

Log-Driven Semantic Tuning for Mixed Integer Programming Solvers

Zhengzhong Ricky You, Kaizhao Sun, Zeping Min, Wotao Yin, Xinshang Wang*

Alibaba Group US, Bellevue, WA, USA

*Correspondence: Xinshang Wang, xinshang.w@alibaba-inc.com

Abstract. Automated configuration of mixed-integer programming (MIP) solvers largely relies on black-box optimization. Each trial requires an expensive solver run and provides only a scalar outcome, so tuning can require many such evaluations. Human experts take a different approach. They inspect solver logs, identify performance bottlenecks, and adjust the parameters that address them. We propose Log-Driven Semantic Tuning (LDST), which uses a large language model (LLM) to automate this process. LDST extracts compact diagnostics from solver logs, uses the LLM to propose sparse edits to the current best configuration based on parameter meanings and observed solver behavior, and validates each proposal before execution. A regime-adaptive objective selects the tuning signal according to the information available from a time-limited run, while a log-based completion signal estimates completion time from branch-and-bound tree growth recorded in ordinary solver logs, so that solved and unsolved runs share one time scale. On 215 instances from five benchmark sets, LDST reduces geometric-mean loss by 43.9% on easy instances and 16.7% on intermediate instances relative to the best of SMAC3, RBFOP, and random search. In both regimes, LDST achieves a lower loss with 10 proposals than any of these baselines achieves with 50. Configurations that LDST tunes on a single instance also transfer better than baseline configurations to unseen instances of the same family.

Key words: mixed-integer programming; algorithm configuration; large language models; parameter tuning; SCIP

1. Introduction

The performance of mixed-integer programming (MIP) solvers such as SCIP (Achterberg 2009) depends heavily on parameter configuration, and instance-specific or class-specific tuning can yield order-of-magnitude speedups (Hutter et al. 2010). Yet most automated configuration methods rely on black-box optimization (BBO; Schede et al. 2022, Hoos 2012, Hutter et al. 2009, 2011, Lindauer et al. 2022, López-Ibáñez et al. 2016, Ansótegui et al. 2009, Xu et al. 2011, Hutter et al. 2014, Eggenberger et al. 2019). These methods typically use only scalar feedback, such as runtime or a derived loss, making the search highly sample inefficient when solver evaluations are expensive. A run that times out because of weak heuristics may be difficult to distinguish from one slowed by excessive linear programming (LP) relaxation work without additional evaluations. Human experts take a different approach. They examine solver logs, infer performance bottlenecks from process signals, and make targeted parameter changes. For example, a run that finds primal solutions early but makes little progress on the dual bound requires a different intervention from one dominated by the computational cost of strong branching.

We show that solver-process information can systematically guide algorithm configuration. We propose Log-Driven Semantic Tuning (LDST), which extracts compact diagnostics from solver logs and uses a large language model (LLM) to propose sparse edits to the current best configuration. Each edit changes only a limited number of parameters from a predefined whitelist, while all other parameter values are inherited from the current best configuration. The tuning loop combines diagnostic summaries, memory centered on the best configuration, and explicit validation of the LLM’s advice, allowing semantic reasoning to guide the search while each validated configuration is evaluated through an actual solver run. By modifying the current best configuration rather than constructing each candidate from scratch, LDST maintains a local and stable search, makes individual parameter changes easier to interpret, and uses information from earlier runs to inform subsequent decisions.

The tuning loop also uses a regime-adaptive objective, as the most informative progress signal depends on the outcome of the default run at the time limit. Case 1 contains easy instances solved by the default

configuration, Case 2 contains intermediate instances with a feasible solution but no proof of optimality, and Case 3 contains hard instances with only dual-bound progress. For easy instances, solve time is the natural objective, but a configuration that reaches the time limit has no observed solve time. Solved status, final gap, and the primal–dual integral cannot be compared with observed solve times, and existing tree-size estimators are built into specific solvers (Hendel et al. 2022). We therefore introduce a log-based completion signal that estimates the completion time of an unfinished run from branch-and-bound tree growth, which MIP solver logs commonly report. Solved and unsolved runs then share one time scale, so a single Case 1 objective ranks both. In a retrospective ranking study, the signal also orders unfinished runs more accurately than a proxy for the primal–dual integral. Across 215 instances from five benchmark sets, LDST achieves final geometric-mean losses of 0.274 and 0.557 in Cases 1 and 2 after 50 proposals, respectively, 43.9% and 16.7% below the best scalar-feedback baselines (0.490 and 0.668). By trial 10, LDST already reaches losses of 0.396 and 0.660, lower than any baseline achieves after 50 proposals, while the corresponding baseline losses at trial 10 remain at 0.588–0.706 and 0.763–0.807.

We make the following contributions.

- *We propose LDST, a log-driven semantic tuner for MIP solvers.* LDST extracts compact diagnostics from solver logs and uses an LLM to propose sparse edits to the current best configuration. Each proposal undergoes schema, domain, novelty, and settings-loading checks before evaluation. This provides a process-aware alternative to scalar-feedback tuning without requiring task-specific training, solver-source modifications, or access to internal solver states.
- *We introduce a regime-adaptive tuning objective with a log-based completion signal for easy instances.* The objective uses estimated completion time for easy instances, the primal–dual integral (Berthold 2013) for intermediate instances with feasible solutions, and normalized dual improvement for hard instances with only dual progress. For easy instances, $\hat{T}_\infty$ estimates the completion time of unfinished runs by fitting a simple surrogate of branch-and-bound tree growth to log snapshots, which places solved and unsolved runs on one time scale. In a retrospective ranking study, it orders 26 of 34 configuration pairs correctly, against 17 for a proxy for the primal–dual integral (Appendix G).
- *We show that log-driven semantic tuning improves configuration quality under matched budgets in the easy and intermediate regimes.* We evaluate LDST on a released benchmark of 215 instances from five benchmark sets, with generated formulations for repeated SCIP tuning and fixed regime labels. In single-instance tuning, LDST reduces geometric-mean loss by 43.9% and 16.7% in the easy and intermediate regimes, respectively, and achieves lower losses with 10 proposals than the baselines attain with 50. On Case 1, mechanism isolation attributes the gain to the combination of parameter semantics and log diagnostics, while configurations tuned by LDST on a single instance outperform baseline configurations when transferred to unseen instances of the same family.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 defines the regime-adaptive objective and the log-based completion signal, and Section 4 presents the LDST framework. Section 5 reports the experimental results, and Section 6 concludes the paper. The appendices provide details on the benchmark, comparison protocol, parameter space, log parser, prompt construction, validation of the completion signal, and additional results. Code, generated benchmark instances, and analysis scripts are available at <https://github.com/Zhengzhong-You/LDST>.

2. Related Work

Automated algorithm configuration. Automated solver tuning has traditionally relied on black-box search. Early work demonstrated that automated tuning can outperform expert-designed default settings for MIP solvers (Hutter et al. 2010). Representative approaches include the SMAC family (Hutter et al. 2011,

Lindauer et al. 2022), racing methods such as irace (López-Ibáñez et al. 2016), evolutionary methods such as GGA (Ansótegui et al. 2009), and surrogate-based optimizers such as RBFOpt (Costa and Nannicini 2018). These methods treat the solver as an opaque mapping from configuration to outcome and search the configuration space through repeated evaluations. In contrast, LDST uses log-derived diagnostics to identify performance bottlenecks and parameter semantics to guide targeted configuration edits.

Feature-based and instance-specific configuration. Classical configurators can also exploit static instance features. SMAC can incorporate instance descriptors into its surrogate model (Hutter et al. 2011, Lindauer et al. 2022); ISAC clusters training instances based on their features and tunes a configuration for each cluster (Kadioglu et al. 2010); and Hydra-MIP constructs portfolios for heterogeneous MIP families and selects among them for new instances (Xu et al. 2011). These approaches use instance features to guide the selection of configurations or portfolios learned from prior instances. In contrast, LDST needs no instance features or configurations learned in advance for a given instance class. Instead, it adapts during tuning using diagnostics extracted from its own solver runs.

Learning-based solver control and configuration. Another line of research uses learned models to guide internal solver decisions, including branching (Gasse et al. 2019, Gupta et al. 2020) and cut selection (Tang et al. 2020). Related work on solver configuration uses graph neural networks or Deep Sets to encode problem instances and predict promising parameter settings (Hosny and Reda 2023, Song et al. 2023). These approaches can exploit problem structure but often require large training sets, closer integration with the solver, or both. Adapting them to new instance distributions may also require retraining. In contrast, LDST operates between solver runs without task-specific training and reads solver behavior from standard logs, avoiding dependence on internal solver states that may be inaccessible without modifying or instrumenting the solver.

Large language models for optimization. Recent work has explored LLMs as optimization agents, including OPRO (Yang et al. 2024), heuristic evolution systems such as FunSearch (Romera-Paredes et al. 2024), and LLM-assisted configuration methods. In hyperparameter optimization, LLMs have been used to propose configurations based on task descriptions and trial histories (Zhang et al. 2023, Liu et al. 2024a), while LLAMBO integrates an LLM into Bayesian optimization for warm starting, surrogate modeling, and candidate sampling (Liu et al. 2024b). For MIP solvers, LLMs have been used to configure cutting-plane separators (Lawless et al. 2025), guide online parameter configuration in re-optimization settings (Liu et al. 2026), and switch among branching rules online based on structured tree-state metrics (Zheng et al. 2026). Execution feedback has also been used to guide LLM-based optimization in other domains, including log-based step-size adaptation for evolution strategies (Custode et al. 2024), parallel-program optimization (Wei et al. 2025), and trajectory summaries for hyperparameter tuning (Naphade et al. 2025). Concurrent work GRIMIP (Luo et al. 2026) and LDST both use LLMs for iterative MIP solver configuration without task-specific training. Within a Bayesian optimization framework, GRIMIP draws on the history of evaluated configurations and their scalar performance scores. In contrast, LDST goes beyond scalar feedback by retaining log-derived diagnostics from successive solver runs and using this process-level history, together with parameter semantics, to guide sparse edits to the best configuration found so far.

Log-based progress extrapolation. Recent work has used solver-process signals to estimate search progress and runtime. Hendel et al. (2022) study branch-and-bound tree-size estimation using signals such as gap, tree weight, and leaf frequency, together with forecasting and learned estimators. Their results demonstrate that search-completion information can be inferred from solver-process signals, although their strongest estimators rely on richer internal measures and trained predictors implemented within the solver. In contrast, our completion signal uses only the node counts and bounds in the progress table, which MIP solver logs commonly report. By fitting a simple surrogate to the observed trajectory, it estimates completion

time using standard log output, without task-specific training, source-code access, or solver modification. In LDST, this signal places unfinished runs on the same time scale as solved runs, so one objective ranks both on easy instances. The same signal can also support solver monitoring and runtime estimation.

3. Problem Formulation

In this section, we formalize the tuning setting used throughout the paper. A single scalar loss does not provide equally informative feedback across all instances, because the progress information available at the time limit differs across instances. For instances that the default configuration solves, solved status, final gap, and the primal–dual integral may provide limited information for ranking truncated runs because none directly estimates the remaining time to proof. For instances with feasible solutions but no proof of optimality, both primal and dual bound trajectories are available and can support a trajectory-based measure. When no feasible solution has been found, primal-based measures are unavailable, leaving dual progress as the basis for comparison. We therefore first classify instances according to their outcomes under the default configuration and define a regime-adaptive loss that selects the most informative progress signal available at the time limit. We then introduce a log-based completion signal for easy instances that estimates remaining search effort from the observed branch-and-bound trajectory.

3.1. Case Split and Regime-Adaptive Loss

For each instance i and configuration p , we run the solver with a fixed time limit τ (e.g., 600 s), compute a case-dependent scalar loss, and retain the execution log $\Lambda_i(p)$ for diagnostic use. Based on the outcome under the default configuration p^{def} , we assign each instance to one of the three cases summarized in Table 1. Case 1 contains instances solved within τ under p^{def} (proved optimal, or proved infeasible for the single infeasible instance). Case 2 contains intermediate instances that are not solved within τ but do have a primal bound at time τ . Case 3 contains hard instances with a dual bound but no feasible primal solution by τ . These assignments remain fixed throughout tuning, preventing changes in configuration from altering the evaluation objective and ensuring that losses are comparable across methods.

Table 1 Case definitions and tuning signals.

Case	Level	Outcome	Signal	Summaries
Case 1	Easy	solved	$\hat{T}_\infty$	solve rate, time
Case 2	Intermediate	feasible, no proof	PDI	primal, dual, gap
Case 3	Hard	dual progress only	norm. dual improv.	final dual, improved-dual coverage

Notes. Cases are defined under the default configuration p^{def} . PDI denotes the primal–dual integral.

The loss for instance i is defined according to the most informative progress signal available in its assigned case. Let $c_i \in \{1, 2, 3\}$ denote the case assigned to instance i under the default configuration p^{def} . For Case 1, we use $\hat{T}_{\infty,i}(p)$, the estimated completion time under configuration p , whose construction and fitting procedure are described in Section 3.2 and Appendix G. The corresponding normalized loss is $L_i^{(1)}(p) = \hat{T}_{\infty,i}(p) / \hat{T}_{\infty,i}(p^{\text{def}})$. Let $z_i^{\text{pr}}(t; p)$ and $z_i^{\text{du}}(t; p)$ denote the primal and dual bounds at time t under configuration p . For Case 2, we use the primal–dual integral $\text{PDI}_i(p)$ reported by SCIP at the end of the run (Berthold 2013). This quantity integrates the relative gap between $z_i^{\text{pr}}(t; p)$ and $z_i^{\text{du}}(t; p)$ over solving time, with SCIP expressing the gap in percent, capping it at 100, and using the same value when either bound is unavailable. The corresponding normalized loss is $L_i^{(2)}(p) = \text{PDI}_i(p) / \text{PDI}_i(p^{\text{def}})$. For Case 3, we measure normalized improvement in the final dual bound, defined as $L_i^{(3)}(p) = 1 + s_i(z_i^{\text{du}}(\tau; p) - z_i^{\text{du}}(\tau; p^{\text{def}})) / (|z_i^{\text{du}}(\tau; p^{\text{def}})| + \varepsilon)$ with $\varepsilon = 10^{-6}$, where $s_i = -1$ for minimization and $s_i = +1$ for maximization. The overall instance-level loss is $L_i(p) = L_i^{(c_i)}(p)$, with every normalized loss floored at 10^{-12} to

ensure that geometric means remain well defined. This regime-adaptive design selects the strongest reliable progress signal available at the time limit while preserving the interpretability of standard solver summaries.

3.2. Log-Based Completion Signal

On Case 1 instances, solve time is the natural objective, but a run that reaches the time limit τ has no observed solve time. Recording τ for every such run would tie configurations that differ substantially in their remaining search effort, and final gaps and primal–dual integrals cannot be compared with the solve times of finished runs. We therefore use the completion-time estimator $\hat{T}_{\infty,i}(p)$, which places finished and unfinished runs on one time scale. If a run is solved within τ , then $\hat{T}_{\infty,i}(p) = T_i(p)$, where $T_i(p)$ denotes the observed solve time. Otherwise, we truncate the parsed SCIP log at $\tau' = \min\{\tau, t_{\max}\}$, where $t_{\max}$ is the last logged time, and form snapshots (t_j, n_j, u_j, g_j) from its progress-table rows. Here t_j is the elapsed time, n_j the number of processed nodes, u_j the number of open nodes, and $g_j = |z_{\text{final}}^{\text{pr}} - z^{\text{du}}(t_j)|$ the absolute difference between the dual bound at t_j and the primal bound $z_{\text{final}}^{\text{pr}}$ reported at the end of the observed run, which for tuning runs is the primal bound available at τ . Following the single-variable branching model of [Le Bodic and Nemhauser \(2017\)](#), as further analyzed by [Anderson et al. \(2021\)](#), each snapshot yields the terminal-tree-size surrogate $\hat{N}_j(x) = n_j + u_j x^{g_j}$. The estimator fits a common terminal tree size b^* on the log scale, estimates the recent time per processed node θ from the last five sampled rows, and predicts $\hat{T}_{\infty,i}(p) = \tau' + \theta \max\{b^* - n_{\text{last}}, 0\}$, where n_{last} is the last sampled processed-node count. The signal is designed for Case 1, where runs are close to proof, and Cases 2 and 3 use the bound-based signals defined above. Appendix G gives the row filtering, the fitting problem, and the safeguards against unstable fits, together with a retrospective validation.

