EXACTLY VERIFIABLE REINFORCEMENT LEARNING FOR LLM-WRITTEN RULES

Anonymous authors
Paper under double-blind review

ABSTRACT

An NP-hard problem is intrinsically difficult to solve optimally, so in practice one resorts to heuristics, hand-coded rules that decide quickly and give solutions of adequate quality. Workload scheduling is an example. When more jobs arrive than a system can serve, rules pick the next job, and they must change as the workload and the goals do. A rule is a short program, so a language model can write one, and re-running a recorded log of requests under the rule in an exact simulator gives a verifiable reward. We present **RuleForge**, a framework that approaches NP-hard problems by learning their heuristics. RuleForge has a library of verified rules and scores it by its best rule for each of many goals, with each goal having criteria set for its metrics. A writer model proposes rules for the goals the library misses, the rules that most raise that score join the library, and a selector model learns which rule to run for each goal. What is new is the writer's reinforcement-learning reward, which credits each rule with what it adds to the library over every goal, so a rule that repeats the library gets nothing and rules that fix the same goal share the credit. With equal simulator budgets, RuleForge outperforms eleven LLM-driven methods on multi-tenant LLM inference scheduling, dynamic job-shop and flow-shop scheduling, bin packing and the traveling salesman problem, and its lead on inference scheduling holds across seven open base models. There, ablations attribute most of the lead to the reward, and RuleForge's compact libraries cover more goals than an archive of tens of thousands of rules built with far more evaluations.

1 INTRODUCTION

Scheduling problems are hard to solve exactly. It is strongly NP-hard to order jobs arriving over time on one machine without preemption for the least total completion time, and to finish the last job of a job shop as early as possible (Pinedo, 2022). Future arrivals are also unknown. Production schedulers therefore decide online with priority rules, short formulas that score each waiting job and run the highest-scoring one (Branke et al., 2016; SchedMD, 2026; Kwon et al., 2023). Take a server that runs a language model for several tenants, the customers or teams that share it. Each tenant is promised a share of the server's capacity. When more requests arrive than the server can process, a rule decides which request runs next. The operator's goals are thresholds on each tenant's share of service, on how long requests wait and on the fraction of requests served. Experts write such rules by hand and rewrite them whenever the workload or the goals change.

Language models can now write these rules. Search systems keep the rules that score well in a simulator (Romera-Paredes et al., 2024; Liu et al., 2024a; Ye et al., 2024; Novikov et al., 2025; Cheng et al., 2025), and some evolve sets of complementary rules (Liu et al., 2026; Zhou et al., 2026). Other systems also train the model with reinforcement learning (Šurina et al., 2025; Huang et al., 2026b; Lv et al., 2026b). They reward each rule by its own score, by its gain over a seed rule or, as in CALM, by its gain over the best parent rule in its prompt. None of them trains the model on what a rule adds to the rules already kept.

What should a rule-writing model be rewarded for? We start from two observations. First, a scheduling problem is hard to solve but easy to check. A simulator replays a recorded workload under any rule and reports exactly whether the schedule meets a goal, and by how much it misses. Second, no single rule meets every goal. A scheduler that must serve many goals needs a library of rules, and

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

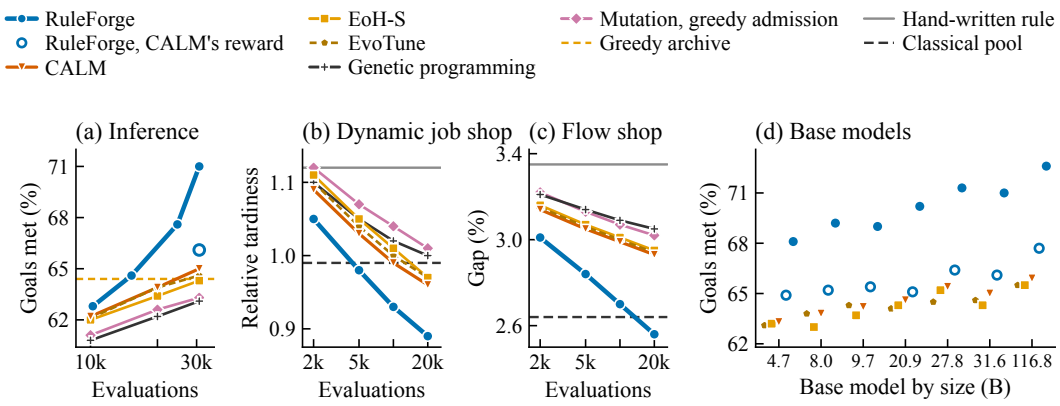

Figure 1: RuleForge leads the other LLM-driven methods at the full budget on three scheduling problems and with seven base models. (a) Held-out inference goals met by the chosen rule. The dashed line is the hand-written rules plus 24 rules chosen greedily from 26,341 grammar-sampled archive rules (Table 14). RuleForge's first point is a one-generation run without GRPO, and its middle points are its libraries after generations 0 and 1. (b) Dynamic job-shop mean tardiness relative to genetic programming. (c) Percent gap to the best-known makespans of Taillard's flow shops, where RuleForge ties the classical pool. (d) Inference goals met with the seven base models, ordered by total parameters, the main model at 31.6 billion. Tables 17 and 21 give paired intervals.

a new rule is worth what it adds to that library. A rule that scores well alone but repeats the library adds nothing, and rewarding it teaches the model to write it again. An earlier version of RuleForge rewarded each rule for its own result on the goal in its prompt, and nearly all of its trained writers' positive rewards went to such repeats (Appendix M).

We introduce **RuleForge**, which grows a *library* of verified rules from a few hand-written ones (Figure 2). It runs three rounds, called generations. In each generation, a *writer* model and random edits of library rules propose rules for the goals the library misses. A *verifier* checks every proposal in the simulator. Greedy admission then adds the rules that most raise the library's value, which counts the training goals some library rule meets, with partial credit for near misses. A *selector* model learns which rule to run for each goal. Both are adapters of one language model, and reinforcement learning trains both. Each scheduling decision runs one readable rule and calls no model.

The key piece is the writer's reward. Each GRPO group (Shao et al., 2024) holds eight rules for one goal the library misses. We ask what the group's verified rules would add to the library's value across every training goal, not only the one they target. We split this gain among the eight rules by their Shapley values. A rule that alone fixes a missed goal receives that goal's gain. Rules that fix the same goal share it. A rule that only repeats the library receives nothing. For on-policy samples, training on this credit ascends in expectation a weighted sum of the gains of groups of one to seven rules, rather than the gain of one rule alone (Section 3).

We evaluate RuleForge with Nemotron 3.5 Lightning (NVIDIA, 2026) on five problems. They are LLM inference scheduling on the public BurstGPT trace (Wang et al., 2025), a dynamic job shop (Holthaus & Rajendran, 2000), a permutation flow shop, and the online bin-packing and traveling-salesman benchmarks of automatic heuristic design (Liu et al., 2024b; Zheng et al., 2025b). With the same base model, starting rules and simulator budget, RuleForge leads eleven LLM-driven methods on all five problems (Table 1). On inference scheduling its chosen rules meet 71.0 percent of held-out goals, against 65.0 percent for the best other method, CALM (Figure 1a). Giving RuleForge's writer CALM's reward instead costs 4.9 of these 6.0 points (Figure 4a). The lead holds with each of seven base models from 4.7 to 116.8 billion parameters, and CALM's reward costs most of it on average (Figure 1d). On every problem, the trained writer beats a frozen writer and the earlier version's reward. Every preregistered main comparison and primary credit contrast passes after Holm's correction, except crediting every goal on the two shops, where the smallest share of a group's gain lands on other goals (Table 11 and Appendix I). We make three contributions.

- A reward for library building, with a map from any group credit to the objective it ascends. Each sample receives its Shapley share of its group's gain in library value over every goal. Prior rewards score each sample alone, credit it in full, split credit on one prompt or split a group's own value without the library (Section 3).
- RuleForge, a loop in which a writer and a selector learn from a simulator's exact verdicts to build and use a library of readable rules (Algorithm 1).
- An evaluation on five problems and seven base models under one protocol, with ablations and preregistered tests that isolate each part of the reward (Table 1 and Figure 4).

2 PROBLEM STATEMENT

We treat a heuristic as a priority rule (Branke et al., 2016). An *instance* x is a window of LLM requests, or a set of generated dynamic job-shop, flow-shop, bin-packing or routing problems of one size or load. A *rule* r scores the options at each decision, such as the waiting jobs, the open bins or the unvisited nodes. The highest-scoring option is taken. Rules are arithmetic expressions over features of the options, with $\min$, $\max$, abs, pow, round and a conditional (Appendix C). Running r on x gives a solution with D metrics $\mu(r, x) \in \mathbb{R}^D$, such as the fraction of requests served.

A *goal* $g = (x, \mathcal{J}, \tau)$ sets a threshold τ_j with a direction $\bowtie_j \in \{\leq, \geq\}$ on one to three metrics $j \in \mathcal{J}$. One run of the simulator on x, an *evaluation*, checks if rule r meets g. The *verifier* returns

$$v(r, g) = \mathbf{1}\big[\, \mu_j(r, x) \bowtie_j \tau_j \text{ for all } j \in \mathcal{J} \,\big]. \tag{1}$$

By construction, half the held-out inference goals are met by no hand-written rule, and a binary verdict rewards no progress on them. We call a training or held-out goal graded if no hand-written rule meets it but some hand-written rule meets all its thresholds except one. On a graded goal we pick the threshold that hand-written rules miss by the least. A rule that meets every other threshold gets a *graded score* $s(r, g)$ below 1, the fraction by which it improves on the best hand-written value of that metric. If that value is a tenant-share error of 0.20 and the goal asks for 0.10, a rule that reaches 0.15 scores 0.25 (Appendix C). In every other case $s(r, g) = v(r, g)$.

A *library* L is a set of rules, and a *selector* σ picks the rule $\sigma(g) \in L$ that runs for goal g. For goals drawn from a distribution $\mathcal{D}$, we report the goals met by the chosen rule. We also report the *library oracle*, the goals some library rule meets, which a perfect selector would reach,

$$\mathrm{Met}(L, \sigma) = \mathbb{E}_{g \sim \mathcal{D}}\, v\big(\sigma(g), g\big), \qquad V^*(L) = \mathbb{E}_{g \sim \mathcal{D}} \max_{r \in L} v(r, g). \tag{2}$$

RuleForge grows libraries to maximize a graded oracle on the training goals G, the *library value*

$$F(L) = \sum_{g \in G} s(L, g), \qquad s(L, g) = \max_{r \in L} s(r, g). \tag{3}$$

We also report a library's *oracle graded score*, the mean of $s(L, g)$ over the held-out inference goals no hand-written rule meets. A rule adds value only on goals where it beats every library rule. Adding a rule therefore never lowers F and adds no more to a larger library, so F is monotone and submodular (Bilmes, 2022).

Budgeted verification. A method's evaluation count covers every simulator run it makes, including the runs that score reinforcement-learning samples. We fixed the budget N before any final run. It is 30,833 evaluations on inference, an earlier version's mean count, and 20,000 on each other problem (Appendix E). On the other four problems, a goal bounds the mean gap or tardiness over a training instance. We report this mean for the rule the selector picks on each test set (Appendix H). Given the hand-written rules L_0 and the training goals G, the task is to choose $L \supseteq L_0$ and σ that maximize $\mathrm{Met}(L, \sigma)$ with at most N evaluations.

3 RULEFORGE

RuleForge runs three generations of the five steps below (Figure 2 and Algorithm 1). Admission and the writer's reward use one quantity, the gain in library value. Admission adds the survivors of largest gain, and the reward splits each group's gain among its rules (Appendix B).

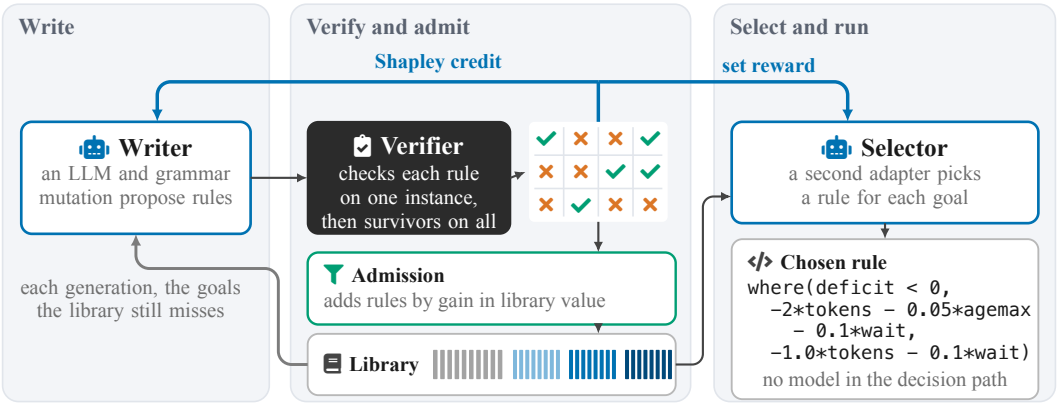

Figure 2: One generation of RuleForge, Steps 1 to 5 of Section 3. The rule shown, from an earlier version of RuleForge (Appendix M), favors short and newer requests.

Algorithm 1 The RuleForge loop.

Require: hand-written rules L_0, training goals G, writer W after supervised fine-tuning
1: **for** generation $t = 0, 1, 2$ **do**
2: $P \leftarrow$ six rules from W per graded goal of G that L_t misses, one mutation per writer rule (Step 1)
3: $C \leftarrow$ rules of P that pass the prescreen, and the surviving GRPO samples of generation $t-1$ (Step 2)
4: $L_{t+1} \leftarrow L_t$ plus up to K rules of C, run on all training instances and added by largest gain (Equation 4)
5: **if** $t < 2$ **then** fine-tune W on its admitted rules and train it with GRPO on R_W (Equation 6)
6: train a selector σ on L_{t+1}, supervised and then with GRPO on R_σ (Equation 7, Step 5)
7: **end for**
8: **return** library L_3 and selector σ

Step 1, writing for the goals the library misses. Each graded training goal the library misses gets a prompt that lists the library's rules and names the closest rule and the threshold it misses (Appendix D). The writer samples six rules per prompt, and grammar mutation, which uses no model, adds one random edit of a library rule for each.

Step 2, verification and admission. The verifier first runs each candidate on the instance of its goal, a cheap check we call the prescreen. A candidate survives if it meets a goal the library misses or comes closer to one. Coming closer means meeting every other threshold and beating the library's best value on the missed threshold by more than 2 percent. Survivors run on every training instance. Admission then adds them one at a time by their gain in library value,

$$\text{gain}(r \mid L) = F(L \cup \{r\}) - F(L) = \sum_{g \in G} \big[s(r, g) - s(L, g)\big]_+, \tag{4}$$

recomputed after each addition, up to $K = 8$ per generation. Since F is monotone and submodular, this greedy choice keeps at least $1 - 1/e$ of the best gain of any K survivors (Bilmes, 2022).

Step 3, the writer's reward. The writer trains on groups of $n = 8$ rules $r_1, \ldots, r_n$ for one missed goal g, the *prompting goal*. These groups are separate from the proposals of Step 1. The writer samples 6.4 rules of a group on average. Earlier verified rules fill the rest, replayed with their cached verdicts at no evaluation cost (Appendix D). On a training goal g', rule r_i adds

$$\Delta_i(g') = \mathbf{1}[\,r_i \text{ survives}\,]\,\big[s(r_i, g') - s(L, g')\big]_+, \qquad \gamma(S) = \sum_{g' \in G} \max_{r_i \in S} \Delta_i(g'), \tag{5}$$

where $\gamma(S)$ is what admitting the survivors in a subset S of the group would add to the library value. Each rule receives its Shapley value (Rozemberczki et al., 2022). This is the gain the rule adds when the group's rules join the library one at a time, averaged over all joining orders. It equals

$$R_W(r_i) = \sum_{g' \in G} \phi_i(g'), \qquad \phi_{(i)}(g') = \sum_{j=1}^{i} \frac{\Delta_{(j)}(g') - \Delta_{(j-1)}(g')}{n - j + 1}, \tag{6}$$

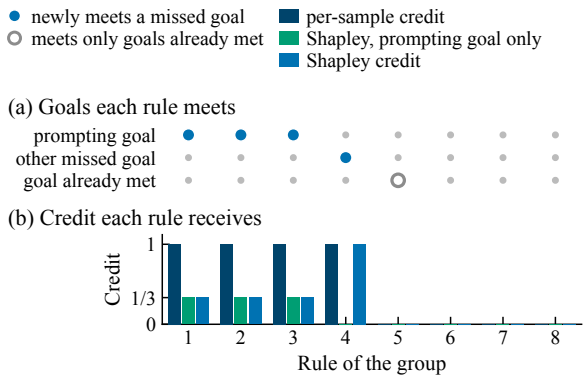

Figure 3: The Shapley credit on one group of eight rules for a missed goal, with binary verdicts. (a) Rules 1 to 3 newly meet this prompting goal, rule 4 another missed goal and rule 5 only goals already met. (b) Per-sample credit gives rules 1 to 3 a point each for one shared goal, and Shapley credit on the prompting goal alone gives rule 4 nothing. The Shapley credit gives rules 1 to 3 a third each and rule 4 a full point, adding up to the two goals the group adds.

where on each goal the rules are sorted by gain, $\Delta_{(1)}(g') \le \cdots \le \Delta_{(n)}(g')$, with $\Delta_{(0)}(g') = 0$, so $\phi_{(i)}$ is the share of the rule in position i. We call R_W the *Shapley credit*. On each goal, it splits every step up in gain equally among the rules that reach it. Suppose one rule raises a goal's score by 0.3 and another by 0.8. They share the first 0.3, and the second rule takes the next 0.5 alone, so they receive 0.15 and 0.65. Figure 3 shows the binary case. Summed over goals, a group's credits add up to what its rules would add to the library. A set's gain follows from its members' gains, so the reward needs no simulator run beyond those that verify each rule.

Step 4, training the writer. The writer and the selector are LoRA adapters on one base model (Hu et al., 2022). The writer first fine-tunes on the hand-written rules, paired with the goals they meet. After admission in generations 0 and 1, it fine-tunes on its own admitted rules. It then takes 100 GRPO steps (Shao et al., 2024) on the reward R_W. Each step centers the credits on the group's mean without dividing by their spread, as in Dr. GRPO (Liu et al., 2025). The steps also use the dynamic sampling, clipping and length-normalized sequence ratio of Appendix D. The surviving GRPO samples join the next admission.

Why this reward. Per-sample credit, a rule's own gain over the library, gives full credit to every rule that meets a shared goal. This pushes the writer toward the rules it already writes well. The Shapley credit counts a shared gain once. Let J_m be the expected gain in library value of m rules that the writer samples for a missed goal. For fresh on-policy samples, Proposition 1 in Appendix B shows that any credit that weighs a rule's marginal gains by subset size alone, centered on the group mean, ascends in expectation a weighted sum of $J_1, \ldots, J_{n-1}$. Per-sample credit ascends only J_1, leave-one-out credit only J_{n-1} and a best-of-k credit only J_k. The centered Shapley credit ascends $\sum_{m=1}^{n-1} J_m/m$, which weighs groups of one to seven rules and gives less weight than per-sample credit to goals the writer already fixes often. A writer that maximizes it keeps at least a third of the best gain of one rule and of eight rules, where per-sample credit guarantees only an eighth for eight and best-of-6 credit a sixth for one (Proposition 2). Centering removes the sum of a group's credits from the update, so the case for the Shapley weights rests on this objective and on their low variance, not on their summing to the gain (Section 4). Replay, dynamic sampling, clipping and the length-normalized ratio depart from this analysis, so Appendix I also tests the credits under plain on-policy GRPO, where it is exact. On one goal, the Shapley credit and its objective are those of TailRL (Ramasubramanian et al., 2026), and of MaxRL for a binary gain (Tajwar et al., 2026). What is new is the game the credit splits, the gain over a library summed over every training goal, and the map from each credit to the objective it ascends.

Step 5, training the selector. No single rule serves every goal. The selector therefore reads a goal and a summary of its instance, on inference the statistics of the preceding trace window. It returns a shortlist ℓ of up to three rules. Only the first runs, and the others let the reward give credit when it misses. Each generation retrains the selector on the new library, first by supervised fine-tuning and then with 400 GRPO steps. The set reward counts the shortlist as a success when one of its rules meets g (Chen et al., 2025), weighted toward earlier positions,

$$R_\sigma(\ell, g) = w_{\mathrm{pos}(\ell,g)}, \qquad w = \left(1, \tfrac{1}{2}, \tfrac{1}{4}\right), \tag{7}$$

Table 1: Results on five problems, means over five seeds with 95 percent intervals, at 30,833 evaluations on inference and 20,000 elsewhere. Inference reports held-out goals met by the chosen rule and the library oracle. The dynamic job shop reports tardiness relative to genetic programming, and the others the percent gap to the best-known makespans, the bin lower bound and the optimal tour. On inference the hand-written row runs the shared selector over the ten hand-written rules, and elsewhere it is one classical rule, the apparent tardiness cost rule ATC, NEH, best fit or nearest neighbor. The classical pool runs the shared selector over each problem's classical heuristics (Appendices G and H). Bold marks each column's best value and underlining the best other LLM-driven method.

	Inference, goals met ↑		Dynamic	Flow	Bin	
Method	Chosen rule	Oracle	job shop ↓	shop ↓	packing ↓	TSP ↓
Hand-written rules	45.2 ±1.1	50.0	1.12 ±0.03	3.35 ±0.18	2.40 ±0.23	24.6 ±0.3
No language model in the search						
Classical pool	57.5 ±1.6	62.0 ±1.5	0.99 ±0.03	2.64 ±0.15	1.41 ±0.14	**7.4 ±0.1**
Genetic programming	63.1 ±1.8	67.8 ±1.7	1.00	3.05 ±0.17	1.05 ±0.10	13.4 ±0.3
Mutation, greedy admission	63.3 ±1.8	68.5 ±1.7	1.01 ±0.03	3.02 ±0.17	1.08 ±0.11	16.3 ±0.3
LLM evolutionary search						
FunSearch	62.4 ±1.7	67.0 ±1.7	1.00 ±0.03	3.00 ±0.17	0.92 ±0.10	13.2 ±0.4
EoH	62.8 ±1.7	68.1 ±1.7	1.00 ±0.03	2.99 ±0.17	1.02 ±0.10	13.1 ±0.3
ReEvo	62.6 ±1.7	67.2 ±1.7	0.99 ±0.03	2.99 ±0.17	1.38 ±0.14	12.6 ±0.3
MCTS-AHD	63.2 ±1.8	68.7 ±1.7	0.98 ±0.03	2.97 ±0.16	0.81 ±0.09	11.5 ±0.3
OpenEvolve (Sharma, 2025)	63.4 ±1.8	68.2 ±1.7	0.99 ±0.03	2.98 ±0.16	0.88 ±0.09	12.9 ±0.4
ShinkaEvolve (Lange et al., 2026)	63.7 ±1.8	68.9 ±1.7	0.98 ±0.03	2.96 ±0.16	0.79 ±0.08	11.9 ±0.3
LEVI (Tanveer, 2026)	63.9 ±1.8	68.9 ±1.7	0.98 ±0.03	2.96 ±0.16	0.83 ±0.08	12.1 ±0.3
Sets of heuristics						
EoH-S	64.3 ±1.8	69.9 ±1.7	0.97 ±0.03	2.95 ±0.16	0.74 ±0.08	11.6 ±0.3
DSevolve	64.1 ±1.8	69.8 ±1.7	0.97 ±0.03	2.96 ±0.16	0.86 ±0.09	12.3 ±0.2
Reinforcement learning in the loop						
EvoTune	64.6 ±1.8	69.2 ±1.7	0.97 ±0.03	2.94 ±0.16	0.76 ±0.08	11.8 ±0.4
CALM	65.0 ±1.8	70.1 ±1.8	0.96 ±0.03	2.93 ±0.16	0.64 ±0.07	11.3 ±0.3
RuleForge, single best rule	61.5 ±1.7	61.5 ±1.7	0.94 ±0.03	2.70 ±0.15	0.55 ±0.06	11.1 ±0.3
RuleForge	**71.0 ±1.8**	**75.4 ±1.8**	**0.89 ±0.03**	**2.56 ±0.14**	**0.38 ±0.05**	9.9 ±0.3
lead over CALM, paired	6.0 ±1.5	5.3 ±1.5	0.068 ±0.021	0.37 ±0.14	0.26 ±0.07	1.38 ±0.37

where $\mathrm{pos}(\ell, g)$ is the position of the first listed rule that meets g. The reward is 0 when no listed rule meets g, and a near miss on a goal the library cannot meet gets graded credit (Appendix D).

4 Evaluation

Problems. LLM inference scheduling is our main problem. Requests from the BurstGPT trace arrive over time from several synthetic tenants, and four to eight slots process them without preemption. A rule scores each waiting request from features such as its true size in tokens, its wait and its tenant's shortfall from an equal share of the served work. The highest score runs next (Appendix C). Goals use four metrics, the L1 and L2 distances of the tenants' shares from equal shares, how unevenly the tenants wait and the fraction of requests served. The 2,816 held-out goals lie on fresh trace windows and set thresholds on two or three metrics. The offered load is three to six times what the server can process, since tenant shares only matter under backlog (Sheng et al., 2024). A clairvoyant CP-SAT relaxation that knows every arrival bounds the held-out goals any schedule can meet at 81.3 percent (Appendix G). In the dynamic job shop, jobs arrive at ten machines at utilization 0.85 or 0.95, and a rule orders each machine's queue to keep jobs from finishing late. In the flow shop, a rule sets the initial job order and the tie-break of the NEH insertion heuristic to shorten the makespan, the time the last job finishes. In bin packing a rule scores the open bins for each arriving item, and in the traveling salesman problem it scores the next city (Appendix H).

Baselines and protocol. We compare with eleven LLM-driven methods, namely seven evolutionary searches, two that evolve sets of heuristics and two that train the model. We also compare with genetic programming, grammar mutation with greedy admission and the classical heuristics of each

Table 2: Shortest first, which is the best hand-written inference rule, and three rules admitted by the earlier version of RuleForge (Appendix M). Met counts the held-out goals a rule meets alone, of that version's 352, and New those no archive rule meets. `tokens` and `wait` are a request's normalized work and wait, `deficit` is its tenant's target share minus its realized share `tput`, and `agemax` the wait of the tenant's oldest queued request.

Rule	What it does	Met	New
`0 - tokens`	*shortest first, hand-written*	151	0
`where(deficit < 0, -2*tokens - 0.05*agemax - 0.1*wait, -1.0*tokens - 0.1*wait)`	*shortest first, twice as strict above share, newer first*	205	3
`where(tput > target, -2*tokens, -1.0*tput - 0.1*tokens) - 0.05*wait`	*below-target tenants first, least served first, newer first*	200	2
`where(deficit > 0, -0.5*tokens - 0.05*wait + 2*deficit, -0.5*wait - 2*tokens)`	*below-share tenants first, furthest below first, newer first*	201	2

problem (Table 1). Every method starts from the same hand-written rules and shares the base model, the verifier with its prescreen, the evaluation budget, greedy admission and the selector. The shared parts therefore cannot explain a lead. RuleForge alone adds grammar mutation, a supervised start and a prompt that names the closest miss, and Appendix E gives CALM the mutation and the prompt. Each baseline keeps its published prompts, changed only to ask for a rule in our grammar. It scores rules in its own search by their gain in library value and is tuned on the inference development goals. CALM instead scores a rule against its best parent (Appendix E). Intervals are 95 percent, from a wild cluster bootstrap over the 44 trace segments or the test instances plus the variance between seeds (Cameron et al., 2008; Webb, 2023). Every lead we state has a paired interval above zero unless we say otherwise, with Holm-adjusted p-values in Table 11.

Observation 1. RuleForge leads eleven LLM-driven methods on all five problems, and on inference scheduling with each of seven base models. On inference scheduling RuleForge's chosen rules meet 71.0 percent of the held-out goals, against 65.0 percent for CALM, the best baseline. Its 34-rule libraries reach a library oracle of 75.4 percent, above the 70.8 percent of an archive of 26,341 rules built with 809,232 evaluations (Tables 1 and 14). On six more base models of 4.7 to 116.8 billion parameters, all LLM-driven methods retuned, RuleForge leads the best other method by 4.7 to 6.7 points (Figure 1d and Table 17). It leads at every budget of Figure 1. On inference, only the full-budget lead is clear of the noise (Table 21), because the two smaller budgets end before reinforcement learning shapes an admitted rule. RuleForge uses 1.6 times CALM's busy GPU-hours on H100s, yet it leads by 4.4 points when capped at CALM's hours, and CALM given RuleForge's hours, mutation and goal prompt still trails by 4.7 (Appendix E). RuleForge also leads baselines that write native Python policies on inference scheduling, run under their own protocols on bin packing and the traveling salesman problem, or use larger frozen models (Appendix G). Against the classical pool, it wins on inference, the dynamic job shop and bin packing, ties on the flow shop and loses on the traveling salesman problem, where farthest insertion builds tours no next-city rule can express.

