LEARNING INSTANCE-ADAPTIVE SOLVER CONFIGURATION FROM EXECUTION FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixed-integer programming solvers expose many parameters whose best settings depend on individual instances, but evaluating candidate configurations is costly. We formulate instance-adaptive tuning as a budgeted sequential decision problem and train a feedback-conditioned Qwen2.5-7B-Instruct policy through reinforcement learning with online SCIP rollouts. Over four turns, the policy uses solver diagnostics, prior outcomes, and the time limit to choose sparse parameter edits or retain the best observed configuration between executions. These actions span presolve, branching, separation, and primal heuristics, allowing targeted changes across solver subsystems without modifying SCIP internals. On a 53-instance validation set, the policy achieves positive paired win–loss margins in all 16 comparisons with Default SCIP, Random Search, Sequential Model-Based Algorithm Configuration (SMAC3), and prompt-only Base Qwen over four per-call solver limits from 100 to 400 seconds. It also achieves positive margins in 23 of 24 comparisons with six prompt-only language-model agents. At the 300-second limit, its returned configurations solve 24 instances, compared with at most 21 for the four key baselines.

1 INTRODUCTION

Modern mixed-integer programming (MIP) solvers expose hundreds of parameters that control presolve, branching, node selection, cutting planes, primal heuristics, and search management. Their effectiveness depends on the instance and the demands of its solution process. Settings that strengthen the root relaxation may be less useful when tree search dominates, while additional heuristic effort may find feasible solutions but leave less time for improving bounds. Choosing an instance-specific configuration requires balancing these competing uses of computation. Yet each candidate requires another solver execution, making per-instance search costly when only a few calls are available.

Classical methods such as Sequential Model-Based Algorithm Configuration (SMAC3) search a mixed discrete–continuous parameter space (Hutter et al., 2011; Lindauer et al., 2022) and may need more per-instance evaluations than a small online budget permits. Learning-enhanced MIP solvers often target an internal decision such as branching or presolve (Gasse et al., 2019; Liu et al., 2024a). We study a policy that uses experience acquired during training to guide a short sequence of configuration decisions for a new instance. It adapts between SCIP executions (Bolusani et al., 2024), using the outcome of one run to choose settings for the next run on the same instance.

Solver logs provide feedback for this interaction. They reveal whether a feasible solution was found, how primal and dual bounds progressed, how large the search tree became, and how much time was spent in presolve, separation, heuristics, and strong branching. These diagnostics describe both progress and the computation used to achieve it. Reducing the work of one subsystem need not improve the overall outcome; the next solver execution establishes the value of an edit, including its interaction with unchanged settings. Diagnostics therefore inform the next decision, while outcomes provide the training signal.

Language models can combine heterogeneous diagnostics, descriptions of parameter roles, and structured actions. Recent work applies them to optimization modeling (Ahmaditeshnizi et al., 2024), black-box optimization (Yang et al., 2024; Liu et al., 2024b), and cutting-plane separator configuration (Lawless et al., 2024). Prompting can propose a plausible edit, but does not train the model on the solver outcome of that edit. In a sequential interaction, the model must also use previous outcomes to

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.

decide what to try next and whether an earlier configuration is worth retaining. We train the policy through solver execution to make those decisions.

The policy begins with diagnostics from Default SCIP and conducts four tuning turns between executions on the same instance. We formulate this interaction as a budgeted sequential decision problem. At each turn, the policy receives the latest feedback, a bounded history, and the best observed configuration. The environment maintains this *configuration incumbent* according to an internal solver signal. The policy either applies sparse edits to the incumbent or preserves it through a hold; unedited parameters retain their incumbent values. Its 45-parameter interface spans presolve, branching and node selection, separation, and primal heuristics, allowing targeted changes across subsystems while keeping useful settings available.

Training links these decisions to the outcome of the returned configuration. We optimize a Qwen2.5-7B-Instruct policy (Yang et al., 2025) through online reinforcement learning with SCIP rollouts and call the resulting policy Full RL. Its terminal objective rewards improvement over Default SCIP and discourages deterioration from the best configuration observed during the interaction. Rollouts span multiple instances and four per-call solver limits, with the available limit included in the input so that the policy can condition its decisions on the time available for each execution.

Evaluation on a 53-instance validation set covers per-call SCIP limits of 100, 200, 300, and 400 seconds. The comparisons include Default SCIP, Random Search, SMAC3, prompt-only Base Qwen, and six prompt-only frontier agents. Full RL has more paired wins than losses against each of the four key baselines at every limit. This advantage extends beyond default settings to online configuration search and prompted tuning, and is accompanied by additional completed solves at the larger limits.

Contributions.

1. We train a feedback-conditioned policy that uses SCIP diagnostics to choose sparse parameter edits or preserve the best observed configuration within a small solver-call budget. Online reinforcement learning trains the sequence of edits and holds against the outcome of the returned configuration, connecting successive tuning decisions to their final solver result.
2. We show that Full RL achieves positive paired win–loss margins in all 16 comparisons with Default SCIP, Random Search, SMAC3, and prompt-only Base Qwen, and in 23 of 24 comparisons with six prompt-only frontier agents. At 300 seconds, its returned configurations solve 24 instances, compared with at most 21 for any key baseline.
3. We examine a joint ablation that removes candidate feedback and applies edits independently to Default SCIP. Full RL wins on 37 of 53 instances at both 100 and 400 seconds. The ablated policy ends with a hold on all 53 instances at each limit, returning Default settings. The ablation clarifies the role of configuration state. A hold can preserve an earlier useful configuration only when that configuration remains available as its base.

2 Sequential Solver Configuration

An episode begins with Default SCIP. The policy uses each subsequent solver outcome to revise a few settings or retain a configuration that has worked well so far.

Let $\mathcal{D}$ be a distribution of MIP instances, $x \sim \mathcal{D}$ an instance, θ_0 the Default SCIP parameter configuration, and b a per-call solver time limit. Given a configuration θ, executing $\mathcal{S}(x, \theta, b)$ returns a native solver outcome y and diagnostics d. The diagnostics describe status, primal and dual bounds, the optimality gap, node counts, time spent in solver subsystems, and the primal-dual integral (PDI). We index the initial Default execution by $t = 0$ and the four tuning turns by $t = 1, \ldots, 4$.

At turn t, the environment supplies a configuration θ_t^{base} that achieved the best internal solver signal among the initial Default run and earlier candidates. This *configuration incumbent* is a set of parameter values; a solver incumbent is instead a feasible MIP solution. The policy π_ϕ sees the latest diagnostics, the configuration incumbent, the available time limit, and a bounded history. Its action a_t specifies a sparse set E_t of validated parameter–value edits or a hold. The candidate θ_t^{cand} copies θ_t^{base} and changes only the parameters named in E_t; a hold has $E_t = \varnothing$ and preserves the base. After SCIP executes the candidate, the environment updates the feedback and its best observed

configuration. The configuration produced on turn four is returned for evaluation. Appendix D gives the exact composition rule.

2.1 Budgeted deployment objective

Let y_0 be the Default SCIP outcome at limit b and y_{final} the outcome of the returned configuration at the same limit. The relation $\succ$ denotes the native outcome ordering described in Section 4. Let c_t be the solver-call allowance for turn t, C the total tuning allowance, and $\mathbb{I}$ the indicator function. The target policy maximizes paired advantage over Default SCIP under the tuning allowance.

$$\max_{\pi_\phi} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{I}\{y_{\text{final}} \succ y_0\} - \mathbb{I}\{y_0 \succ y_{\text{final}}\}\right] \quad \text{s.t.} \quad \sum_t c_t \le C. \tag{1}$$

A win against Default contributes $+1$ to Equation 1, a loss contributes -1, and a tie contributes zero. Section 3 gives the terminal score used to train the policy.

3 Learning from Solver Execution

Figure 1 connects the configuration loop to policy training. The environment turns each SCIP execution into a diagnostic state; the next outcome judges the policy's sparse edit in the context of a complete solver configuration.

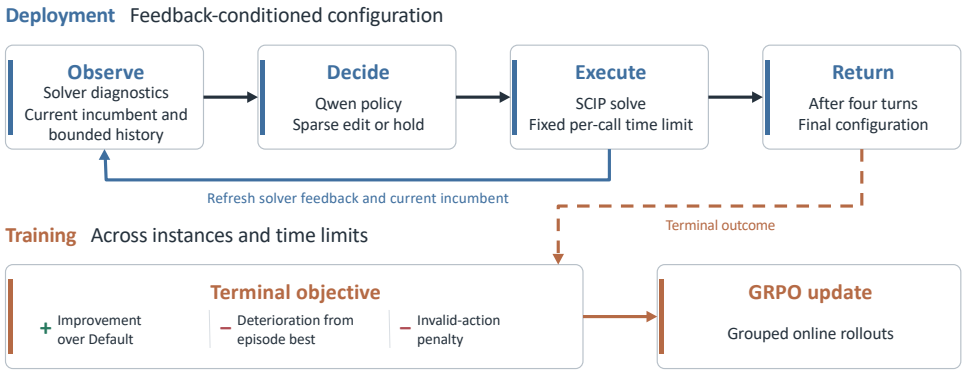


Figure 1: Feedback-conditioned configuration and policy training. The upper loop uses execution diagnostics and the best observed configuration to choose a sparse edit or hold. The lower path updates the policy from grouped online rollouts and the terminal score in Equation 2.