4. Log-Driven Semantic Tuning

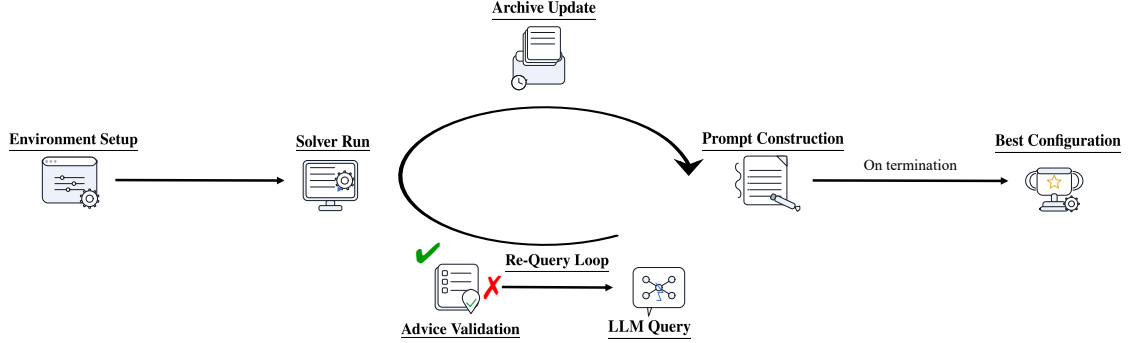
LDST combines log parsing, sparse edit generation around the current best configuration, and advice validation. Let $\mathcal{I}_T$ denote the tuning set and $L(p; \mathcal{I}_T)$ the aggregate tuning objective, defined as the geometric mean of the instance-level losses in Section 3 over $\mathcal{I}_T$. At trial k , evaluating configuration $p^{(k)}$ produces aggregate loss $L^{(k)} := L(p^{(k)}; \mathcal{I}_T)$ and log stream $\Lambda^{(k)}$, while p^* denotes the best configuration found so far. We first describe the overall tuning loop, followed by the diagnostic and prompting design.

4.1. The Tuning Loop

Figure 1 illustrates the LDST tuning loop. Starting from the default configuration, each trial runs the solver on $\mathcal{I}_T$, extracts diagnostics from the resulting logs, and records the results in an archive. The best configuration p^* is updated to the archived configuration with the lowest loss, with exact ties broken in favor of the most recent trial. If tuning budget remains, the next prompt is constructed from the current best configuration, bounded memory, and hard edit constraints. The LLM then proposes sparse edits relative to p^* . Advice validation then checks each proposal before execution, and invalid proposals trigger feedback and re-queries until a valid edit is obtained or the re-query limit is reached. The validated configuration is then evaluated in the next trial. By centering the search on the best configuration found so far, LDST maintains a local and stable search while allowing the LLM to adapt to recent solver behavior.

4.2. Diagnostic State and Prompting

Diagnostic summary. At trial k , the raw log stream $\Lambda^{(k)}$ is summarized in a diagnostic record $r^{(k)} = (L^{(k)}, p^{(k)}, \mathbf{d}^{(k)})$, where $\mathbf{d}^{(k)} = \text{Parse}(\Lambda^{(k)})$. The diagnostic summary $\mathbf{d}^{(k)}$ captures search scale, primal–solution and bound progress, and activity in presolving, root relaxation, strong branching, separation, and major heuristics. Applying the same parser and fixed-key schema to every run ensures comparability across trials without instance-specific prompt engineering. This provides LDST with a consistent view of solver activity and the bottlenecks that dominate each run. Appendix E lists the observable fields used by the parser.

Figure 1 Overview of the LDST tuning loop.

Notes. Each solver run is parsed and added to the archive. The next prompt is constructed from the current best configuration, bounded memory, and edit constraints. The LLM’s proposed edits are validated, with re-queries following failed validation, before the next solver run. Upon termination, the best configuration found is returned.

Memory and validation. The prompt combines task constraints, a parameter whitelist $\mathcal{W}$, the current best configuration, and a bounded memory comprising the best historical records and recent trials. Each non-best configuration p is represented relative to p^* by the edit set $\Delta(p; p^*) := \{(j, p_j) : j \in \mathcal{W}, p_j \neq p_j^*\}$. This representation includes only the whitelist parameters whose values differ from the current best configuration, rather than the full parameter vector. Only the entries in $\Delta(p; p^*)$ are changed; all other parameter values are inherited from the current best configuration. For example, if a previous configuration differs from the current best configuration only in `relpscost/minreliable` and `sbiterquot`, then $\Delta(p; p^*)$ contains only these two entries. Before execution, each proposed edit must pass schema, domain, and novelty checks and load successfully as a SCIP settings file. The novelty check uses a deduplication list of previously evaluated configurations. If validation fails, the resulting feedback is included in the next query. When a family-level formulation text is available, the prompt also includes it (Appendix B). Appendix F provides representative prompts and implementation details.

5. Experiments

The experiments test whether log-driven semantic tuning improves single-instance configuration under matched solver-run budgets, identify which information channels contribute to its gains, and measure how well the tuned configurations transfer to unseen instances within the same benchmark family. Exp. A compares LDST with SMAC3, RBFOPT, and random search (RANDOM) across the three regimes. Exp. B isolates the contributions of scalar history, parsed diagnostics, and parameter semantics on Case 1 instances. Exp. C evaluates within-family transfer from single-instance tuning and multi-instance training splits. All methods share the same solver setup, time limit, editable parameter whitelist, seed convention, and solver-run budgets within each comparison. Appendix G validates $\hat{T}_\infty$, while the remaining appendices provide implementation details, parser fields, and representative prompts.

5.1. Settings

Solver setup and configuration space. We use SCIP 9.2.4 (Bolusani et al. 2024) in single-thread mode with a per-run time limit of $\tau = 600$ s. For reproducibility, we fix randomization/randomseedshift to 42 in all solver runs and set the optimizer seeds of SMAC3, RBFOPT, and RANDOM to 42. Each method runs once per instance under this convention, and the sign tests and bootstrap bands below treat instances as the unit of replication. All methods use the same curated whitelist $\mathcal{W}$ of 45 editable SCIP parameters, detailed in Appendix C. We compare LDST with three black-box configurators, SMAC3, RBFOPT, and RANDOM. All LLM-based methods use GPT-5.2 with medium reasoning effort, and Appendix D provides the remaining LLM settings and default prompt hyperparameters.

Instances and regime assignment. The benchmark comprises 215 instances from five benchmark sets, covering capacitated vehicle routing (CVRP), generalized assignment (GAP), job shop scheduling (JSP), uncapacitated facility location (UFL), and general MIPLIB 2017 instances. Appendix A provides the instance lists and construction details. Following Section 3, each instance is assigned to Case 1, 2, or 3 based on its outcome under the default configuration p^{def} at the time limit τ . These assignments remain fixed across all experiments, regardless of the configurations selected during tuning. Table 2 summarizes the resulting distribution. Appendix I documents the code and data release, licensing information, and hardware and software environment used in the experiments.

Table 2 Case counts by benchmark set with source references.

Set	Source	#	Case 1	Case 2	Case 3	Set	Source	#	Case 1	Case 2	Case 3
CVRP	Lima et al. (2014)	90	11	39	40	UFL	Hoefer (2018)	30	30	0	0
GAP	Beasley (1990)	12	7	5	0	MIPLIB	Gleixner et al. (2021)	59	21	30	8
JSP	Beasley (1990)	24	7	17	0						
Total		215	76	91	48						

5.2. Exp. A: Single-Instance Tuning

A1: Main results. We tune each instance independently for $K = 50$ proposal trials following an initial default run, yielding 51 solver evaluations per method-instance pair. The regime-adaptive loss defined in Section 3, including the Case 1 completion signal $\hat{T}_\infty$, is fixed before comparison and used identically by LDST, SMAC3, RBFOPT, and RANDOM. For each method, the reported loss on instance i is the best-so-far value $b_i(50) = \min_{0 \leq t \leq 50} L_i(p_i^{(t)})$, where trial 0 corresponds to the default configuration. Thus, the reported loss never exceeds 1. Table 3 shows that LDST achieves the lowest geometric-mean loss in Cases 1 and 2, outperforming the best scalar-feedback baseline by 43.9% and 16.7%, respectively. These improvements are also reflected in standard solver performance measures. In Case 1, the shifted geometric-mean solve time decreases from 17.1–17.9s for the baselines to 10.5s for LDST. In Case 2, the final gap decreases from 10.0–10.8% to 4.9%, accompanied by better primal and dual bounds. Paired statistical tests further support these findings. For each pairwise comparison, we apply an exact two-sided sign test after excluding ties. Let n_+ and n_- denote the numbers of instances on which LDST attains a lower and a higher loss, respectively. The resulting p -value equals $\min\{1, 2 \sum_{j=0}^{\min\{n_+, n_-\}} \binom{n_+ + n_-}{j} 2^{-(n_+ + n_-)}\}$. In Case 1, LDST outperforms SMAC3, RBFOPT, and RANDOM on 67, 65, and 67 of the 76 instances, respectively, with all three comparisons significant at $p \leq 1.9 \times 10^{-10}$. In Case 2, LDST wins on 71, 71, and 68 of the 91 instances, with corresponding p -values of 7.2×10^{-8} , 7.2×10^{-8} , and 2.5×10^{-6} .

In Case 3, solver runs provide only dual-bound progress within the time limit, and all four methods achieve aggregate losses within 1.2% of the default. RBFOPT attains the lowest aggregate loss (0.988), mainly through its final proposal on `miplib_irish-electricity`, which improves the dual bound by 30% of its default value (Appendix H). LDST, SMAC3, RBFOPT, and RANDOM improve the dual bound on 43, 43, 41, and 42 of the 48 instances, respectively.

A2: Convergence behavior. Figure 2 shows that LDST establishes an early advantage and maintains it throughout tuning in Cases 1 and 2. In Case 1, LDST achieves the lowest geometric-mean best-so-far loss of 0.396 by trial 10, compared with 0.588–0.706 for the baselines. By trial 50, its loss decreases to 0.274, compared with 0.490 for RBFOPT, 0.499 for SMAC3, and 0.519 for RANDOM. A similar pattern emerges in Case 2. By trial 10, LDST reaches a loss of 0.660, while the baselines remain at 0.763–0.807. By trial 50, its loss decreases to 0.557, compared with 0.668–0.675 for the three baselines. Thus, in both

Table 3 Single-instance tuning results after 50 proposals.

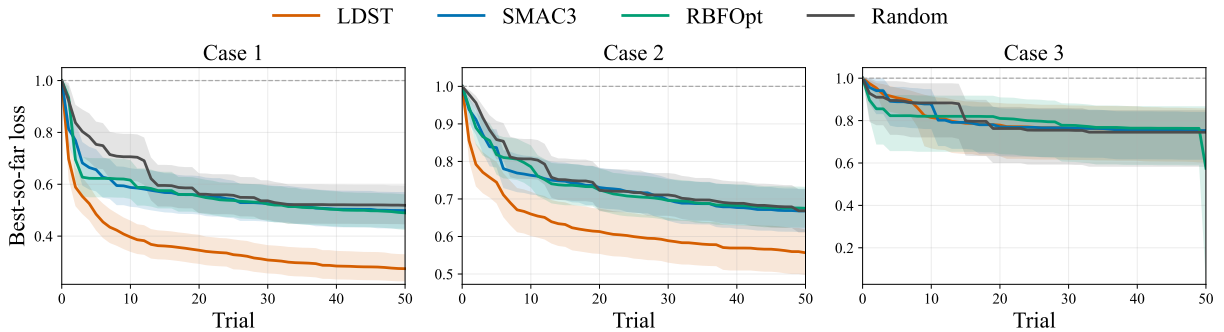
Method	Primary loss $L \downarrow$			Case 1	Case 2		
	Case 1	Case 2	Case 3	Time (s) $\downarrow$	Norm. primal $\downarrow$	Norm. dual $\uparrow$	Gap (%) $\downarrow$
LDST	0.274	0.557	0.993	10.5	1.010	0.990	4.9
SMAC3	0.499	0.668	0.993	17.3	1.034	0.975	10.3
RBFOPT	0.490	0.675	0.988	17.1	1.039	0.975	10.8
RANDOM	0.519	0.669	0.993	17.9	1.030	0.977	10.0

Notes. For each method, the selected configuration is the best among the default (trial 0) and the 50 proposals. Primary losses are geometric means over all 76, 91, and 48 instances in Cases 1, 2, and 3, respectively. “Time” is the shifted geometric mean of solve time (shift 1 s) over the 75 Case 1 instances that every method solves to optimality; the remaining instance, `miplib_neos859080`, is infeasible. For Case 2, let $\tilde{z}_{i,m}^{\text{pr}} = -s_i z_{i,m}^{\text{pr}}$ and $\tilde{z}_{i,m}^{\text{du}} = s_i z_{i,m}^{\text{du}}$ denote the sense-aligned final primal and dual bounds of method m on instance i , and let b_i^* denote the best aligned bound among the compared methods on the 90 instances with finite bounds under all methods. “Norm. primal” is the geometric mean of $1 + (\tilde{z}_{i,m}^{\text{pr}} - b_i^*) / \max\{|\tilde{z}_{i,m}^{\text{pr}}|, |b_i^*|, \varepsilon\}$ (lower is better). “Norm. dual” applies the reciprocal score to the aligned dual bound (higher is better). “Gap” is the shifted geometric mean (shift 0.01) of the final relative gap in percent on the same 90 instances.

regimes, LDST achieves a lower loss with 10 proposals than any baseline attains with 50. Exp. B identifies which information channels produce this early advantage.

For Case 3, Figure 2 applies the display-only transformation $\tilde{L}^{(3)} = 1 + 36(L^{(3)} - 1)$ to make the small differences between methods visible. The late decrease in the RBFOPT curve reflects its trial-50 improvement on `miplib_irish-electricity`, as discussed in A1.

A3: Edit sparsity and LLM cost. LDST changes few parameters per proposal. Across its 10,750 proposals in Exp. A, the evaluated configuration differs from the best configuration at the time of the proposal in a median of 7 of the 45 whitelisted parameters, with medians of 5, 7, and 11 in Cases 1, 2, and 3. Overall, 82% of the proposals change at most 10 parameters, and 1.2% reach the edit cap of 20 (Table 19 in Appendix H). The LLM side is also inexpensive. The 10,964 LLM calls behind these proposals, including re-queries, cost about USD 320 at uncached list prices, or about USD 0.03 per proposal (Table 23 in Appendix I).

Figure 2 Geometric mean of the best-so-far loss in Exp. A.

Notes. For each regime c and trial t , the shaded band is constructed from 2000 bootstrap samples, each containing $|\mathcal{I}_c|$ instances drawn with replacement from $\mathcal{I}_c$. For each sample, the aggregate best-so-far loss at trial t is recomputed using the same formula as the main curve. The lower and upper edges are the 2.5th and 97.5th percentiles of the 2000 bootstrap values at that trial. All curves start from the default configuration at trial 0 (loss 1). The Case 3 panel applies the display-only transformation $\tilde{L}^{(3)} = 1 + 36(L^{(3)} - 1)$.