Observation 2. The lead holds when a real serving engine runs the rules. We replay the held-out requests in vLLM at three to six times overload, with each method's chosen rules ordering the queue (Appendix J). RuleForge's rules meet 61.0 percent of goals set on time to first token, time per output token and tenant share, against 53.1 percent for CALM's rules. The classical pool and VTC, a fair LLM-serving scheduler (Sheng et al., 2024), trail by more.

Observation 3. Rules in this grammar are short and readable. A rule is one expression over a request's features, and it scores a queue in a median of 40 microseconds on one core. The admitted rules of Table 2, from an earlier version of RuleForge, favor short requests and, among similar sizes, newer ones. Each branches on its tenant's share.

Observation 4. On inference scheduling, the writer's reward carries most of the lead, and each of its parts raises goals met. Giving the writer CALM's reward inside the recipe costs 4.9 of the 6.0-point lead over CALM, and 4.4 of a 5.6-point lead averaged over the seven base models

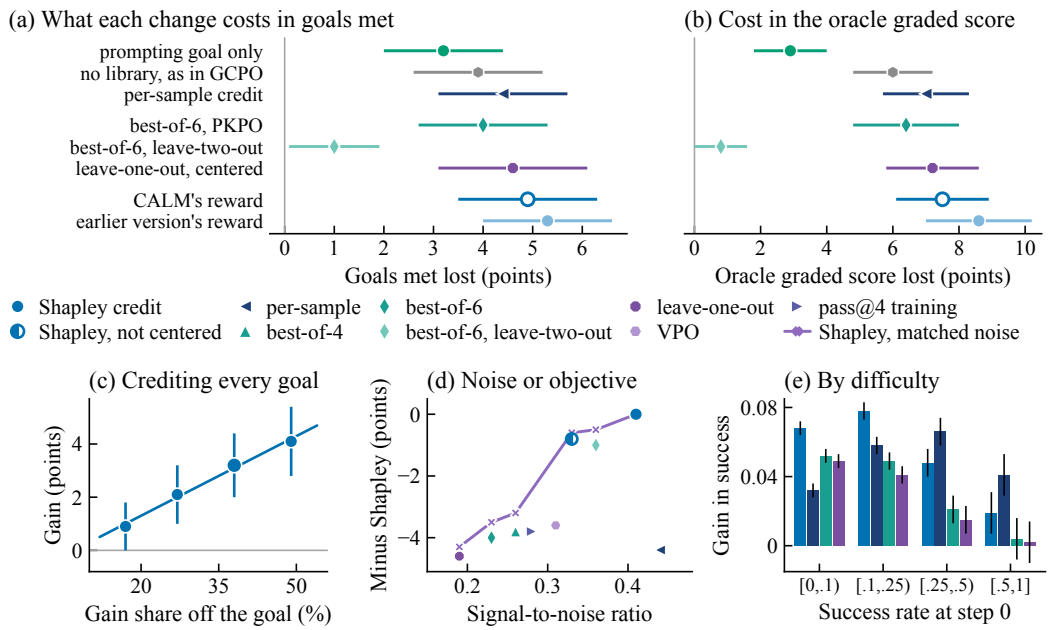

Figure 4: The Shapley split, the gain over the library and crediting every goal each raise goals met on inference scheduling. (a) and (b) RuleForge minus eight variants of the writer's reward (Tables 19 and 20). (c) The gain from crediting every goal against the share of a group's gain that lands on other goals, with the fitted slope. (d) Each credit's goals met minus the Shapley credit's, against its gradient signal-to-noise ratio, the norm of the mean update over its spread across groups. The curve is the Shapley credit with matched noise. (e) Gain in a prompt's success rate p over 100 steps of plain on-policy GRPO, by p at step 0.

(Table 17). A frozen writer, which keeps the supervised start but never trains, stays within noise of CALM, and with a GEPA-optimized prompt (Agrawal et al., 2026) still trails the trained writer (Table 11). Training on the Shapley credit cuts the share of the writer's proposals that repeat a library rule from 37 to 12 percent, while untrained LLM searches spend 38 to 44 percent of evaluations on such repeats (Appendix K). The lead thus comes from training the writer on this reward, and each part of the reward matters. Per-sample credit, as in TestDecision (Wang et al., 2026a), lowers goals met. In a contrast added after the preregistration, so does the Shapley credit without the library, as in GCPO (Chen et al., 2026a). Crediting only the prompting goal, TailRL's credit applied to the gain over the library (Ramasubramanian et al., 2026), ignores the 38 percent of a group's gain that lands on other goals. In a preregistered test that thins or doubles the training goals, this cost grows with that share (Figure 4c). The parts compound. Crediting every goal adds only 0.7 points under per-sample credit, within noise, and the split adds 1.9 on the prompting goal alone. Together they add 5.1, an interaction of 2.5 [1.2, 3.8] (Figure 15). On each of the other four problems, the Shapley credit beats per-sample credit, the earlier version's reward and a frozen writer, and crediting every goal helps except on the two shops, where the smallest share of a group's gain lands on other goals (Appendix H). Inside the code of CALM and EvoTune, the Shapley credit also lowers their bin-packing gaps from 0.74 to 0.62 and from 0.82 to 0.70, and their traveling-salesman gaps (Appendix G).

Observation 5. The Shapley credit beats per-sample credit by rewarding group gains and beats most group credits by being less noisy. Per-sample credit gives gradients about as stable as the Shapley credit's, yet it trails, since it trains on the gain of one rule alone (Section 3). The best-of-k credits and the leave-one-out credit, the group's loss without the rule, train the writer on group gains but are noisier. Once Gaussian noise lowers the Shapley credit's signal-to-noise ratio to theirs, the Shapley credit falls within noise of each (Figure 4d and Table 11). Under plain on-policy GRPO, the credits favor rarely fixed goals in the order Proposition 1 predicts, and the Shapley credit gains

the most on them (Figure 4e). After Holm's correction in Table 11, the only credits within noise of RuleForge are the uncentered Shapley credit and the leave-two-out best-of-6 credit of MaxPO (Takashiro et al., 2026). A preregistered follow-up with 15 seeds on three traces puts the Shapley credit ahead of both, by 0.8 and 1.0 points, and with ten times the training only the leave-two-out gap remains, as its objective predicts (Appendix I). VPO (Bahlous-Boldi et al., 2026) also trails.

Observation 6. The lead lies in complementary rules and the selector, not in a better single rule. On inference scheduling GRPO improves the selector over its supervised start, and its chosen rule meets more held-out goals than the library's best single rule (Table 1 and Appendix K). The best single rules of RuleForge and CALM are nearly tied (Table 15), as are those grown with CALM's reward and with the Shapley credit. Hence complementary rules and a selector that picks among them account for 5.3 of the 6.0-point lead and for 4.7 of the 4.9 points the reward adds (Table 11). The shared selector recovers 68 percent of what RuleForge's library oracle adds over its best single rule, against 49 with CALM's reward and 45 for CALM's library. On goals that exactly one library rule meets, it picks that rule for 88 percent of RuleForge's goals against 74 for CALM's (Appendix K). A preregistered replication on 300 fresh goals of the OR-Library job shop (Beasley, 2026) confirms the selector's GRPO gain on RuleForge's library in goals the shortlist meets (Appendix L).

5 Related work

Language models that write heuristics. FunSearch evolves programs with a language model and an evaluator (Romera-Paredes et al., 2024), and EoH, ReEvo, MCTS-AHD and AlphaEvolve refine that search (Liu et al., 2024a; Ye et al., 2024; Zheng et al., 2025b; Novikov et al., 2025). EoH-S and DSevolve evolve sets of heuristics with a fixed model (Liu et al., 2026; Zhou et al., 2026). EoH-S scores a set by its best member on each instance, as our library value does, and DSevolve and HeurAgenix pair an evolved pool with a learned selector (Yang et al., 2025). EvoTune, CALM and AHD Agent train the model (Šurina et al., 2025; Huang et al., 2026b; Lv et al., 2026b). They reward each program by its own score or its gain over a parent or seed heuristic, not by what it adds to all heuristics kept. Algorithm portfolios grow by what each solver adds (Xu et al., 2010) and value a solver by its Shapley share (Fréchette et al., 2016). RuleForge trains a writer on such a share.

Credit for groups of samples. TailRL and MaxRL split the credit on one prompt equally among the samples that pass each reward threshold (Ramasubramanian et al., 2026; Tajwar et al., 2026), and pass@k training and PKPO optimize the best of k samples (Chen et al., 2025; Walder & Karkhanis, 2025). GCPO splits a group's coverage by Shapley values, ShapE-GRPO splits the reward of the candidates in one response (Chen et al., 2026a; Ai et al., 2026), and TestDecision rewards each test by its gain over a kept suite (Wang et al., 2026a). To our knowledge, RuleForge is the first to train a model on the Shapley split of a library's gain over every goal (Appendix A).

6 Limitations and conclusion

Our inference goals are thresholds on a public trace with synthetic tenants, simulated without batching or a KV cache, which the vLLM replay only begins to address. The rules read true response lengths, and with a length predictor their lead in vLLM shrinks but remains. Like shortest first, they serve few of the largest requests, yet goals that bound the 99th-percentile wait keep the lead. The lead fades as the load falls toward what vLLM can serve (Appendix J). RuleForge uses 1.6 times CALM's busy GPU-hours (Appendix E). On the two shops, where the smallest share of a group's gain lands on other goals, crediting every goal gains little (Appendix H).

We asked what a rule-writing model should be rewarded for. RuleForge rewards it for what its rules add to a library, measured by a simulator's exact verdicts and split among the rules of a group. On inference scheduling, averaged over seven base models, this reward carries most of the lead over the best baseline. It also helps on the four other problems and inside the code of CALM and EvoTune, so it may carry to other training loops where heuristics are short programs a simulator can check.

AI USE STATEMENT

In this work, we used generative AI tools for tasks with required disclosure. AI coding agents helped write and run the experiment code, and they helped draft and tighten the text, including the proofs of Appendix B. We have not used generative AI tools to set the research questions, and data annotation, translation and qualitative analysis are not applicable to this work. Additionally, we used generative AI tools for tasks with recommended disclosure, to find and verify citations and to check the paper for consistency. We have reviewed all AI-assisted work. The authors set the research questions, checked the code, results, proofs and citations, and edited the text, and every citation was checked against Crossref, arXiv or OpenAlex. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only public or generated data and involves no human subjects. Inference scheduling replays arrival times and token counts from the public BurstGPT trace (Wang et al., 2025), and its further checks use the Azure LLM inference trace (Stojkovic et al., 2025) and the Mooncake trace (Qin et al., 2025). These traces contain no prompt text and no user identifiers, and the tenants are a synthetic overlay defined in our code. The other problems use generated instances and public benchmarks, namely the Taillard flow shops as used by Fernandez-Viagas & Framinan (2014), the VRF benchmark (Vallada et al., 2015), the bin-packing and traveling-salesman test sets of Zheng et al. (2025b) and the OR-Library job shops (Beasley, 2026). The language model writes only arithmetic expressions. Every rule compiles from a restricted grammar with a whitelist of functions, so it cannot call other code or reach the system it runs on. Scheduling rules decide who is served, so they bear on fairness. Our goals bound each tenant's share of service, but a goal constrains only one to three metrics, and a rule can meet it while harming a metric it leaves open, such as the service of the largest requests. The simulator reports every metric, so an operator can see such a trade-off before using a rule, and Appendix J reports who each rule serves. No rule from this paper runs on a production system, and we make no claim about production use. For research integrity, we fixed the main comparisons and credit contrasts in protocol files before the final runs, and we report every correction our audits found (Appendix M).

REPRODUCIBILITY STATEMENT

We will release the code and data behind every result. This covers the simulators and rule grammars of the five problems and of the OR-Library selector study, the vLLM scheduling hook, the BurstGPT rows behind every training, development and held-out window with their checksums, the goals of every problem, the 26,341-rule archive with its evaluations, the protocol files with their freeze dates, and the scripts that run every method and build every figure, table and interval from the logged results. Section 3 and Algorithm 1 describe the method, and Appendix D lists every hyperparameter, prompt, model revision and seed. Appendix E gives the shared protocol, the evaluation budget and its accounting, the tuning grid of every baseline and the statistics behind every interval and p-value. Appendices C and H define the rules, goals, metrics and instances of each problem, and Appendix B states the assumptions and complete proofs of both propositions.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Rui Ai, Yu Pan, David Simchi-Levi, and Chonghuan Wang. ShapE-GRPO: Shapley-enhanced reward allocation for multi-candidate LLM training. *arXiv preprint arXiv:2603.29871*, 2026.

Ryan Bahlous-Boldi, Isha Puri, Idan Shenfeld, Akarsh Kumar, Mehul Damani, Sebastian Risi, Omar Khattab, Zhang-Wei Hong, and Pulkit Agrawal. Vector policy optimization: Training for diversity improves test-time search. *arXiv preprint arXiv:2605.22817*, 2026.

J. E. Beasley. OR-Library: Job shop scheduling, file jobshop1. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html`, 2026. Accessed 2026-09-25.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, et al. Llama-Nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*, 2025.

Jeff Bilmes. Submodularity in machine learning and artificial intelligence. *arXiv preprint arXiv:2202.00132*, 2022.

Herbie Bradley, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Grégory Schott, and Joel Lehman. Quality-diversity through AI feedback. *arXiv preprint arXiv:2310.13032*, 2023.

Jürgen Branke, Su Nguyen, Christoph W. Pickardt, and Mengjie Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20 (1):110–124, 2016. doi: 10.1109/TEVC.2015.2429314.

Edmund K. Burke, Matthew R. Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature, PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pp. 860–869, 2006. doi: 10.1007/11844297_87.

A. Colin Cameron, Jonah B. Gelbach, and Douglas L. Miller. Bootstrap-based improvements for inference with clustered errors. *The Review of Economics and Statistics*, 90(3):414–427, 2008. doi: 10.1162/rest.90.3.414.

Haoxuan Chen, Tianming Liang, Wei-Shi Zheng, and Jian-Fang Hu. Breaking Winner-Takes-All: Cooperative policy optimization improves diverse LLM reasoning. *arXiv preprint arXiv:2605.11461*, 2026a.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, Samitha Samaranayake, Carla P. Gomes, and Zhiru Zhang. HeuriGym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, 2026b.

Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. Pass@k training for adaptively balancing exploration and exploitation of large reasoning models. *arXiv preprint arXiv:2508.10751*, 2025.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, Jeff Chen, Lakshya Agrawal, Aditya Desai, Jiarong Xing, Koushik Sen, Matei Zaharia, and Ion Stoica. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025.

Janos Csirik, David S. Johnson, Claire Kenyon, James B. Orlin, Peter W. Shor, and Richard R. Weber. On the Sum-of-Squares algorithm for bin packing. *Journal of the ACM*, 53(1):1–65, 2006. doi: 10.1145/1120582.1120583.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. HSEvo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using LLMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26931–26938, 2025. doi: 10.1609/aaai.v39i25.34898.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

Xingye Dong, Houkuan Huang, and Ping Chen. An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research*, 35(12):3962–3968, 2008. doi: 10.1016/j.cor.2007.05.005.

Rohit Dwivedula, Divyanshu Saxena, Aditya Akella, Swarat Chaudhuri, and Daehyeok Kim. Man-made heuristics are dead. Long live code generators! In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2025.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 835–850, 2021. doi: 10.1145/3453483.3454080.

Victor Fernandez-Viagas and Jose M. Framinan. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45:60–67, 2014. doi: 10.1016/j.cor.2013.12.012.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11794.

Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 94–102, 2020. doi: 10.1145/3377930.3390232.

J. M. Framinan, R. Leisten, and C. Rajendran. Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148, 2003. doi: 10.1080/00207540210161650.

Alexandre Fréchette, Lars Kotthoff, Tomasz Michalak, Talal Rahwan, Holger Hoos, and Kevin Leyton-Brown. Using the Shapley value to analyze algorithm portfolios. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. doi: 10.1609/aaai.v30i1.10440.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient LLM scheduling by learning to rank. In *Advances in Neural Information Processing Systems*, 2024.

Chang Gao, Chujie Zheng, Xiong-Hui Chen, Kai Dang, Shixuan Liu, Bowen Yu, An Yang, Shuai Bai, Jingren Zhou, and Junyang Lin. Soft adaptive policy optimization. *arXiv preprint arXiv:2511.20347*, 2025.

Amirata Ghorbani, Michael P. Kim, and James Zou. A distributional framework for data valuation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3535–3544, 2020.

GLM-5 Team. GLM-5.3-Flash model card. `https://huggingface.co/zai-org/GLM-5.3-Flash`, 2026. MIT license, accessed 2026-09-23.

Pouya Hamadanian, Pantea Karimi, Arash Nasr-Esfahany, Kimia Noorbakhsh, Joseph Chandler, Ali ParandehGheibi, Mohammad Alizadeh, and Hari Balakrishnan. Glia: A human-inspired AI for automated systems design and optimization. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 61–84. ACM, 2026. doi: 10.1145/3786335.3813125.

Emma Hart and Kevin Sim. A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary Computation*, 24(4):609–635, 2016. doi: 10.1162/evco_a_00183.

Oliver Holthaus and Chandrasekharan Rajendran. Efficient jobshop dispatching rules: Further developments. *Production Planning & Control*, 11(2):171–178, 2000. doi: 10.1080/095372800232379.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jin Huang, Qihao Liu, Xinyu Li, Liang Gao, and Yue Teng. Automatic programming via large language models with population self-evolution for dynamic fuzzy job shop scheduling problem. *IEEE Transactions on Fuzzy Systems*, 34(3):896–908, 2026a. doi: 10.1109/TFUZZ.2025.3650586.

Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. CALM: Co-evolution of algorithms and language model for automatic heuristic design. In *International Conference on Learning Representations*, 2026b.

Sheng Jia, Xiao Wang, Shiva Prasad Kasiviswanathan, and Rein Houthooft. Group adaptive clipping policy optimization. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026. To appear, arXiv:2609.00444.

Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC: Instance-specific algorithm configuration. In *Proceedings of the European Conference on Artificial Intelligence*, Frontiers in Artificial Intelligence and Applications, 2010. doi: 10.3233/978-1-60750-606-5-751.

Pawel J. Kalczynski and Jerzy Kamburowski. An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, 35(9):3001–3008, 2008. doi: 10.1016/j.cor.2007.01.020.

Pawel J. Kalczynski and Jerzy Kamburowski. An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research*, 198(1):93–101, 2009. doi: 10.1016/j.ejor.2008.08.021.

Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. doi: 10.1162/evco_a_00242.

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

Kubernetes SIG Scheduling. Kueue: Fair sharing (KEP-1714). `https://github.com/kubernetes-sigs/kueue/tree/main/keps/1714-fair-sharing`, 2024. Documentation, accessed 2026-09-23.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

Yongchan Kwon and James Zou. Beta Shapley: a unified and noise-reduced data valuation framework for machine learning. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 8780–8802, 2022.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, 2026.

Chenyi Li, Yuan Zhang, Bo Wang, Guoqing Ma, Wei Tang, Haoyang Huang, and Nan Duan. SetPO: Set-level policy optimization for diversity-preserving LLM reasoning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL `https://openreview.net/forum?id=tYKGzVE1g7`.

Marius Lindauer, Holger Hoos, Kevin Leyton-Brown, and Torsten Schaub. Automatic construction of parallel portfolios via algorithm configuration. *Artificial Intelligence*, 244:272–290, 2017. doi: 10.1016/j.artint.2016.05.004.

Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249*, 2023.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024a.

Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Qinglong Hu, Ping Guo, Xi Lin, Xialiang Tong, Mingxuan Yuan, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. LLM4AD: A platform for algorithm design with large language model. *arXiv preprint arXiv:2412.17287*, 2024b.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. EoH-S: Evolution of heuristic set using LLMs for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37090–37098, 2026. doi: 10.1609/aaai.v40i43.41038.

Weibo Liu, Yan Jin, and Mark Price. A new improved NEH heuristic for permutation flowshop scheduling problems. *International Journal of Production Economics*, 193:21–30, 2017. doi: 10.1016/j.ijpe.2017.06.026.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. In *Conference on Language Modeling*, 2025.

Shu Luo. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing*, 91:106208, 2020. doi: 10.1016/j.asoc.2020.106208.

Haoze Lv, Ning Lu, Shengcai Liu, Shaofeng Zhang, and Ke Tang. MuEvo: LLM-driven evolution of multi-heuristic ensemble. *arXiv preprint arXiv:2608.03636*, 2026a.

Haoze Lv, Ning Lu, Ziang Zhou, Yew-Soon Ong, and Shengcai Liu. AHD Agent: Agentic reinforcement learning for automatic heuristic design. *arXiv preprint arXiv:2605.08756*, 2026b.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 270–288. ACM, 2019. doi: 10.1145/3341302.3342080.

MiniMax. MiniMax-M1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Jean-Baptiste Mouret and Glenn Maguire. Quality diversity for multi-task optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 121–129, 2020. doi: 10.1145/3377930.3390203.

Su Nguyen, Yi Mei, and Mengjie Zhang. Genetic programming for production scheduling: A survey with a unified framework. *Complex & Intelligent Systems*, 3(1):41–66, 2017. doi: 10.1007/s40747-017-0036-x.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

NVIDIA. NVIDIA-Nemotron-3.5-Lightning-30B-A3B-BF16 model card. https://huggingface.co/nvidia/NVIDIA-Nemotron-3.5-Lightning-30B-A3B-BF16, 2026. Accessed 2026-09-23.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Tal Oved, Roi Pony, Oshri Naparstek, and Udi Barzelay. Evolution or illusion? Rethinking evaluation in LLM evolutionary search. *arXiv preprint arXiv:2609.19799*, 2026.

John Park, Yi Mei, Su Nguyen, Gang Chen, and Mengjie Zhang. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Applied Soft Computing*, 63:72–86, 2018. doi: 10.1016/j.asoc.2017.11.020.

Laurent Perron and Frédéric Didier. CP-SAT. https://developers.google.com/optimization/cp/cp_solver, 2026. Google OR-Tools, accessed 2026-09-23.

Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Claudio P. Santiago, Soo K. Kim, and Joanne T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, sixth edition, 2022. doi: 10.1007/978-3-031-05921-6.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—A KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

Junhao Qiu, Haoyang Zhuang, Fei Liu, Jianjun Liu, and Qingfu Zhang. EvoDR: Evolving dispatching rules via large language model for dynamic flexible assembly flow shop scheduling. *arXiv preprint arXiv:2601.15738*, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2026a. Model cards Qwen3.5-4B and Qwen3.5-9B, Apache-2.0 license.

Qwen Team. Qwen3.8-27B model card. https://huggingface.co/Qwen/Qwen3.8-27B, 2026b. Apache-2.0 license, accessed 2026-09-23.

Shrinivas Ramasubramanian, Daman Arora, Fahim Tajwar, Guanning Zeng, Qingyang Wu, Zhongzhu Zhou, Chenfeng Xu, Haiwen Feng, Yuda Song, Aarti Singh, Ruslan Salakhutdinov, J. Andrew Bagnell, Jeff Schneider, and Andrea Zanette. Tail-likelihood reinforcement learning. *arXiv preprint arXiv:2609.02987*, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Olivér Kiss, Sebastian Nilsson, and Rik Sarkar. The Shapley value in machine learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 5572–5579, 2022. doi: 10.24963/ijcai.2022/778.

Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007. doi: 10.1016/j.ejor.2005.12.009.

SchedMD. Multifactor priority plugin. https://slurm.schedmd.com/priority_multifactor.html, 2026. Accessed 2026-09-25.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Asankhaya Sharma. OpenEvolve: An open-source evolutionary coding agent. GitHub repository, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve. Accessed 2026-09-16.

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, Santa Clara, CA, 2024. USENIX Association.

Kevin Sim, Emma Hart, and Ben Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation*, 23(1):37–67, 2015. doi: 10.1162/evco_a_00121.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1348–1362, 2025. doi: 10.1109/HPCA61900.2025.00102.

Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *Advances in Neural Information Processing Systems*, volume 21, 2008. URL https://proceedings.neurips.cc/paper_files/paper/2008/hash/5751ec3e9a4feab575962e78e006250d-Abstract.html.

Anja Šurina, Amin Mansouri, Lars C. P. M. Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with LLMs: Evolutionary search meets reinforcement learning. In *Conference on Language Modeling*, 2025.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning. In *International Conference on Machine Learning*, 2026.

Shota Takashiro, Soichiro Nishimori, Paavo Parmas, Yongmin Kim, Kohsei Matsutani, Gouki Minegishi, Yusuke Iwasawa, Takeshi Kojima, and Yutaka Matsuo. On advantage estimates for max@K policy gradients. *arXiv preprint arXiv:2606.06080*, 2026.

Temoor Tanveer. LEVI: Stronger search architectures can substitute for larger LLMs in evolutionary search. *arXiv preprint arXiv:2605.09764*, 2026.

Minghao Tian, Yunfei Xie, and Chen Wei. How off-policy can GRPO be? Mu-GRPO for efficient LLM reinforcement learning. *arXiv preprint arXiv:2605.17570*, 2026.

Eva Vallada, Rubén Ruiz, and Jose M. Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3):666–677, 2015. doi: 10.1016/j.ejor.2014.07.033.

Niki van Stein and Thomas Bäck. LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793.

Christian Walder and Deep Tejas Karkhanis. Pass@K policy optimization: Solving harder reinforcement learning problems. In *Advances in Neural Information Processing Systems*, 2025.

Guoqing Wang, Chengran Yang, Xiaoxuan Zhou, Zeyu Sun, Bo Wang, David Lo, and Dan Hao. TestDecision: Sequential test suite generation via greedy optimization and reinforcement learning. *arXiv preprint arXiv:2604.01799*, 2026a.

Jianhong Wang, Yuan Zhang, Tae-Kyun Kim, and Yunjie Gu. Shapley Q-value: A local reward approach to solve global reward games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7285–7292, 2020. doi: 10.1609/aaai.v34i05.6220.

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. ThetaEvolve: Test-time learning on open problems. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=XHP568OuIl.

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. BurstGPT: A real-world workload dataset to optimize LLM serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 5831–5841. ACM, 2025. doi: 10.1145/3711896.3737413.

Matthew D. Webb. Reworking wild bootstrap-based inference for clustered errors. *Canadian Journal of Economics*, 56(3):839–858, 2023. doi: 10.1111/caje.12661.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008. doi: 10.1613/jair.2490.

Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 210–216, 2010. doi: 10.1609/aaai.v24i1.7565.

Xianliang Yang, Ling Zhang, Haolong Qian, Lei Song, and Jiang Bian. HeurAgenix: Leveraging LLMs for solving complex combinatorial optimization challenges. *arXiv preprint arXiv:2506.15196*, 2025.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 27144–27152, 2025. doi: 10.1609/aaai.v39i25.34922.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244, 2025.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=96zNuQrH9Y.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1621–1632, 2020.

Fangfang Zhang, Yi Mei, Su Nguyen, Mengjie Zhang, and Kay Chen Tan. Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, 25(4):651–665, 2021. doi: 10.1109/TEVC.2021.3065707.

Jianghua Zhang, Son Duy Dao, Wei Zhang, Mark Goh, Guodong Yu, Yan Jin, and Weibo Liu. A new job priority rule for the NEH-based heuristic to minimize makespan in permutation flowshops. *Engineering Optimization*, 55(8):1296–1315, 2023. doi: 10.1080/0305215X.2022.2085259.

Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the importance of evolutionary search in automated heuristic design with large language models. In *Parallel Problem Solving from Nature – PPSN XVIII*, volume 15149 of *Lecture Notes in Computer Science*, pp. 185–202. Springer, 2024. doi: 10.1007/978-3-031-70068-2_12.

Shaofeng Zhang, Shengcai Liu, Ning Lu, Jiahao Wu, Ji Liu, Yew-Soon Ong, and Ke Tang. LLM-driven instance-specific heuristic generation and selection. *arXiv preprint arXiv:2506.00490*, 2025.

Wentao Zhang, Haoyu Zhang, Xinke Jiang, Yuxuan Cheng, Yuhan Pan, Miao Li, Zhipeng Qiao, Tao Feng, Zhen Tao, and Dengji Zhao. Fishing out free riders: Shapley-based reward attribution for parallel reasoning via reinforcement learning. *arXiv preprint arXiv:2607.18979*, 2026.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025a.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte Carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 78338–78373. PMLR, 2025b.

XinLei Zhou, Jin Huang, Jie Yang, Xinyu Li, and Liang Gao. DSevolve: Enabling real-time adaptive scheduling on dynamic flexible job shop with LLM-evolved heuristic portfolios. *arXiv preprint arXiv:2603.27628*, 2026.

GUIDE TO THE APPENDIX

Appendix A places RuleForge among prior methods, and Appendix B proves what Section 3 says about the reward. Appendices C and D give the rules, goals and settings, and Appendix E gives the shared protocol, the statistics and the compute, with the paired intervals of Table 11 and Figure 9. Appendices F, G, H, I, J, K and L give the evidence behind the observations of Section 4, and Appendix M reports the earlier version of RuleForge.

