER consistently outperforms strong sampling, model-based, evolutionary, and LLM-based baselines under limited budgets. Ablations and case studies confirm the benefits of rule reuse, contextual assessment, and structured planning, with compact local planners proving sufficient. Overall, RULER shows that experience from online configuration can be learned and transferred across problem instances to guide subsequent configuration.

AI USE STATEMENT

In this work, we used generative AI to assist with code generation, language polishing, and limited manuscript drafting. We reviewed all AI-assisted work and verified the generated code before use. We take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce RULER and the reported experiments throughout the main paper and appendix. Section 4 describes the benchmark suites, problem–solver settings, baselines, evaluation metrics, online configuration protocol, and planner used in the main experiments. Appendix A provides the detailed RULER decision process, while Appendix E specifies the four base optimizers, their exposed configuration spaces, fixed solver components, and optimization procedures. The planner-facing prompt contracts and information-access restrictions are given in Appendix E.3. Finally, Appendix B and Appendix D provide auditable execution traces and concrete examples of rule induction, contextual assessment, target-side validation, and experiment planning.

ETHICS STATEMENT

We have considered the potential ethical implications of this work, including issues related to fairness, privacy, security, and possible misuse. Our work is intended for research purposes, and we encourage responsible use of the proposed methods and careful consideration of their limitations when applied in real-world settings. We have made efforts to report our methodology and experimental results accurately and transparently, and we take responsibility for the integrity of the research presented in this paper.

REFERENCES

Aldeida Aleti and Irene Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys (CSUR)*, 49:1–35, 2016.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

Abdoulatif Cissé, Xenophon Evangelopoulos, Sam Carruthers, Vladimir V. Gusev, and Andrew I. Cooper. Hypbo: Accelerating black-box scientific experiments using experts' hypotheses. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 3881–3889, 2024.

Abdoulatif Cissé, Xenophon Evangelopoulos, Vladimir V. Gusev, and Andrew I. Cooper. Language-based bayesian optimization research assistant (bora). In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 4967–4975, 2025.

Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6: 791–812, 1958.

George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2:393–410, 1954.

George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6: 80–91, 1959.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6:182–197, 2002.

Benjamin Doerr and Frank Neumann. A survey on recent progress in the theory of evolutionary algorithms for discrete optimization. *ACM Transactions on Evolutionary Learning and Optimization*, 1:1–43, 2021.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 1107–1128, 2024.

Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, 3:124–141, 1999.

John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. The MIT press, 1992.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Kexin Huang, Ying Jin, Ryan Li, Michael Y. Li, Emmanuel Candes, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 25372–25437, 2025.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220:671–680, 1983.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23:1–9, 2022.

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. In *Proceedings of the 41st International Conference on Machine Learning*, 2024a.

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

Siyi Liu, Chen Gao, and Yong Li. AgentHPO: Large language model agent for hyper-parameter optimization. In *Conference on Parsimony and Learning*, volume 280, pp. 1146–1169, 2025.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *International Conference on Learning Representations*, volume 2024, pp. 31252–31284, 2024b.

Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pp. 320–353. Springer, 2003.

Zeyuan Ma, Jiacheng Chen, Hongshu Guo, Yining Ma, and Yue-Jiao Gong. Auto-configuring exploration-exploitation tradeoff in evolutionary computation via deep reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1497–1505, 2024.

Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42:55–61, 2000.

Giang Ngo, Dat Phan Trong, Dang Nguyen, Sunil Gupta, and Svetha Venkatesh. Adaptive acquisition selection for bayesian optimization with large language models. In *International Conference on Learning Representations*, volume 2026, pp. 149314–149336, 2026.

Tai Nguyen, Phong Le, André Biedenkapp, Carola Doerr, and Nguyen Dang. On the importance of reward design in reinforcement learning-based dynamic algorithm configuration: A case study on onemax with (1+(λ, λ))-ga. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1162–1171, 2025.

Robbert Reijnen, Yaoxin Wu, Zaharah Bukhsh, and Yingqian Zhang. Graph-supported dynamic algorithm configuration for multi-objective combinatorial optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

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

Zhichu Ren, Zekun Ren, Zhen Zhang, Tonio Buonassisi, and Ju Li. Autonomous experiments using active learning and ai. *Nature Reviews Materials*, 8:563–564, 2023.

Elias Schede, Moritz Seiler, Kevin Tierney, and Heike Trautmann. Deep reinforcement learning for instance-specific algorithm configuration. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1190–1198, 2025.

Jonas Seng, Fabrizio Ventola, Zhongjie Yu, and Kristian Kersting. Hyperparameter optimization via interacting with probabilistic circuits. In *Proceedings of the Fourth International Conference on Automated Machine Learning*, volume 293, pp. 11/1–39, 2025.

Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Ilya M Sobol. Distribution of points in a cube and approximate evaluation of integrals. *USSR Computational mathematics and mathematical physics*, 7:86–112, 1967.