5.3. Exp. B: Mechanism Isolation

Exp. B examines the information channels underlying the gains of LDST. We compare LDST with LLM-STATICCONTEXT, LLM-OUTCOMEONLY, and LLM-INDEXONLY under the single-instance Case 1 proto-

col. Each of the 76 Case 1 instances is tuned independently for $K = 30$ trials, and results are aggregated over all instances. The LDST results use the first 30 trials of its Exp. A runs. All methods follow the shared execution contract in Appendix B, with the controls differing only in the information available to the decision policy. Specifically, LLM-STATICCONTEXT removes the cross-trial archive, including scalar history, along with the diagnostic guide and deduplication list, while retaining the parsed diagnostics of the current best run. LLM-OUTCOMEONLY retains scalar history but removes parsed diagnostics and the diagnostic guide. LLM-INDEXONLY replaces parameter names and defaults with indices and domain hints. Table 16 details these changes to the prompts.

Table 4 shows that LDST performs best across all three metrics, achieving a Best loss of 0.308, early aggregate loss $L_{\leq 5} = 0.485$, and threshold-hit rate $H_{0.8} = 90.8\%$. All three metrics give the same ranking, with LDST followed by LLM-STATICCONTEXT, LLM-OUTCOMEONLY, and LLM-INDEXONLY. Removing parameter semantics causes the largest performance degradation. Despite retaining scalar history and parsed diagnostics, LLM-INDEXONLY achieves a Best loss of 0.523, with $L_{\leq 5}$ at 0.707 and $H_{0.8}$ at 76.3%. This suggests that index-level domain hints alone are insufficient to translate solver feedback into effective parameter edits. Removing scalar history and archive memory has a smaller but noticeable effect. Although LLM-STATICCONTEXT retains the current best run’s diagnostics, it remains weaker than LDST, with a Best loss of 0.342 versus 0.308, 0.509 versus 0.485 in $L_{\leq 5}$, and 86.8% versus 90.8% in $H_{0.8}$. Removing parsed diagnostics while retaining scalar history also reduces performance. LLM-OUTCOMEONLY achieves a Best loss of 0.358, $L_{\leq 5} = 0.520$, and $H_{0.8} = 84.2\%$, a smaller degradation than that caused by removing parameter semantics. All three controls therefore underperform the full method. Parameter semantics contribute the most, while parsed diagnostics provide a larger benefit than cross-trial memory. These results indicate that the advantage of LDST comes from combining knowledge of parameter semantics with diagnostic evidence from solver logs.

Parameter semantics also help maintain a productive search. Without parameter names and defaults, LLM-INDEXONLY exhausts the re-query limit before trial 30 on 6 of the 76 instances, five times by proposing a previously evaluated configuration and once by returning no edits. LLM-STATICCONTEXT, which lacks the archive and the deduplication list, terminates early on one instance, whereas LDST completes all trials on every instance. For early-terminated runs, the last best-so-far loss is carried forward to $b_i(K)$.

Table 4 Mechanism isolation in Exp. B on the 76 Case 1 instances.

Method	Scalar history	Parsed diagnostics	Parameter semantics	Best loss ↓	$L_{\leq 5}$ ↓	$H_{0.8}$ (%) ↑
LDST	✓	✓	✓	0.308	0.485	90.8
LLM-STATICCONTEXT	✗	✓	✓	0.342	0.509	86.8
LLM-OUTCOMEONLY	✓	✗	✓	0.358	0.520	84.2
LLM-INDEXONLY	✓	✓	✗	0.523	0.707	76.3

Notes. The access columns indicate whether scalar history, parsed diagnostics, and parameter semantics are retained relative to LDST. Index-level domain hints in LLM-INDEXONLY are not counted as parameter semantics. Let $b_i(t) := \min_{0 \leq s \leq t} L_i(p_i^{(s)})$, and let GM denote the geometric mean. Best loss is $\text{GM}_{i \in \mathcal{I}} b_i(K)$, early aggregate loss is $L_{\leq 5} := \text{GM}_{i \in \mathcal{I}} b_i(5)$, and the threshold-hit rate is $H_{0.8} := 100|\mathcal{I}|^{-1} \sum_{i \in \mathcal{I}} \mathbf{1}\{b_i(K) \leq 0.8\}$. For early-terminated runs, the last best-so-far loss is carried forward.

5.4. Exp. C: Within-Family Transfer from Single and Multiple Training Instances

C1: Transfer from a single anchor. C1 examines whether a configuration tuned on a single instance transfers to other instances within the same benchmark family. We use the 55 Case 1 instances from CVRP, GAP, JSP, and UFL as anchors. For each anchor and method, we select the configuration obtained in Exp. A and evaluate it, without further tuning, on the remaining Case 1 instances from the same family. LDST improves upon the default configuration on all 55 anchors, whereas SMAC3, RBFOPT, and RANDOM

select the default on 4, 3, and 4 anchors, respectively. Table 5 shows that LDST achieves the lowest anchor and test losses overall and the lowest test loss in every family. Its aggregate test loss is 0.499, compared with 0.798–0.838 for the baselines. Paired sign tests across the 55 anchors further support this advantage, with LDST outperforming SMAC3, RBFOPT, and RANDOM on 48, 49, and 54 anchors, respectively (all $p \leq 1.4 \times 10^{-8}$).

Table 5 C1 one-instance transfer on the 55 Case 1 anchors.

Method	Overall			Family-level Case 1 test			
	Case 1 train	Case 1 test	Test–train	CVRP	GAP	JSP	UFL
LDST	0.238	0.499	0.261	1.190	0.734	0.262	0.385
SMAC3	0.429	0.799	0.370	2.067	0.966	0.657	0.565
RBFOPT	0.427	0.798	0.370	1.978	1.078	0.716	0.547
RANDOM	0.449	0.838	0.390	1.866	1.313	0.635	0.601

Notes. Each anchor configuration is selected as the best among the default configuration and the 50 proposals in Exp. A for that anchor. Entries are geometric means over anchor splits. Test–train denotes $L_{\text{test}} - L_{\text{train}}$. The last four columns report Case 1 test loss separately for CVRP, GAP, JSP, and UFL.

C2: Multi-instance tuning versus independent selection. C2 compares two ways to use multiple training instances from the same family, independent selection and joint tuning. For each eligible benchmark set s and training size $m \in \{2, 3, 4, 5, 6\}$, we construct five deterministic Case 1 train/test splits, provided that s contains at least $m + 4$ Case 1 instances, leaving at least four instances for testing. Independent selection uses the configurations obtained from the individual Exp. A runs and selects the one with the lowest geometric-mean loss across the training subset. Joint tuning instead runs LDST for 50 proposals on the full training subset. The selected configurations are then evaluated on the Case 1 test instances and, when available, on Case 2 and Case 3 instances from the same family.

Table 6 compares the two strategies. With two or three training instances, joint tuning achieves lower Case 1 training and test losses and lower Case 2 and Case 3 losses. Its stronger fit to the training subset increases the train–test gap from 0.150 to 0.226 at $m = 2$ and from 0.099 to 0.143 at $m = 3$, even as the test loss decreases. At $m = 4$, the two strategies achieve nearly identical Case 1 test losses (0.560 for independent selection versus 0.561 for joint tuning), although joint tuning still yields lower training loss and lower losses in Case 2 and Case 3. For larger training subsets, the relative performance depends on the benchmark family. Joint tuning outperforms independent selection on CVRP at every training size and on GAP at both eligible sizes, whereas independent selection performs better on UFL. Consequently, the pooled averages depend on the composition of eligible benchmark families at each training size (Appendix H).

Table 6 Within-family transfer under independent and joint tuning in C2.

m	Case 1 train		Case 1 test		Test–train ↓		Case 2		Case 3	
	Indep.	Joint	Indep.	Joint	Indep.	Joint	Indep.	Joint	Indep.	Joint
2	0.390	0.282	0.540	0.508	0.150	0.226	0.919	0.849	1.010	1.002
3	0.396	0.325	0.495	0.468	0.099	0.143	0.864	0.828	1.018	1.011
4	0.421	0.395	0.560	0.561	0.139	0.166	0.715	0.689	1.025	1.007
5	0.389	0.435	0.543	0.625	0.154	0.190	0.687	0.775	1.030	1.008
6	0.467	0.376	0.474	0.508	0.007	0.132	0.705	0.720	1.009	1.008

Notes. Each metric is reported for both Independent and Joint tuning. Test–train denotes $L_{\text{test}} - L_{\text{train}}$. At $m = 2, 3$, Case 1 includes all four benchmark families, Case 2 includes CVRP, GAP, and JSP, and Case 3 includes only CVRP. At $m = 4, 5, 6$, Case 1 includes CVRP and UFL, while Cases 2 and 3 include only CVRP.

6. Conclusion

We introduced LDST, which uses solver-log diagnostics and an LLM to propose validated sparse edits to the current best configuration. The framework combines memory centered on that configuration, a regime-adaptive loss, and a Case 1 completion-time signal based on branch-and-bound tree growth. Across 215 MIP instances, LDST achieves final geometric-mean losses of 0.274 and 0.557 in the easy and intermediate regimes, respectively, 43.9% and 16.7% below the best scalar-feedback baselines. In both regimes, LDST achieves with 10 proposals a lower loss than any baseline attains with 50. On Case 1, the mechanism study attributes the gain to parameter semantics combined with log diagnostics, and configurations that LDST tunes on a single instance transfer better than baseline configurations to unseen instances of the same family. Semantic reasoning over these diagnostics can make MIP configuration substantially more sample efficient. The reliance on ordinary logs and parameter names suggests extending LDST to other solvers, larger configuration spaces, and other language models.

References

- Achterberg T (2009) SCIP: Solving constraint integer programs. *Mathematical Programming Computation* 1(1):1–41.
- Adams J, Balas E, Zawack D (1988) The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science* 34(3):391–401.
- Anderson D, Le Bodic P, Morgan K (2021) Further results on an abstract model for branching and its application to mixed integer programming. *Mathematical Programming* 190(1-2):811–841.
- Ansótegui C, Sellmann M, Tierney K (2009) A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. Gent IP, ed., *Principles and Practice of Constraint Programming - CP 2009*, 142–157 (Berlin, Heidelberg: Springer).
- Applegate D, Cook W (1991) A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing* 3(2):149–156.
- Augerat P, Naddef D, Belenguer JM, Benavent E, Corberan A, Rinaldi G (1995) Computational results with a branch and cut code for the capacitated vehicle routing problem. Research Report 949-M, Université Joseph Fourier, Grenoble, France.
- Baldacci R, Hadjiconstantinou E, Mingozzi A (2004) An Exact Algorithm for the Capacitated Vehicle Routing Problem Based on a Two-Commodity Network Flow Formulation. *Operations Research* 52(5):723–738.
- Beasley JE (1990) OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society* 41(11):1069–1072.
- Berthold T (2013) Measuring the impact of primal heuristics. *Operations Research Letters* 41(6):611–614.
- Bolusani S, et al. (2024) The SCIP optimization suite 9.0. arXiv preprint arXiv:2402.17702.
- Christofides N, Eilon S (1969) An Algorithm for the Vehicle-dispatching Problem. *Operational Research Quarterly* 20(3):309–318.
- Chu PC, Beasley JE (1997) A genetic algorithm for the generalised assignment problem. *Computers & Operations Research* 24(1):17–23.
- Costa A, Nannicini G (2018) RBFOpt: An open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation* 10(4):597–629.
- Custode LL, Caraffini F, Yaman A, Iacca G (2024) An investigation on the use of large language models for hyperparameter tuning in evolutionary algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1838–1845 (ACM).
- Eggensperger K, Lindauer M, Hutter F (2019) Pitfalls and Best Practices in Algorithm Configuration. *Journal of Artificial Intelligence Research* 64:861–893.
- Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. Muth JF, Thompson GL, eds., *Industrial Scheduling*, 225–251 (Englewood Cliffs, NJ: Prentice-Hall).

- Fisher ML (1994) Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees. *Operations Research* 42(4):626–642.
- Gasse M, Chételat D, Ferroni N, Charlin L, Lodi A (2019) Exact Combinatorial Optimization with Graph Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, volume 32.
- Gleixner A, Hendel G, Gamrath G, Achterberg T, Bastubbe M, Berthold T, Christophel P, Jarck K, Koch T, Linderoth J, Lübbecke M, Mittelmann HD, Ozyurt D, Ralphs TK, Salvagnin D, Shinano Y (2021) MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation* 13(3):443–490.
- Gupta P, Gasse M, Khalil EB, Kumar MP, Lodi A, Bengio Y (2020) Hybrid Models for Learning to Branch. *Advances in Neural Information Processing Systems*, volume 33.
- Hendel G, Anderson D, Le Bodic P, Pfetsch ME (2022) Estimating the Size of Branch-and-Bound Trees. *INFORMS Journal on Computing* 34(2):934–952.
- Hoefer M (2018) UflLib: Library of (uncapacitated) facility location instances. Max-Planck-Institut für Informatik benchmark repository.
- Hoos HH (2012) Automated Algorithm Configuration and Parameter Tuning. Hamadi Y, Monfroy E, Saubion F, eds., *Autonomous Search*, 37–71 (Berlin, Heidelberg: Springer).
- Hosny A, Reda S (2023) Automatic MILP Solver Configuration By Learning Problem Similarities. arXiv preprint arXiv:2307.00670.
- Hutter F, Hoos HH, Leyton-Brown K (2010) Automated Configuration of Mixed Integer Programming Solvers. Lodi A, Milano M, Toth P, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, 186–202 (Berlin, Heidelberg: Springer).
- Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential Model-Based Optimization for General Algorithm Configuration. Coello CAC, ed., *Learning and Intelligent Optimization*, 507–523 (Berlin, Heidelberg: Springer).
- Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36:267–306.
- Hutter F, López-Ibáñez M, Fawcett C, Lindauer M, Hoos HH, Leyton-Brown K, Stützle T (2014) AClib: A Benchmark Library for Algorithm Configuration. Pardalos PM, Resende MG, Vogiatzis C, Walteros JL, eds., *Learning and Intelligent Optimization*, 36–40 (Cham: Springer International Publishing).
- Kadioglu S, Malitsky Y, Sellmann M, Tierney K (2010) ISAC – Instance-Specific Algorithm Configuration. *ECAI 2010*, 751–756 (IOS Press).
- Kendall MG (1938) A new measure of rank correlation. *Biometrika* 30(1-2):81–93.
- Kendall MG (1945) The treatment of ties in ranking problems. *Biometrika* 33(3):239–251.
- Kuehn AA, Hamburger MJ (1963) A Heuristic Program for Locating Warehouses. *Management Science* 9(4):643–666.
- Lawless C, Li Y, Wikum A, Udell M, Vitercik E (2025) LLMs for Cold-Start Cutting Plane Separator Configuration. arXiv preprint arXiv:2412.12038.
- Lawrence S (1984) Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Technical Report (Supplement), Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.
- Le Bodic P, Nemhauser G (2017) An abstract model for branching and its application to mixed integer programming. *Mathematical Programming* 166(1):369–405.
- Lima I, Uchoa E, Pecin D, Pessoa A, Poggi M, Vidal T, Subramanian A (2014) CVRPLIB: Capacitated vehicle routing problem library. Website (Galgos / DI PUC-Rio and collaborators).
- Lindauer M, Eggenberger K, Feurer M, Biedenkapp A, Deng D, Benjamins C, Ruhopf T, Sass R, Hutter F (2022) SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23(54):1–9.