A RELATED WORK IN MORE DETAIL

Table 3 marks which parts of RuleForge each closest prior method shares. TestDecision rewards each sampled test by its marginal gain over the current suite (Wang et al., 2026a), CALM scores a sample against the best parent rule in its prompt (Huang et al., 2026b), AHD Agent against a fixed baseline heuristic (Lv et al., 2026b), and ThetaEvolve only penalizes copies of programs in its database. Only RuleForge splits the gain a group adds to the whole library among its samples, on every goal.

Table 3: Parts of RuleForge in prior work. A set objective scores the set of rules a method keeps by its best member per instance, task or niche, or by what its members cover together. A set-level reward scores several answers together, from a group of samples or, as in VPO and ShapE-GRPO, from inside one response. A library-aware score rates a sample by its gain over the rules already kept, and ○ marks a reference that is only the best parent rule in the prompt or a penalty for copies, while under training the LLM it marks a trained selector only. The rows are FunSearch (Romera-Paredes et al., 2024), EoH (Liu et al., 2024a), ReEvo (Ye et al., 2024), MCTS-AHD (Zheng et al., 2025b), OpenEvolve (Sharma, 2025), ShinkaEvolve (Lange et al., 2026), LEVI (Tanveer, 2026), SATzilla (Xu et al., 2008), Hydra (Xu et al., 2010), EoH-S (Liu et al., 2026), NELLI (Sim et al., 2015), CMA-ME (Fontaine et al., 2020), HeurAgenix (Yang et al., 2025), InstSpecHH (Zhang et al., 2025), MuEvo (Lv et al., 2026a), DSevolve (Zhou et al., 2026), EvoTune (Šurina et al., 2025), CALM (Huang et al., 2026b), AHD Agent (Lv et al., 2026b), TestDecision (Wang et al., 2026a), ThetaEvolve (Wang et al., 2026b), TTT-Discover (Yuksekgonul et al., 2026), pass@k training (Chen et al., 2025), PKPO (Walder & Karkhanis, 2025), VPO (Bahlous-Boldi et al., 2026), SetPO (Li et al., 2026), MaxRL (Tajwar et al., 2026), TailRL (Ramasubramanian et al., 2026), ShapE-GRPO (Ai et al., 2026), GCPO (Chen et al., 2026a) and the free-rider attribution of Zhang et al. (2026).

Method	Set of rules	Set objective	Trains the LLM	Set-level reward	Library-aware score	Learned selector
FunSearch, EoH, ReEvo, MCTS-AHD OpenEvolve, ShinkaEvolve, LEVI						
SATzilla	✓	✓				✓
Hydra	✓	✓			✓	✓
EoH-S	✓	✓			✓	
NELLI	✓	✓			✓	
CMA-ME	✓	✓			✓	
HeurAgenix	✓		○			✓
InstSpecHH	✓					✓
MuEvo	✓					
DSevolve	✓	✓				✓
EvoTune			✓			
CALM			✓		○	
AHD Agent			✓			
TestDecision	✓	✓	✓		✓	
ThetaEvolve			✓		○	
TTT-Discover			✓	✓		
Pass@k training, PKPO, VPO, SetPO			✓	✓		
MaxRL, TailRL, ShapE-GRPO, GCPO			✓	✓		
Free-rider attribution			✓	✓		
RuleForge	✓	✓	✓	✓	✓	✓

Further related work. Genetic programming, deep reinforcement learning and language models also learn dispatching rules and scheduling policies (Branke et al., 2016; Zhang et al., 2021; 2020; Mao et al., 2019; Huang et al., 2026a; Qiu et al., 2026), LLM serving orders requests by hand-designed or learned policies (Sheng et al., 2024; Fu et al., 2024), and language models design systems policies in simulators (Cheng et al., 2025; Dwivedula et al., 2025; Hamadanian et al., 2026). Other methods evolve pools or ensembles of heuristics (Yang et al., 2025; Zhang et al., 2025; Lv et al., 2026a; Hart & Sim, 2016; Park et al., 2018), heuristics for several objectives at once (Yao et al., 2025) or whole metaheuristics (van Stein & Bäck, 2025) or train the model at test time on one open problem (Wang et al., 2026b; Yuksekgonul et al., 2026). Other credits for sets of samples train on the top quantile of a batch (Petersen et al., 2021), credit a sample's leave-one-out contribution to a kernel diversity (Li et al., 2026) or give each path of a parallel-reasoning response its Shapley value as a reward (Zhang et al., 2026). Multi-agent reinforcement learning credits each agent with its counterfactual contribution to a shared reward (Foerster et al., 2018) or with its Shapley value (Wang et al., 2020), as the Shapley credit does for the rules of a group. Online submodular maximization schedules heuristics with guarantees (Streeter & Golovin, 2008), and library learning grows reusable programs across tasks (Ellis et al., 2021). Quality-diversity search keeps an archive of diverse elites (Mouret & Clune, 2015; Fontaine et al., 2020), also with a language model judging quality and diversity (Bradley et al., 2023), and the multi-task MAP-Elites baseline that scores every candidate on every task keeps an archive whose score is our library value (Mouret & Maguire, 2020). In algorithm selection (Kerschke et al., 2019), SATzilla learns a selector over a fixed portfolio (Xu et al., 2008), ISAC configures a solver for each cluster of instances (Kadioglu et al., 2010), parallel portfolios are built by configuration (Lindauer et al., 2017), Hydra (Xu et al., 2010) and NELLI (Sim et al., 2015) grow a portfolio by what each member adds, and a solver's Shapley value measures what it adds to a portfolio (Fréchette et al., 2016). Benchmarks for LLM-designed heuristics (Liu et al., 2024b; Chen et al., 2026b) shaped our protocol, and Appendix J reports results across budgets and seeds, as Oved et al. (2026) recommend.

B Proofs for Section 3

The greedy bound. For one goal g, the map $L \mapsto s(L, g) = \max_{r \in L} s(r, g)$ is monotone, and adding a rule r raises it by $[s(r, g) - s(L, g)]_+$, which can only shrink as L grows. So the map is monotone and submodular, and so is F, a sum of such maps. Fix L and let $h(C) = F(L \cup C) - F(L)$ for a set C of survivors from S. Then h is monotone and submodular with $h(\emptyset) = 0$, admission adds the survivor of largest marginal gain in h at each step (Equation 4), and the greedy bound (Bilmes, 2022) gives $h(A) \geq (1 - 1/e) \max_C h(C)$ over the sets $C \subseteq S$ no larger than the greedy set A. Admission stops before K rules only when every remaining survivor has zero marginal gain, and then by submodularity no set of survivors gains more than A. Dropping survivors that duplicate a library rule removes only rules of zero marginal gain.

The library game and efficiency. Let U be the survivors of a group $\mathcal{N}$ and $\gamma(S) = F(L \cup (S \cap U)) - F(L)$ the gain in library value from admitting the survivors in $S \subseteq \mathcal{N}$. Since $s(L \cup C, g') = \max(s(L, g'), \max_{r \in C} s(r, g'))$, this gain is $\gamma(S) = \sum_{g' \in G} \max_{r_i \in S} \Delta_i(g')$ with $\max \emptyset = 0$, a sum over goals of games whose value is the largest marginal gain. The Shapley value (Rozemberczki et al., 2022) adds over games. On a goal g' the game's value is $\int_0^1 \mathbf{1}[\max_{r_i \in S} \Delta_i(g') \geq t]\, dt$. For each t, let T_t hold the $N(t)$ rules whose gain on g' is at least t. The game $\mathbf{1}[S \cap T_t \neq \emptyset]$ gives each rule of T_t the value $1/N(t)$ by symmetry and efficiency and every other rule 0 as a null player, and $N(t) = n - j + 1$ for $\Delta_{(j-1)}(g') < t \leq \Delta_{(j)}(g')$. Linearity then gives $\phi_i(g') = \int_0^{\Delta_i(g')} dt / N(t)$, which is Equation 6 and the form of TailRL's recurrence (Ramasubramanian et al., 2026). Efficiency makes the credits sum to $\gamma(\mathcal{N})$, symmetry gives rules with equal gains equal credit, and a goal the library meets has $s(L, g') = 1 \geq s(r, g')$, so it adds nothing to any credit.

Semivalue credits. A *semivalue* credit averages a rule's marginal gains over subsets of the other rules of its group, with total weight $q_k \geq 0$ on the subsets of size k and $\sum_k q_k = 1$ (Kwon & Zou, 2022),

$$R_q(r_i) = \sum_{k=0}^{n-1} q_k \binom{n-1}{k}^{-1} \sum_{S \subseteq \{r_j\}_{j \neq i},\ |S| = k} \big[\gamma(S \cup \{r_i\}) - \gamma(S)\big], \tag{8}$$

and the Shapley value, $q_k = 1/n$, gives $R_q = R_W$. Let $J_m(\theta) = \mathbb{E}\,\gamma(\{r_1, \ldots, r_m\})$ for m rules sampled independently from the writer $\pi_\theta(\cdot \mid g)$.

Proposition 1 *If the n rules of a group are sampled independently and each receives a semivalue credit R_q as its advantage, the expected policy-gradient update is*

$$\mathbb{E}\Big[\sum_{i=1}^{n} R_q(r_i)\,\nabla_\theta \log \pi_\theta(r_i \mid g)\Big] = \nabla_\theta \sum_{m=1}^{n} \frac{n\,q_{m-1}}{m}\,J_m(\theta), \tag{9}$$

and centering the credits by their group mean adds $(q_m - q_{m-1})\,\nabla_\theta J_m(\theta)$ to each term, with $q_n = 0$.

The Shapley value weighs every subset size alike, so R_W ascends $\sum_m J_m/m$, or $\sum_{m<n} J_m/m$ once centered. Per-sample credit, $q_0 = 1$, ascends only J_1, centered leave-one-out credit only J_{n-1}, and the best-of-k credit of PKPO (Walder & Karkhanis, 2025) a single J_k. On one goal the Shapley case is the result of TailRL (Ramasubramanian et al., 2026), which reduces to MaxRL for a binary gain (Tajwar et al., 2026), and it extends to the library value, which sums such games over goals.

Proof of Proposition 1. Let γ be any set function on rules with $\gamma(\emptyset) = 0$ that does not depend on θ, such as the library game, and let a semivalue give rule r_i of a group $\mathcal{N}$ of n rules the credit of Equation 8. Let the rules be sampled independently from $\pi_\theta(\cdot \mid g)$, and let Q_k be a set of k further independent samples. Given r_i, every coalition of k other rules is distributed as Q_k, so $\mathbb{E}[R_q(r_i) \mid r_i] = \sum_k q_k\,c_k(r_i)$ with $c_k(r) = \mathbb{E}[\gamma(Q_k \cup \{r\}) - \gamma(Q_k)]$, the expected marginal gain of r against k other rules. For $J_m(\theta) = \mathbb{E}[\gamma(Q_m)]$, the score-function identity over m independent samples and exchangeability give $\nabla_\theta J_m = m\,\mathbb{E}[\gamma(Q_m)\nabla_\theta \log \pi_\theta(r_1 \mid g)]$ for one of the samples r_1. In $\gamma(Q_m) = \gamma(Q_m \setminus \{r_1\}) + [\gamma(Q_m) - \gamma(Q_m \setminus \{r_1\})]$ the first term does not depend on r_1 and has zero expected product with the score, so $\nabla_\theta J_m = m\,\mathbb{E}[c_{m-1}(r_1)\nabla_\theta \log \pi_\theta(r_1 \mid g)]$. Hence

$$\begin{aligned}\mathbb{E}\Big[\sum_{i=1}^{n} R_q(r_i)\nabla_\theta \log \pi_\theta(r_i \mid g)\Big] &= n\,\mathbb{E}\big[\mathbb{E}[R_q(r_1) \mid r_1]\nabla_\theta \log \pi_\theta(r_1 \mid g)\big] \\ &= \sum_{k=0}^{n-1} n\,q_k\,\frac{\nabla_\theta J_{k+1}}{k+1} = \sum_{m=1}^{n} \frac{n\,q_{m-1}}{m}\nabla_\theta J_m,\end{aligned}$$

which for the Shapley value is $\nabla_\theta \sum_m J_m/m$, and $\mathbb{E}[R_W(r_i) \mid r_i] = \frac{1}{n}\sum_k c_k(r_i)$ is the distributional Shapley value of r_i (Ghorbani et al., 2020). Subtracting the group mean $\bar{R}_q$ from every credit subtracts $\mathbb{E}[\bar{R}_q \sum_i \nabla_\theta \log \pi_\theta(r_i \mid g)] = \nabla_\theta \mathbb{E}[\bar{R}_q]$. The average of c_k over r is $J_{k+1} - J_k$ with $J_0 = 0$, so $\mathbb{E}[\bar{R}_q] = \sum_{m=1}^{n}(q_{m-1} - q_m)J_m$ with $q_n = 0$, and centering adds $(q_m - q_{m-1})\nabla_\theta J_m$ to each term. For the Shapley value $\bar{R}_q = \gamma(\mathcal{N})/n$ by efficiency, which removes the term $m = n$. For a binary gain the centered Shapley credit is the practical estimator of MaxRL (Tajwar et al., 2026). Per-sample credit $\gamma(\{r_i\})$ has $q_0 = 1$ and gives $n\nabla_\theta J_1$, or $(n-1)\nabla_\theta J_1$ once centered. Leave-one-out credit $\gamma(\mathcal{N}) - \gamma(\mathcal{N} \setminus \{r_i\})$ has $q_{n-1} = 1$ and gives $\nabla_\theta J_n$, and since its group mean has expectation $J_n - J_{n-1}$, centering turns it into $\nabla_\theta J_{n-1}$. PKPO's best-of-k credit with its leave-one-out-minus-one baseline equals k/n times the semivalue credit with $q_{k-1} = 1$ (Walder & Karkhanis, 2025), and its other baselines change it only by a term computed from the other rules of the group, which leaves the expected update unchanged. These credits weighted by $1/k$ sum to the Shapley credit sample by sample, so their harmonic mixture over $k = 1, \ldots, n$ is the Shapley credit without centering, which ascends $\sum_{m \le n} J_m/m$, and its row in Table 11 tests centering. Applied goal by goal and summed over goals, PKPO's estimator ascends J_k for the library game. Finally, by the uniqueness of the Shapley value under the efficiency, symmetry, null-player and linearity axioms (Rozemberczki et al., 2022), it is the only semivalue that is efficient for every γ. Already the library games in which j rules of a group share one gain on one goal, for $j = 1, \ldots, n$, force $q_k = 1/n$, so only the weights $1/m$ make the credits sum to the gain in library value.

The weights of a binary gain. For a binary gain with success probability p on a goal, $J_m = 1 - (1-p)^m$, and the centered update of R_W weighs $\nabla_\theta p$ by $(1 - (1-p)^{n-1})/p$, per-sample credit by $n-1$ and a single J_k by $k(1-p)^{k-1}$. For a graded gain these weights hold at each level t of gain, since on a goal $\sum_{m<n} \nabla_\theta J_m/m = \int_0^1 \frac{1-(1-p_t)^{n-1}}{p_t}\,\nabla_\theta p_t\,dt$, where p_t is the probability that a rule's gain reaches t (Ramasubramanian et al., 2026). Figure 5 draws them relative to their value at $p = 0$,

as the weight views of MaxRL and TailRL do on one prompt (Tajwar et al., 2026). Without centering the Shapley and per-sample weights are $(1-(1-p)^n)/p$ and n, which agree when successes are rare, while the Shapley weight is up to n times smaller on goals the writer already solves.

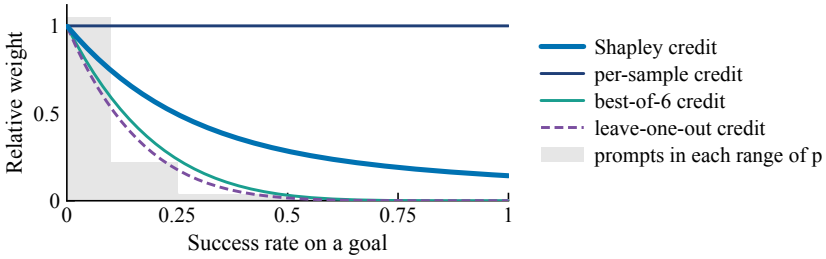

Figure 5: Each credit's weight on a goal the writer improves with probability p, relative to its weight at $p = 0$, exact from Proposition 1 for a binary gain and $n = 8$, with the Shapley and leave-one-out credits centered as in the recipe. The shading gives the distribution of p on the prompting goal among the prompts of generation 1, whose bins hold 70, 22, 6 and 2 percent of them.

Scope. Proposition 1 holds for the plain policy gradient on fresh on-policy samples with the credits as advantages, and after centering efficiency acts only through the group mean $\gamma(\mathcal{N})/n$ that centering removes. Four parts of the recipe depart from it. GSPO's length-normalized ratio scales each score by $1/|r_i|$, so the proposition holds exactly only for rules of equal length. Dynamic sampling removes only zero advantages of the centered credits and so rescales their expected update, while for the uncentered Shapley credit and the PKPO arms it also skips groups of equal nonzero credits, under 1 percent of the groups of generation 1. A replayed rule is not a fresh sample, and one that still adds value lowers the Shapley and leave-one-out credits of the fresh rules of its group but not their per-sample credit. The proof uses only independent players and a γ that does not depend on θ, so it covers replayed rules as players whose distribution does not depend on θ, and the fresh rules' centered Shapley credits then ascend $\sum_{m<n} \bar{J}_m/m$, with $\bar{J}_m$ the expected gain of m rules drawn from the group. We do not analyze clipping or the replayed rules' own off-policy terms, and Appendix I tests the credits under plain on-policy GRPO, where the proposition is exact. We checked the proposition by exact enumeration over small groups and softmax policies, for one goal and for the library value over two goals, for the Shapley and Banzhaf values and the per-sample, leave-one-out and single-size credits, with and without centering.

Relation to admission. Summed over the training goals, a rule's marginal gains are its admission gain, $\sum_{g' \in G}[s(r,g') - s(L,g')]_+ = \mathrm{gain}(r \mid L)$ by Equation 4, and the Shapley credit shares this sum within the group on every goal, so a rule written for one missed goal is credited on the other goals it helps. The Shapley credit on the prompting goal alone, TailRL's credit (Ramasubramanian et al., 2026) on that goal's gain over the library, keeps only the prompting goal's term and ignores the 38 percent of a group's gain in generation 1 that lands on other goals. Since $\phi_i(g') = \int_0^{\Delta_i(g')} dt/N(t)$ and $N(t)$ can only grow as rules join the game, a rule's credit in its group is at least its Shapley share of what all survivors of the generation would add. Equality holds on every goal where no survivor outside the group adds value, which covers 64 percent of the credit of generation 1.

The objective and the next admission. Proposition 1 names the objective each credit ascends, and the next result ties the Shapley credit's objective to the gain that admission can take up.

Proposition 2 *Let γ be monotone and submodular with $\gamma(\emptyset) = 0$, and let J_m be as in Proposition 1. Then J_m/m does not increase with m, and for $n \geq 2$*

$$\frac{n-1}{n} J_n \;\leq\; \sum_{m=1}^{n-1} \frac{J_m}{m} \;\leq\; H_{n-1}\, J_n, \qquad H_{n-1} = \sum_{m=1}^{n-1} \frac{1}{m}. \tag{10}$$

A writer $\hat{\theta}$ that maximizes this sum over a set of writers has $J_1(\hat{\theta}) \geq \frac{H_{n-1}}{n-1} J_1(\theta_1)$ and $J_n(\hat{\theta}) \geq \frac{n-1}{nH_{n-1}} J_n(\theta_n)$, where θ_1 and θ_n maximize J_1 and J_n over the same set. If the survivors of a group $\mathcal{N}$ join the pool of the next admission from L, which adds up to $K \geq n$ rules greedily, that admission raises F by at least $(1 - 1/e)\,\gamma(\mathcal{N})$.

Proof of Proposition 2. Let $r_1, r_2, \ldots$ be sampled independently from $\pi_\theta(\cdot \mid g)$, let $S_k = \{r_1, \ldots, r_k\}$ and $\delta_k = \mathbb{E}[\gamma(S_k) - \gamma(S_{k-1})]$, so that $J_m = \sum_{k \leq m} \delta_k$. By submodularity, $\gamma(S_{k+1}) - \gamma(S_k) \leq \gamma(S_{k-1} \cup \{r_{k+1}\}) - \gamma(S_{k-1})$, whose expectation is δ_k since r_k and r_{k+1} are exchangeable. So $\delta_{k+1} \leq \delta_k$, and J_m/m, the mean of the first m increments, does not increase with m. Then $J_m/m \geq J_{n-1}/(n-1)$ for $m \leq n-1$, so the sum is at least $J_{n-1} \geq \frac{n-1}{n} J_n$, and monotonicity gives $J_m \leq J_n$, so the sum is at most $H_{n-1} J_n$. Write O for the sum. Since $J_1 \leq J_m \leq m J_1$, O lies between $H_{n-1} J_1$ and $(n-1) J_1$, so $J_1(\hat{\theta}) \geq O(\hat{\theta})/(n-1) \geq O(\theta_1)/(n-1) \geq H_{n-1} J_1(\theta_1)/(n-1)$, and Equation 10 gives $J_n(\hat{\theta}) \geq O(\hat{\theta})/H_{n-1} \geq O(\theta_n)/H_{n-1} \geq (n-1) J_n(\theta_n)/(nH_{n-1})$. A single J_k gives the factors $1/k$ and k/n by the same steps. For the admission, $h(C) = F(L \cup C) - F(L)$ is monotone and submodular, the survivors of $\mathcal{N}$ form a set of at most $n \leq K$ rules in the pool with h equal to $\gamma(\mathcal{N})$, and the greedy bound gives $h(A) \geq (1 - 1/e)\,\gamma(\mathcal{N})$ for the admitted set A. If admission stops before K rules because no survivor adds value, every rule of U has zero marginal gain on A, so $h(A) = h(A \cup U) \geq \gamma(\mathcal{N})$ by submodularity and monotonicity. Both bounds of Equation 10 are tight, the upper one for a writer that always samples one rule, where $J_m = J_n$ for every m, and the lower one in the limit where every rule's gain is rare and J_m grows linearly in m.

With $n = 8$, the centered Shapley objective lies between $0.875\, J_n$ and $2.59\, J_n$, so the next admission from a pool holding a group's survivors gains at least $(1 - 1/e)/H_{n-1}$, about 0.24, times that objective in expectation. The Shapley factors are 0.37 and 0.34, while per-sample credit guarantees only $1/8$ of the largest J_n, and the best-of-4, best-of-6 and leave-one-out credits only $1/4$, $1/6$ and $1/7$ of the largest J_1, which matters because a group has few survivors and admission keeps $K = 8$ rules across all groups. Each factor is attained in the limit in a choice between a writer that always samples the same rule and one that spreads its samples over rules that gain on different goals. For weights $w_m \geq 0$, a maximizer of $\sum_m w_m J_m$ gets the factors W_0/W_1 and $W_1/(nW_0)$, with $W_0 = \sum_m w_m$ and $W_1 = \sum_m m\, w_m$, whose product is $1/n$, so no weighting guarantees more than $1/\sqrt{8} = 0.35$ of both. Among uncentered credits efficiency singles out the weights $1/m$, while after centering it fixes only the group mean that centering removes.

A two-kind example. In Figure 6 a writer samples a bold rule with probability p and a safe rule otherwise, with $n = 8$ and one goal. A bold rule gains 1 with probability 0.3 and 0 otherwise, a safe rule always gains 0.5, and the writer cannot tell which bold rules will succeed. Per-sample credit ascends J_1, maximized at $p = 0$, where the best of six samples gains 0.5. Leave-one-out credit ascends J_8, maximized at $p = 0.92$, where the best of six gains 0.89 but one sample only 0.32 on average. The Shapley mixture peaks at $p = 0.64$, where the best of six gains 0.86 and one sample 0.37, so it keeps 96 percent of leave-one-out credit's best-of-six gain with a 17 percent higher mean gain. The best of six is what a goal's six proposals realize, and the mean of one is what a rule written for one goal offers the other goals it may improve. With centering the maxima move to $p = 0.62$ and $p = 0.90$, where leave-one-out credit ascends J_7. These values are exact.

C Rules, goals and trace windows of inference scheduling

This appendix defines the rules, goals and trace windows of inference scheduling, and Appendix H does the same for the other four problems.

Rules are expressions in the grammar

$$\begin{aligned} e \;::=\; & f \mid c \mid d \mid e \circ e \mid \min(e,e) \mid \max(e,e) \mid \mathrm{abs}(e) \\ & \mid \mathrm{pow}(e,e) \mid \mathrm{round}(e) \mid \mathrm{where}(e \lesseqgtr e,\, e,\, e), \qquad \circ \in \{+,-,\times,\div\}, \end{aligned} \tag{11}$$

where f is a feature of the job or of its tenant, c is a constant and $d = 10^{-6}$ keeps divisions finite.

Rules compile from Equation 11 under character, token and name guards and a whitelist of functions, so a rule reads only the features of Table 4 and calls only min, max, abs, pow, round and where.

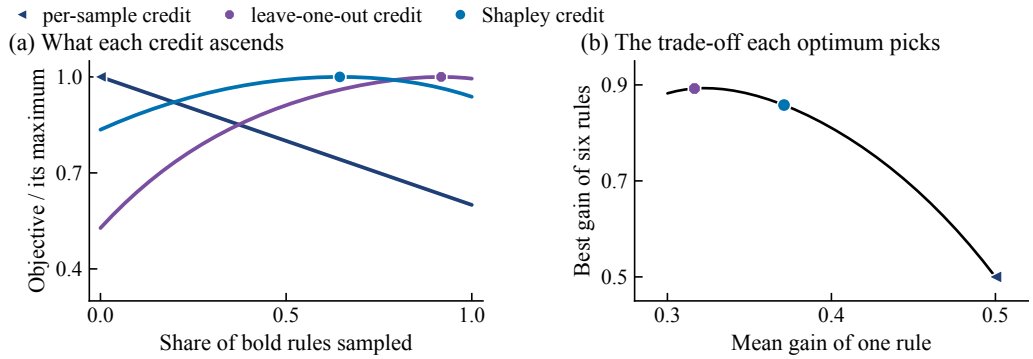

Figure 6: An exact example of Proposition 1 with $n = 8$. With probability p the writer samples a bold rule, which gains 1 with probability 0.3 and 0 otherwise, and else a safe rule, which gains 0.5. (a) The objective each credit ascends without centering, J_1 for per-sample credit, J_8 for leave-one-out credit and $\sum_m J_m/m$ for Shapley credit, each divided by its maximum, with its maximizer marked. (b) The best gain of six rules against the mean gain of one rule as p runs from 0 to 1, with each credit's maximizer marked.

The OR-Library job-shop language drops pow and round and adds an exponential with a clipped argument. Tables 6 and 7 list the hand-written rules, and Table 5 defines the metrics.

Each window holds three slices of 1,000 requests, and an evaluation averages every metric over the three. The 15 training and 9 development windows start every 4,500 rows within the first 120,000 rows of the trace, and the released files list each window's rows, tenant configuration and seeds. In those rows conversation logs have median prompt and response lengths of 514 and 242 tokens and API logs of 70 and 3. The 176 held-out windows come from 44 segments after row 120,000, which no earlier study used, half of them from conversation logs, each under four tenant configurations fixed before the final runs, with 16 goals per window. The selector reads a goal's thresholds and the statistics of the previous window of the same segment and configuration, and rule features are normalized by running means over the requests that have arrived. With A the requests that start before the slice ends at time T, request i from tenant κ_i has a work, a wait and, for each tenant, a realized share of served work,

$$c_i = \alpha h_i + \beta o_i, \qquad \omega_i = \min(t_i, T) - a_i, \qquad \psi_\kappa = \frac{\sum_{i \in A,\, \kappa_i = \kappa} c_i}{\sum_{i \in A} c_i}, \tag{12}$$

where h_i and o_i are its prompt and response tokens, a_i and t_i its arrival and start, and α and β are fit to the main model's prefill and decode times in vLLM. The share metrics are the L1 and L2 distances of the shares ψ_κ from equal shares. A slice ends at its last arrival, when its offered work, the sum of c_i, is three to six times what the slots can serve.

Held-out and development goals constrain two or three of the four metrics, and every wait metric counts all requests, an unserved request waiting until its slice ends. Each threshold is drawn uniformly from the range of the hand-written rules' outcomes on its window, widened by a tenth on each side. Goals alternate between ones some hand-written rule meets and ones no hand-written rule meets but some meets on all thresholds but one. So the hand-written rules meet half of the held-out goals, and each further goal adds 0.036 points of the 2,816 held-out goals. The 720 training goals are the 120 goals of the training windows drawn this way and 600 goals whose one to three thresholds come from quantiles of the hand-written rules' outcomes. No archive rule enters any threshold, while the goal-reward loop drew those 600 thresholds from the hand-written rules plus 12 archive rules, a setting Appendix J reruns.