David Speck, André Biedenkapp, Frank Hutter, Robert Mattmüller, and Marius Lindauer. Learning heuristic selection with dynamic algorithm configuration. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pp. 597–605, 2021.

Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

El-Ghazali Talbi. Machine learning into metaheuristics: A survey and taxonomy. *ACM computing surveys (CSUR)*, 54:1–32, 2021.

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

Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29: 331–345, 2024.

Gerhard Venter and Jaroslaw Sobieszczanski-Sobieski. Particle swarm optimization. *AIAA journal*, 41:1583–1589, 2003.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Zhenxing Xu, Yizhe Zhang, Weidong Bao, Hao Wang, Ming Chen, Haoran Ye, Wenzheng Jiang, Hui Yan, and Ji Wang. Autoep: Llms-driven automation of hyperparameter evolution for metaheuristic algorithms. In *International Conference on Learning Representations*, volume 2026, pp. 139927–139945, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhuo Yang, Daolang Wang, Lingli Ge, Beilun Wang, Tianfan Fu, and Yuqiang Li. Reasoning bo: Enhancing bayesian optimization with the reasoning power of llms. *Materials Genome Engineering Advances*, pp. e70086, 2026.

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.

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

A DETAILED DECISION PROCESS OF RULER

This section provides additional details on the RULER decision process introduced in Section 3. At each decision, the controller observes the current reference trial, recent trajectory statistics, previous configuration–response observations, and remaining budget, while the solver, objective, and feasible configuration space remain fixed. Benchmark optima and post-hoc gaps are never exposed to the controller.

RULER retrieves relevant rules as contextual hypotheses rather than commands. A contextual assessor determines whether each retrieved rule is `matched`, `uncertain`, or `incompatible` using only the observable context and interaction history. The planner then assigns MEMORYVALIDATION, GUIDEDIMPROVEMENT, MECHANISMCONTRAST, or COVERAGEEXPLORATION and generates the corresponding candidate configurations. Contextual assessment and role allocation determine what is tested, but neither creates evidence or assigns an objective value.

At each decision round, all candidates are generated as bounded edits around the same reference configuration. Their observed solver responses are used to evaluate the activated rules and accumulate evidence. Based on these evaluations, the best improving child becomes the reference configuration for the next round; if no child improves upon the current reference, the reference is retained, while the outcomes of the remaining candidates are retained for comparison. A contradicting response rejects the current rule activation without immediately discarding the underlying rule, as the failure may be specific to the present search context. Future activations are therefore governed by the accumulated evidence about when the rule is or is not effective. After each instance, RULER consolidates the newly collected experience by inducing new local rules, merging overlapping ones, revising their applicability or expected effects, and auditing the rule memory for consistency, without consuming additional solver evaluations. Figure 6 summarizes the detailed decision process.