3.1 Diagnostic state

The environment converts raw SCIP output into a compact state describing feasibility, primal and dual progress, search-tree growth, and time spent in solver subsystems. The policy input combines these measurements with the best observed configuration, recent actions and outcomes, the current turn, and the available solver time limit. It distinguishes the latest candidate from the best configuration so an unsuccessful proposal does not erase useful settings from the interaction state. Appendix A lists the diagnostic fields, and Appendix D describes the configuration and history representation.

3.2 Structured action space

The model emits a compact JavaScript Object Notation (JSON) action over 45 documented SCIP parameters covering branching and node selection, presolve, separation, and primal heuristics. It generates parameter identifiers and values directly as tokens; a schema check validates their types and domains before SCIP executes the action. Each edit contains an integer identifier `p` and a typed value `v`; an empty edit list represents a hold. Appendix F shows the recorded prompt, and Appendix G lists the parameter domains.

In one recorded 100-second uncapacitated facility location episode (Appendix B), the second separation action lowers the root and per-node separation-round limits from two to one. Separation time falls from 10.39 to 7.50 seconds, but total solve time rises from 72.88 to 74.00 seconds. A later branching edit takes 77.39 seconds, and the final hold re-executes the earlier configuration in 72.95 seconds. All four runs reach optimality. The example separates savings in one subsystem from improvement of the complete solve and shows how a hold preserves an earlier configuration.

3.3 Learning from solver outcomes

Each rollout is a sampled interaction on one instance at one solver limit, with at most four turns. We train the policy using a terminal score based on the returned configuration. To compare different solver outcomes, we use a Default-normalized, status-aware loss L. Outcome levels $h = 0, 1, 2, 3$ represent runs with no finite bound, only a finite dual bound, a finite primal bound, and a terminal proof, respectively; h_0 is the Default level. The loss is $L = 2^{h_0 - h} q$, where $q \in [0.8, 1.25]$ measures within-level quality using solve time, primal-dual integral, or dual-bound progress. The levels prioritize bound availability and proof status over within-level differences; the Default run has $L = 1$. With L_t denoting the loss recorded at turn t, we define $\Delta^{\text{final}} = 1 - L_4$ and $\Delta^{\text{reg}} = \max\{0, L_4 - \min(1, L_1, \ldots, L_4)\}$. These measure final improvement over Default and deterioration from the best observed loss, including Default.

Let $p_{\text{fmt}} \geq 0$ be the final-format penalty and $\mathbb{I}_{\text{invalid}}$ the invalid-final-action indicator, both with unit weight. Let $\beta \geq 0$ weight deterioration. The terminal score is

$$R = \text{clip}_{[-5,5]}\left(8\Delta^{\text{final}} - \beta\Delta^{\text{reg}} - p_{\text{fmt}} - \mathbb{I}_{\text{invalid}}\right), \tag{2}$$

Full RL uses $\beta = 4$; the ablation without the deterioration penalty sets $\beta = 0$. Appendix B.2 gives the exact comparison-regime mapping, fallback cases, rules for q, and recorded validator weights.

3.4 Training and checkpoint selection

We train the Qwen2.5-7B-Instruct policy with group-relative policy optimization (GRPO) (Shao et al., 2024). For each training prompt, four rollouts sample up to four feedback-conditioned turns. GRPO compares their terminal scores within the group and uses the relative outcomes to update the policy. The learning signal thus evaluates a sequence of edits and holds through the configuration it ultimately returns.

Continuation training begins from a solver-tuned checkpoint shared with the ablation without the deterioration penalty. Each of the 262 training instances is represented at per-call limits of 100, 200, 300, and 400 seconds, and the policy observes the available limit. Appendix B gives the training lineage and checkpoint-selection rule, and Appendix E records implementation settings.

4 Experimental Design

4.1 Data and split

The 315-instance pool combines a 215-instance bank curated for a separate concurrent submission on log-guided, prompt-based SCIP tuning with 100 CVRPLIB-derived XML100 instances. The bank comprises 90 capacitated vehicle routing problem (CVRP) instances from CVRPLIB, 12 generalized assignment problem (GAP) and 24 job-shop scheduling problem (JSP) instances from OR-Library, 30 uncapacitated facility location (UFL) instances from UflLib, and 59 MIPLIB 2017 instances (Lima et al., 2014; Beasley, 1990; Hoefer, 2018; Gleixner et al., 2021). Within this bank, the four domain-specific families were exported as `.mps` files from mathematical formulations, and MIPLIB files were used directly. Appendix E.1 documents the inherited source subsets and models. Table 1 shows the 262/53 training–validation split. Validation instances are excluded from policy-gradient training and used for both checkpoint selection and the reported comparisons.

Source group	Train	Validation	Total
CVRP (CVRPLIB)	72	18	90
XML100 (CVRPLIB)	81	19	100
GAP (OR-Library)	10	2	12
JSP (OR-Library)	22	2	24
UFL (UflLib)	23	7	30
MIPLIB 2017	54	5	59
Total	262	53	315

Table 1: Training and validation split of the 315-instance MIP dataset.

4.2 Comparators and solver limits

At each per-call limit $b \in \{100, 200, 300, 400\}$ seconds, Default SCIP supplies one reference execution and the initial diagnostics. Random Search, SMAC3, prompt-only Base Qwen, Full RL, and the six prompt-only language-model agents in Table 3 each use four tuning turns under the same solver-call and action constraints. Base Qwen uses the Full RL prompt without reinforcement learning. The language-model agents receive the same diagnostics, action schema, and interaction history. Every returned configuration is executed at the corresponding SCIP time limit. Appendix E records the available implementation settings and exact frontier-model identifiers.

Comparison with Default SCIP measures whether the returned settings improve on an untuned solver execution. Random Search and SMAC3 then test the learned policy against online configuration search within the small call budget. Base Qwen examines the value of solver-specific training for the same backbone, while the six additional prompt-only agents ask whether that trained policy also compares favorably with alternative language models.

4.3 Paired outcomes

For each instance and limit, we compare the final SCIP outcome from Full RL with the baseline outcome. The native ordering first considers solver status and available primal or dual bounds. Within comparable outcomes, an instance's preassigned evaluation regime selects solve time, primal–dual progress, or dual-bound quality as the criterion. A win favors Full RL, a loss favors the baseline, and a tie leaves them equal under this ordering. Let W, T, and L be the numbers of wins, ties, and losses. We report them as $W/T/L$ and their net margin $W - L$. Appendix C gives the full ordering and pairing rules.

This ordering distinguishes useful progress even when both executions reach the time limit. Finding a feasible solution can change the outcome rank, while lower PDI or a stronger dual bound can distinguish runs within the same status tier under the assigned regime. Such improvements are invisible to the number of solved instances. The paired counts consequently summarize per-instance preferences without placing solve times, integrals, and objective bounds on a common numerical scale.

We also report the number of solved instances and the mean SCIP time for the final execution of each returned configuration across the 53 validation instances.

5 Main Results

Full RL achieves positive paired win–loss margins in all 16 comparisons with Default SCIP, Random Search, SMAC3, and prompt-only Base Qwen (Table 2). Against Base Qwen at 300 seconds, for example, it records 34 wins, 9 ties, and 10 losses. Base Qwen already has access to execution feedback and can revise settings, so its comparison with Full RL evaluates learned and prompted use of the same interaction. The advantages over Random Search and SMAC3 extend this finding to methods that also devote their available calls to configuration search. Together, these comparisons support the solver-trained policy as a way to use a short tuning interaction to return configurations that more often yield preferred SCIP outcomes across the evaluated solver limits.

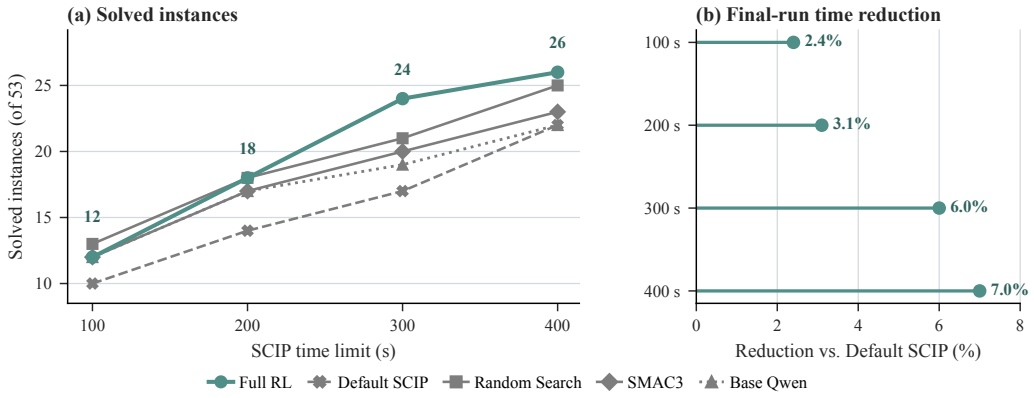


Figure 2: Solver outcomes across four time limits. (a) Number of instances solved by each method among 53. (b) Reduction in mean final-run SCIP time for Full RL relative to Default SCIP; mean times are reported in Table 10.

The paired advantage is accompanied by more completed solves at the larger limits (Figure 2(a)). Full RL solves 24 instances at 300 seconds, compared with 21 for Random Search, the key baseline with the most solved instances. At 400 seconds, the numbers are 26 and 25.