Every graded goal, training or held-out, has one *relaxed metric* k^*, the metric with the smallest relative miss among hand-written rules that meet every other threshold. With u the best hand-written

value on k^* among rules that meet the other thresholds, the graded score of Section 2 is

$$s(r,g) = \begin{cases} 1 & \text{if } v(r,g) = 1, \\ \min\big(0.99,\ \max(0,\ (u - \mu_{k^*}(r,x))/|u|)\big) & \text{if } r \text{ misses only } \tau_{k^*}, \\ 0 & \text{otherwise,} \end{cases} \tag{13}$$

written for a metric where lower is better, with the sign flipped for a metric where higher is better. Its middle case, capped so that a near miss never counts as a met goal, is the fraction by which r improves on the best hand-written value, and a rule that misses a different single threshold gets no credit. Fixing the relaxed metric per goal makes $s(r,g)$ depend only on the rule's own outcome, so F is one fixed set function over all rules. Crediting each rule on its own nearest miss instead leaves the order of the methods unchanged, with RuleForge's chosen rule meeting 70.6 percent of held-out goals against 64.7 for the best other method. A library's oracle graded score applies Equation 13 to all its rules at once, averaged over the 1,408 held-out goals no hand-written rule meets.

Table 4: The features a rule can read, computed for one waiting request or ready operation when a slot or machine frees. Inference times are divided by the running mean work of the requests that have arrived, so no feature reads a later request. The work counts the response as well as the prompt, so a rule sees each request's full size, which a live server would have to predict.

Feature	Meaning
Inference domain, eight features of a waiting request and its tenant	
`deficit`	the tenant's target share minus its realized share of served work
`tokens`	the request's work c_i of Equation 12, which weighs its prompt and response tokens, divided by the running mean work of the requests that have arrived
`wait`	the time since the request arrived
`target`	the tenant's target share, one over the number of tenants
`tput`	the tenant's realized share of served work so far
`qshare`	the tenant's share of the queued requests
`backlog`	the tenant's share of the queued work
`agemax`	the wait of the tenant's oldest queued request
Job-shop dispatching, sixteen features of a ready operation, its job and the shop	
`p`	the operation's processing time
`wr`	the job's remaining work, including this operation
`nr`	the job's remaining operations
`done`	the job's completed work
`due`	the job's due date
`slack`	the due date minus the current time minus the remaining work, which can be negative
`wait`	the time the operation has been ready
`now`	the current time
`tq`	the number of operations ready on this machine
`nq`	the number of operations queued at the job's next machine
`wnq`	the work queued at the job's next machine
`np`	the processing time of the job's next operation, zero for the last operation
`tw`	the job's total work
`k`	the due-date tightness of the unit
`n`	the number of jobs
`m`	the number of machines

D IMPLEMENTATION DETAILS

Figure 7 shows the writer's prompt.

The writer's supervised start. The writer first fine-tunes for two epochs on every pair of a hand-written rule and a training goal it meets, and after admission in generations 0 and 1 on its own admitted rules paired with the training goals they newly meet, a form of expert iteration in which the verifier decides what the writer imitates. No archive rule enters any writer or selector data, training threshold or method's start. The variant with archive rules in the supervised start (Table 20 and Appendix J) adds the goal-reward loop's targets, for each training goal no hand-written rule meets

Table 5: The metrics a goal constrains, one to three per goal and two or three per held-out inference goal. A rule meets a goal when every constrained metric is on the allowed side of its threshold within 10^{-9}, with inference metrics averaged over a window's three slices.

Metric	Goal form	Definition
Inference domain		
L2 share error	$\leq$ threshold	the Euclidean distance between the tenants' realized and target shares of served work, which counts the full work of every request that starts by the slice's end, since a started request runs to completion without preemption
L1 share error	$\leq$ threshold	the sum over tenants of the absolute gap between realized and target share
Wait spread	$\leq$ threshold	the population standard deviation, across every tenant with a request in the slice, of that tenant's mean wait over all its requests, divided by the mean work of the slice's requests, where an unserved request waits until the slice ends
Served fraction	$\geq$ threshold	the share of the slice's requests that start by the slice's end
Job-shop dispatching		
Makespan	$\leq$ threshold	the latest completion time
Mean flow time	$\leq$ threshold	the mean completion time, with every job released at time zero
Total tardiness	$\leq$ threshold	the sum over jobs of completion time minus due date, floored at zero
Maximum tardiness	$\leq$ threshold	the largest tardiness of any job
Tardy fraction	$\leq$ threshold	the share of jobs whose tardiness exceeds 10^{-9}
Utilization	$\geq$ threshold	the total processing time divided by the number of machines times the makespan

Table 6: The ten hand-written inference rules, five simple and five in the full grammar, all compiled through the same path. A higher score is admitted first.

Name	Form	Body
`fifo`	simple	`wait`
`fair_tree`	simple	`deficit`
`sjf`	simple	`0 - tokens`
`deficit_per_token`	simple	`deficit / (tokens + d)`
`deficit_plus_wait`	simple	`deficit + 0.1 * wait`
`rich_gate`	full	`where(deficit > 0.02, deficit / (tokens + d), 0.1 * wait - 0.5 * tput)`
`rich_age`	full	`deficit / (tokens + d) + 0.05 * agemax`
`rich_over`	full	`where(tput > target, 0.1 * wait - tput, deficit / (tokens + d))`
`rich_backlog`	full	`where(backlog > 0.3, deficit / (tokens + d), deficit + 0.1 * wait)`
`rich_floor`	full	`max(deficit, 0) / (tokens + d) + 0.02 * wait`

the first archive rule that meets it, or else the one with the best value on the relaxed metric among those that meet the other thresholds.

The selector's reward. Let $w = (1, \frac{1}{2}, \frac{1}{4})$ be the position weights, so the first listed rule, the one that runs, counts most. On a goal the library can meet, the reward is w_{pos} for the position of the first listed rule that meets it, or 0. On a goal it cannot meet,

$$R_\sigma = w_{\mathrm{pos}}\,\mathbf{1}[\mathrm{pass}] + \lambda \min\Big(1,\ \max\Big(0,\ \frac{u - u_{\mathrm{list}}}{u - u_{\mathrm{lib}}}\Big)\Big), \tag{14}$$

where pos is the position of the first listed rule that passes and $\lambda = 1$. A listed rule passes when it meets every other threshold of the goal and comes within 2 percent of the library's best value on the relaxed metric, and u, u_{lib} and u_{list} are the best values on that metric among the hand-written rules, the library and the list, each over rules that meet the other thresholds. The same reward trains every selector of the final recipe, while the goal-reward loop used other weights (Table 22) and a pass test that did not check the other thresholds (Appendix M).

Table 7: The 20 classical dispatching rules of the OR-Library job shop, where a higher score dispatches first. `atc` scales slack by twice the operation's processing time rather than k times the queue mean, a simplification of the standard ATC rule (Pinedo, 2022).

Name	Family	Body
`spt`	processing time	`0 - p`
`lpt`	processing time	`p`
`fifo`	arrival	`wait`
`lifo`	arrival	`0 - wait`
`mwkr`	work remaining	`wr`
`lwkr`	work remaining	`0 - wr`
`mopnr`	operations remaining	`nr`
`lopnr`	operations remaining	`0 - nr`
`edd`	due date	`0 - due`
`slack`	slack	`0 - slack`
`cr`	critical ratio	`0 - (due - now) / (wr + d)`
`mdd`	modified due date	`0 - max(due, now + wr)`
`atc`	apparent tardiness cost	`exp(0 - max(slack, 0) / (2 * p + d)) / (p + d)`
`covert`	cost over time	`max(0, 1 - max(slack, 0) / (2 * wr + d)) / (p + d)`
`spt_winq`	processing time plus next queue	`0 - (p + wnq)`
`twopt_winq`	processing time plus next queue	`0 - (2 * p + wnq)`
`pt_wkr`	processing time over work	`0 - p / (wr + d)`
`spt_slack`	weighted hybrid	`0 - p - 0.1 * slack`
`slack_per_op`	slack per operation	`0 - slack / (nr + d)`
`mwkr_spt`	weighted hybrid	`wr - 5 * p`

```
You are choosing the admission rule for a multi-tenant fair-share GPU inference service
from workload telemetry alone; no outcome measurements are available.  When a slot
frees, the host applies the chosen rule to pick the next waiting request; the host itself
enforces weighted fair-share accounting and never preempts a running request.  Available
rules (name = priority expression over the features deficit tokens wait target tput qshare
backlog agemax; higher score = admit next):
  fifo = wait
  fair_tree = deficit
  sjf = 0 - tokens
  ... the other hand-written and library rules
Workload telemetry for the upcoming window:
  tenants:  12, target_skew:  uniform+heavy-head, slots:  8, offered_load_rho:  3.0,
tokens_p50:  68.0, tokens_p90:  78.0, arrival_cv:  2.55, requests_per_window:  1000
Operator goals (ALL must hold):
  G1.  tenant share-attainment error (L1) at most 0.0154
  G2.  at least 0.4062 of requests served within the window
No known rule meets every goal here.  The closest, ⟨rule⟩, meets every other goal and
reaches ⟨metric⟩ = ⟨value⟩ against the goal of ⟨threshold⟩.  Write a rule that meets every
goal, or one that moves ⟨metric⟩ further toward the goal while keeping the other goals.
Predict from telemetry alone.  Always commit to exactly one rule, written as one
arithmetic expression over the features above ... Answer with ONE line of JSON:
  {"feasible":  true, "rule":  "<expression>", "candidates":  ["<expression>", ...]}
```

Figure 7: The writer's prompt for one inference training goal the library misses, abridged. The rule list shows the current library. The telemetry summarizes the goal's window, here from API logs with short responses, and its requests_per_window counts one of the window's three slices. The closest-miss paragraph names the library rule that comes closest to the goal. The final recipe keeps only the rule field of each answer, and the selector sees the same context and answers with names of library rules.

The writer's training. Each GRPO group holds 8 rules for one missed goal, of which replayed rules fill 1.6 slots on average, so a run draws 5,120 fresh samples in its 6,400 group slots. The loss follows Dr. GRPO (Liu et al., 2025) with DAPO's dynamic sampling (Yu et al., 2025), GAPO's clipping (Jia et al., 2026) and GSPO's sequence ratio (Zheng et al., 2025a) normalized by length (Table 8). Verified rules of earlier generations are replayed in the groups of their goal with their cached verdicts, which makes part of each group off-policy, like the stale rollouts that GRPO-style training tolerates (Tian et al., 2026), at no evaluation cost, while mutation rules, which have no log-probability under the writer, are never replayed.

Which runs stand behind each result. RuleForge's inference results come from five runs of the final recipe on Nemotron 3.5 Lightning (NVIDIA, 2026), seeds 0 to 4, each stopped at 30,833 counted evaluations and listed seed by seed in Figure 8. Each variant of Tables 19 and 20 changes one part of the recipe and runs five seeds, sharing each seed's generation 0 with RuleForge when it changes the writer's reward, which first acts after admission in generation 0. The four other problems run five seeds per method, ten for the full recipe and the prompting-goal arm on bin packing and the traveling salesman problem, the OR-Library job-shop library comes from five runs with up to 12 admissions per generation, and the other models of Table 18 run five seeds each. Tables 8, 9 and 10 give the settings, and Table 22 those in which the goal-reward loop differs.

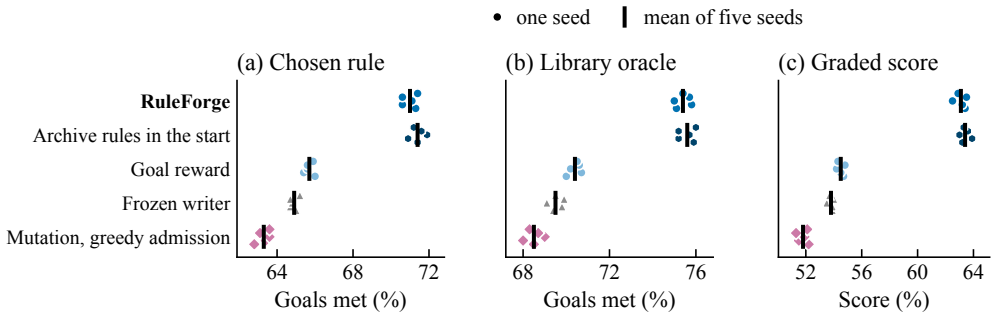

Figure 8: The inference runs behind Tables 1, 19 and 20, seed by seed, for RuleForge, three variants that each change one part of it and mutation with greedy admission, five seeds each on Nemotron 3.5 Lightning, every run stopped at 30,833 counted evaluations. Each mark is one seed and each bar the mean of the five. (a) The share of the 2,816 held-out goals met by the rule the selector picks, (b) the share some rule of the library meets and (c) the graded score on the 1,408 held-out goals no hand-written rule meets.

E Experimental protocol, budgets and compute

One admission and one selector for every method. Every method starts from the hand-written rules, and none uses the rule archive. Each LLM-driven method keeps its own search and published prompts, changed only to ask for an expression in our grammar. It seeds its population with the hand-written rules and scores a rule by its gain in library value on the goals of the windows it was evaluated on. CALM, whose prompt names no goal, compares a rule with its best parent (Appendix I) on the missed goal drawn for its prescreen window. Baselines run from their authors' code or LLM4AD (Liu et al., 2024b), except DSevolve, MuEvo and InstSpecHH, which we implement from their papers, as we do genetic programming, grammar mutation and the DRL dispatcher of Luo (2020). CALM and EvoTune start from the base model as published, without RuleForge's supervised start, CALM and ThetaEvolve train the same rank-32 LoRA adapters with sequence-level ratios as RuleForge's writer, and every method is tuned over the grids of Table 12. A candidate is first evaluated on the window of a goal its library misses, and on every training window only if it meets or improves a missed goal there. Greedy admission adds up to 8 survivors at each of three points with the same evaluation counts for every method, so an inference library holds at most 34 rules, and the selector recipe of Appendix D trains on each library. Prescreened on every training window instead, CALM and EoH-S meet 65.3 and 64.5 percent of the held-out goals, and with the

Table 8: Writer and loop settings of one generation of the final recipe, in inference scheduling and the OR-Library job shop unless a row says otherwise, with the changes of Appendix H on the other problems. s is the run seed and t the generation.

Step	Setting
Start	The library starts as the hand-written rules alone, the ten inference rules or the 20 classical rules of the OR-Library job shop, with no searched rule and no archive rule.
Writer start	Two epochs of supervised fine-tuning with the loss on the completion only, on every pair of a hand-written rule and a training goal it meets, among the 720 inference training goals or the 1,030 OR-Library job-shop training goals, with the run seed passed to the trainer. No archive rule enters the data.
Expert iteration	After admission in generations 0 and 1 and before the writer's GRPO, the same fine-tuning on the writer's admitted rules, each paired with the training goals it newly meets.
Prompted goals	Every training goal the current library misses that has a relaxed metric. Goals the library meets are not prompted.
Writer prompt	The goal's prompt with the window summary and thresholds, the current library listed as name and expression, the closest library rule with its measured value on the missed metric and the threshold, and a request for one JSON line with a verdict, a rule and candidate rules that meet every threshold or move the missed metric further toward it.
Sampling	Six bodies per prompt at temperature 1.0 and top-p 0.95, at most 160 new tokens, batch 16, sampler seed $s + t$. A proposal counts only if it parses to a nonempty rule.
Grammar mutation	Exactly as many candidates as parsed writer proposals, each one mutation of a random library rule assigned to a random training window. The mutation replaces one numeric literal by one of 13 values or scales it by 0.5 or 2 with probability 0.5, swaps one feature for another with probability 0.3, and otherwise appends a weighted feature.
Verification	Candidates are deduplicated by body, and a GRPO sample that repeats an evaluated body reuses its verdicts. Each body is evaluated on the window that proposed it and survives if, on some goal of that window the library misses, it meets every threshold, or meets every other threshold and beats the library's best value on the relaxed metric by more than 2 percent. Survivors are evaluated on all 15 training windows, and a body that fails on any window is dropped.
Admission	Greedy on the value F of Equation 3. The survivor with the largest marginal gain of Equation 4 is admitted and the gains are recomputed, up to 8 admissions per generation in inference scheduling and 12 in the OR-Library job shop. A survivor of zero gain is never admitted.
Duplicate check	A survivor whose outcomes on every training window and metric, rounded to six decimals, equal those of a hand-written, library or admitted rule is discarded.
Proposer	An admitted rule is credited to the source that proposed it first, writer or mutation, and is named `x01`, `x02` and so on in admission order.
Writer GRPO	After admission in generations 0 and 1, 100 GRPO steps with group-mean centering and no division by the spread (Liu et al., 2025) on prompts for the training goals the enlarged library misses, groups of 8 rules for one goal, batch 8 with 4 accumulation steps, a learning rate set per model by a preregistered sweep over 5×10^{-6} and 10^{-5}, no KL term, completions of at most 160 tokens, seed $s + 100 + t$, and a fresh LoRA adapter of rank 32 on the merged writer. Groups whose rewards are all equal are skipped, the clipping range grows with the advantage, and each rule verified in an earlier generation is replayed off-policy with its cached verdicts. A replayed rule's sequence ratio uses the log-probability recorded when it was first sampled, clipped like any other. Groups are sampled at temperature 1.0 without top-p truncation, so every fresh rule is a draw from the writer's policy. Importance ratios are taken per sequence for every base model (Zheng et al., 2025a), as GSPO recommends for mixtures of experts such as Nemotron 3.5 Lightning and gpt-oss-20b.
Writer reward	The marginal gain $[s(r, g') - s(L, g')]_+$ of each surviving rule of a group on every training goal g' against the current library L, split into Shapley credit goal by goal by Equation 6 and summed over goals, so the credits of a group sum to the gain in library value from admitting its survivors. Rules that do not survive score 0. A rule whose every metric lies within 10^{-6} relative of some library rule on that window scores 0. Group advantages are not rescaled. Copies of one body within a group are separate players with equal gains and split that body's share equally, as Proposition 1 assumes.
Frozen writer	Zero writer GRPO steps, so the generation-0 writer proposes in every generation.
Generations	Three per run.

Table 9: Selector settings of the final recipe, with the OR-Library job-shop selectors in the last two rows and each other problem's selector input in Appendix H.

Step	Setting
Training rows	The 720 training goals with answers recomputed under the grown library, keeping the goals that have an exact target.
Selector input	A window summary with the number of tenants, slots, offered load, tenant stickiness, the 50th and 90th percentiles of token length, the coefficient of variation of inter-arrival gaps, the request count and a constant skew label, followed by the goal's thresholds and the whole library listed as name and expression.
Supervised targets	On a feasible goal, every library rule that meets it, ordered by the margin it leaves on the thresholds, with one committed first pick. On an infeasible goal, ranked best-effort rules. Every row adds one two-turn revision example with the loss on the second answer.
Supervised stage	LoRA rank 32, alpha 64, dropout 0.05 on the attention, MLP or expert, and Mamba-2 input and output projections of each model, learning rate 10^{-4}, batch 1 with 16 accumulation steps, maximum length 3,072, bf16, six epochs.
GRPO stage	Dr. GRPO loss, groups of 8, batch 8 with 4 accumulation steps, a learning rate set per model by a preregistered sweep over 5×10^{-6} and 10^{-5}, temperature 1.0, completions of at most 200 tokens, no KL term, clipping at 0.2 and 0.28, cosine schedule with 10 warm-up steps, truncated completions masked, a fresh LoRA adapter of rank 32 and alpha 64 on the merged supervised model, 400 steps.
GRPO reward	The set reward of Equations 7 and 14 with position weights $w = (1, \frac{1}{2}, \frac{1}{4})$ and weight $\lambda = 1.0$ on its second term, so a list scores the weight of its first rule that meets a goal the library can meet, or 0, and up to 2 on a goal it cannot meet. A near miss counts only when the listed rule meets every other threshold and comes within 2 percent of the library's best value on the relaxed metric, relative to that value.
Evaluation	Greedy decoding, at most 300 new tokens, thinking disabled, one turn. The first entry of the answer is the chosen rule, and the first three form the shortlist.
OR-Library job-shop supervised	The same supervised recipe on RuleForge's OR-Library job-shop library or on the 20 classical rules. Over the classical rules, up to three near-miss rules are appended to short lists on 548 of the 1,030 training rows.
OR-Library job-shop GRPO	Dr. GRPO, groups of 8, temperature 1.0, a learning rate set per model by a preregistered sweep over 5×10^{-6} and 10^{-5}, 400 steps from the same run's supervised checkpoint. The set reward of Equations 7 and 14 with the same weights $w = (1, \frac{1}{2}, \frac{1}{4})$, where the library is the selector's rule set, and the first-rule reward credits only the first listed rule.

writer's goal prompt 65.6 and 64.9, all below RuleForge's 71.0. RuleForge alone adds grammar mutation as a second proposer. Without it RuleForge still leads CALM by 4.9 [3.5, 6.3] points, and CALM and EoH-S with one mutation per parsed proposal meet 65.4 and 64.8 (Table 11).

Paired differences. Every claim that RuleForge beats another method or variant rests on the paired difference over the same goals, with each arm averaged over its five seeds. A symmetric wild cluster bootstrap (Cameron et al., 2008) with the six-point weights of Webb (2023) and 9,999 replicates draws one weight per trace segment, we add the between-seed variance of both arms' means, and intervals and p-values use Student's t with 43 degrees of freedom. On the other problems the weights are drawn per test instance within each test set, with Satterthwaite's degrees of freedom for the instance and seed parts of the variance. Table 11 Holm-adjusts p-values within three families fixed before the runs, the 19 ablations on the chosen rule, the same on the graded score and the four selector variants, and every interval is unadjusted. With a planning correlation of 0.011 among the goals of one segment, a per-seed standard deviation of 0.4 points and a paired variance that grows with the share of goals on which two arms disagree, the design was planned to detect 1.3 points in goals met with 80 percent power, 2.1 at the first step of a Holm family of 19 and 1.0 in the pooled contrast. We therefore preregistered two primary contrasts, the Shapley credit against the mean of the six other credits and against its prompting-goal scope, which pass Holm's procedure over the two with both p below 0.001 under any dependence, and the other single contrasts are secondary. Adding the between-seed variance counts the runs' goal-level noise twice, so the intervals are conservative.

Table 10: Models, software, hardware, seeds and compute of the final runs. The goal-reward loop's settings are in Table 22.

Item	Setting
Models	Nemotron 3.5 Lightning, Qwen3.5-4B, Nemotron Nano 8B, the checkpoint Llama-3.1-Nemotron-Nano-8B-v1, Qwen3.5-9B, gpt-oss-20b, Qwen3.8-27B and gpt-oss-120b, and the frozen DeepSeek-V4.1-Flash and GLM-5.3-Flash, each pinned to the revision recorded in its released run configuration. The seven trained models are loaded in bf16, and gpt-oss-120b trains its LoRA adapters on eight H100 GPUs, and DeepSeek-V4.1-Flash and GLM-5.3-Flash are served in the precision of their released weights. Each model runs with thinking disabled through its chat template, or through the chat mode of its encoding script for DeepSeek-V4.1-Flash, except gpt-oss-20b, gpt-oss-120b and GLM-5.3-Flash, whose templates expose a reasoning effort rather than a thinking switch and which run at low reasoning effort. For the gpt-oss models the writer's cap of 160 new tokens counts the final answer, and the analysis channel is capped at 512 tokens.
Software	TRL 1.7.1 and PEFT 0.19.1, the Mamba-2 and linear-attention kernels of the hybrid models, and the public vLLM OpenAI container at the version the main model's card names, recorded with each run.
Hardware	H100 GPUs with 80 GB, one per run for models up to 10 billion parameters, eight per run for larger ones, and sixteen for DeepSeek-V4.1-Flash and GLM-5.3-Flash, whose released weights take about 765 and 643 GB.
Seeds	One seed s per run drives the loop's own sampling and mutation, the writer's sampler at $s + t$ in generation t, the supervised stages and the selector's GRPO at s, and the writer's GRPO at $s + 100 + t$.
Compute	Figure 10 gives each method's busy GPU-hours and generated tokens on inference scheduling, and the samples and wall-clock time of RuleForge and CALM.
CPU cost	Building the inference rule archive took 1,060 s, evaluating it on the 44 windows of the earlier held-out set took 4,534 s on a laptop, and building the OR-Library job-shop archive took 2,745 s.

Budgets. An evaluation is one simulation of one rule on one inference window or one class of training instances. A method's count includes every simulation it runs, of candidates, survivors, the writer's GRPO samples, the hand-written rules and the in-loop evaluations of CALM, EvoTune and ThetaEvolve, while held-out evaluations count for no method. The inference budget of 30,833 evaluations is the mean count of the goal-reward loop's ten confirmatory runs, fixed before any final run, and each other problem gets 20,000. In an inference run, the hand-written rules take 150 evaluations on the 15 training windows. Each distinct candidate takes one prescreen, 9,820 in all (Table 13), and each survivor takes 14 more on the other windows, about 1,490 survivors in all. A GRPO sample that repeats an evaluated body reuses its verdicts. About 15 percent of candidates survive. The share falls from 24 percent in generation 0 to 7 percent of generation 2's proposals and mutations as the library grows, which sets the per-generation counts of Table 14. The DRL dispatcher is retrained outside the budget, and L2D, the learned dispatcher of Zhang et al. (2020), keeps its published weights.

The rule archive. The rule archive holds 26,341 valid rules, 13,501 sampled from four templates of the grammar and 12,840 children of two mutation rounds, and building it took 809,232 evaluations on the training and development windows. It is a reference, not a method. The greedy archive library chooses among archive rules by their training scores, so it costs the construction's evaluations. Random archive subsets and the full archive choose their rule for each goal on the held-out windows themselves, so they are held-out oracles placed at their held-out scoring cost, 4,636,016 evaluations for the full archive (Figure 11).

Compute. Reinforcement learning spends most of its cost on the GPU, so Figure 10 gives each method's busy GPU-hours and generated tokens, and its caption the samples and wall-clock time of RuleForge and CALM. Capped at CALM's 23.4 busy GPU-hours by shortening only the writer's GRPO training, RuleForge's chosen rules meet 69.4 percent of the held-out goals, 4.4 [3.0, 5.8] points above CALM's 65.0. Given RuleForge's 38.5 busy GPU-hours, CALM reaches 65.6 and

Table 11: RuleForge minus each other row on the same 2,816 held-out inference goals, or the 1,408 no hand-written rule meets for the graded score, in points, with 95 percent intervals from the paired bootstrap of Appendix E. The two primary contrasts carry unadjusted p-values, both below 0.001 after Holm's procedure over the two, and the second part gives p-values Holm-adjusted within the three families, while every interval is unadjusted. The Shapley credit without centering, which equals the harmonic mixture of PKPO's credits, RuleForge without grammar mutation and the leave-two-out best-of-6 credit stay within noise of RuleForge after adjustment, and every other ablation and every selector variant but the first-rule reward is below it. The pooled contrast averages per-sample credit, PKPO's best of four and of six, centered leave-one-out credit, CALM's reward and the Shapley credit on the prompting goal alone. The hand-written oracle of 50.0 is exact, and the fixed shortlist runs the library's single best rule. Figure 9 covers the other problems.