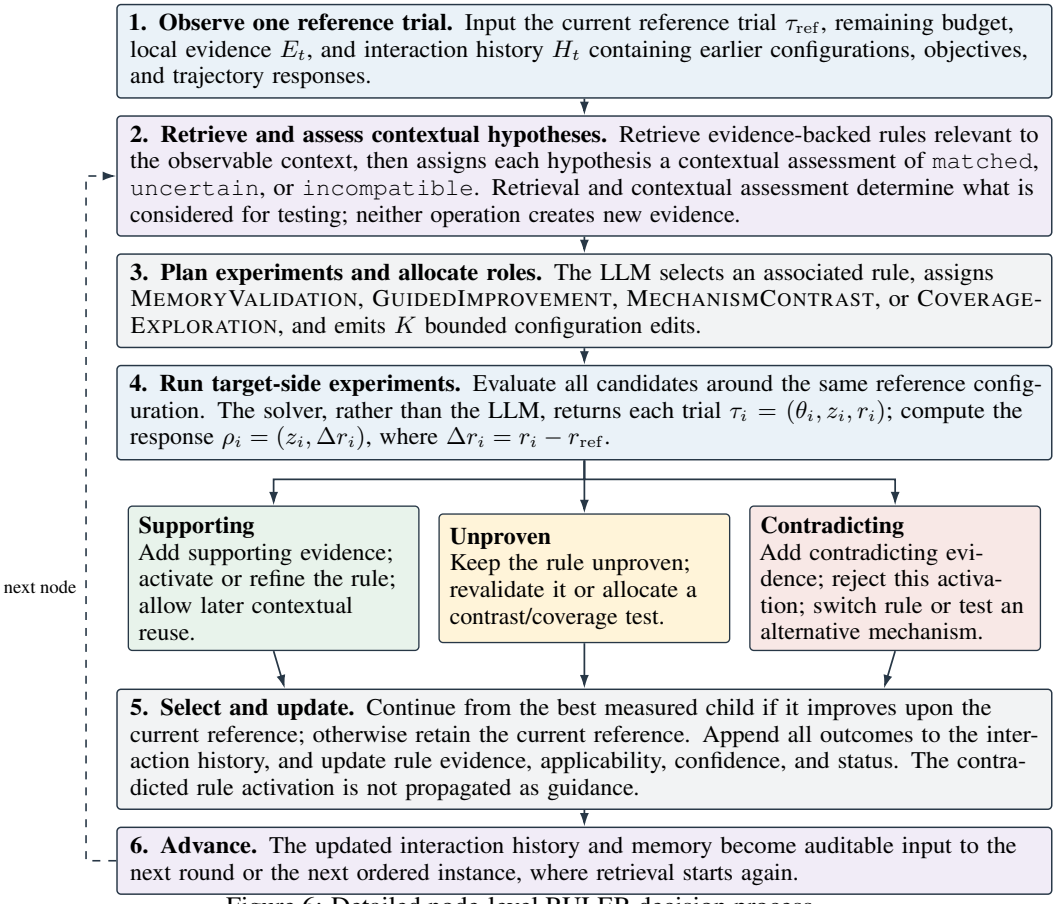

Figure 6: Detailed node-level RULER decision process.

B FINE-GRAINED RULER EXECUTION TRACE

To complement the case study in Section 4.5, we provide a decision-level trace of TSP GA+2opt run on instance `d493`, focusing on decisions round 3 and round 4. The trace records the observable controller context, retrieved rule, role assignments, model proposals, configuration evaluations, and subsequent evidence updates. The official reference objective is used only for post-hoc analysis and is never exposed to the controller.

Controller input at decision round 3. The current reference has objective 35,791 after 320 evaluations, with `unique_ratio` $= 0.600$, `stagnation_fraction` $= 0.0031$, and recent improvement rate $= 0.0156$. Its configuration is $(c, m, i, e) = (0.75, 0.18, 0.08, 0.12)$. RULER retrieves `TSP-P-0001` with memory score 0.17215 and target applicability `large/low-stagnation/medium-diversity`. The retrieved configuration is $(0.95, 0.20, 0.05, 0.08)$, supported by 26 earlier contexts and two measured target rounds. The contextual matcher returns `matched`, indicating that the rule is suitable for testing in the current context; this assessment itself creates no new evidence.

Decision Round 3: local configuration evaluation. The matched rule is assigned to MEMORYVALIDATION, while the remaining lanes are assigned to GUIDEDIMPROVEMENT and COVERAGEEXPLORATION. The coverage-lane model completion is {`c:0.85, m:0.15, i:0.03, e:0.06`}. For this minimization setting, *Ref. gain* denotes the reduction in objective relative to the current reference, so positive values indicate improvement.

Lane / producer	c	m	i	e	Objective	Ref. gain
Memory validation (selected)	0.95	0.20	0.05	0.08	**35,698**	**+93**
Guided improvement	0.90	0.20	0.08	0.18	35,757	+34
Coverage exploration / LLM	0.85	0.15	0.03	0.06	**35,698**	+93

Reference transition. The memory-validation and coverage lanes tie at 35,698; deterministic lane order selects the evidence-backed memory package. The selected memory trial improves the current reference by 93 and is retained as supporting target-side evidence. The reference therefore advances from 35,791 to 35,698. At the next decision, 440 evaluations have been used, `unique_ratio` rises to 0.8375 and `stagnation_fraction` to 0.2273.

Decision Round 4: reassessment after the reference transition. The rule remains `matched` and is again assigned to MEMORYVALIDATION. The coverage-lane model completion is {`c:0.80, m:0.15, i:0.03, e:0.06`}.

Lane / producer	c	m	i	e	Objective	Ref. gain
Memory validation (selected)	0.95	0.20	0.05	0.08	**35,624**	**+74**
Guided improvement	0.90	0.20	0.08	0.18	35,698	0
Coverage exploration / LLM	0.80	0.15	0.03	0.06	35,648	+50

Measured evidence and memory update. The memory lane is now the unique best child and is selected. Its measured reference-relative improvement of 74 objective units is stored as supporting evidence `tsp:d493:d3:c0`; no benchmark reference enters retrieval, proposal, or selection. Across the two decisions, the incumbent improves from 35,791 to 35,624 (-167, or -0.467%). Post hoc, the reference gap decreases from 2.254% to 1.777%, a 0.477 percentage-point (21.2% relative) reduction.

C PERFORMANCE ACROSS DIFFERENT CONFIGURATION BUDGETS

Figure 7 evaluates the effect of the round-level configuration budget under a controlled experimental setting. For each budget, we run RULER and all baselines on the same set of benchmark instances while keeping all other experimental settings unchanged, such that the number of configuration rounds is the only varying factor. For TSP GA+2opt, this budget-scaling study uses the same 15-instance subset as TSP-SA. Across the evaluated budgets, RULER consistently achieves strong performance