- Liu R, He X, Pan Y, Umenberger J (2026) Online Algorithm Configuration for MILP Re-Optimization with LLM Guidance. Submitted to ICLR 2026, OpenReview, <https://openreview.net/forum?id=xbyebs1ZF>.
- Liu S, Gao C, Li Y (2024a) Large language model agent for hyper-parameter optimization. arXiv preprint arXiv:2402.01881.
- Liu T, Astorga N, Seedat N, van der Schaar M (2024b) Large Language Models to Enhance Bayesian Optimization. *International Conference on Learning Representations*.
- López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3:43–58.
- Luo Y, Chen X, Wang C, Zhu F, Zhong T, Yu T (2026) GRIMIP: A General Framework for Instance-Specific Configuration of MIP Solvers Using LLMs. ArXiv:2606.23299 [cs.LG].
- Manne AS (1960) On the Job-Shop Scheduling Problem. *Operations Research* 8(2):219–223.
- Naphade O, Bansal S, Pareek P (2025) Small LLMs with expert blocks are good enough for hyperparameter tuning. arXiv preprint arXiv:2509.15561.
- Queiroga E, Sadykov R, Uchoa E, Vidal T (2022) 10,000 optimal CVRP solutions for testing machine learning based heuristics. *AAAI-22 Workshop on Machine Learning for Operations Research (MLAOR)*.
- Romera-Paredes B, Barekatin M, Novikov A, Balog M, Kumar MP, Dupont E, Ruiz FJR, Ellenberg JS, Wang P, Fawzi O, Kohli P, Fawzi A (2024) Mathematical discoveries from program search with large language models. *Nature* 625(7995):468–475.
- Ross GT, Soland RM (1975) A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming* 8(1):91–103.
- Schede E, Brandt J, Tornede A, Wever M, Bengs V, Hüllermeier E, Tierney K (2022) A Survey of Methods for Automated Algorithm Configuration. *Journal of Artificial Intelligence Research* 75:425–487.
- Song W, Liu Y, Cao Z, Wu Y, Li Q (2023) Instance-specific algorithm configuration via unsupervised deep graph clustering. *Engineering Applications of Artificial Intelligence* 125:106740.
- Tang Y, Agrawal S, Faenza Y (2020) Reinforcement Learning for Integer Programming: Learning to Cut. *Proceedings of the 37th International Conference on Machine Learning*, 9367–9376.
- Wei A, Nie A, Teixeira TSFX, Yadav R, Lee W, Wang K, Aiken A (2025) Improving parallel program performance with LLM optimizers via agent-system interfaces. *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 66155–66177 (PMLR).
- Xu L, Hutter F, Hoos H, Leyton-Brown K (2011) Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. *RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)*.
- Yang C, Wang X, Lu Y, Liu H, Le QV, Zhou D, Chen X (2024) Large Language Models as Optimizers. *International Conference on Learning Representations*.
- Zhang MR, Desai N, Bae J, Lorraine J, Ba J (2023) Using Large Language Models for Hyperparameter Optimization. arXiv preprint arXiv:2312.04528.
- Zheng Z, Gu H, Shao J (2026) Large Language Model Guided Dynamic Branching Rule Scheduling in Branch-and-Bound. Submitted to ICLR 2026, OpenReview, <https://openreview.net/forum?id=8LCdjf7uIk>.

Appendix Roadmap

Appendix A describes the benchmark sources, MPS-construction protocol, default-based regime assignment, and within-family split design used in the main experiments. Appendix B specifies the shared execution contract, scalar-feedback baselines, and Exp. B controls. Appendices C and D document the configuration space and default hyperparameters, while Appendices E and F describe the fixed log parser and prompt construction used by LDST. Appendix G validates the Case 1 completion-time estimator $\hat{T}_\infty$. Appendix H

provides the edit sizes of Exp. A, additional Exp. C results, and an audit of the Case 3 outlier. Appendix I documents the code and data release, licensing information, hardware and software environment, and token-side compute accounting. Finally, Appendix J provides the complete family-specific formulations and benchmark manifests.

A. Benchmark Sources, Formulations, and Regime Assignment

This appendix describes the benchmark design underlying the main experiments, including the source collections, MPS-construction protocol, default-based regime assignment, and within-family split design used in Exp. C. Appendix J provides the complete family-specific formulations and benchmark manifests.

We use five public benchmark sets, CVRPLIB CVRP (Lima et al. 2014), OR-Library GAP and JSP (Beasley 1990), UfLib UFL (Hofer 2018), and MIPLIB 2017 (Gleixner et al. 2021). The final benchmark comprises 215 instances, including 90 CVRP, 12 GAP, 24 JSP, 30 UFL, and 59 MIPLIB instances. For the four domain-specific families, we export standard mixed-integer linear programming (MILP) formulations to .mps files through an automated pipeline. Each generated file is checked against its native model for variable and constraint counts, integrality patterns, and objective agreement within numerical tolerances. For MIPLIB, we initially select 60 instances from the official benchmark-v2 subset using a fixed random seed. We exclude k1mushroom because it does not complete presolving under default SCIP settings within the 600-second time limit, leaving 59 MIPLIB instances for the reported experiments. The code repository provides the final manifests, file-level provenance, and regime labels (Appendix I).

A.1. Default-Based Regime Assignment

Each instance is assigned to a case based on its outcome under the default configuration p^{def} with a common time limit of $\tau = 600\text{s}$, following the definitions in Section 3. Case 1 contains instances solved under p^{def} , including those proved optimal and the single instance `miplib_neos859080`, which is proved infeasible. Case 2 contains instances with a feasible primal solution but no proof of optimality, while Case 3 contains the remaining hard instances evaluated using the dual-progress objective. These assignments remain fixed across all subsequent experiments and are not recomputed after tuning. This ensures consistent regime definitions across methods and prevents tuned configurations from changing the evaluation objective.

Table 7 summarizes the source collections, construction procedures, and default-based regime counts for each benchmark family.

Table 7 Benchmark sources, construction routes, and regime counts.

Family	Source collection	Construction route	#	Case 1	Case 2	Case 3
CVRP	CVRPLIB	two-commodity flow MILP export	90	11	39	40
GAP	OR-Library gapc/gapd	assignment MILP export	12	7	5	0
JSP	OR-Library jobshop1	disjunctive MILP export	24	7	17	0
UFL	UfLib Uniform	facility-location MILP export	30	30	0	0
MIPLIB	MIPLIB 2017 benchmark-v2	direct MPS benchmark instances	59	21	30	8

Notes. Counts cover the 215-instance benchmark. Regime assignments are made under the default configuration and remain fixed.

Table 8 gives the complete per-instance case-assignment manifest used by all experiments. Table 7 summarizes the corresponding counts.

A.2. Within-Family Split Construction

Exp. C uses deterministic within-family train/test splits constructed from the Case 1 instances of CVRP, GAP, JSP, and UFL. MIPLIB is excluded because it contains heterogeneous general-purpose instances rather than a single problem family. For each family s and training size m , a split is eligible only if at least four Case 1

Table 8 Case assignments for all benchmark instances.

Family	Instance	Case	Family	Instance	Case
CVRP	A-n32-k5	2	CVRP	A-n33-k5	2
CVRP	A-n33-k6	2	CVRP	A-n34-k5	2
CVRP	A-n36-k5	2	CVRP	A-n37-k5	2
CVRP	A-n37-k6	2	CVRP	A-n38-k5	2
CVRP	A-n39-k5	2	CVRP	A-n39-k6	2
CVRP	A-n44-k6	2	CVRP	A-n45-k6	3
CVRP	A-n45-k7	2	CVRP	A-n46-k7	2
CVRP	A-n48-k7	2	CVRP	A-n53-k7	2
CVRP	A-n54-k7	3	CVRP	A-n55-k9	3
CVRP	A-n60-k9	3	CVRP	A-n61-k9	3
CVRP	A-n62-k8	3	CVRP	A-n63-k10	3
CVRP	A-n63-k9	3	CVRP	A-n64-k9	3
CVRP	A-n65-k9	3	CVRP	A-n69-k9	3
CVRP	A-n80-k10	3	CVRP	B-n31-k5	2
CVRP	B-n34-k5	2	CVRP	B-n35-k5	2
CVRP	B-n38-k6	2	CVRP	B-n39-k5	2
CVRP	B-n41-k6	2	CVRP	B-n43-k6	2
CVRP	B-n44-k7	2	CVRP	B-n45-k5	2
CVRP	B-n45-k6	3	CVRP	B-n50-k7	2
CVRP	B-n50-k8	2	CVRP	B-n51-k7	3
CVRP	B-n52-k7	2	CVRP	B-n56-k7	2
CVRP	B-n57-k7	3	CVRP	B-n57-k9	3
CVRP	B-n63-k10	3	CVRP	B-n64-k9	3
CVRP	B-n66-k9	3	CVRP	B-n67-k10	3
CVRP	B-n68-k9	3	CVRP	B-n78-k10	3
CVRP	E-n101-k14	3	CVRP	E-n101-k8	3
CVRP	E-n13-k4	1	CVRP	E-n22-k4	1
CVRP	E-n23-k3	1	CVRP	E-n30-k3	2
CVRP	E-n31-k7	1	CVRP	E-n33-k4	2
CVRP	E-n51-k5	2	CVRP	E-n76-k10	3
CVRP	E-n76-k14	3	CVRP	E-n76-k7	3
CVRP	E-n76-k8	3	CVRP	F-n135-k7	3
CVRP	F-n45-k4	2	CVRP	F-n72-k4	2
CVRP	P-n101-k4	2	CVRP	P-n16-k8	1
CVRP	P-n19-k2	1	CVRP	P-n20-k2	1
CVRP	P-n21-k2	1	CVRP	P-n22-k2	1
CVRP	P-n22-k8	1	CVRP	P-n23-k8	2
CVRP	P-n40-k5	1	CVRP	P-n45-k5	2
CVRP	P-n50-k10	3	CVRP	P-n50-k7	2
CVRP	P-n50-k8	3	CVRP	P-n51-k10	3
CVRP	P-n55-k10	3	CVRP	P-n55-k15	3
CVRP	P-n55-k7	2	CVRP	P-n55-k8	2
CVRP	P-n60-k10	3	CVRP	P-n60-k15	3
CVRP	P-n65-k10	3	CVRP	P-n70-k10	3
CVRP	P-n76-k4	3	CVRP	P-n76-k5	3
GAP	gapc-1	1	GAP	gapc-2	1
GAP	gapc-3	1	GAP	gapc-4	1
GAP	gapc-5	1	GAP	gapc-6	1
GAP	gapd-1	1	GAP	gapd-2	2
GAP	gapd-3	2	GAP	gapd-4	2
GAP	gapd-5	2	GAP	gapd-6	2
JSP	abz5	2	JSP	abz6	1
JSP	abz7	2	JSP	abz8	2
JSP	abz9	2	JSP	ft10	2

Notes. Each instance is assigned to a case once under the default configuration p^{def} with time limit $\tau = 600$ s. These assignments remain fixed across all experiments.

Table 8 Case assignments for all benchmark instances (continued).

Family	Instance	Case	Family	Instance	Case
JSP	ft20	2	JSP	1a01	1
JSP	1a02	1	JSP	1a03	1
JSP	1a04	1	JSP	1a05	1
JSP	1a06	2	JSP	1a07	2
JSP	1a08	2	JSP	1a09	2
JSP	1a10	2	JSP	1a11	2
JSP	1a12	2	JSP	orb01	2
JSP	orb02	1	JSP	orb03	2
JSP	orb04	2	JSP	orb05	2
MIPLIB	50v-10	2	MIPLIB	CMS750_4	2
MIPLIB	assign1-5-8	2	MIPLIB	atlanta-ip	2
MIPLIB	b1c1s1	2	MIPLIB	beasleyC3	1
MIPLIB	cbs-cta	1	MIPLIB	chromaticindex512-7	2
MIPLIB	cmflsp50-24-8-8	3	MIPLIB	cod105	1
MIPLIB	comp21-2idx	2	MIPLIB	cryptanalysis	3
MIPLIB	dano3_3	1	MIPLIB	kb128n5obj16	3
MIPLIB	dws008-01	2	MIPLIB	drayage-25-23	1
MIPLIB	gfd-schedule	3	MIPLIB	fhnw-binpack4-48	3
MIPLIB	n180f7d50m30k18	3	MIPLIB	glass-sc	2
MIPLIB	glass4	2	MIPLIB	gmu-35-50	2
MIPLIB	h80x6320d	1	MIPLIB	irish-electricity	3
MIPLIB	lectsched-5-obj	2	MIPLIB	momentum1	2
MIPLIB	mushroom-best	2	MIPLIB	mzzv11	1
MIPLIB	n3div36	2	MIPLIB	neos-1354092	3
MIPLIB	neos-1445765	1	MIPLIB	neos-3046615-murg	2
MIPLIB	neos-3083819-nubu	1	MIPLIB	neos-3627168-kasai	1
MIPLIB	neos-3988577-wolgan	3	MIPLIB	neos-5052403-cygnet	2
MIPLIB	neos-662469	2	MIPLIB	neos-827175	1
MIPLIB	neos-911970	1	MIPLIB	neos8	1
MIPLIB	neos859080	1	MIPLIB	nexp-150-20-8-5	2
MIPLIB	nursesched-sprint02	1	MIPLIB	p200x1188c	1
MIPLIB	piperout-27	1	MIPLIB	radiationm18-12-05	2
MIPLIB	radiationm40-10-02	2	MIPLIB	ran14x18-disj-8	2
MIPLIB	rocI-4-11	1	MIPLIB	rocII-5-11	2
MIPLIB	roll3000	1	MIPLIB	s100	2
MIPLIB	snp-02-004-104	2	MIPLIB	supportcase22	3
MIPLIB	supportcase40	2	MIPLIB	supportcase42	2
MIPLIB	swath1	1	MIPLIB	thor50dday	2
MIPLIB	traininstance2	2	MIPLIB	uct-subprob	2
MIPLIB	unitcal_7	1	UFL	1023UnifS	1
UFL	1123UnifS	1	UFL	1223UnifS	1
UFL	123UnifS	1	UFL	1323UnifS	1
UFL	1423UnifS	1	UFL	1523UnifS	1
UFL	1623UnifS	1	UFL	1723UnifS	1
UFL	1823UnifS	1	UFL	1923UnifS	1
UFL	2023UnifS	1	UFL	2123UnifS	1
UFL	2223UnifS	1	UFL	223UnifS	1
UFL	2323UnifS	1	UFL	2423UnifS	1
UFL	2523UnifS	1	UFL	2623UnifS	1
UFL	2723UnifS	1	UFL	2823UnifS	1
UFL	2923UnifS	1	UFL	3023UnifS	1
UFL	323UnifS	1	UFL	423UnifS	1
UFL	523UnifS	1	UFL	623UnifS	1
UFL	723UnifS	1	UFL	823UnifS	1
UFL	923UnifS	1			

Notes. Each instance is assigned to a case once under the default configuration p^{def} with time limit $\tau = 600$ s. These assignments remain fixed across all experiments.

instances remain for testing. We generate five distinct training subsets for each eligible (s, m) pair using the manifest order and a fixed seed key family $|m|42$. The corresponding test set consists of all remaining Case 1 instances in the same family. When available, Case 2 and Case 3 instances from that family are also used to evaluate transfer performance, but their logs are never exposed during tuning. Table 9 summarizes the eligible splits and test sets. This design enables within-family transfer comparisons under matched solver budgets while preventing information leakage from test-instance logs into the tuning process.

Table 9 Deterministic within-family split design for Exp. C.

Family	Case 1 pool	C2 train sizes m	Splits per m	Case 1 test count	Other transfer targets
CVRP	11	2, 3, 4, 5, 6	5	$11 - m$	Case 2 and Case 3
GAP	7	2, 3	5	$7 - m$	Case 2 only
JSP	7	2, 3	5	$7 - m$	Case 2 only
UFL	30	2, 3, 4, 5, 6	5	$30 - m$	none beyond Case 1

Notes. All split manifests are generated from the fixed case-assignment manifest using base seed 42. Each eligible split leaves at least four Case 1 instances for testing. C1 uses every Case 1 instance from the listed families as a single anchor.

B. Experimental Protocol, Baselines, and Shared Execution Contract

This appendix documents the experimental protocol for method comparisons and the controls used in Exp. B. Table 10 specifies the execution conditions shared across the tuning experiments. The following subsections describe the method-specific feedback channels, search-space representations, and prompt modifications. Appendix F provides representative prompts and outputs. Figure 5 illustrates the prompt components used by LDST, Figure 6 presents a representative JSON response, and Table 16 identifies the components retained or removed in each Exp. B control.