Full RL already has positive paired margins over Random Search at 100 and 200 seconds. Random Search solves one more instance at 100 seconds, and the two methods solve the same number at 200 seconds. Mean final-run SCIP time is also lower for Full RL than for Default SCIP at all four limits (Figure 2(b)). Appendix C reports the number of solved instances and runtime for every method and limit.

Limit (s)	Default SCIP	Random	SMAC3	Base Qwen	Minimum $W - L$
100	46/5/2	35/2/16	36/0/17	30/10/13	+17
200	46/5/2	32/3/18	35/3/15	31/9/13	+14
300	48/5/0	29/3/21	34/3/16	34/9/10	+8
400	47/4/2	32/4/17	35/3/15	25/11/17	+8

Table 2: Paired outcomes on 53 validation instances. Entries report Full RL wins/ties/losses against each baseline; $W - L$ is the win–loss margin.

Full RL's gains over Default SCIP span 46–48 of the 53 instances at each limit, with at most two losses. Random Search and SMAC3 yield closer comparisons, yet Full RL still wins on more than half of the instances against each at every limit. The near-sweep over Default therefore extends into a majority preference when the alternatives also search for instance-specific settings. For a solver user, this supports the trained interaction as a way to choose configurations within the available trials, with an advantage over online search that remains present at the longest per-call limit.

Table 3 compares Full RL with six prompt-only language-model agents outside the Qwen backbone.

Baseline	100 s	200 s	300 s	400 s
GPT-5.6-Sol	34/2/17	36/2/15	37/3/13	34/3/16
GPT-5.6-Terra	42/2/9	40/2/11	36/3/14	33/4/16
GPT-5.6-Luna	38/2/13	35/2/16	37/2/14	32/2/19
Claude Opus 5	33/2/18	33/2/18	28/2/23	23/5/25
Claude Opus 4.8	34/2/17	33/3/17	34/3/16	36/3/14
Claude Sonnet 4.6	32/1/20	27/2/24	37/3/13	25/5/23

Table 3: Full RL wins/ties/losses against six prompt-only language-model agents on the same 53 validation instances.

Across six alternative prompt-only backbones and four solver limits, Full RL has positive paired margins in 23 of 24 comparisons. The exception is Claude Opus 5 at 400 seconds, where Full RL records 23 wins, 5 ties, and 25 losses. With the same diagnostics and action interface available to all agents, these comparisons extend the evidence beyond the prompt-only Qwen baseline. They support training the tuning interaction on SCIP outcomes as an alternative to selecting another prompted backbone. The practical task involves both choosing what to test next and keeping useful evaluated settings available for the final solve, motivating the joint analysis of feedback and configuration retention.

6 Ablations and Analysis

We examine the deployed feedback-and-retention loop and the training objective at 100 and 400 seconds. Figure 3 shows paired outcomes alongside regressions from the best configuration observed earlier in the episode. A regression means that the final execution scores below that earlier best, including Default SCIP, under the internal solver signal.

6.1 No candidate feedback or incumbent reuse

Using the same trained checkpoint, the joint ablation removes candidate feedback and configuration-incumbent reuse. After the initial turn, the policy receives Default SCIP diagnostics in place of candidate outcomes, and each edit is applied independently to Default settings. An empty edit list therefore holds Default rather than an earlier candidate. Appendix D gives the full interaction semantics.

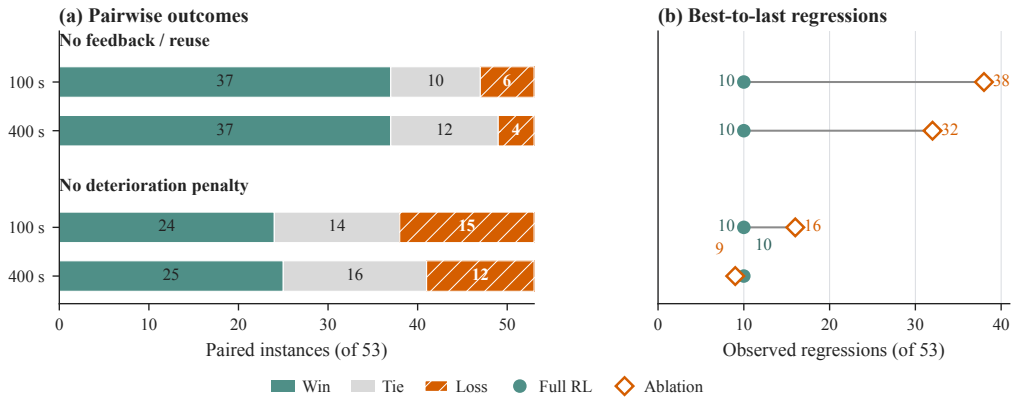


Figure 3: Paired outcomes and best-to-last regressions in two ablations. (a) Full RL wins, ties, and losses against each ablated policy on 53 instances. (b) Observed regressions for Full RL (circles) and the ablation (diamonds).

Limit (s)	Full RL $W/T/L$	$W - L$	Episode-best-to-last regressions	
			Full RL	No feedback/reuse
100	37/10/6	+31	10/53	38/53
400	37/12/4	+33	10/53	32/53

Table 4: Full RL versus the joint removal of candidate feedback and configuration-incumbent reuse. $W/T/L$ gives paired wins/ties/losses; regressions compare the final outcome with the episode best.

Full RL wins on 37 instances at each limit, with six losses at 100 seconds and four at 400 seconds (Table 4). Of those wins, 34 and 33, respectively, occur within the same solver-status tier (Appendix C). The advantage therefore appears mainly in the reported progress measures among outcomes with comparable status.

At both limits, the ablated policy ends with a hold on all 53 instances, so its final outcomes come from new executions of Default settings. Full RL ends with a hold on 47 instances at each limit, preserving its best observed configuration. The regression records show that the ablated interaction often tests a non-Default candidate with a better outcome under the internal solver signal than its final Default execution (Appendix C). Its history retains the proposed actions but omits their outcomes, and earlier non-Default candidates cannot become the base for later edits or holds. The joint comparison therefore concerns the ability to carry evaluated configurations forward, including settings associated with an earlier observed improvement.

6.2 No deterioration penalty

The continuation without the deterioration penalty starts from the same solver-tuned checkpoint as Full RL and changes β in Equation 2 from 4 to 0. Both pipelines use the same training schedule and checkpoint-selection rule. Selection chooses the shared warm start for the ablation and a continuation checkpoint for Full RL.

Limit (s)	Full RL $W/T/L$	$W - L$	Episode-best-to-last regressions Full RL	$\beta = 0$
100	24/14/15	$+9$	10/53	16/53
400	25/16/12	$+13$	10/53	9/53

Table 5: Full RL versus the selected continuation pipeline without the deterioration penalty. $W/T/L$ gives paired wins/ties/losses; regressions use the definition in Table 4.

Full RL has paired win–loss margins of $+9$ at 100 seconds and $+13$ at 400 seconds (Table 5). It has fewer observed best-to-last regressions at 100 seconds (10 versus 16), but one more at 400 seconds (10 versus 9). Paired outcomes compare the two policies under the native ordering; a regression compares each final execution with its own episode best under the internal solver signal. At each limit, seven of Full RL's ten regressions occur when the earlier best configuration is returned unchanged and executed again (Appendix C). The regression count therefore includes worse re-execution outcomes even when the interaction has retained the earlier settings.

The joint comparison links better solver progress within comparable status tiers with an interaction that keeps evaluated configurations available for reuse. Earlier trials can therefore contribute to the final choice even after unsuccessful later proposals. The penalty analysis further distinguishes retaining an evaluated configuration from reproducing its earlier measured outcome when SCIP executes it again.

7 Related Work

Classical algorithm configuration searches expensive mixed-type parameter spaces (Hutter et al., 2011; Lindauer et al., 2022), while dynamic configuration learns policies that adjust controls within a solver run (Adriaensen et al., 2022). Learned MIP components address individual decisions such as branching (Gasse et al., 2019) and presolve (Liu et al., 2024a); similarity-based methods retrieve configurations using relationships between instances (Hosny & Reda, 2023). Our policy adapts between SCIP executions on the same instance. It uses the completed run's diagnostics to edit or retain parameter settings across branching, node selection, presolve, separation, and primal heuristics.

Language models have been used for optimization modeling (Ahmaditeshnizi et al., 2024), black-box proposal generation (Yang et al., 2024), and Bayesian optimization (Liu et al., 2024b). Solver-focused studies consider cutting-plane separator configuration (Lawless et al., 2024), language-model-guided MILP re-optimization (Liu et al., 2025), and branching-rule scheduling (Zheng et al., 2025). Tool-learning methods train multi-step tool-use policies from outcomes (Yu et al., 2025; Feng et al., 2025; Dong et al., 2025). The separate concurrent submission develops log-guided, prompt-based SCIP tuning with diagnostic summaries and sparse edits to the best observed configuration. Within this shared diagnostic and incumbent-edit framework, we train a configuration policy through online reinforcement learning with SCIP rollouts, using terminal outcomes to evaluate sequences of edits and holds.

8 Conclusion

The feedback-conditioned Qwen policy developed here uses SCIP diagnostics over four tuning turns to choose sparse parameter edits or hold the best observed configuration maintained by the environment. Its documented 45-parameter interface allows changes across solver subsystems, while configuration memory keeps useful settings available as further candidates are tested. Online reinforcement learning trains this interaction against the SCIP outcome of the returned configuration.