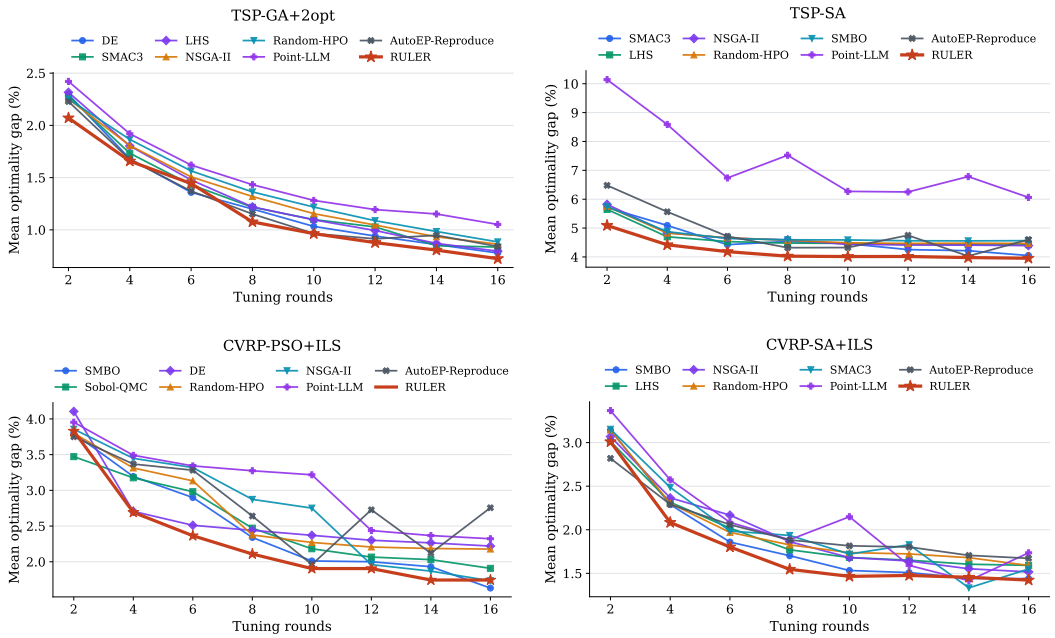

Figure 7: Mean optimality gap across multiple round-level configuration evaluation budgets. RULER remains competitive across all four solver settings and achieves the lowest gap at many intermediate and large budgets.

and overall outperforms the competing methods. In particular, its advantage persists as the available budget increases, indicating that RULER can effectively make use of additional configuration rounds rather than relying on a particular budget choice. These results further show that the gains observed under the default eight-round setting are robust across a broad range of configuration budgets.

D Example Cases of RULER

This section gives concrete examples of the observations, rule cards, contextual validation, role allocation, and memory updates used by the final RULER implementation. Unless stated otherwise, all numerical values come from TSP GA+2opt run on instance `d493`. The controller observes the current reference trial, configuration bounds, and measured trajectory responses; the benchmark reference is used only for post-hoc reporting and is absent from all planner inputs.

D.1 Example of a Local Edit–Response Observation

One local edit–response observation.

At the third displayed decision on `d493`, the current reference has objective 35,791 and configuration

$$\theta_{r-1} = (c, m, i, e) = (0.75, 0.18, 0.08, 0.12),$$

where c, m, i, and e denote crossover rate, mutation rate, immigrant fraction, and elite fraction. The memory-validation lane evaluates

$$\theta_{r,0} = (0.95, 0.20, 0.05, 0.08),$$

which induces the bounded edit

$$\delta_{r,0} = \{c : +0.20,\ m : +0.02,\ i : -0.03,\ e : -0.04\}.$$

The child objective is 35,698, giving a reference-relative objective improvement of 93. Its observed trajectory includes

$$\Delta\texttt{unique_ratio} = +0.2375, \qquad \texttt{stagnation_fraction} = 0.8333.$$

Together, the bounded edit and this measured behavioral and performance response form the local edit–response observation for this trial.

The resulting observation records the edit, the solver trajectory, and the performance change relative to the current reference. It therefore provides local evidence about how this configuration edit affects the search in the current context.

D.2 Example of a Mined Local Edit–Response Rule

The TSP implementation realizes a rule card as an evidence-backed intervention package, represented by a target configuration together with its applicability and response evidence. Immediately before `d493`, one such package is `TSP-P-0001`.

Rule package `TSP-P-0001`.

Intervention $\mathcal{D}_j$:

$$(c, m, i, e) = (0.95, 0.20, 0.05, 0.08).$$

Applicability descriptor ϕ_j: the package stores the observable size, stagnation, diversity, and recent improvement regime in which its evidence was collected.

Expected response $\widehat{\rho}_j$: non-negative target response relative to the current reference under the stored applicability conditions.

Evidence before `d493`: 146 supporting references from 26 earlier contexts, no contradictory reference, status `supported`, and score 0.170945310.

Evidence added on `d493`: supporting target responses from successful activations, together with unproven and contradicting responses observed as the search context changes.

State after `d493`: status `supported`, score 0.176737437, five target support records, and one target contradiction record.

This package is not an unconditional claim that one configuration is optimal. It is a falsifiable intervention with an applicability descriptor and explicit provenance. New target evidence can strengthen its support, leave it unproven, or add a contradiction without erasing the earlier audit trail.