RuleForge minus	Chosen rule	p	Library oracle	Oracle graded	p
Methods of Tables 1, 14 and 16					
Hand-written rules	25.8 [23.7, 27.9]	–	25.4 [23.6, 27.2]	–	
Classical pool, chosen by the selector	13.5 [11.2, 15.8]	–	13.4 [11.1, 15.7]	–	
Genetic programming	7.9 [6.1, 9.7]	–	7.6 [6.0, 9.2]	–	
Greedy archive library	6.6 [5.1, 8.1]	–	5.8 [4.3, 7.3]	–	
Full rule archive	–	–	4.6 [3.1, 6.1]	–	
Mutation, greedy admission	7.7 [6.2, 9.2]	–	6.9 [5.2, 8.6]	–	
FunSearch	8.6 [6.7, 10.5]	–	8.4 [6.7, 10.1]	–	
EoH	8.2 [6.6, 9.8]	–	7.3 [5.6, 9.0]	–	
ReEvo	8.4 [6.6, 10.2]	–	8.2 [6.6, 9.8]	–	
MCTS-AHD	7.8 [5.9, 9.7]	–	6.7 [4.9, 8.5]	–	
OpenEvolve	7.6 [6.0, 9.2]	–	7.2 [5.6, 8.8]	–	
ShinkaEvolve	7.3 [5.9, 8.7]	–	6.5 [5.0, 8.0]	–	
LEVI	7.1 [5.3, 8.9]	–	6.5 [4.8, 8.2]	–	
EoH-S	6.7 [4.9, 8.5]	–	5.5 [3.9, 7.1]	–	
DSevolve	6.9 [5.5, 8.3]	–	5.6 [4.0, 7.2]	–	
EvoTune	6.4 [4.8, 8.0]	–	6.2 [4.6, 7.8]	–	
CALM	6.0 [4.5, 7.5]	–	5.3 [3.8, 6.8]	–	
HSEvo	8.3 [6.7, 9.9]	–	7.5 [5.7, 9.3]	–	
(1+1)-EPS	9.1 [7.3, 10.9]	–	8.3 [6.7, 9.9]	–	
MuEvo	7.0 [5.2, 8.8]	–	6.2 [4.6, 7.8]	–	
ThetaEvolve	6.8 [5.3, 8.3]	–	5.9 [4.6, 7.2]	–	
HeurAgenix	7.4 [5.7, 9.1]	–	6.6 [4.9, 8.3]	–	
InstSpecHH	7.2 [5.5, 8.9]	–	6.4 [4.8, 8.0]	–	
RuleForge, single best rule	9.5 [7.6, 11.4]	–	13.9 [11.8, 16.0]	–	
Controls of the main comparison, with the first term named where it is not RuleForge					
CALM with grammar mutation	5.6 [4.1, 7.1]	–			
CALM with the writer's goal prompt	5.4 [4.1, 6.7]	–			
CALM at RuleForge's busy GPU-hours	5.4 [3.9, 6.9]	–			
CALM with goal prompt, mutation and RuleForge's GPU-hours	4.7 [3.2, 6.2]	–			
CALM prescreened on every window	5.7 [4.3, 7.1]	–			
EoH-S prescreened on every window	6.5 [4.8, 8.2]	–			
EoH-S with the writer's goal prompt	6.1 [4.5, 7.7]	–			
Frozen writer, prompt optimized by GEPA	4.7 [3.4, 6.0]	–			
Frozen writer minus CALM	−0.1 [−0.9, 0.7]	–			
RuleForge capped at CALM's busy GPU-hours, minus CALM	4.4 [3.0, 5.8]	–			
EoH-S with grammar mutation	6.2 [4.9, 7.5]	–			
RuleForge without mutation, minus CALM	4.9 [3.5, 6.3]	–			
Best single rules, RuleForge's minus CALM's	0.7 [−0.1, 1.5]	–			
Best single rule minus the best classical one	7.2 [5.8, 8.6]	–			
Chosen minus best single rule, RuleForge's minus CALM's	5.3 [3.6, 7.0]	–			
EoH-S, frozen gpt-oss-120b	5.5 [4.2, 6.8]	–			
MCTS-AHD, frozen gpt-oss-120b	7.2 [5.7, 8.7]	–			
ShinkaEvolve, frozen gpt-oss-120b	6.5 [5.0, 8.0]	–			
EoH-S, frozen DeepSeek-V4.1-Flash	5.2 [3.7, 6.7]	–			
MCTS-AHD, frozen DeepSeek-V4.1-Flash	6.7 [5.0, 8.4]	–			
ShinkaEvolve, frozen DeepSeek-V4.1-Flash	6.1 [4.6, 7.6]	–			
OpenEvolve, frozen DeepSeek-V4.1-Flash	6.9 [5.2, 8.6]	–			
The two primary credit contrasts, with unadjusted p					
Shapley credit minus the mean of six credits	4.2 [3.2, 5.2]	¡0.001			
Shapley credit minus its prompting-goal scope	3.2 [2.0, 4.4]	¡0.001			

Table 11: Continued, with p-values Holm-adjusted within the three families and the contrasts outside them. The Shapley credit without the library was added after the families were fixed.

RuleForge minus	Chosen rule	Holm p	Library oracle	Oracle graded	Holm p
Ablations of Table 19, Table 20 and Figure 15, Holm families on the chosen rule and on the graded score					
Split and scope of the credit					
Shapley credit, prompting goal only	3.2 [2.0, 4.4]	¡0.001	2.3 [1.1, 3.5]	2.9 [1.8, 4.0]	¡0.001
Per-sample credit, as in TestDecision	4.4 [3.1, 5.7]	¡0.001	4.0 [2.6, 5.4]	7.0 [5.7, 8.3]	¡0.001
Per-sample credit, prompting goal only	5.1 [3.6, 6.6]	¡0.001	4.8 [3.4, 6.2]	8.4 [6.8, 10.0]	¡0.001
Other credits on the same gain					
Best-of-4 credit of PKPO	3.8 [2.7, 4.9]	¡0.001	3.4 [2.2, 4.6]	6.1 [4.7, 7.5]	¡0.001
Best-of-6 credit of PKPO	4.0 [2.7, 5.3]	¡0.001	3.6 [2.3, 4.9]	6.4 [4.8, 8.0]	¡0.001
Centered leave-one-out credit	4.6 [3.1, 6.1]	¡0.001	4.2 [2.9, 5.5]	7.2 [5.8, 8.6]	¡0.001
Shapley credit, not centered	0.8 [−0.01, 1.61]	0.096	0.5 [−0.3, 1.3]	0.7 [−0.2, 1.6]	0.142
Shapley credit with best-of-6's noise	3.5 [2.1, 4.9]	¡0.001	2.8 [1.6, 4.0]	4.2 [2.9, 5.5]	¡0.001
Shapley credit with leave-one-out's noise	4.3 [2.9, 5.7]	¡0.001	3.5 [2.3, 4.7]	5.6 [4.3, 6.9]	¡0.001
Best-of-6 credit, leave-two-out baseline	1.0 [0.09, 1.91]	0.096	0.7 [−0.1, 1.5]	0.8 [0.01, 1.59]	0.142
TailRL's credit on the raw score	5.0 [3.7, 6.3]	¡0.001	4.4 [3.2, 5.6]	7.8 [6.1, 9.5]	¡0.001
CALM's reward	4.9 [3.5, 6.3]	¡0.001	4.3 [3.0, 5.6]	7.5 [6.1, 8.9]	¡0.001
Writer, admission and recipe					
Goal reward	5.3 [4.0, 6.6]	¡0.001	5.0 [3.6, 6.4]	8.6 [7.0, 10.2]	¡0.001
Expert iteration, no writer GRPO	5.9 [4.3, 7.5]	¡0.001	5.4 [3.8, 7.0]	8.6 [7.1, 10.1]	¡0.001
Frozen writer	6.1 [4.6, 7.6]	¡0.001	5.9 [4.4, 7.4]	9.3 [7.9, 10.7]	¡0.001
Random admission	3.7 [2.4, 5.0]	¡0.001	3.0 [1.8, 4.2]	4.5 [3.3, 5.7]	¡0.001
Without grammar mutation	1.1 [0.1, 2.1]	0.096	0.8 [−0.02, 1.62]	0.8 [−0.02, 1.62]	0.142
No replay	2.4 [1.4, 3.4]	¡0.001	1.8 [0.7, 2.9]	2.9 [1.7, 4.1]	¡0.001
Plain GRPO with replay	2.0 [1.0, 3.0]	¡0.001	1.5 [0.5, 2.5]	2.1 [1.0, 3.2]	0.002
Archive rules in the supervised start, outside the families and the budget	−0.4 [−1.2, 0.4]	–			
Selectors of Table 20, the third Holm family					
First-rule reward	0.8 [−0.04, 1.64]	0.061	–	–	
Supervised selector	1.9 [0.9, 2.9]	¡0.001	–	–	
Shuffled rewards	4.9 [3.5, 6.3]	¡0.001	–	–	
Fixed shortlist	9.5 [7.6, 11.4]	¡0.001	–	–	
Outside the families, plain on-policy GRPO, the Shapley credit minus					
Per-sample credit	3.1 [2.0, 4.2]	–			
Best-of-6 credit	2.8 [1.5, 4.1]	–			
Leave-one-out credit	3.4 [2.3, 4.5]	–			
Outside the families, other contrasts					
Shapley credit not centered minus best-of-6	3.2 [2.0, 4.4]	–			
Plain on-policy GRPO, no replay	4.1 [2.8, 5.4]	–	3.8 [2.7, 4.9]	6.1 [4.6, 7.6]	–
VPO	3.6 [2.5, 4.7]	–	3.6 [2.4, 4.8]	5.9 [4.6, 7.2]	–
Pass@4 training	3.8 [2.7, 4.9]	–	3.6 [2.2, 5.0]	6.2 [4.7, 7.7]	–
Shapley credit without the library, as in GCPO	3.9 [2.6, 5.2]	–	3.3 [2.1, 4.5]	6.0 [4.8, 7.2]	–
Best single rules, RuleForge's minus with CALM's reward	0.2 [−0.6, 1.0]	–			
Reward's lead beyond the best single rule	4.7 [3.1, 6.3]	–			
Outside the families, the Shapley credit with noise matched to each credit's ratio minus that credit					
Best-of-4 credit of PKPO	0.6 [−0.3, 1.5]	–			
Best-of-6 credit of PKPO	0.5 [−0.3, 1.3]	–			
Best-of-6 credit, leave-two-out baseline	0.5 [−0.4, 1.4]	–			
Centered leave-one-out credit	0.3 [−0.5, 1.1]	–			
Shapley credit, not centered	0.2 [−0.6, 1.0]	–			

EoH 63.7, and with the writer's goal prompt, grammar mutation and those GPU-hours at once, CALM's chosen rules meet 66.3 percent, 4.7 [3.2, 6.2] points below RuleForge's.

F INFERENCE LIBRARIES

Table 14 gives RuleForge's inference libraries by generation against the baselines, and Figure 11 their quality against evaluations. Of the 8 rules admitted per generation, the writer supplies 2.1, 5.6 and 6.6 in generations 0, 1 and 2 and grammar mutation the rest, 5.9, 2.4 and 1.4, means over the five runs.

Pooling the 18,300 survivors of the eleven baselines and the mutation loop and adding 24 of them greedily by their gain on the training goals gives a library oracle of 71.9 percent, 3.5 [2.2, 4.8] points below RuleForge's. The whole pool as a held-out oracle reaches 73.2 percent, so the trained writer finds rules that the searches do not.

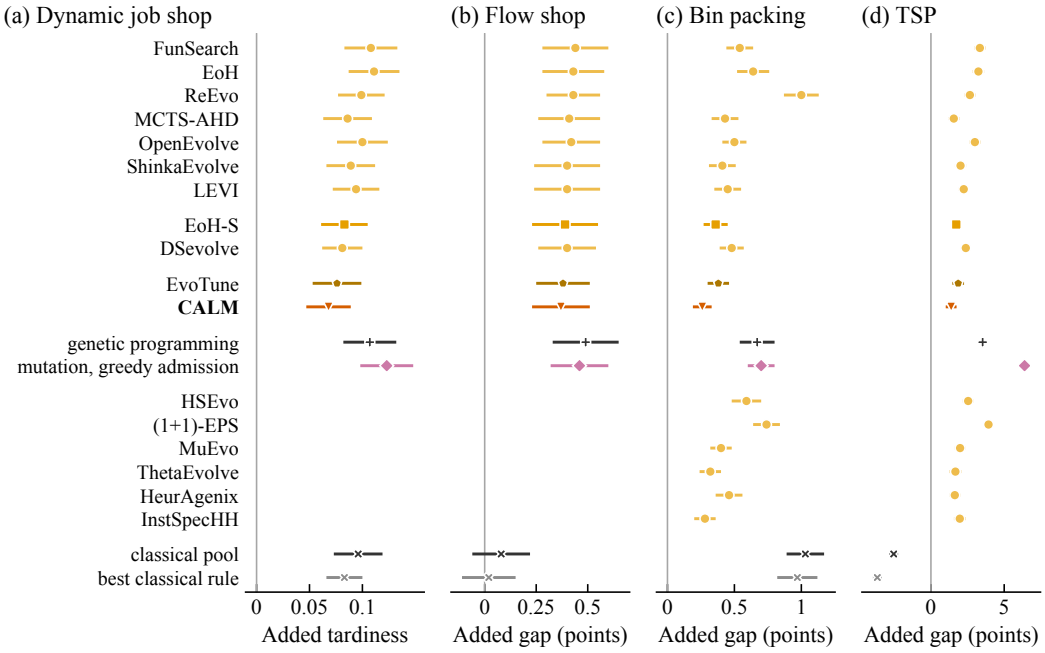

Figure 9: Each method minus RuleForge on the four other problems, where lower is better, so a mark right of zero is a method that RuleForge beats. Intervals are 95 percent wild cluster bootstrap intervals whose weights are drawn per test instance within each test set, add the variance between seeds and use Student's t with Satterthwaite's degrees of freedom, about 8 for a paired difference on the traveling salesman problem (Appendix E). The six methods of Table 16 ran on bin packing and the traveling salesman problem only. The classical pool runs the shared selector over each problem's classical heuristics, and the last row sets the best single classical heuristic against RuleForge's single best rule.

Table 12: Tuning grids on the development goals. Each configuration runs two seeds at a third of the budget, the setting whose chosen rule meets the most development goals is kept, no tuning run counts toward a budget, and every grid reruns for each base model. Unlisted settings keep their published values. RuleForge's recipe comes from the longer development of the goal-reward loop (Appendix M), a larger effort than any baseline's grid.

Methods	Grid
FunSearch, EoH, ReEvo, MCTS-AHD, OpenEvolve, ShinkaEvolve, LEVI	population or island size at half, one and two times the published value, sampling temperature 0.7 or 1.0
EoH-S, DSevolve	set or population size at half, one and two times the published value, temperature 0.7 or 1.0
EvoTune	learning rate 5×10^{-7}, 5×10^{-6} or 3×10^{-5}, the ends and middle of its published sweep
CALM	learning rate 5×10^{-6}, 10^{-5} or 5×10^{-5}, the last its published value, group size 4 or 8
Genetic programming	population 200 or 500, tournament size 3 or 7
RuleForge	writer learning rate 5×10^{-6} or 10^{-5}
Other credits of Figure 19	writer learning rate 2×10^{-6}, 5×10^{-6}, 10^{-5}, 2×10^{-5} or 5×10^{-5}, the last CALM's published value

Table 13: Candidate rules each method evaluated per run, means over five seeds. RuleForge's count covers the writer's proposals and the mutations, and its last row adds the distinct bodies among the writer's GRPO samples, whose survivors also join admission. Repeat gives the share of inference candidates whose prescreen outcomes equal a library rule's, for RuleForge its writer's proposals (Appendix K). The last column gives the share of inference proposals that parse in the grammar, and a dash marks a problem the method does not run on.

Method	Inference	Dynamic job shop	Flow shop	Bin packing	TSP	Parse rate (%)	Repeat (%)
FunSearch	12,890	18,220	9,620	13,450	16,740	91	44
EoH	12,160	18,050	9,130	13,010	16,460	93	42
ReEvo	11,820	17,970	8,900	12,800	16,330	92	43
MCTS-AHD	12,510	18,130	9,370	13,230	16,600	93	40
OpenEvolve	12,660	18,170	9,470	13,310	16,650	90	41
ShinkaEvolve	11,690	17,940	8,820	12,720	16,270	92	39
LEVI	12,370	18,100	9,270	13,140	16,540	93	38
EoH-S	11,950	18,000	8,990	12,880	16,380	94	35
DSevolve	12,090	18,030	9,080	12,970	16,430	92	36
EvoTune	10,340	17,560	7,880	11,810	15,660	95	31
CALM	10,650	17,650	8,100	12,020	15,810	95	29
HSEvo	12,020	–	–	12,920	16,410	92	42
(1+1)-EPS	13,710	–	–	13,920	17,020	94	45
MuEvo	11,500	–	–	12,600	16,190	90	41
ThetaEvolve	10,750	–	–	12,100	15,860	95	32
HeurAgenix	11,880	–	–	12,700	16,120	91	40
InstSpecHH	12,240	–	–	13,050	16,530	93	41
Mutation, greedy admission	21,140	19,370	14,790	17,250	18,790	100	33
RuleForge	**7,050**	**12,500**	**4,860**	**7,700**	**10,820**	**97**	**12**
RuleForge with GRPO samples	9,820	15,110	7,750	10,420	13,660		

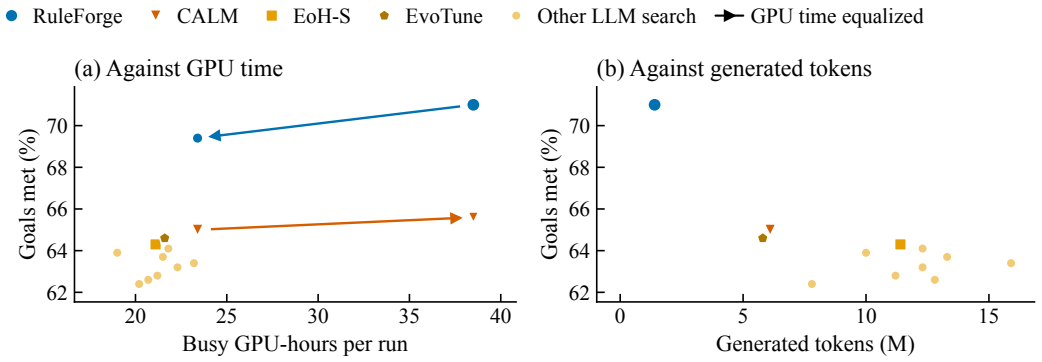

Figure 10: Compute of each method on inference scheduling, per run, against the held-out goals met by its chosen rule. Every method stops at 30,833 counted evaluations, which include the simulations that score reinforcement-learning samples. (a) Busy GPU-hours, the H100 time the method's own model spends sampling and training. RuleForge uses 38.5 busy GPU-hours against CALM's 23.4, and the arrows show the runs that equalize GPU time. Capped at CALM's hours, RuleForge's chosen rules meet 69.4 percent of the held-out goals, and given RuleForge's hours, CALM's meet 65.6. (b) The tokens the method's model generates, counting the reflection and meta calls of ReEvo and ShinkaEvolve. RuleForge's writer draws 9,270 samples against CALM's 12,600, and a run of RuleForge holds eight H100s for 11.6 hours, 92.8 allocated GPU-hours, against 9.8 hours and 78.4 for CALM. The shared selector, which every method trains alike, adds 38,400 samples and 6.2 busy GPU-hours to every run. On bin packing and the traveling salesman problem a run of RuleForge uses 21.4 and 26.8 busy GPU-hours and CALM 19.7 and 24.5.

Table 14: Inference libraries by generation against the baselines, on the 2,816 goals of the fresh held-out windows. Chosen rule counts the goals met by the rule the selector picks, library oracle the goals some rule of the library meets, and the graded score averages the 1,408 held-out goals no hand-written rule meets, all in percent. The classical pool runs the shared selector per goal over the seven classical orderings of Table 15 and shortest first. LLM-driven methods and mutation with greedy admission start from the ten hand-written rules, pass their candidates through the same admission and stop at 30,833 counted evaluations. RuleForge's rows count the evaluations up to each admission, so the writer's GRPO samples after an admission count in the next row. The last two rows choose among the whole archive. The greedy archive library counts the archive's construction, 809,232 evaluations, and the full archive's oracle counts the 4,636,016 held-out evaluations that choose its rule for each goal. Means over five seeds.

Library	Rules	Evaluations	Chosen rule	Library oracle	Graded score
Hand-written rules	10	–	45.2	50.0	0.0
Classical pool, chosen by the selector	8	–	57.5	62.0	26.0
Mutation with greedy admission	34	30,833	63.3	68.5	51.8
OpenEvolve	34	30,833	63.4	68.2	52.0
EoH	34	30,833	62.8	68.1	51.3
DSevolve	34	30,833	64.1	69.8	52.9
CALM	34	30,833	65.0	70.1	53.9
RuleForge, generation 0	18	15,290	64.6	68.9	49.0
RuleForge, generation 1	26	24,590	67.6	72.1	55.8
RuleForge, generation 2	**34**	**30,833**	**71.0**	**75.4**	**63.1**
Chosen among the whole archive					
Greedy archive library	34	809,232	64.4	69.6	52.9
Full archive, held-out oracle	26,341	4,636,016	–	70.8	56.6

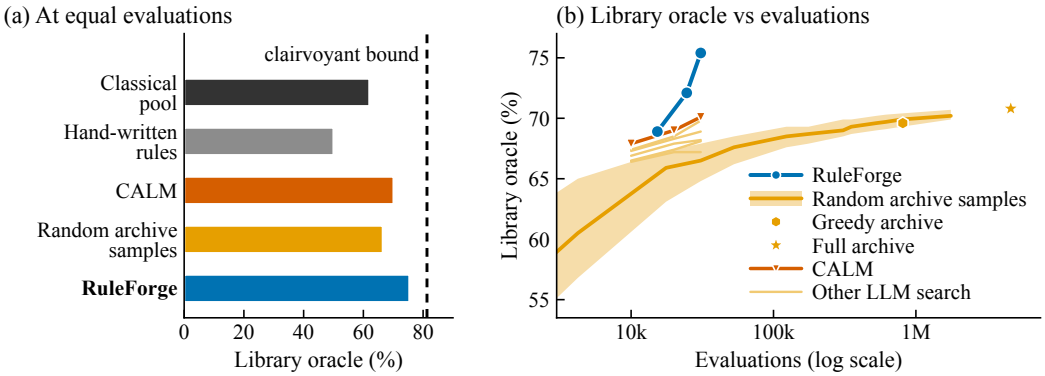

Figure 11: Library quality in inference scheduling. (a) Held-out goals met by the library oracle of the best method of each family at the same number of evaluations, where a random archive sample holds the 175 rules that 30,833 held-out evaluations score, with the clairvoyant bound dashed. (b) Goals met by the library oracle against evaluations for RuleForge by generation, random archive samples, the greedy and full archives, and six LLM-driven methods, with CALM, the best of them, in its own color. The greedy archive library sits at the archive's construction cost, and random samples and the full archive at their held-out scoring cost (Appendix E).

G More baselines, a clairvoyant bound and other models

Each baseline on its own. Table 15 scores each inference baseline under the shared protocol, with each classical ordering rendered as one rule over the features of Table 4. Slurm's multifactor priority (SchedMD, 2026) weighs a request's wait, its tenant's deficit and its work under the documentation's example weights and under weights tuned on the training goals, a fair-sharing queue orders tenants

by deficit, VTC (Sheng et al., 2024) charges a tenant for tokens as they are served, alone and with shortest first inside each tenant, and shortest first with the starvation prevention of Fu et al. (2024) promotes a request skipped a tuned number of times, ranking by true work. The classical pool of Table 1 runs the shared selector over these six orderings, first come, first served as in vLLM (Kwon et al., 2023) and shortest first. All of them run in our slot simulator, and Appendix J replays the chosen rules in vLLM itself.

Table 15: Each inference baseline on the 2,816 fresh held-out goals under the shared selector recipe, with the library's single best rule and oracle. The classical orderings are our renderings of documented policies as one rule, and the classical pool runs the selector over them and shortest first. LLM-driven methods and mutation start from the ten hand-written rules and keep at most 34 rules after 30,833 evaluations. Means over five seeds.

Method	Rules	Chosen rule (%)	Single best (%)	Library oracle (%)	Graded score (%)
Classical orderings, one rule each					
Slurm multifactor, example weights (SchedMD, 2026)	1	44.6	44.6	44.6	12.3
Slurm multifactor, tuned weights	1	51.2	51.2	51.2	18.4
Fair-sharing queue (Kubernetes SIG Scheduling, 2024)	1	13.4	13.4	13.4	2.1
VTC (Sheng et al., 2024)	1	12.2	12.2	12.2	1.8
VTC, shortest first within each tenant	1	49.7	49.7	49.7	16.9
Shortest first, starvation prevention (Fu et al., 2024)	1	54.3	54.3	54.3	21.2
First come, first served (Kwon et al., 2023)	1	5.4	5.4	5.4	0.0
Classical pool, chosen by the selector	8	57.5	54.3	62.0	26.0
No language model					
Mutation with greedy admission	34	63.3	58.7	68.5	51.8
Greedy archive library	34	64.4	60.9	69.6	52.9
LLM search					
FunSearch	34	62.4	59.4	67.0	50.9
EoH	34	62.8	59.9	68.1	51.3
ReEvo	34	62.6	59.6	67.2	51.1
MCTS-AHD	34	63.2	60.2	68.7	51.8
OpenEvolve	34	63.4	60.0	68.2	52.0
ShinkaEvolve	34	63.7	60.4	68.9	52.4
LEVI	34	63.9	60.3	68.9	52.6
Heuristic sets and reinforcement learning in the loop					
EoH-S	34	64.3	60.1	69.9	53.1
DSevolve	34	64.1	59.8	69.8	52.9
EvoTune	34	64.6	60.5	69.2	53.4
CALM	34	65.0	60.8	70.1	53.9
Hand-written rules and RuleForge					
Best hand-written rule, shortest-first	1	42.6	42.6	42.6	0.0
Hand-written rules	10	45.2	42.6	50.0	0.0
RuleForge, single best rule	1	61.5	61.5	61.5	43.8
RuleForge	**34**	**71.0**	**61.5**	**75.4**	**63.1**

Six more baselines. Table 16 adds six LLM-driven methods under the shared protocol, HSEvo (Dat et al., 2025), the (1+1)-EPS baseline of Zhang et al. (2024), MuEvo (Lv et al., 2026a), ThetaEvolve (Wang et al., 2026b), which trains its model during the search, HeurAgenix (Yang et al., 2025), which pairs an evolved pool with a fine-tuned selector, and InstSpecHH (Zhang et al., 2025). None of them beats CALM on any measure, and on inference the seven evolutionary searches of Table 1 lie between 0.9 points below and 0.6 points above grammar mutation with greedy admission. Under their own protocols on our base model, MCTS-AHD and CALM reproduce their published gaps, and at 20,000 evaluations our grammar costs their single best rules 0.06 to 0.08 points of gap on bin packing and 0.5 on the traveling salesman problem. We therefore also run RuleForge's loop over Python functions of the full state under their published protocols, with each training instance as a goal of zero gap and the best seed heuristic's gap as u in Equation 13, a supervised start on the published seed heuristics, no grammar mutation and 10 or 100 GRPO steps per generation. Its

admitted function with the smallest mean training gap beats CALM by 0.14 [0.07, 0.21] and 0.81 [0.39, 1.23] points of gap at 2,000 evaluations and by 0.18 [0.11, 0.25] and 0.96 [0.60, 1.32] at 20,000. Larger frozen models do not close the gap, since EoH-S, MCTS-AHD and ShinkaEvolve with gpt-oss-120b meet 65.5, 63.8 and 64.5 percent of the held-out inference goals, and EoH-S, MCTS-AHD, ShinkaEvolve and OpenEvolve with DeepSeek-V4.1-Flash 65.8, 64.3, 64.9 and 64.1, each paired difference from RuleForge excluding zero.

Native representations on inference scheduling. The shared protocol holds every baseline to our grammar, prescreen and admission. We therefore also run CALM, EvoTune, EoH-S and ShinkaEvolve on inference scheduling in their native form. Each writes Python priority functions over the request features and the queue state, keeps its own population management and fitness, the training goals its functions meet, and skips the shared prescreen. Its final population then passes through the shared admission and selector at 30,833 evaluations. Their chosen rules meet 65.8, 65.3, 64.9 and 64.3 percent of the held-out goals, 0.6 to 0.8 points above their grammar versions. RuleForge in the grammar leads the best of them, CALM, by 5.2 [3.7, 6.7] points, and RuleForge's loop over the same Python functions meets 71.4 percent, 5.6 [4.3, 6.9] points above native CALM.

The reward in other pipelines. Inside CALM's own code and budget of 2,000 model queries on bin packing and the traveling salesman problem, we replace its reward with the Shapley credit, with its population as the library and each training instance as a goal of zero gap with the gap of CALM's best base heuristic as u, so a copy of a kept heuristic gains nothing, which replaces CALM's penalty for copies. The final gap falls from 0.74 to 0.62 on bin packing, 0.12 [0.06, 0.18], and from 11.8 to 11.1 on the traveling salesman problem, 0.70 [0.29, 1.11]. Inside EvoTune's code at its budget of 22,400 programs, with its database as the library and one prompt's outputs as the group, splitting its DPO pairs by the Shapley credit instead of each program's own score lowers the gap from 0.82 to 0.70, 0.12 [0.06, 0.18], and from 12.1 to 11.5, 0.60 [0.23, 0.97]. Holm's procedure over these four transplants, which had no preregistered bar, gives p of 0.001, 0.008, 0.003 and 0.008. Under the protocol of EoH-S, with ten heuristics scored by their best member on its bin-packing test sets and its test sets of 50, 100 and 200 cities and 2,000 evaluations, the first ten Python functions admitted by RuleForge's loop beat EoH-S rerun with a frozen DeepSeek-V4.1-Flash, which lands near its published values (Liu et al., 2026), by 0.14 [0.08, 0.20] and 1.22 [0.85, 1.59]. The third block of Table 16 scores sets the same way under the shared protocol.

A clairvoyant bound. A clairvoyant CP-SAT model (Perron & Didier, 2026) that knows every arrival and token count and keeps only the thresholds on served fraction and share error bounds what any admission schedule can meet at 81.3 percent of the 2,816 held-out goals, against 75.4 percent for RuleForge's library oracles. The bound overstates the true ceiling, since it drops the wait-spread thresholds and counts goals that CP-SAT neither meets nor proves infeasible within its time limit as met.

Mutation at scale. Run as its own loop with admission unchanged, grammar mutation needs more than 30 times the candidates that the shared budget buys it, more evaluations than built the whole archive, to match RuleForge's libraries (Figure 12).