On 53 validation instances, Full RL achieves positive paired margins in all 16 comparisons with four key baselines and in 23 of 24 comparisons with six prompt-only agents across four per-call limits from 100 to 400 seconds. At 300 seconds, its returned configurations solve 24 instances versus at most 21 for the key baselines. Against the joint removal of candidate feedback and incumbent reuse, Full RL wins on 37 of 53 instances at both 100 and 400 seconds; the ablated policy's final holds return Default settings. The ablation shows that a hold can preserve an earlier useful candidate only when the interaction keeps that candidate available as its base.

Online feedback incurs the cost of repeated SCIP executions, while the current interaction has no learned stopping decision. A budget-adaptive extension could add a stop/continue action conditioned on the latest diagnostics, execution history, and best observed configuration. Stopping would return that configuration; continuing would permit another sparse edit or hold. Training could balance prospective solver improvement against the cost of another execution, with evaluation measuring configuration quality together with total tuning time.

The current interface also restricts adaptation to selected SCIP parameters between executions. A possible extension is to represent diagnostics and actions by subsystem role, with solver-specific mappings to measurement fields, parameter names, and admissible values. This design would preserve the feedback/edit/hold structure while allowing its observations and edits to use another solver's interface. Future work would need to develop these mappings, validate their semantic compatibility, and evaluate the adapted policies on additional solvers and instance mixtures.

Artificial intelligence (AI) use statement

Generative AI tools assisted with research methodology feedback, software and figure development, literature organization, and drafting and editing of this manuscript. The reported solver outcomes are drawn from recorded executions rather than generated by these tools. The authors take responsibility for the final methods, evidence, claims, and text, including AI-assisted material.

References

Steven Adriaensen, André Biedenkapp, Gresa Shala, Noor Awad, Theresa Eimer, Marius Lindauer, and Frank Hutter. Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research*, 75:1633–1699, 2022. doi: 10.1613/jair.1.14040.

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596, 2024. URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html.

R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52 (5):723–738, 2004. doi: 10.1287/opre.1040.0111.

J. E. Beasley. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. doi: 10.1057/jors.1990.166.

Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, et al. The SCIP optimization suite 9.0. Technical report, Optimization Online, 2024. URL https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-Star: Empowering LLM-brained multi-tool reasoner via reinforcement learning, 2025.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs, 2025.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021. doi: 10.1007/s12532-020-00194-3.

Martin Hoefer. UflLib: Library of (uncapacitated) facility location instances. Max-Planck-Institut für Informatik benchmark repository, 2018. URL https://resources.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/.

Abdelrahman Hosny and Sherief Reda. Automatic MILP solver configuration by learning problem similarities, 2023.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pp. 507–523. Springer, 2011. doi: 10.1007/978-3-642-25566-3_40.

Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. LLMs for cold-start cutting plane separator configuration, 2024.

Ivan Lima, Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. CVRPLib: Capacitated vehicle routing problem library. Galgos / DI PUC-Rio benchmark repository, 2014. URL https://galgos.inf.puc-rio.br/cvrplib/.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhopf, René Sass, and Frank Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL https://jmlr.org/papers/v23/21-0888.html.

Chang Liu, Zhichen Dong, Haobo Ma, Weilin Luo, Xijun Li, Bowen Pang, Jia Zeng, and Junchi Yan. L2P-MIP: Learning to presolve for mixed integer programming. In *International Conference on Learning Representations*, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/1e6e0c2edb159b2ad2f9419b898f56d3-Abstract-Conference.html.

Ruohong Liu, Xuan He, Yuxin Pan, and Jack Umenberger. Online algorithm configuration for MILP re-optimization with LLM guidance. OpenReview preprint, 2025. URL https://openreview.net/forum?id=xbyebbS1ZF.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *International Conference on Learning Representations*, 2024b. URL https://arxiv.org/abs/2402.03921.

Alan S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960. doi: 10.1287/opre.8.2.219.

G. Terry Ross and Richard M. Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8(1):91–103, 1975. doi: 10.1007/BF01580430.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.03409.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. StepTool: Enhancing multi-step tool usage in LLMs via step-grained reinforcement learning. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, 2025. doi: 10.1145/3746252.3761391.

Zetao Zheng, Hanjie Gu, and Jie Shao. Large language model guided dynamic branching rule scheduling in branch-and-bound. OpenReview preprint, 2025. URL https://openreview.net/forum?id=8LCdjf7uIk.

A Policy Interface and Execution Semantics

A.1 Compact solver observations

The policy interface separates the parameter vocabulary from the changing execution state. The system prompt specifies parameter identifiers, types, domains, action families, and the JavaScript Object Notation (JSON) response format. Subsequent observations provide the current step, the observed incumbent, and a bounded execution history. Table 6 documents the compact diagnostic names used in the implementation; these are measurements of solver executions, not labels identifying the best action.

Field	Meaning
`T`, `st`	Elapsed solve time and terminal solver status.
`n`, `left`, `d`	Processed nodes, remaining nodes, and maximum search depth.
`g`	Rendered final optimality gap.
`sb`, `sep`	Strong-branching and separation time.
`pre`, `heur`	Presolve and heuristic time.
`pdi`	Primal-dual integral.
`prim`, `dual`	Primal and dual bounds.
`rdg`	Recorded root dual-bound gain.

Table 6: Compact diagnostic fields exposed by the solver interface.

The serializer rounds time fields to two decimal places, renders bounds and the primal-dual integral (PDI) with six significant digits, and rounds root dual gain to two decimal places. Displayed gap values below 100 are formatted to one decimal place with a percent suffix; otherwise, the display is `INF` (infinity). This compact rendering serves as policy input and does not replace the native evaluation comparator.

A.2 Sparse parameter edits

The vocabulary contains 45 indexed parameters, organized as in Table 7. An action specifies a bottleneck family and an edit list. Each edit contains an integer parameter identifier `p` and a typed value `v`, both generated by the model as tokens. Domains include categorical choices, Boolean values, integer counts or frequencies, and real-valued ratios or thresholds. Domain membership and value type are distinct constraints. A parameter identifier is not a textual solver-family name, and a frequency is not a separator name.

Family	Identifiers	Parameter roles
Branch/search	1–10	Branching scores, reliability and strong-branching settings, and node selection.
Separation	11–20	Separation rounds, cut limits, and separator frequencies.
Presolve	21–23	Presolve rounds, restarts, and abort factor.
Heuristics	24–45	Feasibility pump, relaxation-induced neighborhood search (RINS), local branching, and relaxation-enforced neighborhood search (RENS) controls.
Hold	Empty list	Retain the currently observed incumbent.

Table 7: Action vocabulary grouped by solver subsystem. The groups describe the parameter interface, not a measured attribution of performance gains.

The prompt permits at most 20 edits and encourages one to three targeted changes; this preference does not imply that every action has that size. Sparse edits are interpreted relative to the configuration supplied by the interaction environment. A hold does not terminate the four-turn evaluation. Evaluation returns the last configuration, including when the last action retains an incumbent. The evaluator does not retrospectively replace an inferior final configuration with an earlier one. Training may reuse a cached hold outcome, whereas the reported endpoint evaluations re-execute held configurations. Consequently, identical settings need not yield identical measured outcomes across runs.

B TRAINING AND CHECKPOINT SELECTION

B.1 TRAINING STRUCTURE

The policy starts from Qwen2.5-7B-Instruct and uses a solver-tuned warm start for the reported continuation stage. The warm start was selected following a ten-update pilot at a 100-second solver limit. The continuation stage has 60 updates, four sampled rollouts per training prompt, and up to four interaction turns per rollout. Each assistant response has a 192-token cap. A response terminated at the length limit can end a training rollout early, so a four-turn maximum does not imply that every training trajectory is complete. The evaluation protocol separately requires four recorded turns.

The continuation stage uses AdamW with a learning rate of 2×10^{-7}, one proximal policy optimization (PPO) epoch per update, a global training batch of 16 prompts, and a microbatch of one per graphics processing unit (GPU). Four rollouts are sampled per prompt at temperature 1.0. GRPO advantages are normalized by their within-group standard deviation. All actor parameters are trainable under Fully Sharded Data Parallel (FSDP), with no parameter-efficient adapter configured. Gradient checkpointing is enabled, and the gradient norm is clipped at 1.3. The saved evaluation records retain the complete four-turn action/feedback sequence for every endpoint episode; the prompt and action schema are transcribed in Appendix F.

The continuation without the deterioration penalty has 38 missing records among 15,360 nominal interaction records across 11 updates; none were backfilled.

For concreteness, one saved 100-second episode on `ufl_2523UnifS.mps.gz` is reproduced below. Each diagnostic reports the fresh result after executing the action in that row; parameter identifiers resolve through the catalog in Appendix G. The final hold re-executes the incumbent rather than terminating early or reusing a cached result.

Turn	Recorded action	Returned diagnostic
1	`sep: (11,2),(12,2)`	`T=72.88 st=optimal n=686 left=0 d=16 g=0.0% sep=10.39 pdi=744.4`
2	`sep: (11,1),(12,1)`	`T=74.00 st=optimal n=825 left=0 d=19 g=0.0% sep=7.50 pdi=750.18`
3	`branch: (1,s),(4,4)`	`T=77.39 st=optimal n=691 left=0 d=15 g=0.0% sep=13.45 pdi=771.82`
4	`hold`	`T=72.95 st=optimal n=686 left=0 d=16 g=0.0% sep=10.37 pdi=746.77`

Training includes all 262 training instances at each of the four limits 100, 200, 300, and 400 seconds. Equal representation in the expanded training set is not a claim of exact four-way balance in every sampled minibatch. The terminal score in Equation 2 is the optimized rollout objective. Auxiliary per-turn quantities retained in the logs are not additional terms in that objective. The ablation without the deterioration penalty changes its coefficient from four to zero during continuation training; both pipelines retain the shared warm-start training history.