B.1. Shared Execution Contract

Table 10 defines the common execution contract for all reported tuning comparisons, including solver calls, objective accounting, the editable parameter space, advice validation, and randomness controls. These conditions are fixed independently of the method-specific tuning policy.

B.2. Black-Box Scalar-Feedback Baselines

The scalar-feedback baselines receive only the loss returned by the regime-adaptive objective. They have no access to raw solver logs, parsed diagnostics, formulation text, prompt memory, or LLM-side validation feedback. Table 11 summarizes each method’s search-space encoding, feedback channel, and handling of invalid configurations under the shared execution contract in Table 10.

B.3. Prompt-Based LLM Controls

Let Π^{LDST} denote the full LDST prompt. It comprises (i) a static header specifying the objective, parameter whitelist, current best configuration, edit cap, and output schema, (ii) optional formulation text, (iii) validator feedback during re-queries, (iv) archive memory, (v) a deduplication list recording previously evaluated configurations, and (vi) a diagnostic-feature guide linking parsed log fields to potential solver bottlenecks. Figure 3 presents a cropped prompt screenshot from an actual Exp. A run, highlighting the components relevant to the Exp. B controls. The three LLM controls selectively remove or restrict information available to the decision policy while preserving the same outer tuning loop.

Table 10 Shared execution contract for the reported tuning comparisons.

Component	Contract
Solver call	SCIP 9.2.4, one thread, one blocking solver run per evaluated configuration and instance, with per-run time limit $\tau = 600$ s.
Objective and normalization	The regime-adaptive loss in Section 3 is fixed before comparison and used identically by LDST, LLM controls, and scalar-feedback baselines; each method’s own default run (trial 0) defines its normalization reference.
Reported selection	For every method, the reported configuration after t trials is the best-so-far configuration over trial 0 (the default) and proposals $1, \dots, t$.
Evaluation budget	Each tuning trial consumes one sequential solver evaluation per active instance. Experiment-specific values of K are stated in the corresponding experiment descriptions and split manifests.
Editable space	All methods search the same curated whitelist $\mathcal{W}$ of 45 SCIP parameters in Appendix C; numeric entries use the native linear scale.
LLM proposal contract	LLM-based methods use edit cap $M = 20$, edits around the current best configuration, the same JSON schema, the same re-query limit $V = 3$, and the same archive novelty check.
Validation	Candidate edits must pass schema validation, whitelist/domain checks, edit-cap truncation, archive-novelty filtering, and successful loading as a SCIP settings file before a solver call.
Randomness control	Solver calls fix randomization/randomseedshift=42; stochastic scalar-feedback optimizers use optimizer seed 42.

Table 11 Implementation details of the scalar-feedback baselines.

Method	Search-space encoding	Feedback	Settings and invalid handling
SMAC3	Typed CONFIGSPACE over the whitelist; native linear numeric domains.	scalar loss	Deterministic, seed 42, K sequential evaluations; non-finite values noncompetitive.
RBFOPT	Native float/int bounds; Boolean and categorical dimensions integer-encoded.	scalar loss	Seed 42, no local search, genetic global search, K sequential evaluations; non-finite values noncompetitive.
RANDOM	Independent typed sampling; positive real intervals log-uniform, all other numeric intervals uniform.	scalar loss	Seed 42, K sequential samples, no surrogate; the default enters only through trial 0.

Notes. All three methods optimize only the scalar loss returned by the current case objective.

LLM-StaticContext. LLM-STATICCONTEXT retains the static components of Π^{LDST} and the parsed diagnostics of the current best run. These include whitelist parameter names, defaults, and domains, edits defining the current best configuration, task and edit constraints, optional formulation text, and a single block of current diagnostics. In Figure 3, this control retains regions A and B but removes the cross-trial archive in C, the diagnostic guide, and the deduplication list. Thus, the LLM receives information about the current best configuration and its solver behavior without access to the history of previous trials.

LLM-OutcomeOnly. LLM-OUTCOMEONLY replaces the full archive in Π^{LDST} with a scalar-only memory $\mathcal{M}^{\text{SCALAR}}$ containing the trial id, loss, solve time, final gap, and finite/feasible status whenever these quantities are defined. In Figure 3, this control retains region A and the scalar feedback in region C but removes the parsed diagnostic fields and the diagnostic guide. It therefore preserves outcome feedback across trials while removing process-level log structure.

LLM-IndexOnly. LLM-INDEXONLY retains the archive, log-derived diagnostics, diagnostic guide, and deduplication list but restricts the semantic parameter context. Relative to Figure 3, it retains region C while replacing the parameter names, groups, and defaults in region A with editable parameter indices, per-index domain hints, and the indices corresponding to Boolean parameters. These

Figure 3 Archived prompt excerpt for CVRP instance A-n37-k6 after trial 2.

```

=== LDST Prompt ===
You are tuning SCIP parameters for a mixed-integer programming (MIP) solver.
Objective: min loss (intuition: ~ min primal-dual integral) (loss is normalized)
Current best loss: 0.2931
Edit cap M: 20
Whitelist names: branching: scorefunc(1;p), preferbinary(2;false), relpscost/{minreliable(3;1), maxreliable(4;5), sbiterquot(5;0.50), sbiterofs(6;100000)}
| nodeselection: estimate/stdpriority(7;200000), dfs/stdpriority(8;0), bfs/stdpriority(9;100000), childsel(10;h) | separating: maxrounds(11;-1),
maxroundsroot(12;-1), maxcuts(13;100), maxcutsroot(14;2000), gomory/freq(15;10), cmir/freq(16;10), flowcover/freq(17;10), clique/freq(18;0),
knapsackcover/freq(19;10), oddcycle/freq(20;-1) | presolving: maxrounds(21;-1), maxrestarts(22;-1), abortfac(23;0.00) | heuristics: feaspump/{freq(24;20
freqofs(25;0), maxdepth(26;-1), maxlpiterquot(27;0.01), maxlpiterofs(28;1000), beforecuts(29;true)}, rins/{nodesofs(30;500), nodesquot(31;0.30),
minnodes(32;50), maxnodes(33;5000), nwaitingnodes(34;200), minfixingrate(35;0.30)}, localbranching/{neighborhoodsize(36;18), nodesofs(37;1000),
nodesquot(38;0.05), lplimfac(39;1.50)}, rens/{nodesofs(40;500), nodesquot(41;0.10), minnodes(42;50), maxnodes(43;5000), minfixingrate(44;0.50),
startsols(45;l)}
Note: format is name(index;default).
Boolean params must use true/false (not 0/1).
Bool params: branching/preferbinary(2), heuristics/feaspump/beforecuts(29)
Tuning set: 1 instance (single-instance); loss = normalized single-instance loss; diagnostics: raw per-run values

Incumbent edits (delta based on defaults):
{ (11)=5, (12)=500, (13)=500, (14)=50000, (15)=-1, (16)=-1, (17)=-1, (18)=-1, (19)=-1 }

Problem formulation
We adopt the two-commodity network-flow formulation of \cit{aldacci2004exact}, which is a compact mixed-integer model that can be solved to optimality
without column or row generation:
\begin{equation*}
\begin{aligned}
&\min \sum_{i,j \in V} c_{ij} x_{ij} \\
&\text{s.t.} \\
&\sum_{j \in V} y_{ji} - \sum_{i \in V} y_{ij} = 2d_i, \\
&\forall i \in V \\
&\sum_{j \in V} y_{ji} = d_i, \quad \forall i \in V \\
&\sum_{j \in V} y_{ij} = KQ - d_i, \quad \forall i \in V \\
&\sum_{j \in V} y_{ij} + \sum_{i \in V} y_{ji} = Q, \quad \forall i \in V \\
&\forall i,j \in V \\
&\sum_{i,j \in V} x_{ij} \\
&+ \sum_{i,j \in V} x_{ji} = 2, \\
&\forall i,j \in V \\
&y_{ij} \geq 0, \quad \forall i,j \in V \\
&x_{ij} \in \{0,1\}, \quad \forall i,j \in V
\end{aligned}
\end{equation*}
Here,  $V$  is a complete undirected graph with two depot copies, where  $V = \{0,1,\dots,n,n+1\}$  and the
customer set is  $C = V \setminus \{0,n+1\}$ . The edge set is
 $E = \{i,j \in V : i,j \in C, i < j\} \cup \{i,n+1\} : i \in C\}$ .
The travel cost on edge  $(i,j) \in E$  is  $c_{ij}$ . Each customer  $i \in C$  has demand  $d_i \geq 0$ , and
 $d_i$ . Vehicles have capacity  $Q$ , and  $K$  denotes the fleet size. Binary variables  $x_{ij}$ 
indicate whether edge  $(i,j) \in E$  is used, while continuous variables  $y_{ij}, y_{ji} \geq 0$  represent the two commodities (vehicle load and residual
capacity) flowing on edge  $(i,j) \in E$ .

[MEMORY] 3 best + 3 recent (after dedup)
cols: trial | loss | delta | solve_time | solving_nodes | nodes_left | max_depth | nsols | num_improvements |
final_primal | final_dual | final_gap | primal_dual_integral | presolve_time | presolved_vars | presolved_cons_max | root_dual_end | root_dual_gain | root_gain_rel | root_first_lp_iters | root_first_lp_time | root_final_root_iters | root_lp_estimate | sb_time | sb_calls |
sb_iters | sb_root_calls | sb_root_iters | cuts_selected_root | sep_gomory_time | sep_gomory_applied | sep_clique_time | sep_clique_applied |
sep_oddcycle_time | sep_oddcycle_applied | sep_cmir_applied | sep_flowcover_applied | sep_knapsackcover_applied | heur_feaspump_time | heur_feaspump_calls |
heur_feaspump_found | heur_feaspump_best | heur_rins_time | heur_rins_calls | heur_rins_found | heur_rins_best | heur_localbranching_time |
heur_localbranching_calls | heur_localbranching_found | heur_localbranching_best | heur_rens_time | heur_rens_calls | heur_rens_found | heur_rens_best |
- 2 | 0.29 | {} | 600 | 298922 | 180760 | 108 | 23 | 12 | 25.25 | 1043 | 964 | 838.07 | 125.93 | 10991.56 | 0.06 | 2106 | 777 | 5689 | 797.49 | 801.25 |
3.76 | 0.00 | 1632 | 0.04 | 3025 | 932.07 | 11.24 | 11653 | 188191 | 34 | 948 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.55 | 19 | 0 | 0 | 1.92 | 63 |
4 | 0 | 0 | 0 | 0 | 0 | 0.02 | 1 | 0 | 0 |
- 1 | 0.92 | {} | (11)=-1, (12)=300, (14)=20000, (15)=0, (16)=1, (17)=1, (19)=1 | 600 | 41473 | 19092 | 75 | 8 | 6 | 304.06 | 1296 | 998 | 876.96 | 121.04 |
34401.07 | 0.06 | 2106 | 777 | 5143 | 797.49 | 865.28 | 67.79 | 0.09 | 1632 | 0.04 | 9595 | 1043.07 | 41.01 | 5342 | 416249 | 34 | 13304 | 975 | 0.03 |
1516 | 0 | 0 | 0 | 0 | 75815 | 7984 | 675 | 2.43 | 52 | 0 | 0 | 2.48 | 16 | 4 | 4 | 0 | 0 | 0 | 0 | 1.11 | 1 | 0 | 0 |
- 0 | 1 | {} | (11)=-1, (12)=-1, (13)=100, (14)=2000, (15)=10, (16)=10, (17)=10, (18)=0, (19)=10 | 600 | 43008 | 15158 | 77 | 10 | 6 | 352.03 | 1029 | 968 |
882.06 | 85.94 | 37502.32 | 0.06 | 2106 | 777 | 4563 | 797.49 | 865.28 | 67.79 | 0.09 | 1632 | 0.04 | 9595 | 1020.89 | 45.25 | 6049 | 443851 | 34 |
13304 | 975 | 5.93 | 19091 | 0 | 0 | 0 | 0 | 66741 | 7229 | 422 | 2.73 | 56 | 0 | 0 | 2.19 | 18 | 5 | 5 | 0 | 0 | 0 | 0 | 1.11 | 1 | 0 | 0

```

Notes. The cropped screenshot is taken from the archived prompt and spans the prompt header through the archive-memory block. Red boxes identify the semantic and static context (A), the incumbent (current best configuration) and formulation context (B), and archive memory containing parsed diagnostics (C).

hints support valid parameter edits without revealing their meanings. The tuning loop, validator, and JSON schema remain unchanged, except that the optional textual whitelist field is omitted from the returned edits. In Exp. B, LLM-INDEXONLY terminates early on gap_gapc-4, gap_gapd-1, jsp_la01, miplib_neos-827175, ufl_1023UnifS, and ufl_2423UnifS after trials 20, 16, 18, 23, 20, and 21, respectively. LLM-STATICCONTEXT also terminates early on miplib_cbs-cta after trial 26.

Formulation and leakage control. All reported LLM-based runs use formulation=auto, which includes the built-in formulation text for CVRP, GAP, JSP, and UFL and none for MIPLIB. The formulation text is a family-level mathematical description retrieved from utilities/formulations.py. It depends only on the benchmark family and is identical for all instances within that family. No formulation text is constructed

using test-instance logs or target-instance solve trajectories. In Exp. C, the LLM receives only logs from the training instances during tuning. Test instances from Cases 1, 2, and 3 are evaluated only after a configuration has been selected, and their logs are never fed back into the tuning prompts. Table 12 summarizes the information available to the black-box baselines, LLM controls, and full LDST framework.

Table 12 Information-access matrix for the main control families.

Method	Scalar hist.	Parsed diag.	Param. sem.	Form. text	Rationale	Validator
BBO baselines	✓	✗	✗	✗	✗	✗
LLM-STATICCONTEXT	✗	✓	✓	optional	✓	✓
LLM-OUTCOMEONLY	✓	✗	✓	optional	✓	✓
LLM-INDEXONLY	✓	✓	✗	optional	✓	✓
LDST	✓	✓	✓	optional	✓	✓

Notes. The BBO row groups SMAC3, RBFOPT, and RANDOM. Column headers abbreviate scalar history, parsed diagnostics, parameter semantics, optional formulation text, visible rationale field, and the prompt-side validator, re-query, and deduplication contract. Black-box baselines sample directly from the typed whitelist and do not use this prompt-side contract. SMAC3 and RBFOPT fit surrogate models to scalar history, whereas RANDOM ignores that history. For LLM-INDEXONLY, index-level domain hints are not counted as parameter semantics because parameter names and defaults are hidden.

C. Configuration Space: SCIP Parameter Whitelist

We use the same curated whitelist $\mathcal{W}$ across all experiments and controls. The whitelist is small enough that edits remain interpretable and semantically checkable, while covering branching, node selection, cut separation, presolving, and primal heuristics, the main solver components reported in SCIP logs. Table 13 provides the complete specification of all 45 parameters used in the implementation, including their SCIP parameter paths, types, default values, allowed domains, and scales. All numeric parameters are edited on their native linear scales, without parameter-specific transformations. In addition to the typed whitelist, the LLM-side validator checks JSON-schema validity, truncates proposals to the edit cap, checks archive novelty, and verifies that the proposed configuration loads as a SCIP settings file.

D. LLM Hyperparameters and Shared Settings

The LLM serves only as a constrained proposal mechanism. Table 14 summarizes the model-side settings not specified in the shared execution contract in Appendix B. Experiment-specific trial counts are provided in the corresponding experiment descriptions and Appendix I.

E. SCIP Log Parser and Diagnostics

For each run k , the parser converts the raw SCIP log $\Lambda^{(k)}$ into a diagnostic vector with a fixed schema,

$$\mathbf{d}^{(k)} = \text{Parse}(\Lambda^{(k)}),$$

which is passed to the prompt builder described in Section 4. The parser requires no access to internal solver states and uses only standard SCIP log sections, including the branch-and-bound progress table, root-node report, presolve summary, heuristic table, separator table, and final solving summary. Figure 4 illustrates how the raw SCIP log is converted into the compact diagnostic state used by LDST.