Other models. Table 18 runs the writer and the selector with the seven models of the roster, open models from four families with 4.7 to 116.8 billion parameters, the main model Nemotron 3.5 Lightning (NVIDIA, 2026), Qwen3.5-4B and Qwen3.5-9B (Qwen Team, 2026a), Nemotron Nano 8B (Bercovich et al., 2025), gpt-oss-20b and gpt-oss-120b (OpenAI, 2025) and Qwen3.8-27B (Qwen Team, 2026b), with frozen DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) and GLM-5.3-Flash (GLM-5 Team, 2026) writers as untrained baselines. On every trained model the trained writer's graded score exceeds the frozen writer's with a paired interval above zero. Table 17 runs the methods of Table 1 on each base model with every tuning grid rerun. RuleForge leads CALM, the preregistered comparator, by 4.8 to 6.7 points and the best other method of each model by 4.7 to 6.7 points, each with a paired interval above zero. Inside the recipe, CALM's reward costs 4.4 of the 5.6-point mean lead over CALM and 3.2 to 5.1 points on single models, and per-sample credit 3.2 to 4.6, each interval above zero. Pooled over the models, the cost of CALM's reward exceeds half of the lead by 1.6 [1.1, 2.0] points, so the writer's reward carries most of the lead. The other methods change

Table 16: Six more LLM-driven methods under the shared protocol of Table 1, beside CALM and RuleForge. The second block reruns MCTS-AHD and CALM under their own protocols, with Python functions of the full state and one heuristic for all test sets, at their published budgets and at 20,000 evaluations, beside RuleForge's loop under the same protocols. The third block scores sets by their best member on each test instance under the shared protocol, and the fourth runs the protocol of EoH-S, where the traveling salesman gap is to the tours of LKH. Inference gives held-out goals met, bin packing the gap to the L1 lower bound and the traveling salesman problem the gap to the optimal tour, in percent, means over five seeds.

	Inference, goals met ↑		Bin	
Method	Chosen rule	Oracle	packing ↓	TSP ↓
HSEvo	62.7	67.9	0.97	12.4
(1+1)-EPS	61.9	67.1	1.12	13.8
MuEvo	64.0	69.2	0.78	11.9
ThetaEvolve	64.2	69.5	0.70	11.6
HeurAgenix	63.6	68.8	0.84	11.5
InstSpecHH	63.8	69.0	0.66	11.9
Under their own protocols, one heuristic for all test sets				
MCTS-AHD, published budget	–	–	0.92 ±0.09	11.9 ±0.3
published	–	–	0.89	11.7
20,000 evaluations	–	–	0.80 ±0.08	11.3 ±0.3
CALM, 2,000 queries	–	–	0.74 ±0.07	11.8 ±0.3
published	–	–	0.71	11.7
20,000 evaluations	–	–	0.62 ±0.07	11.1 ±0.2
RuleForge, Python functions, 2,000	–	–	**0.60 ±0.07**	**11.0 ±0.4**
20,000 evaluations	–	–	**0.44 ±0.06**	**10.1 ±0.3**
The shared protocol with grammar rules, a set scored by its best member on each instance of our test sets				
EoH-S, ten heuristics, DeepSeek-V4.1-Flash	–	–	0.79 ±0.08	11.2 ±0.3
EoH-S, ten heuristics, main model	–	–	0.86 ±0.09	11.6 ±0.4
CALM's first ten rules, 2,000	–	–	0.81 ±0.09	11.4 ±0.3
RuleForge's first ten rules, 2,000	–	–	**0.58 ±0.07**	**10.3 ±0.3**
CALM's library, 20,000	–	–	0.55 ±0.06	10.9 ±0.3
RuleForge's library, 20,000	–	–	**0.31 ±0.05**	**9.3 ±0.3**
The protocol of EoH-S, on its training instances and its test sets of up to 200 cities				
EoH-S, published, DeepSeek-V3	–	–	0.53	6.5
EoH-S, DeepSeek-V4.1-Flash	–	–	0.55 ±0.06	6.7 ±0.6
RuleForge's first ten Python functions	–	–	**0.41 ±0.05**	**5.5 ±0.5**
CALM	65.0	70.1	0.64	11.3
RuleForge	**71.0**	**75.4**	**0.38**	**9.9**

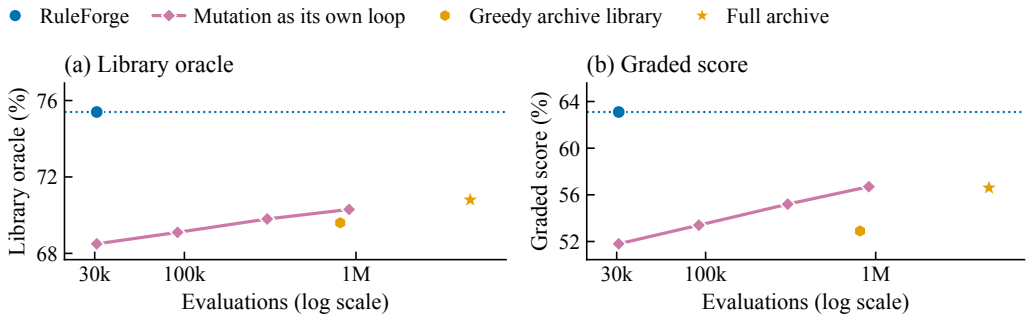

Figure 12: Grammar mutation run as its own loop, with the writer's candidates replaced by mutations and admission unchanged, at 1, 3, 10 and 30 times the candidates that the shared budget of 30,833 evaluations buys it, against RuleForge at that budget and the greedy and full archives of Table 14. (a) The library oracle on the 2,816 held-out inference goals and (b) the graded score on the 1,408 no hand-written rule meets, means over five seeds. The dotted line marks RuleForge's level, which mutation does not reach with 911,700 evaluations.

places from model to model, and EvoTune ties or edges CALM on 2 of them. The reward rows had no preregistered bar.

Table 17: The methods of Table 1 other than genetic programming on each base model of the roster, ordered by total parameters, on inference scheduling, in held-out goals met by the chosen rule, means over five seeds with every tuning grid rerun and each selector an adapter of the same model. Below RuleForge are its lead over the best other method of each model, named in the row above, and over CALM, the preregistered comparator, and the bottom rows swap the Shapley credit for CALM's reward or per-sample credit, with the cost against RuleForge. Leads and costs carry the half-width of their 95 percent paired intervals, and the Nemotron 3.5 Lightning column repeats Table 1.

Method	Qwen3.5 4B	Nemotron Nano 8B	Qwen3.5 9B	gpt-oss 20b	Qwen3.8 27B	Nemotron 3.5 Lightning	gpt-oss 120b
No language model in the search, each model's selector							
Hand-written rules	44.3	44.8	45.2	45.2	45.5	45.2	45.8
Classical pool	56.8	57.2	56.7	57.7	57.9	57.5	57.3
Mutation, greedy admission	62.4	62.6	63.2	63.2	63.5	63.3	63.5
LLM-driven methods							
FunSearch	61.3	61.3	61.8	61.9	63.5	62.4	62.9
EoH	60.8	61.8	61.9	62.8	63.3	62.8	63.7
ReEvo	61.3	61.1	61.3	61.6	62.8	62.6	63.4
MCTS-AHD	61.9	61.8	62.5	63.1	63.9	63.2	63.8
OpenEvolve	61.1	62.6	62.7	62.9	64.1	63.4	64.0
ShinkaEvolve	61.6	62.4	63.2	63.3	63.3	63.7	64.5
LEVI	62.2	63.2	62.8	63.4	64.2	63.9	65.2
EoH-S	63.2	63.0	63.7	64.3	65.2	64.3	65.5
DSevolve	62.7	63.3	63.3	63.9	64.6	64.1	65.2
EvoTune	63.1	63.8	64.3	64.1	64.5	64.6	65.5
CALM	63.3	63.8	64.2	64.6	65.4	65.0	65.9
RuleForge	**68.1**	**69.2**	**69.0**	**70.2**	**71.3**	**71.0**	**72.6**
best other method	CALM	CALM	EvoTune	CALM	CALM	CALM	CALM
lead over it	4.8 ±1.3	5.4 ±1.4	4.7 ±1.3	5.6 ±1.5	5.9 ±1.6	6.0 ±1.5	6.7 ±1.4
lead over CALM	4.8 ±1.3	5.4 ±1.4	4.8 ±1.5	5.6 ±1.5	5.9 ±1.6	6.0 ±1.5	6.7 ±1.4
RuleForge with one part of the writer's reward changed							
With CALM's reward	64.9	65.2	65.4	65.1	66.4	66.1	67.7
cost	3.2 ±1.3	4.0 ±1.2	3.6 ±1.3	5.1 ±1.5	4.9 ±1.5	4.9 ±1.4	4.9 ±1.3
With per-sample credit	64.9	65.3	64.7	65.6	67.3	66.6	68.1
cost	3.2 ±1.2	3.9 ±1.3	4.3 ±1.2	4.6 ±1.2	4.0 ±1.2	4.4 ±1.3	4.5 ±1.4

Table 18: RuleForge with the seven models of the roster and two frozen writers. The writer columns give the generation-2 graded score on the 1,408 held-out inference goals no hand-written rule meets and its gain over a frozen writer of the same model, and the selector column the gain in goals met within three attempts on the 72 OR-Library development goals over the supervised selector, each with its 95 percent paired half-width. Each writer gain excludes zero, and the selector gains exclude it on 4 of the seven models. Parameters are in billions, total and active, and DeepSeek-V4.1-Flash counts its 16 billion active parameters at decode. Means over five seeds.

Model	Total	Active	Writer graded score	Gain over frozen	Selector gain
Nemotron 3.5 Lightning	31.6	3.0	63.1	9.3 ±1.4	4.4 ±4.0
Qwen3.5-4B	4.7	4.7	58.2	5.8 ±1.5	2.2 ±3.9
Nemotron Nano 8B	8.0	8.0	59.2	8.0 ±1.7	5.8 ±3.9
Qwen3.5-9B	9.7	9.7	59.8	6.6 ±1.2	1.9 ±3.9
gpt-oss-20b	20.9	3.6	61.6	9.6 ±1.7	6.7 ±4.0
Qwen3.8-27B	27.8	27.8	62.7	8.2 ±1.5	3.6 ±4.1
gpt-oss-120b	116.8	5.1	64.2	9.8 ±1.5	5.0 ±3.9
Frozen writers, no training					
DeepSeek-V4.1-Flash	552	16	52.1	–	–
GLM-5.3-Flash	320	18	51.6	–	–

The selector's cost. The selector runs once per goal, before scheduling starts. In the OR-Library job shop its full decision, from the prompt to a parsed shortlist of three rules and their evaluations, takes a median of 0.41 seconds on one H100 at batch size one, and a classical rule then schedules

an instance in a median 2.8 milliseconds on one CPU core. On inference scheduling a chosen rule scores a queue in a median of 40 microseconds on one CPU core, over queues of 12 to 310 requests from the 5th to the 95th percentile.

H THE FOUR OTHER PROBLEMS

This appendix describes the four other problems and gives each test set (Figure 13). The protocol is that of Appendix E, with a goal pairing a class of training instances with a threshold on the problem's metric and one evaluation running one rule on one class. The selector reads the item count and capacity on bin packing, the city count and statistics of the coordinates on the traveling salesman problem, the utilization and the threshold on the dynamic job shop, and the numbers of jobs and machines on the flow shop. At test time it reads each test set's class with the tightest threshold of that class's training goals. Every rule runs on the same simulation seeds, so verdicts are exact for the recorded instances and seeds. With 24 to 48 training goals, each generation spends a third of the 20,000 evaluations, prompting the missed goals in turn with six rules per prompt, and a library holds at most 24 admitted rules. Every method runs five seeds with Nemotron 3.5 Lightning, ten for the full recipe and the prompting-goal arm on bin packing and the traveling salesman problem, and Table 1 reports seeds 0 to 4 of every method. Each baseline keeps the better on the training goals of its settings tuned on inference and its published settings, and RuleForge keeps its recipe untuned, so our rows differ from the published numbers, which use other models and budgets. On the flow shop and the traveling salesman problem, running every rule of a library and keeping the best solution gives 2.43 and 9.3 percent for RuleForge's libraries against 2.55 and 10.9 for CALM's.

H.1 ONLINE BIN PACKING

We use the online bin packing task of Romera-Paredes et al. (2024) with the six Weibull test sets of MCTS-AHD (Zheng et al., 2025b) and CALM (Huang et al., 2026b), 1,000, 5,000 and 10,000 items with bin capacity 100 and 500, five instances each. A rule scores every open bin for the arriving item from its size, the bin's remaining capacity and the capacity, and the gap is the percentage of bins above the L1 lower bound. The 36 training goals each pair one item count and capacity with a gap threshold. The hand-written rule is best fit, and our run reproduces its published 2.40 percent average. Published averages with GPT-4o-mini run from 0.89 for MCTS-AHD to 1.71 for ReEvo (Zheng et al., 2025b), and CALM reports 0.71 with a 4-bit Qwen2.5-7B-Instruct after 2,000 model queries (Huang et al., 2026b). The Sum-of-Squares algorithm (Csirik et al., 2006) reaches 1.52 percent, the genetic-programming rule of Burke et al. (2006) 2.56, and the classical pool over these two, best fit and first fit 1.41.

H.2 TRAVELING SALESMAN

A rule scores every unvisited city for the next step of a tour (Liu et al., 2023; Ye et al., 2024; Zheng et al., 2025b) from its distances to the current city, the start and the other unvisited cities and the share of cities left. The test sets are those of MCTS-AHD with 50, 100 and 200 cities, 1,000 instances each, with the gap to the optimal tour, and the 36 training goals each pair one city count with a gap threshold. The hand-written rule is nearest neighbor, and our run reproduces its published gaps. MCTS-AHD and CALM report 11.7 percent on average with one heuristic for all sizes (Zheng et al., 2025b; Huang et al., 2026b), while RuleForge's single best rule reaches 11.1 and its selector 9.9 with our model and 20,000 evaluations. Insertion heuristics, which a next-city rule cannot express, do better still, farthest insertion, as benchmarked by Kool et al. (2019), with 7.4 percent and random insertion with 9.2, and the classical pool, which chooses among nearest neighbor and the nearest, farthest and random insertion heuristics, picks farthest insertion on every set and seed. The seeds carry most of the variance, so Student's t uses about 6 degrees of freedom for a level and 8 for a paired difference.

H.3 DYNAMIC JOB SHOP

Jobs arrive in a Poisson stream at ten machines at utilization 0.85 and 0.95, a setting modeled on the ten-machine dynamic job shop that genetic-programming studies commonly use (Holthaus & Rajendran, 2000; Nguyen et al., 2017; Branke et al., 2016). Each job visits every machine once in

random order, with operation times uniform on 1 to 49 and a due date of its release plus four times its total work. Each replication discards 500 warm-up jobs and records the next 2,000, and a test point averages 50 replications. A rule orders each machine's queue from the features of that literature, and the 24 training goals each pair a utilization with a threshold on mean tardiness. The hand-written rule is ATC (Pinedo, 2022) with look-ahead $k = 2$ over the queue's mean operation time, and mean tardiness is divided by that of genetic programming at the full budget, 38.4 and 191.7 time units at the two utilizations. The best single classical rule, 2PT+WINQ+NPT (Holthaus & Rajendran, 2000), reaches 1.02, genetic programming with a population of 1,024 over 51 generations outside the budget 0.96, the classical pool over 2PT+WINQ+NPT, ATC, COVERT, SPT, EDD, MOD, CR+SPT, PT+WINQ+SL and RR 0.99, and the deep Q-network of Luo (2020), retrained outside the budget, 1.03. On unseen due-date allowances of three and six times a job's total work, RuleForge's chosen rules reach 0.90 and 0.91 against 0.97 and 0.98 for CALM's.

H.4 Permutation flow shop

In the flow shop a rule is a pair of expressions that runs NEH, one scoring jobs for its initial order, the part Framinan et al. (2003) vary, and one scoring tied insertion positions, the part Fernandez-Viagas & Framinan (2014) vary, from each job's processing times and each position's idle time, completion times and place. Both cost $O(m)$, so a rule keeps NEH's $O(n^2m)$ time, and no local search follows. The test set is the 120 Taillard instances in twelve classes of ten, with the gap to the best-known upper bounds in a list dated September 2026 and released with our code, which puts every rerun of a published rule 0.02 above its published mean. The 48 training goals each pair one class of generated instances with a gap threshold, and the selector picks one rule per class. Training instances are generated as Taillard's with other seeds, and their gap is to Taillard's lower bound. NEH reaches 3.35 percent and NEHLJP2 (Zhang et al., 2023), the best single classical rule, 2.72, which every LLM-driven baseline of Table 1 trails. The classical pool, which picks per class among NEH, NEH-D (Dong et al., 2008), NEHKK1 and NEHKK2 (Kalczynski & Kamburowski, 2008; 2009), NEHLJP (Liu et al., 2017), NEHLJP2 and the tie-breaking of NEHFF (Fernandez-Viagas & Framinan, 2014), reaches 2.64, against 2.56 for RuleForge's chosen rules, a tie, while RuleForge's single best rule, at 2.70, ties NEHLJP2. NEH with 25 random initial orders reaches 2.42 at 25 times the computation, so our claim concerns single-pass rules of NEH's complexity. On the 240 large VRF instances (Vallada et al., 2015), RuleForge's chosen rules tie the classical pool, 0.07 [−0.01, 0.15], and its single best rule ties NEHLJP2, 0.05 [−0.04, 0.14], and inside the iterated greedy algorithm of Ruiz & Stützle (2007) its best pair beats a start from NEH by 0.09 [0.02, 0.16] and ties the best published tie-breaking, 0.02 [−0.03, 0.07].

Baselines. The baselines run as in Appendix E, with the same base model, training instances, prescreen, admission, selector and budget as RuleForge. The single best rules of the shared libraries of CALM, EvoTune, EoH-S and MCTS-AHD reach 0.70, 0.83, 0.81 and 0.86 on bin packing and 11.6, 12.0, 11.9 and 11.8 on the traveling salesman problem, against 0.55 and 11.1 for RuleForge's. Genetic programming evolves rules in the same grammar with a population of 200, tournament selection, subtree crossover and mutation at the full budget, and its rules pass through the same admission and selector.

Parts of the recipe. Figure 14 repeats per-sample credit, the goal reward and the frozen writer on the four problems, the best-of-6 and leave-one-out credits on bin packing and the traveling salesman problem, and the Shapley credit on the prompting goal alone everywhere. Crediting only the prompting goal raises the gap on bin packing and the traveling salesman problem, where 33 and 27 percent of a group's gain lands off the prompting goal, and its intervals include zero on the two shops, where only 24 and 18 percent does. These four contrasts form a preregistered family with ten seeds per arm on bin packing and the traveling salesman problem, whose Holm-adjusted p are 0.001 on bin packing, 0.001 on the traveling salesman problem, 0.96 on the dynamic job shop and 0.96 on the flow shop. Each ablation is worse than the full recipe on every problem, and all 12 paired intervals of the three writer ablations exclude zero.

I Policy-gradient algorithms and writer credits

Table 19 gives the values behind Figure 4a and b, and Table 20 changes the recipe's other parts.

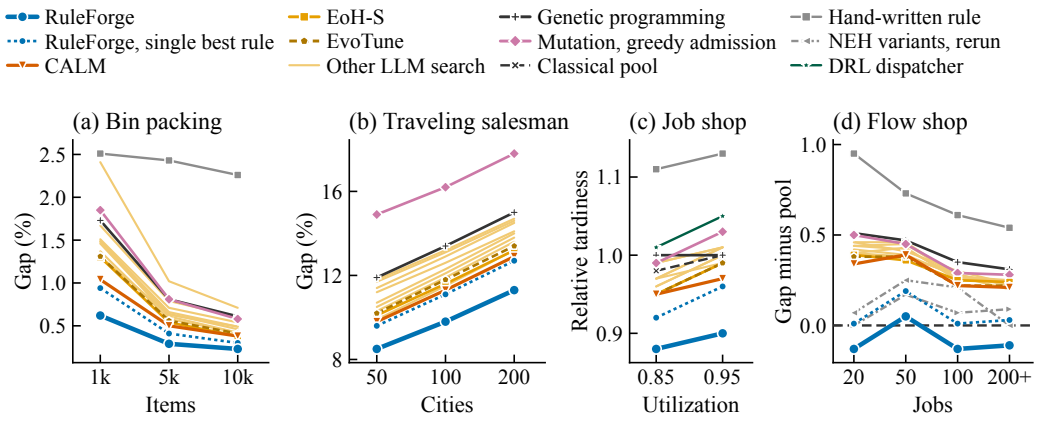

Figure 13: Every test set of the four other problems, means over five seeds with Nemotron 3.5 Lightning as every method's language model, lower is better in every panel. (a) The gap to the L1 lower bound on the Weibull bin-packing sets of 1,000, 5,000 and 10,000 items, each the mean over capacities 100 and 500. (b) The gap to the optimal tour with 50, 100 and 200 cities. Nearest neighbor lies above the axis at 22.6 to 26.3 percent. Panels (a) and (b) leave out the classical pool, whose mean gaps are 1.41 and 7.4 percent. (c) Dynamic job-shop mean tardiness relative to genetic programming at the full budget, at utilization 0.85 and 0.95. (d) The flow-shop gap to the best-known upper bounds of the Taillard instances with 20, 50 and 100 jobs and with 200 or 500 jobs, 30 instances each, minus the gap of the classical pool, which chooses a published NEH pair for each class of ten instances, since the classes differ far more than the methods. The eight other LLM search methods are light lines, and the NEH variants are NEHLJP and NEHLJP2 rerun on our bounds (Liu et al., 2017; Zhang et al., 2023).

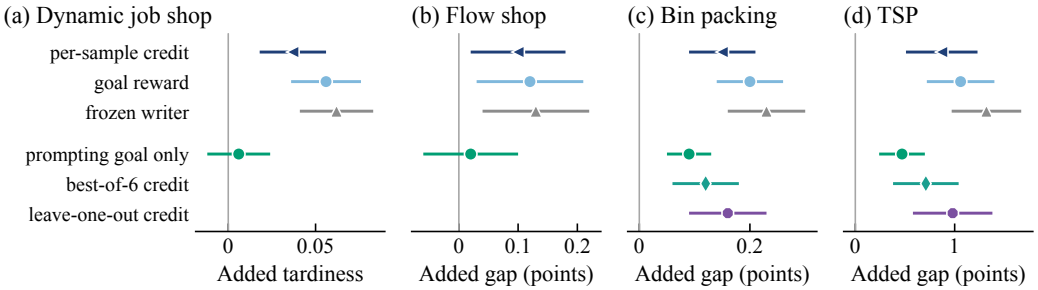

Figure 14: Writer ablations on the four other problems, where lower is better. Each mark is the ablation minus the full recipe on the same test instances, so a mark right of zero is an ablation that does worse, with its 95 percent wild cluster bootstrap interval over instances plus the variance between seeds. The upper rows change the writer, per-sample credit, the earlier version's goal reward and a frozen writer, over five seeds. The lower rows credit the prompting goal alone, whose four differences form a preregistered Holm family with ten seeds per arm on bin packing and the traveling salesman problem, and replace the Shapley credit with the best-of-6 and leave-one-out credits, which ran on those two problems only. A group's gain in generation 1 lands off its prompting goal for 24, 18, 33 and 27 percent in panels (a) to (d), where the full recipe reaches 0.89, 2.56, 0.38 and 9.9.

Table 19: Ablations on inference scheduling, means over five seeds on the 2,816 held-out goals, or the 1,408 no hand-written rule meets for the graded score, with Δ, RuleForge minus the row, and its 95 percent interval. Credits are centered by their group mean except the rows of PKPO (Walder & Karkhanis, 2025) and of the leave-two-out baseline of MaxPO (Takashiro et al., 2026), which use their own baselines. J_m marks the objective a credit ascends in expectation, the gain in library value of m rules sampled for one goal (Appendix B). Table 11 adds the library oracle, Holm-adjusted p-values and more ablations. The row without the library, as in GCPO, was added after the preregistration and lies outside the Holm families of Table 11.

	Chosen rule		Oracle graded	
Variant	Goals met	Δ	Score	Δ
RuleForge, Shapley credit	**71.0**		**63.1**	
How the gain is split, over which goals and against what				
Shapley credit, prompting goal only	67.8	3.2 [2.0, 4.4]	60.2	2.9 [1.8, 4.0]
Per-sample credit, as in TestDecision, J_1	66.6	4.4 [3.1, 5.7]	56.1	7.0 [5.7, 8.3]
Shapley credit without the library, as in GCPO	67.1	3.9 [2.6, 5.2]	57.1	6.0 [4.8, 7.2]
Other splits of the same gain				
Best-of-6 credit of PKPO, J_6	67.0	4.0 [2.7, 5.3]	56.7	6.4 [4.8, 8.0]
Best-of-6 credit, leave-two-out baseline, J_6	70.0	1.0 [0.09, 1.91]	62.3	0.8 [0.01, 1.59]
Centered leave-one-out credit, J_7	66.4	4.6 [3.1, 6.1]	55.9	7.2 [5.8, 8.6]
Other rewards				
CALM's reward	66.1	4.9 [3.5, 6.3]	55.6	7.5 [6.1, 8.9]
Goal reward of an earlier version, Appendix M	65.7	5.3 [4.0, 6.6]	54.5	8.6 [7.0, 10.2]

Table 20: More ablations, one part of RuleForge changed at a time, means over five seeds with 95 percent intervals, whose paired differences are in Table 11. The job-shop columns give goals met within three attempts on the 72 OR-Library development goals over RuleForge's library, whose oracle is 62.5 percent, and over the 20 classical rules, whose oracle is 50.0 percent, with intervals over the five seeds, which the deterministic fixed shortlist over the classical rules lacks.

	Inference			OR-Library job shop	
Variant	Chosen rule	Oracle	Oracle graded	Library	Classical
RuleForge	**71.0** ±1.8	**75.4** ±1.8	**63.1** ±2.5	**59.4** ±4.2	**45.8** ±3.3
Credits and recipe					
Shapley credit, not centered	70.2 ±1.8	74.9 ±1.8	62.4 ±2.5	–	–
Shapley credit with best-of-6's noise	67.5 ±1.8	72.6 ±1.8	58.9 ±2.6	–	–
Per-sample credit, prompting goal only	65.9 ±1.8	70.6 ±1.8	54.7 ±2.6	–	–
VPO	67.4 ±1.8	71.8 ±1.8	57.2 ±2.6	–	–
Pass@4 training	67.2 ±1.8	71.8 ±1.8	56.9 ±2.6	–	–
Shapley credit without the library, as in GCPO	67.1 ±1.8	72.1 ±1.8	57.1 ±2.6	–	–
Best-of-4 credit of PKPO	67.2 ±1.8	72.0 ±1.8	57.0 ±2.6	–	–
Best-of-6 credit, leave-two-out baseline	70.0 ±1.8	74.7 ±1.8	62.3 ±2.5	–	–
Shapley credit with leave-one-out's noise	66.7 ±1.8	71.9 ±1.8	57.5 ±2.6	–	–
TailRL's credit on the raw score	66.0 ±1.8	71.0 ±1.8	55.3 ±2.6	–	–
Expert iteration, no writer GRPO	65.1 ±1.8	70.0 ±1.8	54.5 ±2.6	–	–
Frozen writer	64.9 ±1.8	69.5 ±1.7	53.8 ±2.6	–	–
Random admission	67.3 ±1.8	72.4 ±1.8	58.6 ±2.6	–	–
Without grammar mutation	69.9 ±1.8	74.6 ±1.8	62.3 ±2.5	–	–
No replay of verified rules	68.6 ±1.8	73.6 ±1.8	60.2 ±2.6	–	–
Plain GRPO with replay	69.0 ±1.8	73.9 ±1.8	61.0 ±2.6	–	–
Plain on-policy GRPO, no replay	66.9 ±1.8	71.6 ±1.8	57.0 ±2.6	–	–
Archive rules in the supervised start, outside the budget	71.4 ±1.8	75.6 ±1.8	63.4 ±2.5	–	–
Selector					
First-rule reward	70.2 ±1.8	–	–	56.7 ±4.0	45.0 ±3.3
Supervised selector	69.1 ±1.8	–	–	55.0 ±4.4	43.9 ±3.3
Selector trained on shuffled rewards	66.1 ±1.8	–	–	50.3 ±4.0	41.7 ±3.8
Fixed shortlist	61.5 ±1.7	–	–	52.2 ±3.2	43.1

Figure 19 separates the policy-gradient algorithm from the credit. Beside plain GRPO with the Dr. GRPO loss (Liu et al., 2025), DAPO (Yu et al., 2025) drops groups of equal rewards and clips asymmetrically, GSPO (Zheng et al., 2025a) clips whole sequences with a length-normalized ratio, SAPO (Gao et al., 2025) gates softly, CISPO (MiniMax, 2025) clips the importance weights, and GAPO (Jia et al., 2026) widens clipping with the advantage. The recipe combines DAPO's sampling, GAPO's clipping, GSPO's ratios and replay. Pass@k training (Chen et al., 2025), with $k = 4$, and PKPO (Walder & Karkhanis, 2025) credit the best of k samples, with PKPO's leave-one-out-minus-one baseline for $k = 4$ and 6, and VPO (Bahlous-Boldi et al., 2026) scores three answers of one rollout by their best member under random weightings of the gains on the goals of the rule's window. Each credit replaces the Shapley credit inside the recipe. CALM's reward (Huang et al., 2026b) is -0.6 for a rule that ties a rule in the prompt, 1 plus its relative gain over the best parent, clipped at 1, when it beats that parent, -0.375 times its relative shortfall, clipped at 1, when it falls behind, and -0.75 to -1.0 when it does not parse, with graded scores on the prompting goal as scores and the raw relaxed metric when the best parent scores 0. Table 1's CALM uses the same rule. Every other credit arm sweeps the five writer learning rates of Table 12, CALM's published rate among them, and reports its best, while the Shapley credit keeps the two rates of its preregistered sweep. Under the Shapley credit the six algorithms span 1.2 [0.4, 2.0] points and under the goal reward 1.4, while replacing the goal reward with the Shapley credit raises the score by 8.4 to 8.6 points under every algorithm, 7.2 [5.4, 9.0] more than their largest difference, and by 8.6 under the recipe, each paired interval above zero.