B.2 INTERNAL SOLVER LOSS AND TERMINAL VALIDATOR PENALTY

The terminal score uses a Default-normalized, status-aware solver loss and penalties on the final response. We assign outcome levels $h = 0, 1, 2, 3$ to runs with no finite bound, only a finite dual bound, a finite primal bound, and a terminal proof, respectively. Let h_0 denote the outcome level of the Default run. The levels distinguish bound availability and proof status, while a bounded quality factor captures finer differences in solver performance.

Let d denote the dual bound and g the final gap, with d_0 and g_0 denoting the corresponding Default values. Within the dataset's fixed comparison regime, the raw loss ℓ is solve time divided by Default solve time for a Default-solved instance, PDI divided by Default PDI for an instance with a finite Default primal bound, and $1 + s(d - d_0)/(|d_0| + 10^{-6})$ otherwise, where $s = -1$ for minimization and $s = 1$ for maximization. Ratios are nonnegative. A nonterminal candidate in the solve-time regime has infinite raw loss. If Default PDI is below 10^{-6}, the PDI regime instead uses $(g + 10^{-6})/(g_0 + 10^{-6})$ when both final gaps are finite, and uses 1 otherwise. The dual-bound regime uses 1 if either d or d_0 is nonfinite.

The within-level quality factor q and the normalized loss L are

$$q = \begin{cases} \mathrm{clip}_{[0.8,1.25]}(\ell), & h = h_0, \\ 0.8 + 0.45u, & h > h_0, \\ 1, & h < h_0, \end{cases} \qquad L = 2^{h_0 - h} q. \tag{3}$$

Here, $\mathrm{clip}_{[a,b]}$ denotes clipping to the interval $[a, b]$. For $h > h_0$, u is obtained by clipping a level-specific quantity to $[0, 1]$. That quantity is solve time divided by the time limit at level 3, PDI divided by 100 times the time limit at level 2, and 0.5 at level 1. Thus, $q \in [0.8, 1.25]$, the Default run has $L = 1$, and outcome-level changes have priority over within-level quality.

Let L_t denote the loss at turn t. Final improvement over Default and deterioration from the best observed loss, including Default, are measured by

$$\Delta^{\mathrm{final}} = 1 - L_4, \qquad \Delta^{\mathrm{reg}} = \max\{0, L_4 - \min(1, L_1, \ldots, L_4)\}. \tag{4}$$

With a nonnegative final-format penalty p_{fmt}, an invalid-final-action indicator $\mathbb{I}_{\mathrm{invalid}}$, and a deterioration weight $\beta \geq 0$, the terminal score is

$$R = \mathrm{clip}_{[-5,5]}\left(8\Delta^{\mathrm{final}} - \beta\Delta^{\mathrm{reg}} - p_{\mathrm{fmt}} - \mathbb{I}_{\mathrm{invalid}}\right). \tag{5}$$

Full RL uses $\beta = 4$; the ablation without the deterioration penalty uses $\beta = 0$.

The final-format penalty p_{fmt} is the sum of the final response's recorded validator contributions listed in Table 8. This sum and the invalid-final-action indicator enter the terminal reward with unit weight. An unparseable final action also sets $\mathbb{I}_{\mathrm{invalid}} = 1$.

Recorded validator condition	Penalty contribution
Unparseable response	0.05
Noncompact JSON	0.10
Text after the JSON object	0.60
Leaked conversation text	0.80
Markdown fence	0.30
Repaired JSON	0.02
Missing or invalid action-family label	0.04 or 0.08
Diagnostics–family mismatch	0.35
Family-inconsistent edits	0.20 each, up to 0.60
Disallowed rationale key	0.25
Empty non-hold action	0.08
Disallowed hold or ineffective edit	0.02
Invalid edits	0.05 each, up to 0.20
Edits beyond 20	0.005 each, up to 0.05
Edits beyond eight	0.005 each, up to 0.05

Table 8: Nonzero contributions to the final response's recorded validator penalty p_{fmt}.

B.3 Frozen mixed-limit selection rule

Checkpoints are evaluated using a fixed assignment of the 53-instance validation set to four solver-limit groups of sizes 14, 13, 13, and 13. Each instance appears once in this mixed selection set at its assigned limit. This differs from the final grid, where every one of the 53 instances is evaluated at every limit. The assignment remains fixed across checkpoints.

Let $\mathcal{B} = \{100, 200, 300, 400\}$ and let $\mathcal{J}$ contain Default SCIP, Random Search, SMAC3, and Base Qwen. For checkpoint k, define W_{kbj} and L_{kbj} as the numbers of wins and losses on the assigned subset for solver limit b and baseline j, and let $m_{kbj} = W_{kbj} - L_{kbj}$. Let $q_{kb} = \min_{j \in \mathcal{J}} m_{kbj}$ denote the worst baseline margin within that subset. The selector maximizes the following tuple in lexicographic order,

$$\left(\sum_{b \in \mathcal{B}} \mathbb{I}\{q_{kb} > 0\}, \quad q_{k,(2)}, \quad \min_{j \in \mathcal{J}} M_{kj}, \quad \sum_{j \in \mathcal{J}} M_{kj}, \quad -\sum_{j \in \mathcal{J}} L_{kj}, \quad -k \right), \tag{6}$$

where $q_{k,(2)}$ is the second-largest of the four subset margins, $M_{kj} = \sum_b m_{kbj}$, and $L_{kj} = \sum_b L_{kbj}$. Earlier coordinates take priority, and the final coordinate favors the earlier checkpoint in an otherwise exact tie. Selection therefore uses paired native outcomes and a specified tie-breaking order, rather than visual inspection of a training curve or a frontier-model comparison.

The rule selects update 60 for Full RL and update 0 for the continuation pipeline without the deterioration penalty. Here, update 0 is the shared solver-tuned warm start, not the prompt-only Base Qwen baseline. The resulting ablation compares training-and-selection pipelines; it is not a comparison at equal selected update counts. The subset margins used for selection are not the final 53-instance per-limit results.

B.4 Evaluation sequence and scope

1. Fix the mixed-limit instance assignment and the checkpoint ranking rule.
2. Apply the same rule to the candidate checkpoints from each training pipeline.
3. Freeze the selected checkpoint before running its complete evaluation grid.
4. At each fixed limit, evaluate all 53 instances and retain the native outcome of each final configuration.
5. Pair outcomes by instance and limit, then report wins, ties, and losses.

The mixed selection set and final evaluation grid share the same instance identities. The final-run SCIP time measure excludes language-model inference and orchestration.

Both ablations have 53 unique four-turn validation identities at each endpoint. Three padded duplicate episodes from distributed evaluation are excluded from the reported counts.

C Paired Evaluation and Result Interpretation

C.1 Native outcome ordering

The comparator first assigns each outcome to a tier. Terminal statuses in {`optimal`, `infeasible`, `unbounded`, `inforunbd`, `solved`} take precedence, followed by outcomes with a finite primal bound, outcomes with only a finite dual bound, and outcomes with neither finite bound. Higher tiers take precedence over all subsequent numerical comparisons. Within the same tier, the stored regime of the instance determines the comparison in Table 9. Regimes remain fixed when default runs are refreshed.

The regimes are static dataset tags assigned from the original Default run. The tag is `case1` when it terminated optimally, `case2` otherwise when both primal and dual bounds were finite, and `case3` when only a finite dual bound was available or neither preceding condition held. The completed endpoint artifacts contain 2, 28, and 23 instances in these respective regimes.

Regime	Preferred outcome
`case1`	Lower solve time.
`case2`	Lower PDI; on a tie, lower parsed gap.
`case3`	Higher dual bound for minimization; lower dual bound for maximization.

Table 9: Within-tier native comparison rules. These rules are applied to final outcomes independently of the signal used to choose the episode incumbent.

For two finite numeric values u and v, the comparator treats the values as tied when $|u - v| \leq 10^{-9} \max(1, |u|, |v|)$. A finite value takes precedence over a missing or nonfinite value, while two missing or nonfinite values tie.

C.2 Pairing and completeness checks

A complete method–limit evaluation contains 53 unique instance paths and uses the specified per-call time limit. Four-turn agent evaluations must additionally contain turns one through four and

identify turn four as the final outcome. Pairing is based on instance identity and solver limit, not row position or a coincidental equality of final scores. When legacy validation records require trajectory matching, the matching procedure checks the complete episode's actions and recorded feedback and rejects ambiguous matches. Padding episodes introduced by distributed evaluation are not additional evaluation instances.

For each comparison, exactly one outcome category is assigned to each paired instance, so $W + T + L = 53$. The net margin $W - L$ lies in $[-53, 53]$. Positive values indicate that more instances favor Full RL than the comparator. For example, the 300-second comparison against Random Search is $29/3/21$, giving a margin of $+8$; the 400-second comparison against Claude Opus 5 is $23/5/25$, giving -2.