D.3 Example of Memory Filtering and Rule Validation

At the start of `d493`, persistent memory contains six viable packages. The three highest-scoring entries are summarized below.

Memory at target entry.

```
package       score       source contexts   support refs
TSP-P-0001    0.170945310        26              146
TSP-P-0282    0.070159159        17               61
TSP-P-0111    0.039100249         6               33

memory limit: 6 packages
eligible status: supported or validated
target context: tsp:d493
```

The executable memory layer first removes packages whose evidence status is no longer `supported` or `validated`, ranks the viable packages by their evidence score and support count, and retains at most six. Retrieval then combines stored applicability with the observable target context. At the first `d493` decision, `TSP-P-0001` has not yet been tested in this context. It is therefore returned as a hypothesis for an explicit MemoryValidation experiment, with structural match score 0.300; it is not directly authorized for exploitation.

The target-side test reaches objective 36,016 and is recorded as supporting evidence. RULER marks `d493` as a tested context. Subsequent retrieval uses the measured target evidence, rather than source support alone. In the logged later decisions the contextual match is 0.700, and reuse remains conditional on the accumulated target-side evidence.

D.4 Example of Contextual Hypothesis Assessment

Before one decision on d493, RULER retrieves the historically supported package TSP-P-0001. The package contains the target configuration

$$(c, m, i, e) = (0.95, 0.20, 0.05, 0.08),$$

together with an applicability condition describing the solver contexts in which this configuration previously produced non-negative target responses relative to their references. Before entering d493, the package is supported by 146 evidence records from 26 earlier contexts and has no contradictory record.

The instance identifier d493 is shown only for trace readability and is not exposed to the matcher. The observable target context is

```
evaluations:                320
reference objective:        35791
unique_ratio:               0.6000
stagnation_fraction:        0.0031
recent_improvement_rate:    0.0156
retrieved package:          TSP-P-0001
```

The contextual matcher compares these observable context features with the package condition and checks whether the stored edit remains meaningful under the current configuration bounds and interaction history. It returns

$$m_{t,j} = \texttt{matched},$$

with contextual match score 0.700. This result means only that TSP-P-0001 is a suitable hypothesis for the current context. It does not add supporting evidence and does not establish that the package will improve the objective. The hypothesis is passed to the experiment-planning stage, which decides whether and how it should be tested.

D.5 Example of Hypothesis-Guided Experiment Planning

At each node, a role states why a solver lane is informative; it does not assign an objective value and does not impose a fixed number of parameter changes. At the third displayed d493 decision, RULER allocates three complementary lanes:

```
role                  (c, m, i, e)          objective   ref. gain
memory_validation     (.95,.20,.05,.08)       35698          +93
guided_improvement    (.90,.20,.08,.18)       35757          +34
coverage_exploration  (.85,.15,.03,.06)       35698          +93
```

The role assignment determines what is tested; only the resulting measured responses enter the target-side evidence.

All three configurations are complete, bounded, distinct, and evaluated around the same reference configuration. The memory and coverage lanes reach the same objective; deterministic lane order selects the evidence-backed package, while all evaluated outcomes remain in the interaction history as comparative evidence. At the next decision, the same memory hypothesis is tested again after the context changes and again produces a positive reference-relative response.

The fourth canonical role appears in the same archived run on burma14. There, MechanismContrast changes the reference configuration

$$(0.95, 0.20, 0.05, 0.08) \quad \text{to} \quad (0.78, 0.36, 0.00, 0.20),$$

deliberately contrasting the crossover–mutation–selection regime. The trial is not selected and is retained as unproven evidence. This example shows the intended separation among the four roles: MEMORYVALIDATION tests transfer, GUIDEDIMPROVEMENT follows measured target response, MECHANISMCONTRAST tests a qualitatively different mechanism, and COVERAGEEXPLORATION covers an under-tested region.

D.6 END-TO-END RULER EXECUTION TRACE

We finally summarize the complete information flow for `d493`. The trace starts with a memory hypothesis learned from earlier instances, follows its target validation and reuse, records a later contradiction, and shows how the incumbent can switch to a non-memory role.

1. Retrieve a testable memory hypothesis. Before the target run, `TSP-P-0001` is supported by 26 source contexts. RULER retrieves its complete configuration $(0.95, 0.20, 0.05, 0.08)$ for an explicit MEMORYVALIDATION test. Only observable context, prior evidence, configuration bounds, and recent responses are visible to the controller.

2. Establish target-side evidence. The first target test obtains objective 36,016 and provides supporting target-side evidence. The package is selected, the evidence reference `tsp:d493:d0:c0` is stored, and later reuse is conditioned on the accumulated target-side evidence.

3. Allocate complementary roles at each node. The next coverage experiment improves the incumbent to 35,791. The following two decisions compare memory validation, guided improvement, and coverage exploration around the current reference. The memory package is selected at 35,698 and then again at 35,624, with positive reference-relative responses recorded as target-side evidence.

4. Record contradiction and change direction. At displayed round 6, the memory activation produces a contradicting measured response. RULER stores this contradiction and selects the coverage-exploration child at 35,526 instead. The failed activation is not propagated as the incumbent.