Table 15 defines the diagnostic quantities used by the prompt builder. Let $\mathcal{R}^{(k)}$ denote the ordered progress-table rows parsed from run $\Lambda^{(k)}$. Each row $r \in \mathcal{R}^{(k)}$ contains elapsed time t_r , processed-node count n_r , primal bound z_r^{pr} , and dual bound z_r^{du} when these fields are available. For any row field x , let $\text{first}_f(x)$ and $\text{last}_f(x)$ denote its first and last finite parsed values, respectively, and set $\eta = 10^{-12}$. Let $\mathcal{S} = \{\text{gomory}, \text{clique}, \text{oddcycle}, \text{cmir}, \text{flowcover}, \text{knapsackcover}\}$ and $\mathcal{H} =$

Table 13 Full typed SCIP parameter whitelist used in the experiments.

SCIP parameter path	Type	Default	Allowed domain	Scale
branching/scorefunc	cat	p	{s, p, q}	lin.
branching/preferbinary	bool	false	{false, true}	lin.
branching/relpscost/minreliable	float	1	[0, 1000000000]	lin.
branching/relpscost/maxreliable	float	5	[0, 1000000000]	lin.
branching/relpscost/sbiterquot	float	0.5	[0, 1000000000]	lin.
branching/relpscost/sbiterofs	int	100000	[0, 1000000000]	lin.
nodeselection/estimate/stdpriority	int	200000	[-536870912, 1000000000]	lin.
nodeselection/dfs/stdpriority	int	0	[-536870912, 1000000000]	lin.
nodeselection/bfs/stdpriority	int	100000	[-536870912, 1000000000]	lin.
nodeselection/childsel	cat	h	{d, u, p, i, l, r, h}	lin.
separating/maxrounds	int	-1	[-1, 1000000000]	lin.
separating/maxroundsroot	int	-1	[-1, 1000000000]	lin.
separating/maxcuts	int	100	[0, 1000000000]	lin.
separating/maxcutsroot	int	2000	[0, 1000000000]	lin.
separating/gomory/freq	int	10	[-1, 1000000000]	lin.
separating/cmirt/freq	int	10	[-1, 1000000000]	lin.
separating/flowcover/freq	int	10	[-1, 1000000000]	lin.
separating/clique/freq	int	0	[-1, 1000000000]	lin.
separating/knapsackcover/freq	int	10	[-1, 1000000000]	lin.
separating/oddcycle/freq	int	-1	[-1, 1000000000]	lin.
presolving/maxrounds	int	-1	[-1, 1000000000]	lin.
presolving/maxrestarts	int	-1	[-1, 1000000000]	lin.
presolving/abortfac	float	0.0008	[0, 1]	lin.
heuristics/feaspump/freq	int	20	[-1, 1000000000]	lin.
heuristics/feaspump/freqofs	int	0	[0, 1000000000]	lin.
heuristics/feaspump/maxdepth	int	-1	[-1, 1000000000]	lin.
heuristics/feaspump/maxlpiterquot	float	0.01	[0, 1000000000]	lin.
heuristics/feaspump/maxlpiterofs	int	1000	[0, 1000000000]	lin.
heuristics/feaspump/beforecuts	bool	true	{false, true}	lin.
heuristics/rins/nodesofs	int	500	[0, 1000000000]	lin.
heuristics/rins/nodesquot	float	0.3	[0, 1]	lin.
heuristics/rins/minnodes	int	50	[0, 1000000000]	lin.
heuristics/rins/maxnodes	int	5000	[0, 1000000000]	lin.
heuristics/rins/nwaitingnodes	int	200	[0, 1000000000]	lin.
heuristics/rins/minfixingrate	float	0.3	[0, 1]	lin.
heuristics/localbranching/neighborhoodsize	int	18	[1, 1000000000]	lin.
heuristics/localbranching/nodesofs	int	1000	[0, 1000000000]	lin.
heuristics/localbranching/nodesquot	float	0.05	[0, 1]	lin.
heuristics/localbranching/lplimfac	float	1.5	[1, 1000000000]	lin.
heuristics/rens/nodesofs	int	500	[0, 1000000000]	lin.
heuristics/rens/nodesquot	float	0.1	[0, 1]	lin.
heuristics/rens/minnodes	int	50	[0, 1000000000]	lin.
heuristics/rens/maxnodes	int	5000	[0, 1000000000]	lin.
heuristics/rens/minfixingrate	float	0.5	[0, 1]	lin.
heuristics/rens/startsol	cat	1	{n, 1}	lin.

Notes. The whitelist is generated directly from the code under SCIP 9.2.4.

Table 14 Model-side hyperparameters for the LLM-based tuning loops.

Item	Value
Best and recent records in prompt memory (B, R)	3, 3
LLM model	GPT-5.2
Reasoning effort	medium
Max output tokens	20,000

{feaspump, rins, localbranching, rens} denote the tracked separator and heuristic families, respectively. Summary quantities are extracted from the final SCIP summary when available, with the row-based fallback rules specified in the table.

Figure 4 Representative raw SCIP log excerpt and corresponding parsed diagnostic summary.

<pre> === RAW SCIP LOG (excerpt) === presolved problem has 953 vars, 358 cons presolving time: 0.12 time node left gap 0.2s 1 0 12.44% 0.3s 1 0 4.50% 7.5s 4200 3971 0.23% 10.1s 6691 6318 0.17% 23.7s 18580 17424 0.11% </pre>	<pre> === PARSED DIAGNOSTICS (excerpt) === t_total = 600.0 nodes_last = 432241 nodes_left = 319848 max_depth = 108 num_improvements = 12 first_inc_time = 0.0 gap_final = 1500.39 pdi = 76.83 presolve_time = 0.12 root_dual_gain = 9.06e-03 sb_time = 6.05 sb_iters = 116575 cuts_selected_total = 95704 sep_cmir_applied = 29937 heur_rins_time = 0.41 </pre>
--	---

Notes. The left box shows an excerpt from a real Case 2 run. The parser also reads the corresponding bound, LP, cut, separator, heuristic, and summary sections from the same log. The right box shows the compact, fixed-schema diagnostic state returned by the parser and consumed by LDST.

The parser converts detailed solver logs into a compact diagnostic state for the decision policy. In single-instance mode, the resulting summary is included directly in the prompt archive. In multi-instance mode, the same schema is applied to each instance, and the resulting diagnostics are presented as aligned rows. This preserves comparability across instances without requiring the LLM to process raw solver logs.

F. Prompt Construction and Representative Outputs

This section presents a representative prompt and LLM response and explains their relationship to the prompt ablations in Exp. B. It specifies the information available to the decision policy, the components removed in each ablation, and how the proposed parameter edits are supported by log-derived evidence.

Figure 5 illustrates how the prompt is constructed around the current best configuration rather than the most recent trial. The archive summarizes both the best and the most recent runs using a fixed schema, providing the LLM with compact, comparable information across trials instead of raw solver logs.

Table 16 summarizes the information channels examined in Exp. B. All variants share the same tuning loop and differ only in their access to process diagnostics, scalar history, and parameter semantics.

G. Validation of the Case 1 Completion Signal

This appendix describes the Case 1 completion-time estimator $\hat{T}_\infty$ as implemented and evaluates its absolute accuracy and ranking fidelity. All quantities are derived solely from standard SCIP logs, without access to internal solver states or modifications to the solver source code. In both retrospective studies, we simulate truncation of a completed long run at a specified cutoff and provide the estimator with only the progress-table rows and primal and dual bounds available at that time.

Figure 7 presents an XML100 SCIP log excerpt used in the retrospective validation. Although the full run proves optimality, it still has a nonzero gap at the simulated cutoff $\tau = 600$ s. The estimator therefore uses only the progress-table fields available up to that point, including elapsed time, processed and open node counts, and primal and dual bounds. The final summary of the completed run provides the true completion time against which the estimate is evaluated.

Table 15 Diagnostic quantities used by the prompt builder.

Block	Parser fields exposed to the policy	Implementation definition	SCIP source
Run size and status	status, t_total (shown as solve_time), nodes_last (shown as solving_nodes), nodes_left, max_depth	t_total is the reported solving time, or last _f (t) if missing; nodes_last is the reported solving-node count, or last _f (n) if missing. nodes_left and max_depth are the B&B-tree summary values.	final summary, B&B tree, progress rows
Bounds and integral	primal_final, dual_final, gap_final, pdi (shown as primal_dual_integral)	primal_final and dual_final are final-summary bounds, falling back to last _f (z ^{pr}) and last _f (z ^{du}). gap_final = primal_final − dual_final . pdi = PDI ^{SCIP} (Λ ^(k)) is the SCIP-reported primal–dual integral parsed from the Integrals block.	final summary, Integrals block
Incumbents	nsols, num_improvements, first_inc_time (shown as first_incumbent_time), first_primal	nsols and num_improvements are solution-summary counts. first_inc_time and first_primal use the first-solution summary when present; otherwise they use the time and value at first _f (z ^{pr}).	solution summary, progress rows
Presolve	presolve_time, presolved_vars, presolved_cons_init, presolved_cons_max	Direct parser entries for presolving time, presolved-variable count, initial presolved-constraint count, and maximum presolved-constraint count.	presolve summary
Root progress	root_dual0, root_dual_end, root_dual_gain, root_gain_rel, root_first_lp_iters, root_first_lp_time, root_final_root_iters, root_lp_estimate	root_dual0 is the first root-LP value and root_dual_end is the final root dual bound. The derived fields are root_dual_gain = root_dual_end − root_dual0 and root_gain_rel = root_dual_gain / max{ root_dual0 , η}.	root-node report
LP and branching effort	sb_time, sb_calls, sb_iters, sb_root_calls, sb_root_iters	sb_time, sb_calls, and sb_iters are parsed from the LP-statistics row for strong branching; sb_root_calls and sb_root_iters are parsed from the root-node LP row.	LP statistics
Cut separation	cuts_selected_total, cuts_selected_root, and applied-cut or time fields for c ∈ S	cuts_selected_total = ∑ _q selected _q and cuts_selected_root = ∑ _q rootselec _q over cut selectors q. For each separator family c, the parser records the applied-cut count and records execution time when the SCIP table exposes that entry.	cut-selector and separator tables
Heuristic activity	time, call, found-solution, and best-solution counts for h ∈ H	For each heuristic family h, the parser exposes SCIP’s calls, exectime, found, and best columns as prompt diagnostics.	primal-heuristics table

Notes. The quantities are parsed from standard SCIP logs. Aliases in parentheses give the field names rendered in the archived prompts.

Table 16 Prompt-content ablations used in Exp. B.

Prompt block in Figure 5	LDST	STATICCONTEXT	OUTCOMEONLY	INDEXONLY
Header, whitelist, current best, edit cap, schema	semantic names, defaults, domains	same	same	indices and domains
Optional formulation text	optional	optional	optional	optional
Validator feedback during re-query	kept	re-query only	kept	kept
[MEMORY] archive summary	scalar + diagnostics	current diagnostics only	scalar only	full, indexed
Multi-instance diagnostic rows	kept	removed	removed	kept
[TIPS] diagnostic-feature guide	kept	removed	removed	kept
[DEDUP] prior configurations	kept	removed	kept	kept

Notes. The controls are illustrated using the same prompt excerpt as Figure 5. Each control retains the same outer tuning loop and modifies only the specified prompt blocks. The “current diagnostics only” control removes archive memory while retaining the diagnostic block of the current best run. The “scalar only” control retains trial id, loss, solve time, final gap, and finite/feasible status. The “full, indexed” control retains the diagnostic archive but replaces semantic parameter names with indices.

Figure 5 Representative single-instance prompt excerpt issued after trial 2.

```

=== LDST Prompt ===
You are tuning SCIP parameters for a mixed-integer programming solver.
Objective: min loss (intuition: ~ min solution time for opt) (loss is normalized)
Current best loss: 0.4436
Edit cap M: 20
Whitelist names: branching: scorefunc(1;p), preferbinary(2;false), ...
| nodeselection: ... | separating: ... | presolving: ... | heuristics: ...
Tuning set: 1 instance (single-instance);
loss = normalized single-instance loss; diagnostics: raw per-run values

Incumbent edits (delta based on defaults):
{ (2)=true, (4)=1, (5)=0.10, (6)=5000, (11)=0, (12)=50,
  (13)=200, (14)=5000, ... }

Problem formulation
We adopt the two-commodity network-flow formulation of \citet{baldacci2004exact} ...

[MEMORY] 3 best + 3 recent (after dedup)
cols: trial | loss | delta | solve_time | solving_nodes | nodes_left | ...
- 2 | 0.44 | {} | 1.18 | 1074 | 0 | 26 | ...
- 1 | 0.57 | { (4)=2, (5)=0.20, (6)=20000, (11)=-1, (12)=-1 }
  | 1.52 | ...
- 0 | 1 | { (2)=false, (4)=5, (5)=0.50, (6)=100000, ... }
  | 2.66 | ...

[TIPS]
Use the diagnostics and memory to identify ONE dominant bottleneck before ...

[DEDUP]
Avoid these (loss, delta based on incumbent)s other than already listed above
(most recent first):
(empty)

[TASK]
Propose the next configuration as a sparse edit set Delta p relative to ...
Use <= 20 edits and only whitelist indices.
Return ONLY valid JSON with this schema:
{"edits": [{"index": "<int>", "whitelist": "<str>", "value": "<val>"}],
 "rationale": "<brief reasoning tied to the observed diagnostics>"}

```

Notes. The excerpt is abridged at "...". The prompt includes static context, edits defining the current best configuration (labeled "incumbent" in the prompt), formulation text when available, archive memory, the diagnostic-feature guide, the deduplication list, and the output schema.

G.1. Completion-Time Estimator

The estimator is implemented as `predict_time()` in `utilities/est_time.py`. If a run is solved within the time limit τ , the estimator returns the observed solve time. Otherwise, it truncates the parsed SCIP progress table at $\tau' = \min\{\tau, t_{\max}\}$, where $t_{\max}$ is the last logged time. After discarding the startup prefix with $\text{node} \leq 1$, it retains only rows with finite values for time, processed-node count, open-node count, primal bound, and dual bound. The resulting snapshots are $\mathcal{S} = \{(t_j, n_j, u_j, g_j)\}_{j=1}^m$, where t_j is elapsed time, n_j is the processed-node count, u_j is the number of open nodes, and $g_j = |z_{\text{final}}^{\text{pr}} - z^{\text{du}}(t_j)|$ is the absolute difference between the primal bound reported at the end of the observed run and the dual bound at time t_j . For tuning runs, the observed run ends at τ , so $z_{\text{final}}^{\text{pr}}$ is the primal bound available at the time limit.

Following the single-variable branching (SVB) model of [Le Bodic and Nemhauser \(2017\)](#), each snapshot yields a candidate terminal tree size $\hat{N}_j(x) = n_j + u_j x^{g_j}$. The implementation draws up to 25 stratified

Figure 6 Representative LLM response with a sparse parameter edit set.

```

=== LLM_RESPONSE_JSON (excerpt) ===
{
  "edits": [
    {"index": "3", "whitelist": "branching: relpscost/minreliable", "value": "2"},
    {"index": "4", "whitelist": "branching: relpscost/maxreliable", "value": "4"},
    {"index": "5", "whitelist": "branching: relpscost/sbiterquot", "value": "0.05"},
    {"index": "6", "whitelist": "branching: relpscost/sbiterofs", "value": "50"}
  ],
  "rationale":
    "1) Incumbent already fixes the dominant issue at root
    (root_dual_gain 4.84, nodes drop to 126), and separation times are tiny;
    so root cutting is not the bottleneck anymore.
    2) The remaining dominant cost is expensive branching:
    sb_time=16 out of 28 s with very large sb_iters (128118) despite the small tree.
    3) Therefore, reduce strong-branching effort by making pseudocosts
    become 'reliable' sooner (lower minreliable/maxreliable) and by
    tightening the SB LP-iteration limits (lower sbiterquot/sbiterofs).
    This should cut sb_time substantially
    while accepting a modest risk of more nodes."
}

```

Notes. The response comes from a different single-instance run than Figure 5. The proposed edits are tied to the parsed diagnostics of that run, making the proposal interpretable and allowing it to be validated before the next solver call.