For Full RL versus no feedback/incumbent reuse, status-tier changes contribute $3/0/1$ and $4/0/1$ wins/ties/losses at 100 and 400 seconds, while within-tier metrics contribute $34/10/5$ and $33/12/3$. For Full RL versus no deterioration penalty, the corresponding status-tier counts are $2/0/1$ and $3/0/2$, and the within-tier counts are $22/14/14$ and $22/16/10$. Terminal outcomes are either `optimal` or `timelimit`. Full RL records 12/41 and 26/27 at the two limits; no feedback records 10/43 and 22/31; and no penalty records 12/41 and 24/29. These decompositions sum to the reported 53 paired instances.

C.3 Different questions answered by the reported statistics

Table 10 reports two direct solver metrics in addition to paired outcomes. The number of solved instances is an absolute terminal-status measure. Mean runtime is the duration of the final SCIP execution for the returned configuration, averaged over all 53 instances; it is not the total four-turn agent latency, and a smaller mean can partly reflect more early terminations. We do not aggregate gaps because the frozen parser does not provide comparable finite gap values for every method, and we do not average PDI across regimes in which PDI is not the selected native metric.

Limit	Full RL	Default SCIP	Random	SMAC3	Base Qwen
100	12 / 88.8	10 / 91.0	13 / 87.8	12 / 89.6	12 / 88.8
200	18 / 162.6	14 / 167.8	18 / 157.4	17 / 162.7	17 / 162.0
300	24 / 224.8	17 / 239.0	21 / 224.1	20 / 230.0	19 / 226.0
400	26 / 281.5	22 / 302.8	25 / 284.2	23 / 294.0	22 / 283.4

Table 10: Secondary native metrics on the fixed 53-instance set. Each cell reports solved instances / mean final-run SCIP time in seconds. Runtime excludes model inference and the preceding solver calls, so it does not represent an end-to-end speedup.

Relative to Default SCIP, Full RL solves 2, 4, 7, and 4 additional instances at the four limits. Its mean final-run time is lower by 2.4%, 3.1%, 6.0%, and 7.0%, respectively. The differences in the number of solved instances relative to the strongest non-RL method at each limit are -1, 0, $+3$, and $+1$. At the larger limits, the paired advantage is accompanied by additional completed solves.

Paired final-outcome counts compare Full RL with a baseline on the same instance. Episode-best-to-last regression instead compares the final outcome with the observed incumbent from that episode under the internal solver signal. The two measures use different reference points and different orderings. A method can win more pairs against a baseline without having fewer observed regressions. The comparison without the deterioration penalty at 400 seconds illustrates this distinction. Full RL has 25 wins, 16 ties, and 12 losses, yet has 10 regressions compared with nine for the ablation.

Held settings are executed again during evaluation. At both endpoints, seven of Full RL's ten observed regressions return the same configuration as an earlier episode-best run. The corresponding counts for the no-feedback/reuse ablation are three of 38 at 100 seconds and one of 32 at 400 seconds. For the ablation without the deterioration penalty, the episode-best configuration is returned unchanged in nine of 16 regressions at 100 seconds and seven of nine at 400 seconds. Final hold counts are 47/53 at both limits for Full RL and 44/53 and 50/53 for the ablation without the deterioration penalty, respectively.

D Configuration Composition and Interaction State

D.1 Three configurations with different roles

The default configuration, the observed incumbent, and the current candidate have distinct roles. The default provides a common starting configuration and reference execution. The incumbent stores a configuration associated with the best internal signal observed so far. A candidate is the configuration obtained by applying a new sparse action to the current base of the interaction. Conflating these objects would make it unclear whether the policy is modifying its best observed configuration, repeating a previous unsuccessful proposal, or starting again from default.

Let θ_t^{base} denote the base supplied at turn t, and let E_t be a set of validated identifier–value pairs. The coordinate-wise rule below defines sparse composition. Here i indexes a parameter and $\text{ids}(E_t)$ is the set of identifiers in E_t.

$$\theta_{t,i}^{\text{cand}} = \begin{cases} v, & (i, v) \in E_t, \\ \theta_{t,i}^{\text{base}}, & i \notin \text{ids}(E_t). \end{cases} \tag{7}$$

The notation describes configuration composition and does not assume independent parameter effects. Applying two individually plausible changes together can alter solver behavior differently from either change alone. The feedback loop observes the resulting execution and does not estimate each coordinate's contribution from its name.

D.2 Illustrative edit and hold semantics

Consider a symbolic base with two editable coordinates and a proposal that changes only the first. The second coordinate retains its base value and is not reset simply because it is absent from the JSON. Applying the same sparse proposal to a different base can therefore produce a different complete configuration. An action string alone is therefore insufficient to identify the solver configuration that was executed.

An empty edit list preserves the base selected by the environment. In the full-policy interaction, this represents retention of the observed incumbent. In the no-candidate-feedback/no-incumbent-reuse intervention, proposed edits are applied independently to default; an empty list therefore returns default. The policy receives the original Default diagnostics on turn one. On turns two through four, its previous actions remain in history, but it again receives the Default observation without candidate outcomes or a selected configuration incumbent. The same empty JSON edit list has the same coordinate-update meaning in both cases, while the complete configuration depends on the base.

D.3 Memory and duplicate avoidance

The user-message builder renders the incumbent as a sparse difference from default and records previous configurations relative to the incumbent when forming duplicate-avoidance messages. It traverses visible memory in reverse step order, removes repeated delta strings, and displays at most eight distinct deltas. The resulting instruction warns the policy against repeating visible candidates but does not guarantee that every repeated configuration is prevented. The latest-feedback line distinguishes the most recent visible record from the incumbent, allowing the prompt to indicate that the latest candidate did not become best even if an earlier candidate was useful.

D.4 An action is not an execution record

The raw response, parsed edits, executed configuration delta, and observed diagnostics are stored separately. This separation allows us to check whether a generated action was parseable and what configuration it actually induced. It also avoids interpreting a change in JSON formatting as a change in the solver configuration. Parsing, type handling, and any recorded repairs belong to the interface; comparative quality is determined from the executed outcome. Conversely, an unchanged configuration does not guarantee an unchanged measured outcome when it is re-executed under a wall-clock limit.

E Recorded Training Settings and Reproduction Order

Table 11 lists the settings available in the saved Full RL training-protocol record. These are recorded run settings, not values inferred from an execution environment. The protocol describes the reported multibudget continuation stage; the warm-start lineage is described in Appendix B.

Setting	Recorded value
Training mode	Multibudget
Optimizer / actor learning rate	AdamW / 2×10^{-7}
Global batch / microbatch per GPU	16 / 1
PPO epochs per update	1
GRPO normalization	Within-group standard deviation
Trainable parameters	All actor parameters (FSDP; no adapter)
Gradient clipping	1.3
Continuation updates	60
Rollouts per prompt	4
Maximum interaction turns	4
Training solver limits	100, 200, 300, 400 seconds
Validation solver limits	100, 200, 300, 400 seconds
Checkpoint interval	10 updates
Validation interval	10 updates
Returned configuration	Last configuration
Terminal improvement coefficient	8.0
Final-configuration deterioration coefficient	4.0
Explicit hold caching during training	Enabled
Recorded training seed	1913552801

Table 11: Settings transcribed from the saved Full RL continuation protocol.

E.1 Dataset and identity preparation

The separate concurrent submission on log-guided, prompt-based SCIP tuning curated the 215-instance bank, specified a shared benchmark protocol, and reported numerical reference results. We reuse the instances for policy training and validation. Table 12 summarizes the source subsets and model choices documented for the inherited bank.

Family	Source subset	Mathematical model or input
CVRP (90)	CVRPLIB sets `A` (27), `B` (23), `E` (13), `F` (3), and `P` (24) (Lima et al., 2014).	Two-commodity flow formulation (Baldacci et al., 2004) with binary edge variables and nonnegative continuous flows, a duplicated depot, and fleet size fixed from the instance name.
GAP (12)	OR-Library `gapc` and `gapd`, six instances each (Beasley, 1990).	Binary assignment minimizing total assignment cost, with exactly one agent per job and agent-capacity constraints (Ross & Soland, 1975).
JSP (24)	OR-Library `jobshop1`, comprising `abz5`–`abz9`, `ft10`, `ft20`, `la01`–`la12`, and `orb01`–`orb05` (Beasley, 1990).	Disjunctive makespan minimization (Manne, 1960) with nonnegative start times, binary order variables, job precedence, and machine non-overlap. The big-M horizon equals the total processing time.
UFL (30)	All `*UnifS` instances in the Uniform family of UflLib (Hoefer, 2018).	Binary facility and assignment variables, with exactly one assignment per customer and assignments restricted to open facilities. The objective minimizes fixed opening costs plus assignment costs.
MIPLIB (59)	A sample from MIPLIB 2017 `benchmark-v2` (Gleixner et al., 2021).	Original `.mps` files used directly.

Table 12: Source subsets and documented model construction for the inherited 215-instance bank.

During construction of the inherited bank, 60 instances were sampled uniformly from MIPLIB 2017 `benchmark-v2` with random seed 42. The instance `k1mushroom` was excluded because it did not complete presolving within 600 seconds under default SCIP settings, leaving 59 instances.

XML100 comprises 100 CVRPLIB-derived instances whose Default SCIP runs certified optimality within 1000 seconds. These instances were selected for a separate retrospective completion-signal study in the concurrent manuscript and are included here for policy training and validation.