5. Preserve the full evidence trail. The final selected objective is 35,506, compared with the initialized objective 43,157. After the instance, `TSP-P-0001` contains five target support records and one target contradiction record and remains `supported`. Its final memory score is 0.176737437. The final incumbent uses the coverage configuration $(0.85, 0.15, 0.03, 0.06)$, demonstrating that memory guides the search without forcing every later decision to reuse the same rule.

Condensed selected-incumbent trace.

displayed round	selected role	objective
1	memory_validation	36016
2	coverage_exploration	35791
3	memory_validation	35698
4	memory_validation	35624
6	coverage_exploration	35526
8	coverage_exploration	35506

E EXPERIMENT IMPLEMENTATION DETAILS

This section provides implementation details for the underlying configurable optimizers and the prompt contracts used by RULER mentioned in Section 4. For each optimizer, we specify the exposed configuration space and the corresponding optimization procedure. We then summarize the planner-facing prompt contracts used in the four experimental settings.

E.1 BENCHMARK INSTANCES

For reproducibility, we report the exact benchmark instances used in each problem–solver setting. The TSP instances are drawn from TSPLIB95 and the CVRP instances from CVRPLIB. The instance identifiers used in the main experiments are listed below.

TSP GA+2opt *63 instances*

`burma14, ulysses16, ulysses22, att48, eil51, berlin52, st70, eil76, pr76, gr96, rat99, kroA100, kroB100, kroC100, kroD100, kroE100, rd100, eil101, lin105, pr107, pr124, bier127, ch130, pr136, gr137, pr144, ch150, kroA150, kroB150, pr152, u159, rat195, d198, kroA200, kroB200, gr202, ts225, tsp225, pr226, gr229, gil262, pr264, a280, pr299, lin318, linhp318, rd400, fl417, gr431, pr439, pcb442, d493, att532, ali535, u574, rat575, p654, d657, gr666, u724, rat783, dsj1000, and pr1002.`

TSP-SA *15 instances*

`eil51, berlin52, pr76, rd100, kroA150, pr226, a280, lin318, rd400, pcb442, att532, rat575, u724, dsj1000, and pr1002.`

CVRP PSO+ILS *11 instances*

`P-n21-k2, E-n22-k4, B-n50-k7, E-n51-k5, M-n101-k10, X-n101-k25, M-n200-k17, X-n200-k36, X-n491-k59, X-n502-k39, and X-n513-k21.`

CVRP-SA+ILS *11 instances*

`P-n21-k2, E-n22-k4, B-n50-k7, E-n51-k5, M-n101-k10, X-n101-k25, M-n200-k17, X-n200-k36, X-n303-k21, X-n502-k39, and X-n513-k21.`

E.2 OPTIMIZERS IMPLEMENTATION DETAILS

We evaluate RULER with four configurable base optimizers TSP GA+2opt, TSP-SA, CVRP PSO+ILS, and CVRP-SA+ILS corresponding to the solver settings used in the main experiments mentioned in Section 4. In each setting, RULER controls only the exposed configuration variables listed below, while the remaining solver components, operators, and evaluation procedure are fixed. We summarize the configuration space and optimization procedure for each solver.

E.2.1 TSP GA+2OPT

Configuration space. The TSP GA+2opt solver exposes four configuration variables controlling crossover, mutation, random immigration, and elite survival. The population size (80), offspring batch size (80), tournament size (5), order crossover, inversion mutation, and exhaustive 2-opt applied to every non-immigrant offspring are fixed throughout the experiments.

Hyperparameter (implementation key)	Symbol	Range
Crossover probability (`crossover_rate`)	p_c	$[0.40, 0.95]$
Inversion-mutation probability (`mutation_rate`)	p_m	$[0.02, 0.60]$
Random-immigrant fraction (`immigrant_fraction`)	p_{imm}	$[0, 0.20]$
Elite-survivor fraction (`elite_fraction`)	p_{elite}	$[0.02, 0.20]$

Algorithm 1 TSP GA+2opt

Require: TSP instance I; configuration
$\theta = (p_c, p_m, p_{\text{imm}}, p_{\text{elite}}) \in \Theta_{\text{TSP-GA}}$; offspring-evaluation budget B
Ensure: Best tour $x^\star$

Construct nearest-neighbor tours from ten sampled start vertices
Repeat the constructed tours to form a population P of size 80
Evaluate P and set $x^\star \leftarrow \arg\min_{x \in P} f_I(x)$
while fewer than B offspring have been evaluated **do**
    Insert $\lfloor p_{\text{imm}}|P| \rceil$ random-permutation immigrants into offspring set O
    while $|O|$ is below the offspring batch size **do**
        $x_a, x_b \leftarrow \text{TOURNAMENTSELECT}(P, 5)$
        $x \leftarrow \text{ORDERCROSSOVER}(x_a, x_b)$ with probability p_c; otherwise $x \leftarrow x_a$
        Apply inversion mutation to x with probability p_m
        $x \leftarrow \text{EXHAUSTIVETWOOPTDESCENT}(x, I)$
        Evaluate x, append it to O, and update $x^\star$ if improved
    end while
    $P \leftarrow \text{ELITISTREPLACEMENT}(P, O, p_{\text{elite}})$
end while
return $x^\star$

E.2.2 TSP-SA

Configuration space. The TSP-SA solver starts from a fixed nearest-neighbor tour. Its configuration controls the annealing schedule, Markov-chain length, reheating behavior, neighborhood selection, and uphill-acceptance mechanism. The effective Markov-chain length is the problem-size-scaled base length multiplied by λ.