The credits in training. The semivalue credits coincide in groups where at most one rule adds value on each goal, and 24 percent of the groups of generation 1 have two or more rules that add value on a common goal. Over generation 1 (Figure 18), per-sample credit raises J_1 fastest and J_6 least, leave-one-out credit raises J_1 least, and the Shapley credit raises J_6 most. The gradient signal-to-noise ratio of a credit is the norm of its mean update of the writer's adapter over the spread of the update across groups, averaged over five steps of generation 1 on 64 held-out prompts. It is 0.41 [0.37, 0.45] for the Shapley credit and 0.44 [0.40, 0.48] for per-sample credit, against 0.26 and 0.23 for PKPO's best of four and six, 0.19 for leave-one-out credit, 0.33 for the uncentered Shapley credit and 0.36 for the leave-two-out best-of-6 credit, since PKPO's estimator credits few samples of each group and leave-one-out credit credits a rule only on goals where it alone attains the group's best gain, a stronger form of the mutual masking that GCPO finds shrinks leave-one-out credit under its coverage value (Chen et al., 2026a). In the OR-Library job shop, the set reward lets the selector meet 2.7 points more goals within three attempts than the first-rule reward, within the five-seed intervals of Table 20.

Set-level credits. VPO and pass@4 training meet 67.4 and 67.2 percent of the held-out goals, 3.6 [2.5, 4.7] and 3.8 [2.7, 4.9] points below the Shapley credit, at gradient signal-to-noise ratios of 0.31 and 0.28, below the noise curve of Figure 4d, so noise alone does not account for their loss.

Credit and realized gain. A credit is a proxy for the gain a rule realizes at admission, which keeps K rules across all groups. Over the surviving fresh rules of generation 1, the rank correlation between a rule's credit and its gain against the library L' of the next admission, $F(L' \cup \{r\}) - F(L' \setminus \{r\})$, is 0.58 [0.55, 0.61] for the Shapley credit, against 0.44 [0.41, 0.47] for per-sample credit, 0.49 [0.45, 0.52] for the best-of-6 credit and 0.41 [0.38, 0.45] for leave-one-out credit.

Variance and plain GRPO. The noise-matched arms add zero-mean Gaussian noise to each Shapley advantage after dynamic sampling, scaled so that the gradient signal-to-noise ratio matches a credit's. Matched to each of the five group credits, the noisy Shapley credit leads it by 0.2 to 0.6 points with an interval that includes zero (Table 11). At the best-of-6 credit's ratio it meets 67.5 percent of the held-out goals and leads that credit by at most 1.3 points, so the best-of-6 credit's objective accounts for at most that much of its 4.0-point loss. The best-of-6 credit with MaxPO's leave-two-out baseline (Takashiro et al., 2026) keeps its objective at a ratio of 0.36 and trails the Shapley credit by 1.0 [0.09, 1.91] points, with no detected deficit in J_6 but a lower J_1 at step 100, 0.005 [0.001, 0.009], as the two-kind example of Appendix B predicts. Under plain on-policy GRPO, where Proposition 1 is exact, the Shapley credit leads per-sample, best-of-6 and leave-one-out credit by 3.1 [2.0, 4.2], 2.8 [1.5, 4.1] and 3.4 [2.3, 4.5] points, and dropping replay, sampling, clipping and ratios together costs it 4.1 [2.8, 5.4] points. Figure 4e bins these on-policy arms by a

prompt's success rate p on its prompting goal, estimated at step 0 from 64 samples of the shared writer. Relative to per-sample credit, whose weight is flat, the gain below $p = 0.1$ over the gain between 0.1 and 0.25 is 2.17, 1.92 and 1.58 for the leave-one-out, best-of-6 and Shapley credits, in the order of the values 2.27, 1.99 and 1.42 that Figure 5 predicts. The noisier best-of-k credits gain less than their weights suggest, so below $p = 0.25$, where 92 percent of the prompts start, the Shapley credit gains the most, 0.016 [0.010, 0.022] more than the best other credit below 0.1.

A powered test of the split. Two credits stay within noise of the Shapley credit in the Holm families, its uncentered form and the best-of-6 credit with the leave-two-out baseline. A follow-up, preregistered before its runs, trains 15 seeds of each and scores them on the pooled held-out goals of BurstGPT, Azure and Mooncake, 8,448 goals on 132 segments. The Shapley credit leads the leave-two-out credit by 1.0 [0.6, 1.4] points and the uncentered credit by 0.8 [0.4, 1.2], with Holm-adjusted p below 0.001 for both. A gap from noise should shrink with training, and a gap from the objective should not. Under plain on-policy GRPO with ten times the steps, outside the budget, the gap to the leave-two-out credit holds at 0.9 [0.5, 1.3]. The gap to the uncentered credit falls to 0.3 [-0.1, 0.7], and the gap to PKPO's best-of-6 credit falls from 2.8 to 1.2 [0.7, 1.7], toward the leave-two-out credit's. The leave-two-out credit thus trails on its objective, J_6, and the uncentered credit, whose objective adds only J_n/n, mostly on noise.

Group size and the batch split. Proposition 1 predicts how the Shapley credit's weight on a prompt's success rate p changes with the group size n. Under plain on-policy GRPO at a fixed number of samples, groups of 4, 8 and 16 rules meet 66.2, 66.9 and 67.1 percent of the held-out goals, within noise of one another. Their gain below $p = 0.1$ over their gain between 0.1 and 0.25, relative to per-sample credit, is 1.27, 1.58 and 2.19, and it rises with n as the predicted 1.14, 1.42 and 1.97 do. The group split ignores the survivors of other groups, which admission weighs. Splitting each goal's gain over all groups of a step instead, with centering per group, needs no more simulator runs and changes goals met by -0.3 [-1.0, 0.4] points, so the group split loses nothing measurable.

The scope of the credit. Crediting every training goal matters in proportion to how much of a group's gain lands off its prompting goal. Thinning or doubling the training goals of each inference window moves that share from 17 to 49 percent and the gain from crediting every goal from 0.9 to 4.1 points, a slope of 0.10 [0.05, 0.15] points per percentage point that meets the preregistered bar (Figure 4). With the goals fixed, crediting a random quarter or half of the other goals gives a slope of 0.076 [0.038, 0.114] per percentage point of credited share, so the gain grows with the credited share at a fixed goal density too. On the other problems the scope gains survive Holm's correction on bin packing and the traveling salesman problem, with p of 0.001 and 0.001, and include zero on the two shops, where the least of a group's gain lands off the prompting goal and the family detects only about 0.03 and 0.14 at 80 percent power (Figure 14). Figure 15 crosses the scope with the split. Crediting every goal adds 3.2 [2.0, 4.4] points under the Shapley split and 0.7 [-0.2, 1.6] under per-sample credit, which credits a gain that several rules share on another goal to each in full. Their difference, 2.5 [1.2, 3.8] points from the paired bootstrap of Appendix E, excludes zero, so the split matters more when every goal is credited. The Shapley split of the group's own value with no library in its baseline, as in GCPO (Chen et al., 2026a), trails the recipe by 3.9 [2.6, 5.2] points.

Diversity and survival. Over generation 1, the writer's samples under the Shapley credit grow slightly more varied than a frozen writer's, while per-sample credit and the goal reward narrow them, and more of its samples survive verification (Figure 17). Its library also ends generations 1 and 2 with the highest oracle graded score, from a shared generation 0 (Figure 16).

J Other inference settings, traces and budgets

This appendix tests whether RuleForge's lead depends on its start, the inference setup, the trace or the budget.

With archive rules in the supervised start. Adding the goal-reward loop's archive targets to the writer's first fine-tuning (Appendix D), at the same 30,833 evaluations, gives a chosen rule that meets 71.4 percent of the held-out goals and libraries that meet 75.6 percent as library oracles with

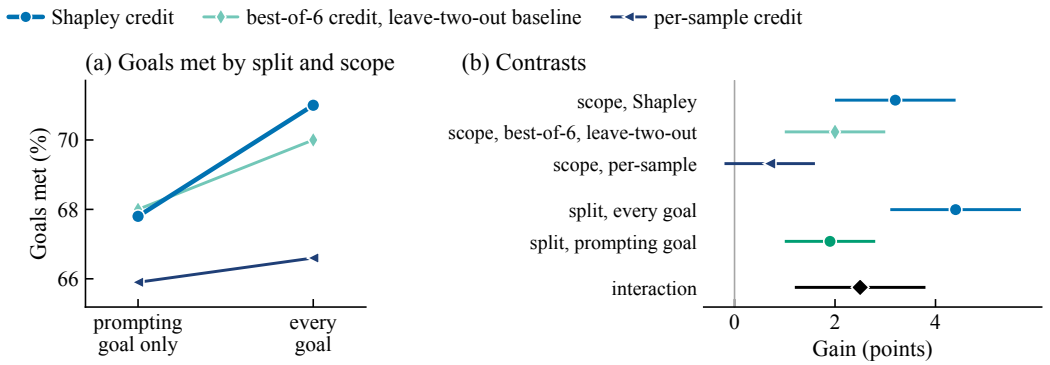

Figure 15: The split of a group's gain crossed with the goals it is credited on, in held-out inference goals met by the chosen rule, means over five seeds. (a) The Shapley credit, the best-of-6 credit with the leave-two-out baseline and per-sample credit, each credited on the prompting goal alone and on every goal. (b) Paired contrasts with 95 percent intervals, the scope gain of each split, the Shapley credit minus per-sample credit at each scope, and their interaction, the scope gain under the Shapley split minus the scope gain under per-sample credit, 2.5 [1.2, 3.8] points.

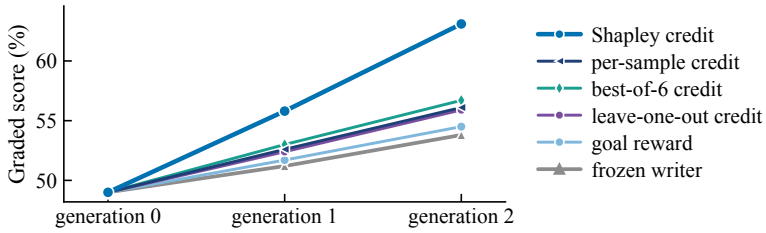

Figure 16: Oracle graded score of the library by generation on the 1,408 held-out goals no hand-written rule meets, for the Shapley, per-sample, best-of-6 and leave-one-out credits, the goal reward and a frozen writer inside the RuleForge recipe.

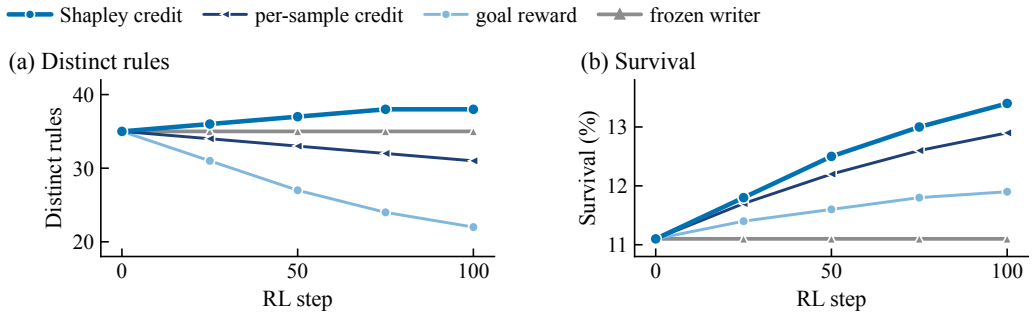

Figure 17: The writer's samples over the GRPO steps of generation 1 under the Shapley credit, per-sample credit and the goal reward, and for a frozen writer. (a) Distinct rules among 48 samples of eight prompts. (b) The share of these samples that survive verification. All four start from the same writer at the start of generation 1, which the frozen writer keeps.

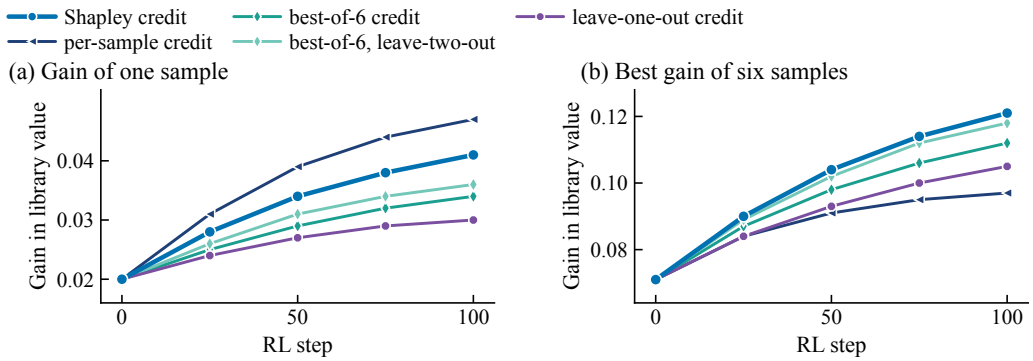

Figure 18: The objectives of Proposition 1 over the writer's GRPO steps of generation 1, estimated on 64 held-out prompts with 16 samples each. (a) J_1, the expected gain in library value of one sample. (b) J_6, that of six samples, the number the writer proposes per missed goal.

a graded score of 63.4. The archive start thus changes goals met by 0.4 [−0.4, 1.2] points for the 809,232 evaluations that built the archive, which no budget counts, so the final recipe leaves it out.

The earlier held-out set. On the 352 goals on which none of the goal-reward loop's preregistered tests passed (Appendix M), whose 11 trace segments include nine that also supply development rows, RuleForge's chosen rules meet 69.4 percent against 64.0 for the best other LLM-driven method.

Other inference settings. Figure 20 repeats the comparison with the best other LLM-driven method under thirteen changes to the setting. With predicted lengths, every rule and classical ordering reads a length predictor fit on the training windows from the prompt length and the tenant instead of the true response length, and RuleForge's chosen rules then lead the classical pool by 13.2 [10.8, 15.6] points. Other settings bound the 99th-percentile wait of all requests or the 90th-percentile wait of served requests, fix thresholds per tenant configuration, use the goal-reward loop's training thresholds, draw held-out thresholds from the outcomes of every method's library instead of the hand-written rules', let the selector read the window it schedules, change the offered load to 0.8 or 1.5, or carry unserved requests into the next window. The paired interval excludes zero in every setting but an offered load of 0.8, where the rules' outcomes differ little.

Starvation. Under overload the slots stay busy, and starting one of a tenant's waiting requests rather than another of equal work leaves the tenant's mean wait unchanged, so only a threshold on the 99th-percentile wait, as in Figure 20, registers starvation within a tenant. Figure 21 reports who is served. Starvation prevention brings shortest first's oldest waiter from 97 to 83 percent of the span, while under RuleForge's and CALM's chosen rules the oldest waiter is older, at 90 and 91 percent. Under the 99th-percentile bound RuleForge's chosen rules lead the classical pool, which includes that prevention, by 7.1 [5.4, 8.8] points. Like shortest first and CALM's chosen rules, RuleForge's chosen rules serve few of the largest requests, 7.9 percent of the largest tenth against 9.8 for CALM's and 23.9 under first come, first served, trading the service of long requests for tenant shares and waits a little more than CALM's rules do.

Replay in vLLM. We replay each method's chosen rules and the classical orderings as the scheduling policy of vLLM (Kwon et al., 2023), serving the main model on one H100 with chunked prefill at the default limit on running sequences, where memory for the KV cache binds. The rule scores the waiting requests at every step, and when the cache fills vLLM preempts the running request with the lowest score. Outputs are fixed to their trace lengths, offered load is scaled to the simulator's three to six times overload, and goals set thresholds on the 90th-percentile time to first token, the mean time per output token and the tenant share error, drawn as in Appendix C from the hand-written rules' measured outcomes on the 176 fresh windows, each verdict averaging three passes. Each selector reads a vLLM goal without retraining, as a simulator goal whose wait-spread

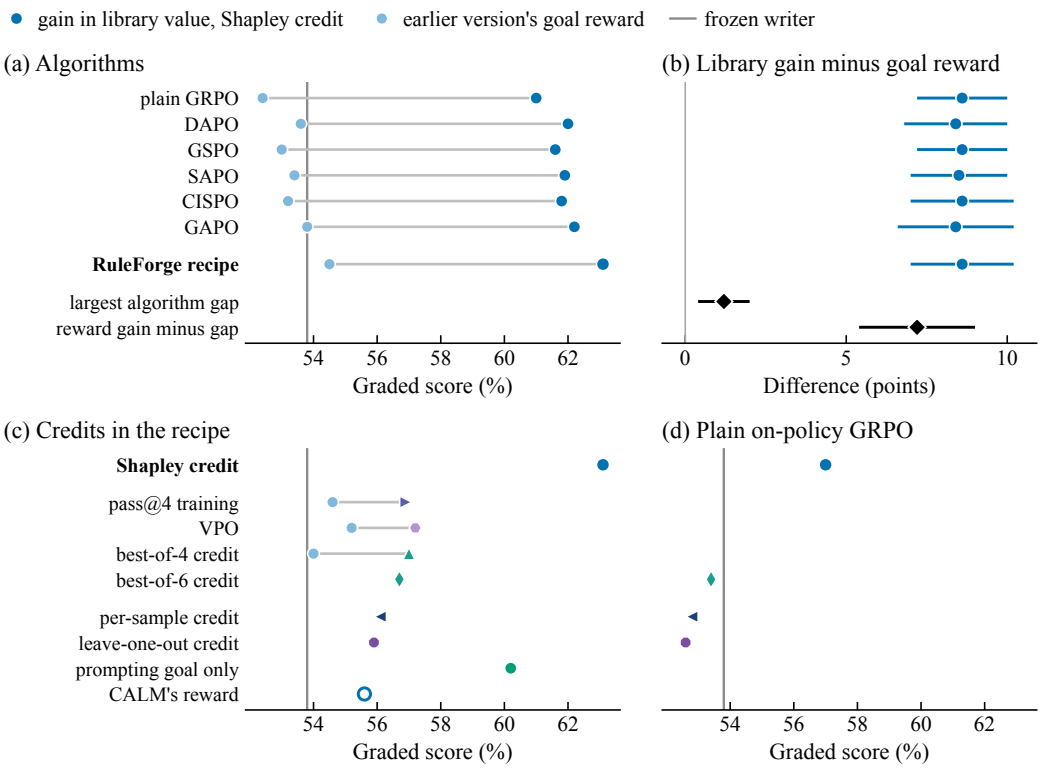

Figure 19: Generation-2 graded score on the 1,408 held-out inference goals no hand-written rule meets, means over five seeds, with the frozen writer as a gray line. (a) Six policy-gradient algorithms, which keep the recipe's replay, and the RuleForge recipe, each with the Shapley credit on the gain in library value and with the earlier version's goal reward. (b) The paired difference between the two rewards under each, with its 95 percent interval, then the largest difference between two algorithms under the Shapley credit, GAPO minus plain GRPO, and the smallest gain from the reward minus that difference, 7.2 [5.4, 9.0] points. (c) Each credit in place of the Shapley credit inside the recipe, on the same marginal gains except CALM's reward, which scores a rule against the best parent in its prompt, with pass@4 training, VPO and the best-of-4 credit also on the goal reward. (d) Four of these credits under plain on-policy GRPO without replay, where Proposition 1 is exact. Table 11 gives the paired differences of the rows it shares. On the 72 OR-Library development goals, the selector's set reward meets 59.4 and 45.8 percent within three attempts over RuleForge's library and over the 20 classical rules, against 56.7 and 45.0 for the first-rule reward.

threshold is the 90th-percentile time to first token divided by the slice's mean work, whose share threshold is unchanged and which leaves out time per output token. RuleForge's chosen rules meet 61.0 percent of the goals, against 53.1 for CALM's, 51.8 for the classical pool and 38.4 for VTC, paired differences of 7.9 [5.9, 9.9], 9.2 [7.5, 10.9] and 22.6 [20.3, 24.9] points (Figure 22a). With time to first token over served requests only they meet 63.4 percent against 57.1 for CALM's, 6.3 [4.6, 8.0] points, and with predicted lengths for every ordering, as in Figure 20, 58.9 against 52.0, 6.9 [5.1, 8.7]. At offered loads of 0.7, 0.9, 1.1 and 1.5, with bounds fixed per load, the lead over CALM is 1.0 [−0.1, 2.1], 2.9 [1.8, 4.0], 5.5 [4.1, 6.9] and 7.9 [6.1, 9.7] points (Figure 22c), so it fades as the load falls toward what vLLM can serve.

New traces. Without retraining, we replay the final libraries and selectors on 2,816 held-out goals on 176 windows built the same way from the 2024 Azure LLM inference trace (Stojkovic et al., 2025) and the Mooncake trace (Qin et al., 2025), whose prefix hashes the simulator ignores. RuleForge's chosen rules meet 67.3 and 64.9 percent of the goals, against 61.2 and 58.9 for the best other

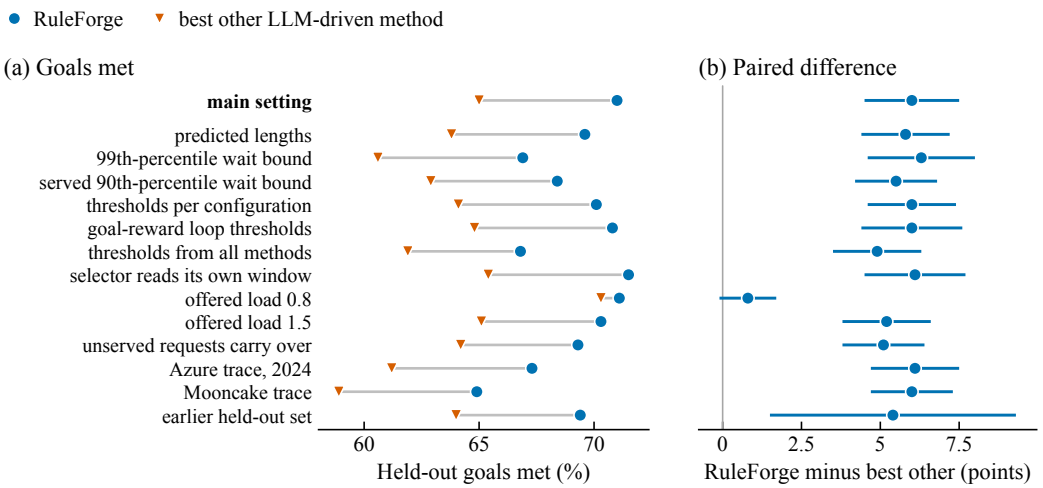

Figure 20: RuleForge's chosen rules against the chosen rules of the best other LLM-driven method under other inference settings, on 2,816 held-out goals over 44 segments except the earlier held-out set, 352 goals over 11 segments. (a) Held-out goals met. (b) The paired difference with its 95 percent interval, which excludes zero in 12 of the 13 settings, all but an offered load of 0.8, where the rules' outcomes differ little. The main setting repeats Table 1.

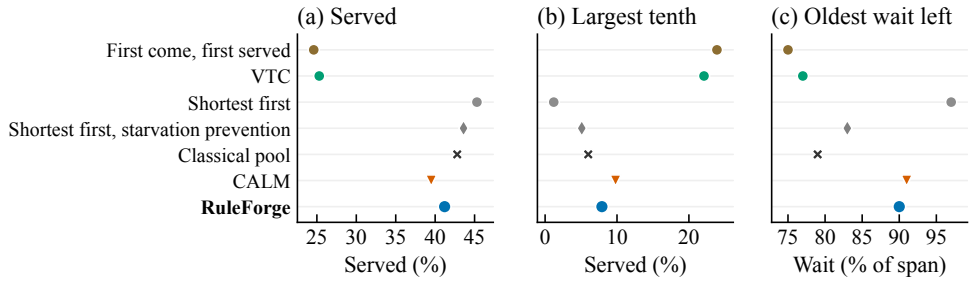

Figure 21: Who is served on the fresh held-out inference windows under each ordering, means over the 176 windows. (a) The share of requests that start by their slice's end, as in Table 5. (b) The share served of the tenth of requests with the most work. (c) The age of the oldest request still waiting when the slice ends, in percent of the slice's span. A size-neutral order admits about one request in ρ, the ratio of offered work to capacity. The starvation prevention is that of Fu et al. (2024), and CALM and RuleForge run their chosen rules.

LLM-driven method's, and as library oracles RuleForge's libraries meet 71.6 and 69.0 percent, the greedy archive library 66.1 and 63.6, and the best other method's 65.3 and 62.7.

Smaller budgets on bin packing and the traveling salesman problem. At 1,000 to 20,000 evaluations under the shared protocol, down to the budgets of MCTS-AHD and CALM, RuleForge has the smallest mean gap at every budget, and its paired interval against the next best method excludes zero at every budget but 1,000 and 2,000 evaluations on the traveling salesman problem (Figure 23).

Figure 1 in paired form. In Table 21, RuleForge's mean leads CALM's at every budget on inference scheduling, with intervals that exclude zero at the full budget only, and on both shops at every budget.

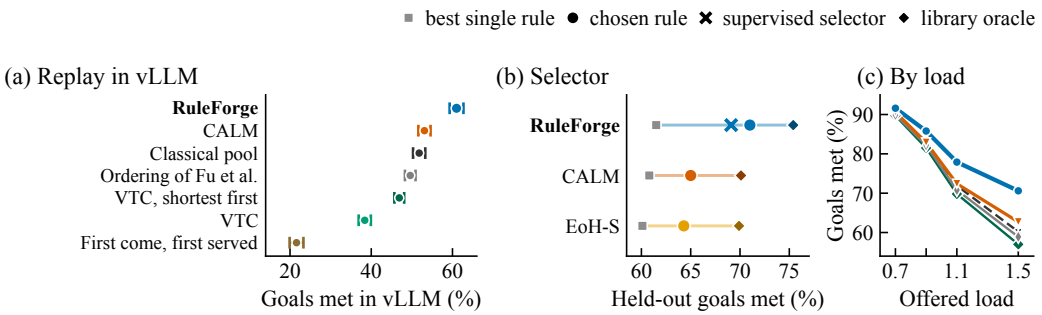

Figure 22: Scheduling with the chosen rules. (a) Goals on time to first token, time per output token and tenant share error met when vLLM runs each ordering on the held-out requests at three to six times overload, with 95 percent intervals. (b) Held-out inference goals met by each library's best single rule, by the rule the selector picks and by the library oracle, with the supervised selector over RuleForge's library as a cross. (c) Goals met in vLLM against offered load, with bounds fixed per load and the classical pool dashed.

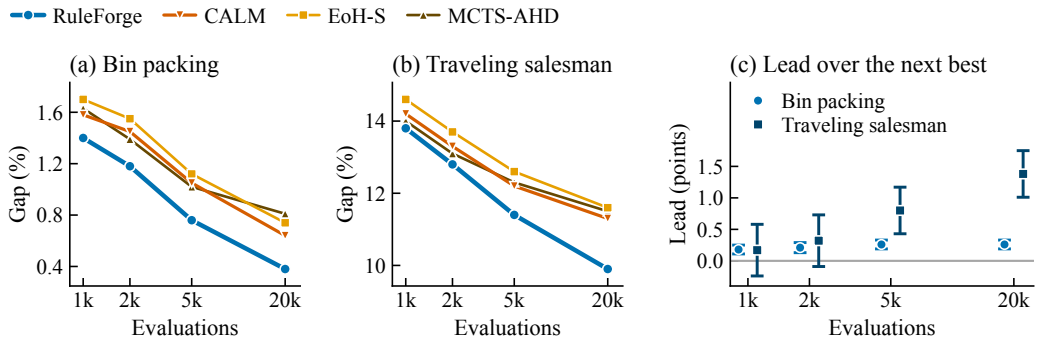

Figure 23: Bin packing and the traveling salesman problem at smaller budgets under the shared protocol, where RuleForge scales its proposals and GRPO steps to the budget. (a) and (b) The mean gap over five seeds against evaluations, lower is better. (c) The next best method minus RuleForge at each budget, with its paired 95 percent interval, positive where RuleForge leads. MCTS-AHD's own budget is 2,000 evaluated heuristics on bin packing and 1,000 on the traveling salesman problem, and CALM's is 2,000 model queries.

Table 21: Figure 1 in paired form, with 95 percent intervals and a positive value favoring RuleForge. On inference scheduling RuleForge's smallest run at 10,240 evaluations and its generation-0 libraries at 15,290 face CALM's runs of 10,000 and 20,000, and the last row is RuleForge's generation 1, at 24,590 evaluations, minus the greedy archive library.