We use the fixed split of 262 training and 53 validation instances in Table 1. Each instance retains its source label, objective direction, and evaluation regime across methods. Comparisons match underlying instances independently of method-specific row ordering.

E.2 Model selection before comparison

The warm-start checkpoint is shared by the continued-training pipelines. Candidate checkpoints are then ranked using the frozen mixed-limit assignment and the lexicographic rule in Appendix B. Selection precedes the complete per-limit grid. Prompt-only frontier results do not participate in this checkpoint ranking. A faithful reproduction distinguishes the source backbone, the solver-tuned warm start, and the selected continuation checkpoint instead of referring to all three as the same base model.

E.3 Execution and aggregation

At each fixed solver limit, preserve the requested turn horizon, returned configuration rule, and instance identities. Keep the default-reference execution separate from candidate execution and from model inference time. Retain raw outputs and structured native diagnostics so that the reported counts can be checked without regenerating model responses. Aggregate only after checking identity coverage and endpoint completeness. The frozen numeric comparator is then applied to paired final outcomes, with its tie and missing-value rules unchanged.

The six prompt-only frontier baselines use the recorded application programming interface (API) identifiers `gpt-5.6-sol`, `gpt-5.6-terra`, `gpt-5.6-luna`, `claude-opus-5`, `claude-opus-4-8`, and `claude-sonnet-4-6`.

The seed in Table 11 identifies the recorded training run. Solver calls used PySCIPOpt with `lp/threads=1`, and training used one eight-GPU node. The exact SCIP/PySCIPOpt build identifiers and CPU and accelerator model names were not retained.

F Recorded Prompt Excerpts

The following excerpts are taken from the saved Full RL evaluation input at the 100-second limit. They document the actual prompt interface, not a newly written replacement prompt. The static parameter block is identical across all records in that saved validation file. Its complete 45-entry vocabulary is transcribed into the catalog below with the recorded types, domains, defaults, and descriptions. Full SCIP paths are resolved using the indexed parameter definitions. In these excerpts, LP denotes linear programming, BEST denotes the observed incumbent, and LAST denotes the most recent candidate result. RINS and RENS are defined with the action-family vocabulary in Appendix A.

These are selected prompt blocks, not the complete conversation. Internal scalar-objective instructions and scalar-valued history lines are omitted. Omitted content is not presented as verbatim text. Line wrapping is included for readability. The preserved action contract is the no-rationale variant used in these records; it should not be confused with the optional rationale-bearing variant supported by the prompt builder.

F.1 Task and family-decision instructions

```
Choose one action family from heur, pre, sep, branch/search, or hold
    according to diagnostics and previous turns.
```

```
Use heur for no incumbent, weak primal bound, or feasible-solution
    discovery.
Use pre for presolve-dominated time.
Use sep for cut/separation work.
Use branch/search for tree growth, many open nodes, or weak bound
    progress.
Use hold only when BEST clearly improves default and LAST created or
    preserved that improvement.
[FAMILY_DECISION]
Within branch/search, change scorefunc/reliability/strong-branching
    budget according to tree growth and bound progress; avoid one fixed
    branch tuple.
Within sep, distinguish reducing separator work from allowing more
    root/node cuts; use cut rounds/counts only when separation is the
    bottleneck.
Within pre, adjust rounds/restarts/abort only when presolve consumes
    time or blocks progress; avoid presolve edits on search-dominated
    states.
Within heur, choose between feasibility pump timing and
    RINS/local-branching/RENS budgets according to incumbent/primal
    signals; avoid one fixed heuristic tuple.
Use hold when the current BEST is already clearly useful and the
    previous action created or preserved it.
```

The family labels partition the action vocabulary for instruction purposes. They indicate where the model is asked to consider a small intervention, not an oracle assignment of the optimal solver subsystem. In particular, an incumbent is a configuration already observed during the current episode, not an optimum supplied by a separate search. The policy still chooses specific identifiers and values, and the subsequent solve provides the next execution feedback.

F.2 Output contract and value constraints

```
Output exactly one compact JSON object:
    {"bottleneck":"heur|pre|sep|branch|hold","e":[{"p":<id>,"v":<val>}]}
Do not include rationale, reason, comments, or any extra fields.
No prose, no markdown, no code fence, no explanation.
The first character must be { and generation must stop after the final
    }.
Max 20 edits. Only parameter ids 1-45. Values must be in domain.
Prefer 1-3 high-confidence edits; avoid broad parameter sweeps.
Do not copy a memorized global bundle; choose edits from the current
    diagnostics.
Do not default to any single family; support the family by diagnostics,
    LAST result, and BEST state.
At STEP 1, test a small targeted action for the dominant workload
    instead of a full bundle.
Never use words like "branch", "sep", "pre", or "heur" as p. The p
    field must be an integer id.
Bottleneck must be exactly one of "branch","sep","pre","heur","hold";
    never use "cut" or a parameter name.
Separator ids 11-20 take integer frequencies/counts only; never use
    names like gomory, cmir, clique, flowcover as values.
Heuristic bool edits are only bool ids such as 29; do not put
    true/false into integer ids like 24,28,30.
The schema example above is only a format shape; it is not an action
    template.
Family scope: heur edits may tune ids 24-45 for primal heuristics such
    as feasibility pump, RINS, local branching, and RENS.
Family scope: pre edits may tune ids 21-23 for presolve rounds,
    restarts, and abort factor.
Family scope: sep edits may tune ids 11-20 for separation rounds, cuts,
    and separator frequencies.
```

```
Family scope: branch/search edits may tune ids 1-10 for branching and
    node selection.
Family scope: hold uses {"e":[]} only to preserve a clearly improved
    BEST.
Bad values: {"p":11,"v":"flowcover"}, {"p":16,"v":"cmir"},
    {"p":30,"v":false}, {"p":38,"v":true}.
Avoid recurring bad values: p15-20=1, p28 fractional values,
    p30=true/false/200, and ids outside 1-45.
Do not edit a param to its current/default value; that has no effect
    and is bad.
Do not reuse the same parameter-id set for every instance in a family;
    vary ids and values according to LAST diagnostics.
Value guidance, not a menu: for heur ids 24-45 choose legal in-domain
    values by scale and direction; use disable/rare/frequent calls,
    low/medium/high LP or node budgets, and bools only for bool ids.
Value guidance, not a menu: for pre ids 21-23 choose legal
    low/medium/high rounds, restarts, or abort factors according to
    presolve time and progress; avoid one fixed presolve tuple.
Value guidance, not a menu: for sep ids 11-20 choose legal
    none/low/medium/high cut rounds, cut caps, or separator frequencies
    according to cut activity; avoid one fixed sep tuple.
Value guidance, not a menu: for branch/search ids 1-10 choose a legal
    score rule plus low/medium/high reliability or strong-branching
    budget values according to tree growth.
Categorical values must be JSON strings, e.g. {"p":1,"v":"q"}.
In JSON, bool values must be lowercase true/false, never T/F.
Use {"e":[]} only when no safe edit exists.
```

The exclusion examples above are preserved as instructions, including the recorded warnings about recurring values. They do not constitute a separate mathematical description of the parameter domains. A value can lie within a numeric domain while still being discouraged by the prompt. Conversely, a JSON object can be syntactically valid yet fail to satisfy a parameter's type or semantic constraints. This distinction matters when reproducing the model input rather than merely reconstructing a generic JSON schema.

F.3 BUDGET, TURN, AND QUERY-LOCAL REMINDER

The visible budget, turn, and memory fields from the recorded first query are shown below; `[M]` marks the memory summary, and `B` and `R` denote the configured best and recent window sizes.

```
BUDGET:100s
STEP:1/4
[M] B=3,R=3
```

These lines are a structural excerpt; the incumbent and memory contents between them are omitted here. The memory labels do not imply that six distinct past solver calls exist at the first turn. The current budget is an input to the policy, and the step counter indicates the position within a four-turn episode, not a training update number. The query ends with the following local response reminder.

```
Return ONLY JSON:
    {"bottleneck":"heur|pre|sep|branch|hold","e":[{"p":<int
    1-45>,"v":<domain value>}]}
Validator rejects separator names as values, bools in numeric ids,
    fractional int values, and out-of-domain values.
Use 1-3 targeted edits from one diagnosed family; stop after the final
    }.
No rationale/reason key. No extra fields.
```

The reminder repeats the output contract close to generation. It does not add a second action or a fifth turn. In the saved no-rationale protocol, the absence of a natural-language explanation is intentional;

a short structured proposal is sufficient to invoke the solver. The evaluator records the proposal and its execution separately.

G Complete Indexed Parameter Catalog

The catalog preserves all 45 entries from the recorded prompt. "Default" denotes the default displayed in that prompt and does not imply that every SCIP release uses that default. An empty displayed numeric domain indicates that no numeric range was printed for that entry; it does not imply that the underlying solver parameter is unrestricted. `INF` denotes the prompt's range notation, not a JSON value to insert literally into an action. Tables 13–17 group the entries by solver subsystem.

The displayed default for presolve restarts is `-1`, whereas its printed interval begins at zero. Both are transcribed as recorded, without silently repairing the historical interface. Solver-side defaults, special sentinel values, and action validation must therefore be distinguished. The descriptions explain the controls exposed to the policy, not the measured effects of changing any individual parameter.