Hyperparameter (implementation key)	Symbol	Range / choices
Initial temperature (`initial_temperature`)	T_0	$[0.05, 100]$ (log)
Geometric cooling rate (`cooling_rate`)	α	$[0.95, 0.9995]$
Markov-chain length factor (`markov_chain_factor`)	λ	$[0.25, 4]$
Reheating stagnation threshold (`reheat_threshold`)	h	$[0.10, 0.80]$
Reheating multiplier (`reheat_factor`)	γ	$[1, 5]$
2-opt move probability (`two_opt_bias`)	$p_{2\text{opt}}$	$[0.70, 1]$
Secondary move operator (`secondary_move`)	m_s	{`relocate`, `swap`}
Uphill-acceptance rule (`acceptance_mode`)	a	{`metropolis`, `threshold`}

Algorithm 2 TSP-SA

Require: TSP instance I; configuration
$\theta = (T_0, \alpha, \lambda, h, \gamma, p_{2\text{opt}}, m_s, a) \in \Theta_{\text{TSP-SA}}$; move-evaluation budget B
Ensure: Best tour $x^\star$ and cumulative annealing state

$x \leftarrow \text{NEARESTNEIGHBORTOUR}(I)$; $x^\star \leftarrow x$; $T \leftarrow T_0$
for $b = 1, \ldots, B$ **do**
    Choose 2-opt with probability $p_{2\text{opt}}$; otherwise choose secondary move m_s
    Sample a candidate-guided or global-random move and compute cost change Δ
    if $\Delta \le 0$ or $\text{ACCEPTUPHILL}(\Delta, T, a)$ **then**
        Apply the move to x and update $x^\star$ if improved
    end if
    if the current Markov chain of length factor λ is complete **then**
        $T \leftarrow \alpha T$
    end if
    if the normalized no-improvement horizon reaches h **then**
        $T \leftarrow \gamma T$ and reset the stagnation counter
    end if
end for
return $x^\star$ and the cumulative SA state

E.2.3 CVRP PSO+ILS

Configuration space. The CVRP PSO+ILS solver uses a fixed swarm of 32 particles. Particle positions are represented as giant-tour customer permutations, and velocities are represented as bounded sequences of swap operations. The exposed configuration controls particle movement, random perturbation, bounded local search, and the incumbent-level iterated local search (ILS).

Hyperparameter (implementation key)	Symbol	Range / choices
Velocity-inertia retention rate (`inertia_rate`)	w	$[0,\ 1]$
Personal-best attraction rate (`cognitive_rate`)	c_1	$[0,\ 1]$
Global-best attraction rate (`social_rate`)	c_2	$[0,\ 1]$
Random-swap rate (`random_swap_rate`)	p_{swap}	$[0,\ 0.60]$
Bounded-local-search probability (`local_search_rate`)	p_{LS}	$[0.05,\ 0.90]$
Maximum velocity length (`velocity_limit`)	$v_{\max}$	integers $[4,\ 80]$
Random-perturbation strength (`perturbation_strength`)	q	$[0,\ 0.25]$
ILS history length (`ils_history_length`)	H	integers $[20,\ 600]$
ILS neighborhood size (`ils_num_neighbours`)	N_{ILS}	integers $[8,\ 100]$
Exhaustive search on a new best (`ils_exhaustive_on_best`)	e_{ILS}	{`true`, `false`}

Algorithm 3 CVRP PSO+ILS

Require: CVRP instance I; configuration
$\theta = (w, c_1, c_2, p_{\text{swap}}, p_{\text{LS}}, v_{\max}, q, H, N_{\text{ILS}}, e_{\text{ILS}})$
$\in \Theta_{\text{CVRP-PSO}}$; particle-update budget B; ILS budget L
Ensure: Best feasible route set $x^\star$

```
Initialize 32 giant-tour particles using sweep, nearest-neighbor, and random customer orders
Decode and evaluate all particles; initialize personal and global bests
while fewer than B particle updates have been evaluated do
    for each selected particle p do
        Retain each existing velocity swap with rate w
        Append swap sequences toward the personal best with rate c1
      and toward the global best with rate c2
        Add random swaps controlled by p_swap and q; truncate to v_max
        Apply the velocity and run bounded local search with probability p_LS
        Decode and evaluate the resulting giant tour; update personal and global bests
    end for
end while
x* ← IncumbentILS(global best, I, H, N_ILS, e_ILS, L)
return x*
```

E.2.4 CVRP-SA+ILS

Configuration space. The CVRP-SA+ILS solver starts from a giant tour constructed by the sweep heuristic. Candidate giant tours are converted into feasible CVRP solutions by the route decoder before objective evaluation. The exposed configuration controls the annealing schedule, reheating behavior, route decoding, neighborhood mixture, and uphill-acceptance mechanism; an incumbent-level ILS is applied after the annealing phase.