	Budget in evaluations			
Problem	2,000 or 10,000	5,000 or 20,000	10,000	20,000 or 30,833
Inference, goals met	0.6 [−0.2, 1.4]	0.7 [−0.1, 1.5]	–	6.0 [4.5, 7.5]
Dynamic job shop	0.044 [0.025, 0.063]	0.051 [0.031, 0.071]	0.057 [0.037, 0.077]	0.068 [0.047, 0.089]
Flow shop	0.13 [0.03, 0.23]	0.21 [0.12, 0.30]	0.29 [0.18, 0.40]	0.37 [0.23, 0.51]
Generation 1 minus the greedy archive library, goals met				3.2 [2.0, 4.4]

Budget frontier. A single budget can rank search methods wrongly (Oved et al., 2026), so for RuleForge, CALM, EvoTune and EoH-S we split each total budget between seeds and iterations on a grid and keep the split that scores best on the training goals. RuleForge's smallest run, one generation with four rules per prompt instead of six and no GRPO steps at 10,240 evaluations, meets 62.8 percent of the held-out goals with its chosen rule and 68.1 as a library oracle, and its frontier lies above the baselines' at every budget, inside the noise at the smallest (Figure 24).

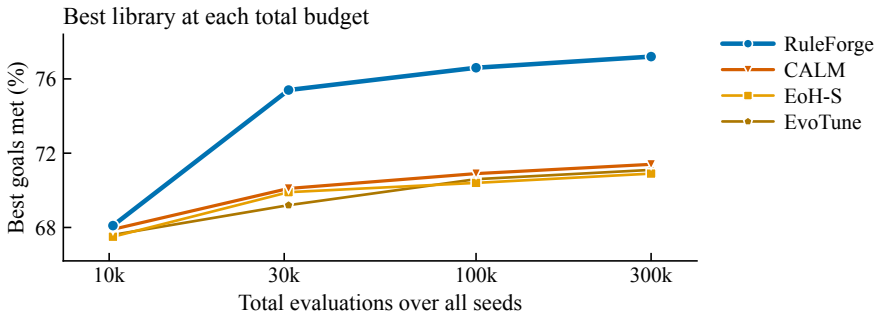

Figure 24: The best held-out goals met by a library oracle on inference scheduling at each total number of evaluations, over the splits of the budget into seeds and iterations (Oved et al., 2026).

K THE SELECTOR AND THE WRITER IN INFERENCE SCHEDULING

Figure 25 compares the writers at generation 2 and Figure 16 follows the credits by generation. A frozen Nemotron 3.5 Lightning writer whose prompt GEPA (Agrawal et al., 2026) optimizes for the admission gain of its rules meets 66.3 percent of the held-out goals, 4.7 [3.4, 6.0] points below the trained writer (Table 11).

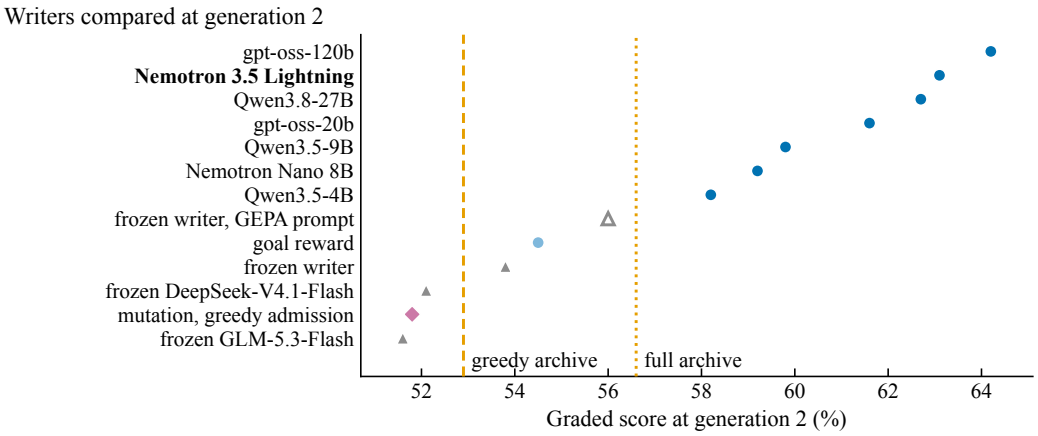

Figure 25: Writers at generation 2, by the oracle graded score of their libraries on the 1,408 fresh held-out goals no hand-written rule meets. Blue circles are the trained writers of the seven models, the light blue circle the goal reward, the gray triangles frozen writers, the hollow triangle a frozen writer with a GEPA-optimized prompt (Agrawal et al., 2026) and the diamond mutation with greedy admission, with the greedy archive library dashed and the full archive dotted.

The inference selector. The selector chooses one of the 34 library rules for each goal. Its chosen rule meets 71.0 percent of the held-out goals, against 69.1 for the supervised selector, 61.5 for a

fixed shortlist, which runs the library's best single rule, and 75.4 for the library oracle (Figure 22b), and its gain over the supervised selector is 1.9 [0.9, 2.9] points.

Why the selector recovers more of RuleForge's library. The chosen rule recovers 68 percent of what RuleForge's library oracle adds over its best single rule, against 49 percent with CALM's reward inside the recipe, 45 percent for CALM's library, 47 for EvoTune's and 43 for EoH-S's. The Shapley credit rewards a rule only for goals the library misses and splits each such goal among the rules that fix it, so each admitted rule tends to own goals of a recognizable kind. On the held-out goals that exactly one library rule meets, 24 percent of RuleForge's goals against 17 percent of CALM's, the selector picks that rule for 88 percent of RuleForge's goals, against 79 percent with CALM's reward and 74 percent for CALM's library, differences of 9 [4, 14] and 14 [8, 20] points. Reinforcement learning on CALM's reward inside the recipe reaches 66.1 percent of the held-out goals, no more than the 66.3 percent of GEPA's prompt optimization with the frozen writer.

Repeats of the library. The verifier logs whether a candidate's prescreen outcomes equal a library rule's on its window to six decimals. Such a repeat costs an evaluation and adds nothing. The seven evolutionary searches spend 38 to 44 percent of their inference evaluations on repeats, grammar mutation 33 percent, CALM 29 and RuleForge's writer 12 (Table 13). Across RuleForge, mutation and the eleven baselines of Table 1, the repeat share and the library oracle have a rank correlation of -0.89. The writer's share follows its reward. The frozen writer repeats the library in 37 percent of its proposals. Trained on the goal reward, CALM's reward, per-sample credit and the Shapley credit, it repeats it in 34, 27, 14 and 12 percent. The two credits that score a rule against the whole library cut repeats the most.

Controls. Each learned part has a control that removes it, a frozen writer, mutation in place of the writer, the goal reward in place of the Shapley credit, a fixed shortlist in place of the selector, and a selector trained on shuffled rewards. The greedy archive library, a reference no budget buys, adds to the hand-written rules the 24 archive rules of largest gain on the training goals by Equation 4.

L THE OR-LIBRARY JOB SHOP

Each of the 82 job-shop instances of the OR-Library (Beasley, 2026) receives due dates $d_j = k \sum_o p_{jo}$ with tightness k of 1.3, 1.6 or 2.0, which gives 246 units, with training goals on 48 base instances and test goals on the other 34, which include the largest. A unit's goals set thresholds on one to three of the six metrics of Table 5. The selectors use Nemotron 3.5 Lightning, except in Figure 26a, train on 1,030 goals and test on the 72 development goals of the earlier study, half of which no classical rule meets, and a preregistered replication tests on 300 fresh goals. RuleForge's job-shop library comes from five runs of the recipe starting from the 20 classical rules with up to 12 admissions per generation. The primary measure, success@k, counts the goals one of the first k listed rules meets, at $k = 3$, with an oracle of 62.5 percent over RuleForge's library and 50.0 over the classical rules. A secondary measure, near-miss success@k, also counts a goal no classical rule meets when a listed rule meets its other thresholds and comes within 2 percent of the best classical value on the relaxed metric.

Where the selector gains. With Nemotron 3.5 Lightning, GRPO raises goals met within three attempts on the 72 development goals, which guided the recipe, from 55.0 to 59.4 percent over RuleForge's library and from 43.9 to 45.8 percent over the classical rules, the latter within the five-seed intervals. Counting near misses, the selector over the classical rules rises from 70.0 to 80.3 percent, and the selector over RuleForge's library reaches 83.3 percent (Figure 27). On the 36 goals no classical rule meets, CP-SAT with an hour per goal proves 18 infeasible and meets 5 (Figure 28), and RuleForge's library meets 9 of them, the 5 CP-SAT meets and 4 it leaves unresolved, none that CP-SAT proves infeasible. Since the instances are known in advance, running all of a library's rules reaches the oracle in a median 0.16 s, so the study measures how well a selector learns to choose. Pooled over the seven models, GRPO gains 4.2 points over the supervised selector with an interval above zero, from 1.9 to 6.7 on single models (Figure 26a and Table 18).

A complete replication. A preregistered replication retrains the selectors with Nemotron 3.5 Lightning for ten seed pairs and tests them on 300 fresh goals, four per unit on the three tightness

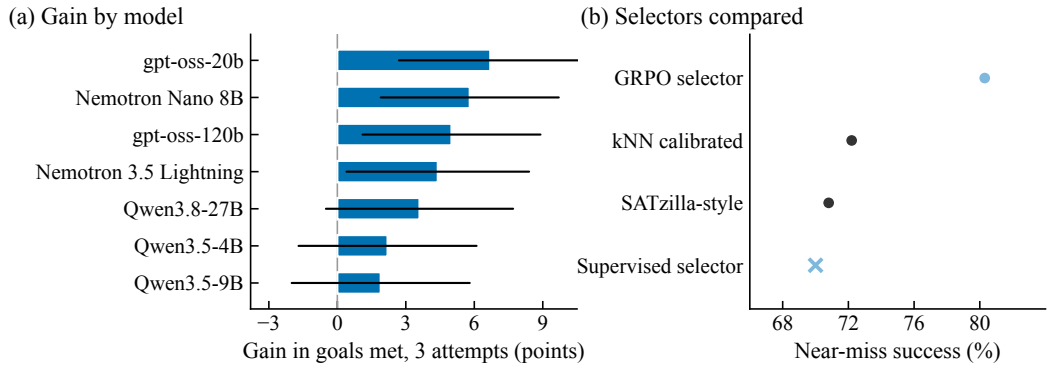

Figure 26: Selectors on the OR-Library job shop. (a) Gain in goals met within three attempts over the supervised selector on the 72 development goals, with 95 percent paired intervals, for the selector over each model's RuleForge library, where the gain pooled over the seven models has an interval above zero. (b) Near-miss success within three attempts on the same goals for four selectors over the 20 classical rules, the supervised and GRPO selectors of Nemotron 3.5 Lightning and two without a language model.

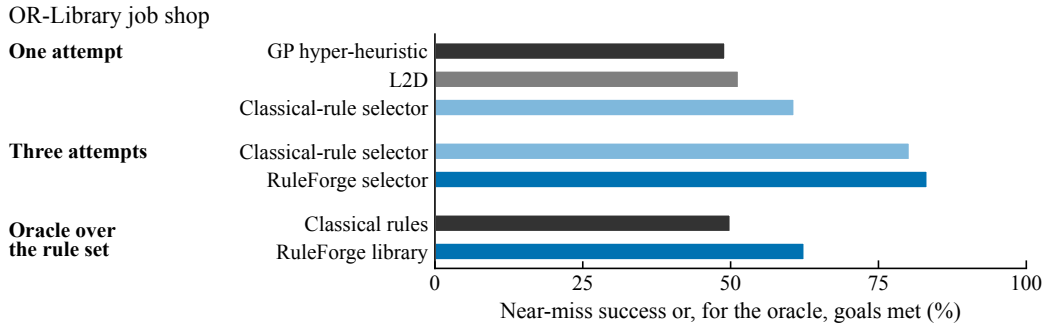

Figure 27: Near-miss success on the 72 OR-Library development goals with one and three attempts, and goals met under each rule set's oracle. The single-attempt rows are a rule evolved by genetic programming, L2D and the selector over the 20 classical rules, and the three-attempt rows set that selector beside the selector over RuleForge's library, both with Nemotron 3.5 Lightning. The oracle rows count goals met only.

factors of the 25 test base instances outside the nine of the development goals, drawn with a new seed after the recipe was frozen, on a library grown on training instances only. Over RuleForge's library, goals met within three attempts rise by 4.1 [1.8, 6.4] points over the supervised selector, from a wild cluster bootstrap over the base instances plus the variance between the ten seed-pair means. Over the 20 classical rules they rise by 6.3 [3.7, 8.9] points over shuffled-reward controls but by only 1.1 [−0.3, 2.5] over the supervised selector, while near-miss success@3 rises by 4.6. All ten pairs finished, and both preregistered gains have intervals above zero. The earlier, incomplete replication on 320 goals is in Appendix M.

An exact solver on the development and replication goals. Figure 28 gives CP-SAT's results on the 72 development goals and the 320 goals of the earlier replication, each solved from scratch at each budget. Every schedule is checked against all thresholds with the simulator's metric code, while an infeasible verdict is the solver's own. The model lets a machine idle while an operation waits, which no non-delay dispatching rule does, so a met goal shows what some schedule can reach,

not what our grammar can express. Hinted with the best schedule of RuleForge's library, CP-SAT meets 41.3 percent of the 300 replication goals it does not prove infeasible within an hour.

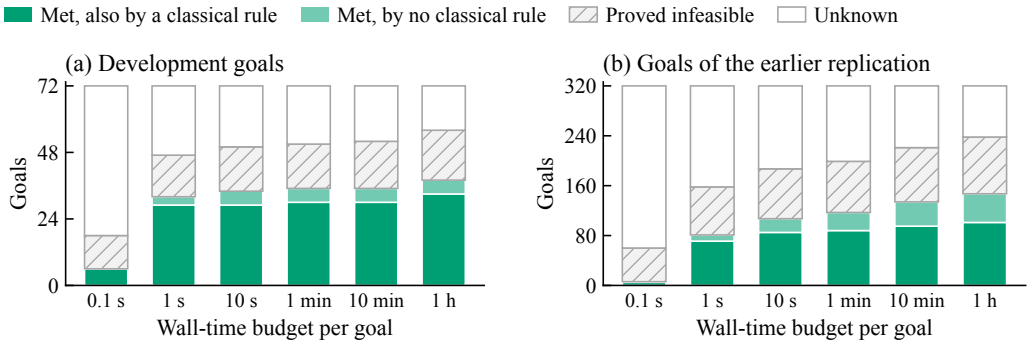

Figure 28: An exact CP-SAT solver on the OR-Library job-shop goals, (a) the 72 development goals and (b) the 320 goals of the earlier replication, solved from scratch at each wall-time budget with every threshold a hard constraint, four workers and deliberate idling allowed. Met counts goals with an independently verified schedule, split by whether some classical rule meets the goal, and infeasible counts the solver's own proofs. A schedule found but not verified within the budget counts as unknown, one goal of the 72 at 0.1 s and five and three of the 320 at 0.1 s and 1 s. Some classical rule meets 36 of the 72 goals and 116 of the 320. The median time to a verified schedule on the 72 goals runs from 0.07 s at the smallest budget to 0.33 s at an hour, while a classical rule schedules an instance in a median 2.8 ms on one CPU core.

M THE EARLIER VERSION OF RULEFORGE

Before the Shapley credit, RuleForge rewarded its writer with the *goal reward*, which scores a rule r by its own result on its prompting goal g, with η the share of the gap to the threshold that r closes,

$$R_{\text{goal}}(r) = \begin{cases} 1 & \text{if } r \text{ meets } g \in M_L, \\ 0.5 + 0.5\,\eta & \text{if } r \text{ improves } g \in M_L, \\ 0.3 & \text{if } r \text{ meets a goal the library already meets,} \\ 0 & \text{otherwise,} \end{cases} \tag{15}$$

where M_L is the set of training goals the library L misses, and the third case cannot arise in the final recipe, which prompts only missed goals. This version, the goal-reward loop, admitted rules by a thresholded count, 3 for each goal the library cannot meet that a survivor newly meets and 1 for each such goal on which it beats the best earlier rule by more than 2 percent, which depends on the order of admission, so the greedy bound of Appendix B does not cover it (Table 22). Its ten confirmatory runs, five trained and two frozen Qwen3.5-4B writers and three with Nemotron Nano 8B (Bercovich et al., 2025), grew libraries that meet 69.5 percent of the 352 held-out goals as library oracles, 0.4 points below the greedy archive library and 1.5 below the full archive, and reinforcement learning did not improve the writer. Job-shop results in this appendix use near-miss success@k.

Why the reward changed. The goal reward credits a rule for what it does on its own. Among the generation-2 proposals of a matched control, 787 of the 847 positive rewards of trained writers and 148 of the 160 of frozen writers came from its 0.3 branch, a rule that meets a goal the library already meets and adds nothing to it. A writer fine-tuned on hand-written rules alone spent all 2,712 of its generation-0 proposals per run on two or three rules the library already held, and with archive targets the writer's share of the eight admissions still fell from 7.0 in generation 0 to 1.4 in generation 2. On inference scheduling the selector added only 0.09 to 0.34 points of goals met over 20 runs, and the preregistered test failed, since libraries of overlapping rules leave little to choose among. These results led to the Shapley credit, which gives a repeat nothing, and to greedy admission on F, which grows a library for coverage.

Table 22: Settings in which the goal-reward loop differs from the final recipe of Tables 8 and 9. Every other setting is shared.

Step	Setting in the goal-reward loop
Prompted goals	Every training goal the library misses that has a relaxed metric, plus a random sample of goals the library meets one third as large and at least one.
Admission	A count of 3 for each training goal the library cannot meet that the survivor newly meets and 1 for each such goal on which it meets the other thresholds and beats the best of the library and the earlier admissions on the relaxed metric by more than 2 percent. The best survivor with a positive count is admitted and the counts are recomputed, up to 8 admissions per generation in inference scheduling and 12 in the OR-Library job shop.
Writer reward	The goal reward, from the proposed rule's evaluation on the goal that prompted it. 0 if every metric lies within 10^{-6} relative of some library rule on that window, 1.0 if it meets every threshold of a goal the library cannot meet, 0.3 if it meets every threshold of a goal the library meets, $0.5 + 0.5\eta$ if it meets the other thresholds and beats the library's best graded value by more than 2 percent, where η is the fraction of the gap to the threshold it closes, and 0 otherwise, as in Equation 15.
Writer GRPO	The Dr. GRPO loss with fixed clipping, no dynamic sampling and no replay, on prompts for every training goal.
Selector	Six supervised epochs and 400 GRPO steps with position weights $w = (1, 1, 1)$ and shaping weight $\lambda = 1.0$ in the confirmatory runs, and four epochs, 200 steps, $w = (1.0, 0.8, 0.6)$ and $\lambda = 0.5$ in the first launch. The pass test of Equation 14 did not check the other thresholds (Appendix M).
Label-fixing variant	From generation 1 on, a feasible training goal keeps the previous generation's selector target while that rule still meets it. At generation 1 this kept 482 and 484 of 719 labels in two runs.
Selector rate	The preregistration sets 10^{-5} for the Llama-3.1-Nemotron-Nano-8B-v1 selector, and the launch environment of the confirmatory runs does not record the rate, so it is not confirmed.
Models	Qwen3.5-4B, Qwen3.5-9B and Llama-3.1-Nemotron-Nano-8B-v1, loaded in bf16 with no pinned revision.
Hardware	One A100 GPU with 80 GB per job, and eight-H100 nodes for the 14 runs of the matched hand-written-only control.
Seed caveat	The supervised trainer did not receive s and built its adapter before the seed was reset, so runs of different recipes with the same seed do not share an initial adapter.
Generations	Three per run, five in two development runs.

What the admitted rules look like. The three admitted rules of Table 2 come from the loop's confirmatory libraries, and Table 23 gives the best rule of each library. Alone, the median admitted rule meets 175 of the 352 held-out goals, the best rule of each library 200 to 212 and shortest-first, the best hand-written rule, 151 (Figure 29). Removing the negative wait terms of the three rules of Table 2, which favor newer requests among similar sizes, cuts the goals they meet from 205, 200 and 201 to 170, 171 and 171, and their share gate is worth 33 goals to the first rule and 3 and 2 to the others. Of the 240 admitted rules, 105 branch on the tenant's share of service, 44 percent against 17 percent of the archive, 12 are verbatim archive rules, and 47 meet a held-out goal no archive rule meets. Figure 30 follows one library through three generations, in which most rules admitted after generation 0 are one edit from an earlier one.

The earlier selector studies. With Qwen3.5-4B (Qwen Team, 2026a) and the original near-miss check, GRPO raised the near-miss success@3 of the selector over the 20 classical rules on the 72 development goals from 69.4 to 79.4 percent, while shuffled rewards lowered it from 74.3 to 59.7. An earlier preregistered replication on all 320 held-out goals of the test file, which include the 72, retrained with the corrected check. On the four of its five seed pairs that finished, near-miss success@3 rose by 3.75 points, 1.39 to 6.02 from a cluster bootstrap that omits the variance between seed pairs and is therefore too narrow, against a bar of an interval above zero and a mean of at least 3.0, and by 7.58 points, 3.15 to 12.11, over shuffled-reward controls, while goals met rose by only 0.23 to 0.63 points. Both verdicts stay incomplete because the fifth pair did not finish, and a later descriptive re-run of it gives 3.81 points, 1.30 to 6.37.

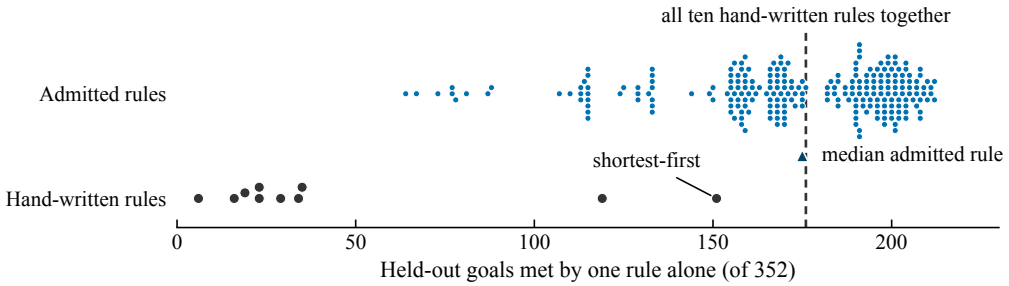

Figure 29: Held-out goals, of 352, that each of the 240 rules admitted by the goal-reward loop's ten confirmatory libraries meets alone, one dot per rule, above the ten hand-written rules, which meet 6 to 151. The dashed line marks the 176 goals the hand-written rules meet together, and the triangle the median admitted rule.

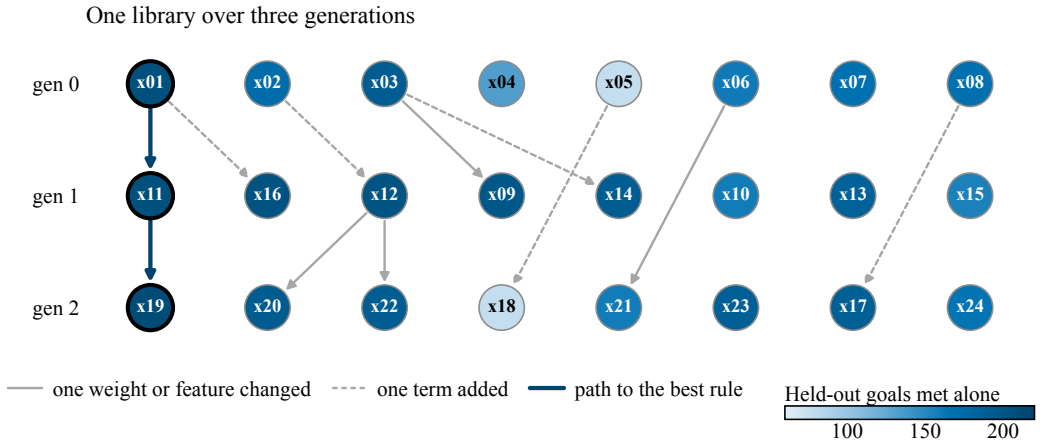

Figure 30: One library of the goal-reward loop over three generations, from the confirmatory run with a trained Nemotron Nano 8B writer and seed 0. Each node is an admitted rule, shaded by the held-out goals it meets alone, and an edge joins it to an earlier rule that differs by one weight or feature, solid, or by one added term, dashed, as read from the rule text. Eleven of the 16 rules admitted after generation 0 are one such edit from an earlier rule, and the best rule, which meets 211 goals alone, is two edits from generation 0.

Preregistered tests. The earlier program preregistered 47 tests with numeric bars, which received 56 verdicts, 14 pass, 23 fail, 17 inconclusive and 2 incomplete (Table 24). None of its confirmatory tests on the 352 held-out inference goals passed, which is why the final method changed its reward and admission.

Corrections of record. Audits during the study found defects, and the supplementary material lists each with its effect. Two bear on this paper. The goal-reward loop's writer start drew 357 of its 694 targets from the rule archive and its training thresholds came partly from archive rules, so the final recipe drops the archive from both (Appendices C and D). The OR-Library near-miss check accepted a rule within 2 percent of the best hand-written value even when it broke another threshold. The corrected check changes no pass of any real-reward run and removes one pass at every budget from one shuffled-reward control, and the final recipe, its evaluation and both replications use it, while the earlier selectors other than the replications' trained with the original check.

Table 23: The rule of each of the goal-reward loop's ten confirmatory libraries that meets the most of the 352 held-out goals alone. Beyond archive counts those goals no archive rule meets, and better graded the goals no hand-written rule meets on which it beats the archive's best graded value by more than 2 percent. None is an archive rule or scores like one on every goal, and seven branch with `where`.

Run	Gen.	Rule	Goals met	Beyond archive	Better graded
Qwen3.5-4B libraries					
trained 0	0	`-0.025*wait - 0.1*tokens - 0.1*target - 4.0*tput`	207	0	4
trained 1	0	`where(deficit > 0, -0.5*tokens - 0.1*wait + 2*deficit, -0.05*wait - 2*tokens)`	212	0	3
trained 2	2	`where(deficit > 0, 2*deficit - 1*tokens, -0.05*target + 0.5*tput - 2*tokens) - 0.1*wait`	209	0	2
trained 3	1	`-0.025*wait - 0.1*tokens - 8.0*tput`	210	0	4
trained 4	1	`where(deficit > 0, 1*tokens - 0.5*target - 2*tokens, 0.5*target + 1*deficit - 2*tokens) - 0.1*wait`	210	0	2
frozen 0	1	`where(deficit > 0, 2*deficit - 0.5*tokens, 0.02*target + 0.1*qshare - 2*tokens) - 0.1*wait`	212	0	3
frozen 1	1	`-0.5*tokens - 0.05*wait - 0.05*qshare + 2*deficit`	200	0	0
Nemotron Nano 8B libraries					
trained 0	2	`where(deficit < 0, -2*tokens - 0.05*qshare - 0.1*wait, -1.0*tokens - 0.1*wait)`	211	0	2
trained 1	1	`where(deficit > 0, 1*target - 1*tokens, 0.02*target + 0.5*tput - 2*tokens) - 0.1*wait`	211	0	2
frozen 0	0	`where(deficit > 0, 0.02*agemax - 1*tokens, 0.02*target + 0.5*tput - 2*tokens) - 0.1*wait`	208	1	3

Table 24: Preregistered tests of the earlier program, one row per study, each fixed with a numeric bar before its runs. A verdict is inconclusive when none was recorded when the test fell due, and incomplete when a planned run did not finish.

Study	What its tests asked	Tests	Verdicts
Reference-library selector and first launch, inference	whether a trained selector over a fixed reference library beats the hand-written rules, and whether the first loop's libraries, writer share of admissions and survival grow by generation	7	2 pass, 2 fail, 5 inconclusive
Development versions of the selector and the writer, inference	whether revised selector training and writer rewards raise survival, the writer's share of admissions, library value and selector scores	8	4 fail, 6 inconclusive
Confirmatory runs and the label-fixing variant, 352 held-out goals	whether generation-2 libraries beat a search library and their own generation 0, whether trained writers supply more admissions than frozen ones, and whether selector training beats its supervised start	7	3 fail, 4 inconclusive
Selector over the 20 classical rules, OR-Library job shop	whether the job shop leaves room to choose among rules, whether selector training beats its supervised start and shuffled-reward controls, whether a replication confirms it, and whether written rules go beyond the classical ones	11	8 pass, 3 fail, 1 inconclusive, 2 incomplete
Graded-score writing and larger rule sets, OR-Library job shop	whether writers and selectors raise the graded score on goals no classical rule meets, and whether selector training still gains on 50- and 60-rule sets	8	3 pass, 6 fail
Writers trained outside the loop, inference	whether writers trained to answer a goal directly with a rule beat a lookup of known rules, with two related tests of selector targets and history features	6	1 pass, 5 fail, 1 inconclusive
Total		47	14 pass, 23 fail, 17 inconclusive, 2 incomplete