G.1 Branching and node selection

ID	SCIP parameter and role	Prompt schema
1	`branching/scorefunc` Branching score formula; changes how candidate variables are ranked.	PROMPT `scorefunc` TYPE `cat[s,p,q]` DEFAULT `p`
2	`branching/preferbinary` Prefer branching on binary variables.	PROMPT `preferbinary` TYPE `bool` DEFAULT `false`
3	`branching/relpscost/minreliable` Min observations before pseudo-cost branching is trusted.	PROMPT `minreliable` TYPE `int[0,INF]` DEFAULT `4`
4	`branching/relpscost/maxreliable` Max reliability threshold before reducing strong branching checks.	PROMPT `maxreliable` TYPE `int[0,INF]` DEFAULT `5`
5	`branching/relpscost/sbiterquot` Strong-branching LP iteration budget as a quotient.	PROMPT `sbiterquot` TYPE `real[0,1]` DEFAULT `0.5`
6	`branching/relpscost/sbiterofs` Strong-branching LP iteration budget offset.	PROMPT `sbiterofs` TYPE `int[0,INF]` DEFAULT `100000`
7	`nodeselection/estimate/stdpriority` Priority of estimate node selection.	PROMPT `est_prio` TYPE `int` DEFAULT `0`
8	`nodeselection/dfs/stdpriority` Priority of depth-first node selection.	PROMPT `dfs_prio` TYPE `int` DEFAULT `-1000000`
9	`nodeselection/bfs/stdpriority` Priority of best-first node selection.	PROMPT `bfs_prio` TYPE `int` DEFAULT `0`
10	`nodeselection/childsel` Child node selection rule after branching.	PROMPT `childsel` TYPE `cat[d,u,p,i,l,r,h]` DEFAULT `h`

Table 13: Recorded parameter specifications for branching and node selection.

G.2 SEPARATION

ID	SCIP parameter and role	Prompt schema
11	`separating/maxrounds` Max separation rounds per node.	PROMPT `sep/maxrounds` TYPE `int[-1,INF]` DEFAULT `-1`
12	`separating/maxroundsroot` Max separation rounds at root.	PROMPT `sep/maxroundsroot` TYPE `int[-1,INF]` DEFAULT `-1`
13	`separating/maxcuts` Max cuts per separation round.	PROMPT `sep/maxcuts` TYPE `int[0,INF]` DEFAULT `100`
14	`separating/maxcutsroot` Max cuts at root separation.	PROMPT `sep/maxcutsroot` TYPE `int[0,INF]` DEFAULT `2000`
15	`separating/gomory/freq` Gomory separator call frequency.	PROMPT `gomory/freq` TYPE `int[-1,65534]` DEFAULT `10`
16	`separating/cmir/freq` Complemented mixed-integer rounding separator call frequency.	PROMPT `cmir/freq` TYPE `int[-1,65534]` DEFAULT `10`
17	`separating/flowcover/freq` Flow-cover separator call frequency.	PROMPT `flowcover/freq` TYPE `int[-1,65534]` DEFAULT `10`
18	`separating/clique/freq` Clique separator call frequency.	PROMPT `clique/freq` TYPE `int[-1,65534]` DEFAULT `0`
19	`separating/knapsackcover/freq` Knapsack-cover separator call frequency.	PROMPT `knapsack/freq` TYPE `int[-1,65534]` DEFAULT `-1`
20	`separating/oddcycle/freq` Odd-cycle separator call frequency.	PROMPT `oddcycle/freq` TYPE `int[-1,65534]` DEFAULT `-1`

Table 14: Recorded parameter specifications for separation.

G.3 PRESOLVE AND FEASIBILITY PUMP

ID	SCIP parameter and role	Prompt schema
21	`presolving/maxrounds` Max presolve rounds.	PROMPT `pre/maxrounds` TYPE `int[-1,INF]` DEFAULT `-1`
22	`presolving/maxrestarts` Max presolve restarts.	PROMPT `pre/maxrestarts` TYPE `int[0,INF]` DEFAULT `-1`
23	`presolving/abortfac` Presolve abort factor; larger stops presolve earlier.	PROMPT `pre/abortfac` TYPE `real[0,1]` DEFAULT `0.0008`
24	`heuristics/feaspump/freq` Feasibility-pump call frequency.	PROMPT `fp/freq` TYPE `int[-1,65534]` DEFAULT `20`
25	`heuristics/feaspump/freqofs` Feasibility-pump frequency offset.	PROMPT `fp/freqofs` TYPE `int[0,65534]` DEFAULT `0`
26	`heuristics/feaspump/maxdepth` Feasibility-pump max search depth.	PROMPT `fp/maxdepth` TYPE `int[-1,65534]` DEFAULT `-1`
27	`heuristics/feaspump/maxlpiterquot` Feasibility-pump LP iteration quotient.	PROMPT `fp/lpiterquot` TYPE `real[0,INF]` DEFAULT `0.01`

Continued on next page

Table 15 continued

ID	SCIP parameter and role	Prompt schema
28	`heuristics/feaspump/maxlpiterofs` Feasibility-pump LP iteration offset.	PROMPT `fp/lpiterofs` TYPE `int[0,INF]` DEFAULT `1000`
29	`heuristics/feaspump/beforecuts` Run feasibility pump before cuts.	PROMPT `fp/beforecuts` TYPE `bool` DEFAULT `true`

Table 15: Recorded parameter specifications for presolve and feasibility pump.

G.4 RINS AND LOCAL BRANCHING

ID	SCIP parameter and role	Prompt schema
30	`heuristics/rins/nodesofs` RINS node budget offset.	PROMPT `rins/nodesofs` TYPE `int[0,INF]` DEFAULT `500`
31	`heuristics/rins/nodesquot` RINS node budget quotient.	PROMPT `rins/nodesquot` TYPE `real[0,1]` DEFAULT `0.3`
32	`heuristics/rins/minnodes` RINS minimum node budget.	PROMPT `rins/minnodes` TYPE `int[0,INF]` DEFAULT `50`
33	`heuristics/rins/maxnodes` RINS maximum node budget.	PROMPT `rins/maxnodes` TYPE `int[0,INF]` DEFAULT `5000`
34	`heuristics/rins/nwaitingnodes` RINS waiting-node trigger.	PROMPT `rins/nwait` TYPE `int[0,INF]` DEFAULT `200`
35	`heuristics/rins/minfixingrate` RINS minimum fixing rate.	PROMPT `rins/minfixrate` TYPE `real[0,1]` DEFAULT `0.3`
36	`heuristics/localbranching/neighborhoodsize` Local-branching neighborhood size.	PROMPT `lb/nbrsize` TYPE `int[1,INF]` DEFAULT `18`
37	`heuristics/localbranching/nodesofs` Local-branching node budget offset.	PROMPT `lb/nodesofs` TYPE `int[0,INF]` DEFAULT `1000`
38	`heuristics/localbranching/nodesquot` Local-branching node budget quotient.	PROMPT `lb/nodesquot` TYPE `real[0,1]` DEFAULT `0.05`
39	`heuristics/localbranching/lplimfac` Local-branching LP limit factor.	PROMPT `lb/lplimfac` TYPE `real[1,INF]` DEFAULT `1.5`

Table 16: Recorded parameter specifications for RINS and local branching.

G.5 RENS

ID	SCIP parameter and role	Prompt schema
40	`heuristics/rens/nodesofs` RENS node budget offset.	PROMPT `rens/nodesofs` TYPE `int[0,INF]` DEFAULT `500`
41	`heuristics/rens/nodesquot` RENS node budget quotient.	PROMPT `rens/nodesquot` TYPE `real[0,1]` DEFAULT `0.1`
42	`heuristics/rens/minnodes` RENS minimum node budget.	PROMPT `rens/minnodes` TYPE `int[0,INF]` DEFAULT `50`

Continued on next page

Table 17 continued

ID	SCIP parameter and role	Prompt schema
43	`heuristics/rens/maxnodes` RENS maximum node budget.	PROMPT `rens/maxnodes` TYPE `int[0,INF]` DEFAULT `5000`
44	`heuristics/rens/minfixingrate` RENS minimum fixing rate.	PROMPT `rens/minfixrate` TYPE `real[0,1]` DEFAULT `0.5`
45	`heuristics/rens/startsol` RENS start solution source.	PROMPT `rens/startsol` TYPE `cat[l,n]` DEFAULT `l`

Table 17: Recorded parameter specifications for RENS.

G.6 READING THE CATALOG AS A CONFIGURATION INTERFACE

Parameter names are hierarchical. The prefix identifies a solver subsystem, and the suffix identifies a control within that subsystem. The compact identifier serves as the action space key, while the full path identifies the corresponding solver setting. A sparse proposal need not repeat the complete catalog. Unedited entries are inherited from the configuration to which the interaction applies the proposal.

The integer fields serve different roles. A frequency controls when a component is called; a round count limits repeated work; and a node offset or node cap controls the amount of subsearch. These controls are not interchangeable merely because they share an integer type. Real-valued quotients and factors likewise belong to their named components rather than forming a generic continuous vector without semantics. Categorical values must use the displayed code, and Boolean entries must be distinguished from integer-valued zero or one in the JSON representation.

The catalog includes related controls whose effects can interact. Root separation and node separation have distinct round and cut limits. RINS and RENS each expose node-budget and fixing-rate controls, but they remain different heuristic components. The presence of these controls motivates feedback-conditioned small edits; it does not establish that a specific family is always the correct intervention for a particular benchmark class.