Figure 7 SCIP log excerpt for XML100 instance XML100_3311_15_35.

SCIP> read XML100_3311_15_35.mps.gz

snapshot fields (t_j, n_j, u_j)

time	node	left
420s	41100	1305
480s	47300	1237
540s	52900	1069
580s	56400	883
600s	57900	725

bound fields at each snapshot

dualbound	primalbound	gap	compl.
1.629251e+04	1.645100e+04	0.97%	49.99%
1.630569e+04	1.645100e+04	0.89%	56.04%
1.632154e+04	1.645100e+04	0.79%	67.09%
1.633211e+04	1.645100e+04	0.73%	73.33%
1.633899e+04	1.645100e+04	0.69%	77.45%

final summary used to audit completion and final primal value

SCIP Status : problem is solved [optimal solution found]
 Solving Time (sec) : 636.04
 Primal Bound : +1.64510000000000e+04
 Dual Bound : +1.64510000000000e+04
 Gap : 0.00 %

Notes. The completed run proves optimality at 636.04s, but its simulated truncation at $\tau = 600$ s retains a 0.69% gap and 725 open nodes. Red boxes identify the progress-table fields used to construct snapshots and the final summary used to record the true completion time.

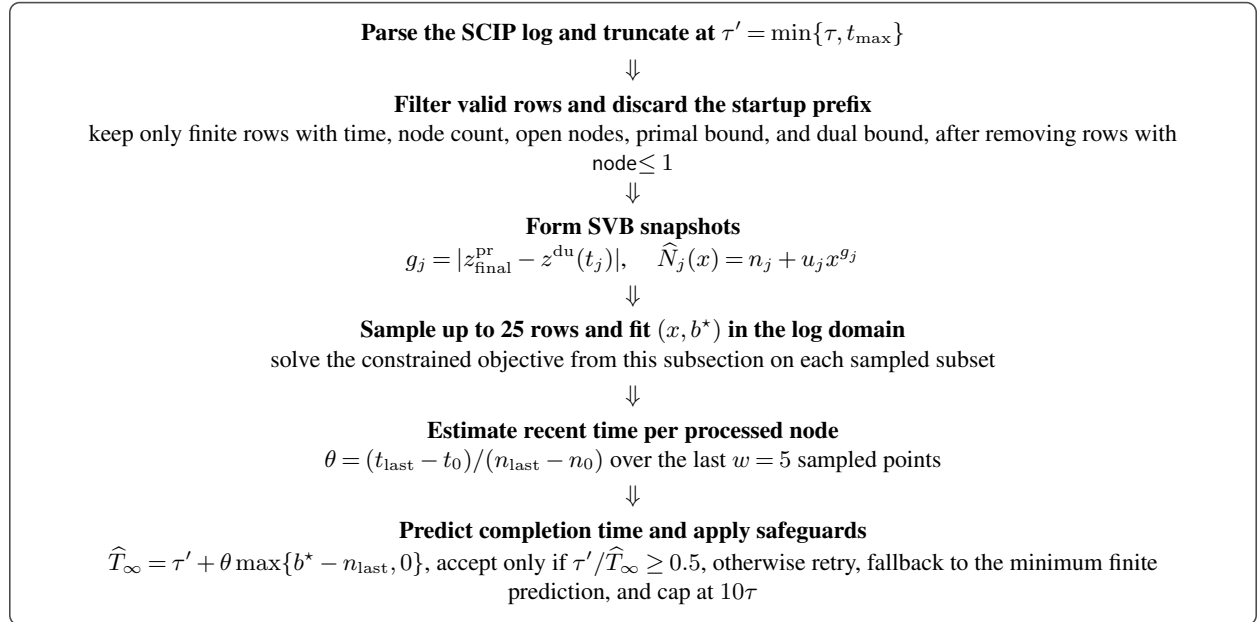
samples from the valid rows, requiring at least 8 rows for fitting. For a sampled subset $\mathcal{T} \subseteq \mathcal{S}$, it solves the following optimization problem on the log scale:

$$\begin{aligned}
& \min_{x, b, \{z_j\}_{j \in \mathcal{T}}, v} \quad \sum_{j \in \mathcal{T}} z_j + \lambda v \\
& \text{s.t.} \quad z_j \geq \log b - \log \hat{N}_j(x), \quad \forall j \in \mathcal{T}, \\
& \quad \quad z_j \geq \log \hat{N}_j(x) - \log b, \quad \forall j \in \mathcal{T}, \\
& \quad \quad v \geq \sum_{j \in \mathcal{T}} z_j - k \log b, \quad v \geq 0, \\
& \quad \quad x \in [1.0001, 10^5], \\
& \quad \quad b \in [b_{\text{lb}}, 2b_{\text{lb}}],
\end{aligned}$$

where $b_{\text{lb}} = n_{\text{last}} + u_{\text{last}}$, $k = 0.15|\mathcal{T}|$, and $\lambda = 10$. Here, the node counts are taken from the latest sampled row. Let b^* denote the fitted terminal tree size. Using the last $w = 5$ sampled points, let (t_0, n_0) and $(t_{\text{last}}, n_{\text{last}})$ denote the earliest and latest observations, respectively. The estimator computes the recent time per processed node as $\theta = (t_{\text{last}} - t_0)/(n_{\text{last}} - n_0)$, using a default value of 0.01 when $n_{\text{last}} \leq n_0$. The predicted completion time is then $\hat{T}_\infty = \tau' + \theta \max\{b^* - n_{\text{last}}, 0\}$.

To limit unstable extrapolations, a candidate prediction is accepted only if $\tau'/\hat{T}_\infty \geq 0.5$. The implementation attempts at most 10 independently sampled fits and returns the first accepted prediction. If no candidate satisfies the acceptance criterion but finite predictions are available, it returns the smallest predicted value. Every prediction is capped at 10τ to prevent extreme outliers. If all fitting attempts fail, the internal failure value 10^{12} is subject to the same cap, assigning the run a completion-time estimate of 10τ (6,000 s at $\tau = 600$ s). Such a run is therefore noncompetitive with any solved run. The tuning procedure uses $\hat{T}_\infty$ to rank configurations, and the following ranking study evaluates its ability to do so. Figure 8 summarizes the estimator.

Figure 8 Six-step flow of the Case 1 completion-time estimator $\hat{T}_\infty$.



Notes. The estimator is implemented in `utilities/est_time.py` and uses only standard solver-log observables. It retries fitting on multiple sampled subsets and assigns the capped value 10τ when all fits fail, making such runs noncompetitive.

G.2. Retrospective Accuracy on Completed Runs

Following the offline simulation protocol of Hendel et al. (2022), we construct a set of 100 two-commodity CVRP models with 35 customers, built from instances of the CVRPLIB XML100 collection (Lima et al. 2014, Queiroga et al. 2022), each solved to proven optimality within an extended time limit of 1,000 s under the default configuration p^{def} . This yields 100 completed default runs. For each instance with ground-truth time to optimality T^* , we truncate its log at fractions $t_k/T^* \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ and set $\tau = t_k$. We then apply the estimator using only the information available at the cutoff, including the primal bound at t_k , to obtain $\hat{T}_\infty(k)$. We assess prediction accuracy using the ratio $r_k = \hat{T}_\infty(k)/T^*$ and report the symmetric mean absolute percentage error (SMAPE) and geometric-mean ratio Q , where $Q = 1$ indicates perfect calibration on the geometric-mean scale. Table 17 summarizes prediction accuracy, the proportion of truncations yielding an accepted fit, and the median fitting time. Prediction error decreases as more of the run is observed, with SMAPE falling from 0.149 at half of the run to 0.068 at 90%. Accepted fits account for 88%–99% of truncations, and Q lies within 3% of 1 from the 0.6 cutoff onward (0.879 at the halfway cutoff).

Table 17 Retrospective accuracy of $\hat{T}_\infty$ on truncated XML100 default-run logs.

t_k/T^*	n	acc.	SMAPE	Q	mean r	med r	med. fit (s)
0.50	100	0.88	0.149	0.879	0.889	0.909	0.104
0.60	100	0.92	0.145	0.981	0.997	1.026	0.123
0.70	100	0.97	0.145	1.028	1.044	1.040	0.122
0.80	100	0.99	0.110	1.022	1.035	1.009	0.115
0.90	100	0.99	0.068	1.017	1.024	1.003	0.102

Notes. Predictions use only information available at each simulated cutoff. The ratio $r = \hat{T}_\infty/T^*$ compares predicted and true completion times. “acc.” is the proportion of truncations for which a sampled fit satisfies the acceptance rule; the remaining truncations use the minimum-prediction fallback, and no truncation has all fitting attempts fail. SMAPE is $2|\hat{T}_\infty - T^*|/(\hat{T}_\infty + T^*)$ averaged over all 100 truncations. $Q = \text{GM}(r)$ measures multiplicative calibration, and “med. fit” denotes the median fitting time per truncation.

G.3. Ranking Fidelity

The usefulness of $\hat{T}_\infty$ for tuning depends on its ability to rank configurations by completion time. We evaluate ranking fidelity using XML100 runs collected with a 1,000 s time limit under the default configuration and ten fixed whitelist settings, CFG01–CFG10. Each setting emphasizes a particular solver strategy, such as primal-first search, depth-first search, extensive root cutting, or increased strong branching. The settings files are released with the code. We retain the default configuration and CFG02, CFG03, CFG04, CFG05, CFG07, CFG09, and CFG10. The three more extreme settings, CFG01, CFG06, and CFG08, solve only 26, 23, and 24 of the 100 instances within 1,000 s, respectively. Ground-truth comparisons are restricted to the 49 instances solved by all eight retained configurations, yielding 392 configuration runs.

At a simulated cutoff of 600 s, 350 of these runs have already finished, for which $\hat{T}_\infty$ returns the observed solve times. The remaining 42 runs require extrapolation. We evaluate ranking on these 42 runs. For each run, we reconstruct the log summary using only progress rows at or before the cutoff, including the primal bound available at that time, and refit $\hat{T}_\infty$. All 42 fits succeed. We restrict pairwise comparisons to the 11 instances with at least two censored configurations, yielding 34 within-instance configuration pairs. We compare $\hat{T}_\infty$, the final logged gap, and a logged-gap integral, all computed at the same cutoff, against the true completion times. The logged-gap integral serves as a proxy for the primal–dual integral and is computed by left-step integration of the percentage gaps reported in the progress table up to the cutoff, using the first visible gap for the initial interval and 100 for missing gaps. This proxy differs from the primal–dual integral reported

by SCIP and used in the Case 2 loss. Table 18 reports the number of correctly ordered pairs and the mean within-instance Kendall rank correlation (Kendall 1938, 1945)

$$\tau_b = \frac{C - D}{\sqrt{(C + D + T_x)(C + D + T_y)}},$$

where C and D denote the numbers of concordant and discordant pairs, respectively, and T_x, T_y count ties in the two rankings.

Table 18 Prediction-only ranking at a 600 s cutoff.

Signal at the cutoff	Correctly ordered pairs	Mean within-instance τ_b
Completion estimate $\hat{T}_\infty$	26/34	0.515
Final logged gap	25/34	0.485
Logged-gap integral (PDI proxy)	17/34	0.121

Notes. The evaluation includes the 11 XML100 instances with at least two censored configurations. All ranking signals are computed using only information available at the cutoff and are compared on the same 34 within-instance configuration pairs.

The completion-time estimate places unfinished runs on the same time scale as the observed solve times of completed runs, so a single Case 1 objective ranks solved and unsolved configurations together, which neither gap-based signal provides. Among the three signals, it also orders the most configuration pairs correctly (26 of 34, against 17 for the logged-gap integral) and achieves the highest mean within-instance rank correlation.

H. Additional Results, Transfer Breakdowns, and Hard-Regime Audit

This section provides supplementary analyses for the experiments in Section 5, including the edit sizes of Exp. A, split-level and benchmark-level breakdowns of Exp. C, and an audit of the Case 3 outlier in Exp. A.

Exp. A edit sizes. Table 19 reports the size of the edit sets that LDST proposes in Exp. A, computed from the archived tuning histories of all 215 instances. Each proposal is measured against the best configuration at the time it is made. Edits are smallest in Case 1 and largest in Case 3.

Table 19 Edit-set sizes of LDST proposals in Exp. A.

Case	Proposals	Median	Mean	90th pct.	Max	≤ 10 (%)	At cap (%)
Case 1	3,800	5	5.00	8	19	97.8	0.0
Case 2	4,550	7	7.35	12	20	86.5	0.4
Case 3	2,400	11	11.71	18	20	49.3	4.7
All	10,750	7	7.49	13	20	82.2	1.2

Notes. Edit-set size is the number of whitelisted parameters for which the evaluated configuration differs from the best configuration at the time of the proposal. Each proposal may edit at most 20 of the 45 whitelisted parameters.

Exp. C split-level and benchmark-level breakdowns. The aggregate results of Exp. C conceal substantial variation across benchmark families. For Case 1 test loss, joint tuning outperforms independent selection at $m = 2$ and $m = 3$, achieves nearly identical performance at $m = 4$, and performs worse on the eligible pooled subsets at $m = 5$ and $m = 6$. Table 20 reports the Case 1 test loss by benchmark family and training size, while Table 21 provides the transfer results across regimes at $m = 3$. Joint tuning consistently outperforms independent selection on CVRP across all five training sizes and on GAP at both eligible sizes. In contrast, independent selection performs better on UFL at every training size. Results for JSP are mixed, with independent selection slightly better at $m = 2$ and joint tuning better at $m = 3$. The pooled results at larger m therefore reflect the contrasting performance of the two remaining eligible families, CVRP and UFL.

Table 20 Benchmark-specific Case 1 test losses across training sizes in Exp. C.

Benchmark	$m = 2$		$m = 3$		$m = 4$		$m = 5$		$m = 6$	
	Indep.	Joint	Indep.	Joint	Indep.	Joint	Indep.	Joint	Indep.	Joint
CVRP	1.122	0.796	1.014	0.874	0.985	0.885	1.028	0.877	0.769	0.675
GAP	0.775	0.673	0.841	0.680	–	–	–	–	–	–
JSP	0.265	0.325	0.217	0.204	–	–	–	–	–	–
UFL	0.369	0.382	0.323	0.395	0.318	0.355	0.287	0.445	0.292	0.382

Notes. Each entry is the geometric mean of split-level Case 1 test losses for the corresponding benchmark family, tuning strategy, and training size. A dash (–) indicates that the family is ineligible for the given training size under the requirement of at least four test instances. Lower values indicate better performance.

Table 21 Benchmark-specific transfer results in Exp. C at $m = 3$.

Benchmark	Indep. C1	Joint C1	Indep. C2	Joint C2	Indep. C3	Joint C3
CVRP	1.014	0.874	0.776	0.749	1.018	1.011
GAP	0.841	0.680	1.030	0.913	–	–
JSP	0.217	0.204	0.807	0.830	–	–
UFL	0.323	0.395	–	–	–	–

Notes. Columns C1, C2, and C3 report Case 1 test loss, same-family Case 2 loss, and same-family Case 3 loss, respectively. Each entry is the geometric mean over repeated splits for the corresponding benchmark family and tuning strategy. A dash (–) indicates that the family has no test instances in that regime. Lower values indicate better performance.