Hyperparameter (implementation key)	Symbol	Range / choices
Initial temperature (`initial_temperature`)	T_0	$[0.1,\ 100]$ (log)
Geometric cooling rate (`cooling_rate`)	α	$[0.95,\ 0.9995]$
Markov-chain length factor (`markov_chain_factor`)	λ	$[0.5,\ 8]$
Reheating stagnation threshold (`reheat_threshold`)	h	$[0.10,\ 0.80]$
Reheating multiplier (`reheat_factor`)	γ	$[1,\ 5]$
Target route-fill fraction (`target_fill`)	η	$[0.75,\ 1]$
Route-splitting policy (`split_policy`)	s	{`greedy`, `target_fill`}
Neighborhood move mixture (`move_mix`)	m	{`mostly_reverse`, `balanced`, `mostly_relocate`, `mostly_swap`}
Uphill-acceptance rule (`acceptance_mode`)	a	{`metropolis`, `threshold`}

Algorithm 4 CVRP-SA+ILS

Require: CVRP instance I; configuration
$\theta = (T_0, \alpha, \lambda, h, \gamma, \eta, s, m, a) \in \Theta_{\mathrm{CVRP-SA}}$
move-evaluation budget B; ILS budget L
Ensure: Best feasible route set $x^\star$ and cumulative annealing state
Initialize a sweep giant tour and decode it using policy s and target fill η
Set the current and best solutions; $T \leftarrow T_0$
for $b = 1, \ldots, B$ **do**
    Sample a reverse, relocate, or swap move according to move mixture m
    Decode the candidate using (s, η) and compute cost change Δ
    if $\Delta \leq 0$ or ACCEPTUPHILL(Δ, T, a) **then**
        Accept the candidate and update $x^\star$ if improved
    end if
    Cool by α after each Markov chain of factor λ
    Reheat by γ when the normalized stagnation horizon reaches h
end for
$x^\star \leftarrow$ INCUMBENTILS$(x^\star, I, L)$
return $x^\star$ and the cumulative SA state

E.3 PROMPT CONTRACTS

We provide detail prompt implementation in Section 4. All planners receive only observable optimization state, valid configuration bounds, measured responses, and evidence-backed memory. Instance identity, benchmark reference objectives, optimality gaps, and hidden benchmark metadata are excluded. Each evaluated proposal is a complete configuration. The shared role vocabulary is `memory_validation`, `guided_improvement`, `mechanism_contrast`, and `coverage_exploration`.

TSP GA+2opt. The model controls the COVERAGEEXPLORATION lane:

```
Propose one globally distinct complete TSP configuration.  Explore a
materially different four-parameter interaction, not a copy of any
tested or reserved configuration.  Infer values from the supplied
state, bounds, and measured responses.  Return a complete in-bound
configuration and a short observable search hypothesis.
```

Rule retrieval, MEMORYVALIDATION, and GUIDEDIMPROVEMENT are performed outside this prompt.

TSP-SA. The model produces a GUIDEDIMPROVEMENT draft and a MECHANISMCONTRAST draft. Role assignment is an independent call:

```
Use matched transfer memory only when its observable conditions fit.
Do not blindly copy it or repeat any tested or reserved configuration.
Return one complete TSP-SA configuration.  Classify its experimental
purpose using exactly one of the four allowed roles.
```

An additional call summarizes at most two hypotheses and may cite a trial as support only when its measured target-side response provides supporting evidence.

CVRP PSO+ILS. The reasoning pipeline first selects zero to two retrieved rules, then allocates two complementary lanes, drafts their configurations, and reviews the pair:

```
A nearby rule is a hypothesis worth testing, not a fact.  Select no
rule when coverage exploration is more useful.  Assign each final lane
one allowed role, one concrete PSO action detail, and a distinct
experimental intent.  Preserve selected rule IDs, avoid evaluated
```

```
configurations, and ground exact values in measured trajectory
evidence.
```

The memory call cites exact evidence references; executable validation derives the supported values and rejects unsupported summaries.

CVRP-SA+ILS. The model chooses at most one transfer rule and one of the four experimental roles before producing the evaluated draft:

```
Prefer supporting target-side evidence that matches current route
structure and move productivity.  Reject a rule when its mechanism
conflicts with current evidence.  Preserve productive boundary values,
avoid tested configurations, and separate guided improvement from
mechanism contrast.
```

Invalid drafts are repaired without changing the allocated role. Hypothesis summarization retains both supporting and contradicting target-side evidence.