Case 3 outlier audit. The aggregate Case 3 results in Exp. A are strongly influenced by a single instance, `miplib_irish-electricity`. Table 22 reports the ten instances with the largest absolute differences in the selected Case 3 loss between LDST and RBFOPT, using the same best-so-far selection rule as Table 3. The largest difference occurs on `miplib_irish-electricity`, where RBFOPT achieves a loss lower by 0.256. On `miplib_neos-1354092`, LDST achieves a loss lower by 0.079, while all remaining differences are below 0.007. Across all 48 instances, the geometric-mean losses are 0.9931, 0.9931, 0.9882, and 0.9929 for LDST, SMAC3, RBFOPT, and RANDOM, respectively. Excluding only `miplib_irish-electricity` yields losses of 0.9939, 0.9947, 0.9955, and 0.9950, respectively. The aggregate advantage of RBFOPT in Case 3 therefore comes from its improvement on a single instance. The main results in Table 3 retain all 48 instances.

Table 22 Largest instance-level loss differences between LDST and RBFOPT in Case 3.

Instance	LDST	SMAC3	RBFOpt	Random	Δ
<code>miplib_irish-electricity</code>	0.9533	0.9218	0.6977	0.8991	+0.2556
<code>miplib_neos-1354092</code>	0.9215	0.9863	1.0000	1.0000	−0.0785
<code>cvrp_B-n51-k7</code>	0.9922	0.9804	0.9855	0.9860	+0.0067
<code>cvrp_B-n45-k6</code>	0.9942	0.9893	0.9894	0.9895	+0.0048
<code>cvrp_A-n62-k8</code>	0.9964	0.9918	0.9919	0.9918	+0.0046
<code>cvrp_P-n50-k10</code>	0.9919	0.9959	0.9960	0.9960	−0.0041
<code>cvrp_B-n67-k10</code>	0.9964	0.9920	0.9923	0.9919	+0.0041
<code>cvrp_P-n65-k10</code>	0.9927	0.9958	0.9967	0.9954	−0.0041
<code>cvrp_A-n63-k9</code>	0.9953	0.9993	0.9993	0.9993	−0.0040
<code>cvrp_A-n69-k9</code>	0.9972	0.9941	0.9932	0.9936	+0.0040

Notes. The comparison covers the 48 Case 3 instances, with the default configuration eligible for selection by every method. Differences are defined as $\Delta = L_{LDST} - L_{RBFOPT}$, so positive Δ indicates a lower loss for RBFOPT. All reported losses are the original Case 3 values without display magnification.

I. Code and Data Availability, Environment, Compute Accounting, and Licensing

Code and data. The LDST source code is available at <https://github.com/Zhengzhong-You/LDST>. The repository includes the tuning loop, typed parameter whitelist, prompt construction, log parser, completion-time estimator, baseline implementations, experiment entry points, benchmark manifests, split assignments, representative prompts, and analysis scripts. It also provides instructions for obtaining the original benchmark instances and regenerating and verifying the derived .mps files, along with regime assignments and file-level provenance. The repository also includes per-instance results for Exp. A, Exp. B, C1, and the XML100 validation studies, together with a script that verifies the corresponding reported summaries.

Licenses and upstream assets. The benchmark instances are obtained from the public sources cited in Appendix A and remain subject to their respective upstream terms. For each source, the repository records the citation, upstream URL, and licensing status, including cases where the publicly available files do not include a separate license. The generated .mps files are deterministic reformulations of the original benchmark instances, and their source collections are documented in the repository. The LDST source code is released under the MIT license. Table 23 summarizes the hardware, software, and token-side compute accounting used for reproducibility.

Table 23 Hardware, software, and LLM token accounting for the reported experiments.

Item	Value
Operating system	Linux (Ubuntu; kernel 6.14.0-33-generic; x86_64)
CPU	13th Gen Intel(R) Core(TM) i7-13700K (24 logical CPUs)
System memory	62 GiB total RAM
Python	3.12.4
SCIP	9.2.4
Threads per solver run	1
Solver-side seed convention	randomization/randomseedshift = 42
LLM model	GPT-5.2
Total LLM calls in Exp. A	10,964
Total tokens in Exp. A	62,958,576
Approx. uncached list-price cost	USD 320.08
Invalid-JSON warnings in Exp. A	69

Compute accounting convention. The main text reports the exact solver-run budgets, and exact token accounting for Exp. A is reported in Table 23. All solver calls are single-threaded and blocking. Exp. A uses 51 solver evaluations per method-instance pair, comprising one default run and $K = 50$ proposals. Exp. B uses 31 evaluations per method-instance pair, comprising one default run and $K = 30$ proposals. Exp. C uses the same budget per training run for independent and joint tuning, with additional evaluations on test instances determined by the fixed split manifests.

Seed convention. All reported solver runs fix randomization/randomseedshift to 42, and the stochastic black-box baselines use an optimizer seed of 42. This convention makes every solver run reproducible. The LLM policy uses a fixed model and API parameter settings, as documented in Appendix D.

J. Full Benchmark Manifests and Family-Specific Formulations

This appendix provides the family-specific formulations, source breakdowns, and benchmark manifests supporting Appendix A. The experimental benchmark comprises 90 CVRP, 12 GAP, 24 JSP, 30 UFL, and 59 MIPLIB instances, 215 in total. Appendix A describes the benchmark sources, construction pipeline, regime assignment, and split protocol; this appendix provides the detailed mathematical formulations and source tables for each benchmark family.

J.1. CVRP: Capacitated Vehicle Routing Problem

Problem definition. The CVRP seeks a set of vehicle routes of minimum total cost, each starting and ending at a depot, such that every customer is served exactly once and each vehicle respects the capacity limit Q . Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a complete undirected graph with two copies of the depot, where $\mathcal{V} = \{0, 1, \dots, n, n+1\}$ and $\mathcal{C} = \mathcal{V} \setminus \{0, n+1\}$ denotes the customer set. Each customer i has demand d_i , and K vehicles are available.

Formulation. Let c_{ij} denote the cost of using edge $\{i, j\}$. We adopt the two-commodity flow formulation of Baldacci et al. (2004), with binary edge variables x_{ij} and nonnegative continuous flow variables $y_{ij} \geq 0$:

$$\begin{aligned}
 \min \quad & \sum_{\{i,j\} \in \mathcal{E}} c_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{j \in \mathcal{V} \setminus \{i\}} (y_{ji} - y_{ij}) = 2d_i, \quad \forall i \in \mathcal{C}, \\
 & \sum_{j \in \mathcal{C}} y_{0j} = d(\mathcal{C}), \\
 & \sum_{j \in \mathcal{C}} y_{j0} = KQ - d(\mathcal{C}), \\
 & \sum_{j \in \mathcal{C}} y_{n+1,j} = KQ, \\
 & y_{ij} + y_{ji} = Q x_{ij}, \quad \forall \{i, j\} \in \mathcal{E}, \\
 & \sum_{\substack{j \in \mathcal{V} \\ j > i}} x_{ij} + \sum_{\substack{j \in \mathcal{V} \\ j < i}} x_{ji} = 2, \quad \forall i \in \mathcal{C}, \\
 & y_{ij} \geq 0, y_{ji} \geq 0, \quad \forall \{i, j\} \in \mathcal{E}, \\
 & x_{ij} \in \{0, 1\}, \quad \forall \{i, j\} \in \mathcal{E}.
 \end{aligned}$$

Here $d(\mathcal{C}) = \sum_{i \in \mathcal{C}} d_i$. We use the 90 CVRPLIB instances summarized in Table 24.

Instance selection. We select CVRP instances from the A, B, E, F, and P families in CVRPLIB (Lima et al. 2014), originating from Augerat et al. (1995) (sets A/B/P), Christofides and Eilon (1969) (set E), and Fisher (1994) (set F). CVRPLIB instance names follow the format <set>-n<N>-k<K>, where N denotes the number of nodes, including the depot, and K is the number of routes in the reference solution. For an instance with N nodes, our implementation sets $n := N - 1$ customers, represents the depot by nodes 0 and $n+1$, and fixes the fleet size to K . Table 24 summarizes the family counts and size ranges of the 90 selected instances.

Table 24 Summary of CVRP instances used in the experiments.

Set	#	Range of N	Source
A	27	32–80	Augerat et al. (1995)
B	23	31–78	Augerat et al. (1995)
E	13	13–101	Christofides and Eilon (1969)
F	3	45–135	Fisher (1994)
P	24	16–101	Augerat et al. (1995)
Total	90		

J.2. GAP: Generalized Assignment Problem

Problem definition. The GAP assigns each job $j \in \mathcal{J}$ to exactly one agent $i \in \mathcal{I}$. Assigning job j to agent i consumes a_{ij} units of the agent’s capacity b_i and incurs cost c_{ij} . The objective is to minimize the

total assignment cost while respecting all agent capacities (Ross and Soland 1975). The binary variable x_{ij} indicates whether job j is assigned to agent i .

Formulation.

$$\begin{aligned}
\min \quad & \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i \in \mathcal{I}} x_{ij} = 1, \quad \forall j \in \mathcal{J}, \\
& \sum_{j \in \mathcal{J}} a_{ij} x_{ij} \leq b_i, \quad \forall i \in \mathcal{I}, \\
& x_{ij} \in \{0, 1\}, \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}.
\end{aligned}$$

We use 12 OR-Library instances from gapc and gapd.

Instance selection. We select the large-scale type-C and type-D instances from the OR-Library GAP collection (Beasley 1990), introduced by Chu and Beasley (1997) and distributed as gapc and gapd. Each file contains six minimization instances with $|\mathcal{I}| \in \{5, 10, 20\}$ and $|\mathcal{J}| \in \{100, 200\}$. Instances are identified by their source file and within-file index (e.g., gapc-3). Table 25 summarizes the two source files and their instance sizes.

Table 25 Summary of GAP instances used in the experiments.

File	#	Sizes ($ \mathcal{I} \times \mathcal{J} $)	Source
gapc	6	$\{5, 10, 20\} \times \{100, 200\}$	Chu and Beasley (1997)
gapd	6	$\{5, 10, 20\} \times \{100, 200\}$	
Total	12		

J.3. JSP: Job Shop Scheduling Problem

Problem definition. Let $\mathcal{J}$ and $\mathcal{M}$ denote the sets of jobs and machines. The JSP schedules jobs, each comprising an ordered sequence of operations. Each operation o requires processing time p_o on a designated machine $\mu(o)$. Each machine can process at most one operation at a time, and preemption is not allowed. The objective is to minimize the makespan $C_{\max}$ (Manne 1960).

Formulation. We use the valid scheduling horizon $H = \sum_o p_o$. Let $\mathcal{P}$ denote the within-job precedence arcs and $\mathcal{D}$ the set of unordered pairs of operations requiring the same machine. Using start-time variables s_o and binary ordering variables $\delta_{oo'}$ for $(o, o') \in \mathcal{D}$, we adopt the following standard disjunctive MILP formulation:

$$\begin{aligned}
\min \quad & C_{\max} \\
\text{s.t.} \quad & s_{o'} \geq s_o + p_o, \quad \forall (o, o') \in \mathcal{P}, \\
& s_o + p_o \leq s_{o'} + H(1 - \delta_{oo'}), \quad \forall (o, o') \in \mathcal{D}, \\
& s_{o'} + p_{o'} \leq s_o + H\delta_{oo'}, \quad \forall (o, o') \in \mathcal{D}, \\
& C_{\max} \geq s_o + p_o, \quad \forall o, \\
& s_o \geq 0, \quad \forall o, \\
& \delta_{oo'} \in \{0, 1\}, \quad \forall (o, o') \in \mathcal{D}.
\end{aligned}$$

We use 24 instances from the OR-Library jobshop1 collection, covering the abz, ft, la, and orb families.

Instance selection. We select 24 benchmark instances from the OR-Library jobshop1 collection (Beasley 1990), spanning the abz, ft, la, and orb families (Adams et al. 1988, Fisher and Thompson 1963, Lawrence 1984, Applegate and Cook 1991). Table 26 summarizes the number of instances, size ranges, and sources for each family.

Table 26 Summary of JSP instances used in the experiments.

Family	#	Range of $(\mathcal{J} , \mathcal{M})$	Source
abz(5–9)	5	(10–20, 10–15)	Adams et al. (1988)
ft(10, 20)	2	(10–20, 5–10)	Fisher and Thompson (1963)
la(01–12)	12	(10–20, 5)	Lawrence (1984)
orb(01–05)	5	(10, 10)	Applegate and Cook (1991)
Total	24		

J.4. UFL: Uncapacitated Facility Location Problem

Problem definition. The UFL seeks to minimize the total facility opening and customer assignment costs, where each facility incurs a fixed opening cost and every customer must be assigned to an open facility (Kuehn and Hamburger 1963). Let $\mathcal{F}$ and $\mathcal{C}$ denote the facility and customer sets, respectively. Opening facility j incurs cost f_j , and assigning customer i to facility j incurs cost c_{ij} . The binary variables y_j and x_{ij} indicate whether facility j is open and whether customer i is assigned to it, respectively.

Formulation.

$$\begin{aligned}
 \min \quad & \sum_{j \in \mathcal{F}} f_j y_j + \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{F}} c_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{j \in \mathcal{F}} x_{ij} = 1, & \forall i \in \mathcal{C}, \\
 & x_{ij} \leq y_j, & \forall i \in \mathcal{C}, \forall j \in \mathcal{F}, \\
 & x_{ij} \in \{0, 1\}, & \forall i \in \mathcal{C}, \forall j \in \mathcal{F}, \\
 & y_j \in \{0, 1\}, & \forall j \in \mathcal{F}.
 \end{aligned}$$

We use 30 UfLib instances from the Uniform family.

Instance selection. We select uncapacitated facility location instances from UfLib (Hoefer 2018). Following our benchmark protocol, we include the complete Uniform family, comprising all 30 *UnifS instances. We exclude the *EuclS instances, which are typically solved at the root node under default settings and therefore provide limited information for tuning. Table 27 summarizes the UFL instances included in the benchmark.

Table 27 Summary of UFL instances used in the experiments.

Family	#	$(\mathcal{F} , \mathcal{C})$	Source
*UnifS	30	(100, 100)	Hoefer (2018)
Total	30		

J.5. General MILP Benchmark: MIPLIB 2017

Benchmark description. To complement the domain-specific models, we include instances from the official MIPLIB 2017 benchmark set (Gleixner et al. 2021). MIPLIB 2017 is the sixth release of the library and was constructed through a data-driven selection process. From an initial pool of 5,721 submissions, it provides a collection of 1,065 instances, including a curated benchmark subset of 240 instances for solver evaluation (benchmark-v2).

Instance selection. We uniformly sample 60 instances from the official MIPLIB 2017 benchmark subset using a fixed random seed (42). We exclude k1mushroom because it does not complete presolving under default SCIP settings within the 600-second time limit. The final experimental subset therefore contains 59 instances, listed in Table 28.

Table 28 Final 59-instance MIPLIB 2017 benchmark subset.

Final selected instances (benchmark-v2)		
50v-10	assign1-5-8	atlanta-ip
b1c1s1	beasleyC3	cbs-cta
chromaticindex512-7	cmflsp50-24-8-8	CMS750_4
cod105	comp21-2idx	cryptanalysiskb128n5obj16
dano3_3	drayage-25-23	dws008-01
fhnw-binpack4-48	gfd-schedulen180f7d50m30k18	glass4
glass-sc	gmu-35-50	h80x6320d
irish-electricity	lectsched-5-obj	momentum1
mushroom-best	mzzv11	n3div36
neos8	neos859080	neos-662469
neos-827175	neos-911970	neos-1354092
neos-1445765	neos-3046615-murg	neos-3083819-nubu
neos-3627168-kasai	neos-3988577-wolgan	neos-5052403-cygnnet
nexp-150-20-8-5	nursesched-sprint02	p200x1188c
piperout-27	radiationm18-12-05	radiationm40-10-02
ran14x18-disj-8	rocI-4-11	rocII-5-11
roll3000	s100	snp-02-004-104
supportcase22	supportcase40	supportcase42
swath1	thor50dday	traininstance2
uct-subprob	unitcal_7	

Notes. The subset used in the experiments was sampled from the official benchmark-v2 collection after removing k1mushroom, which does not finish presolving under the default 600-second run.