Forge: Quality-Aware Reinforcement Learning for NP-Hard Optimization in LLMs

Anonymous ACL submission

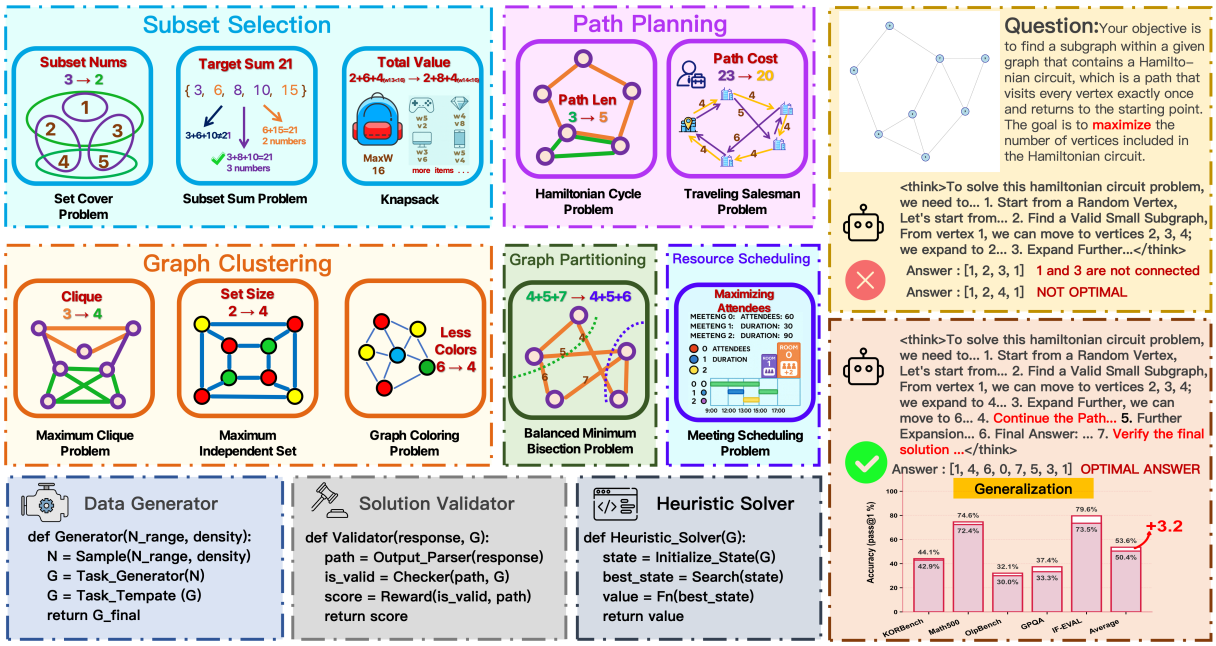

Figure 1: Overview of the FORGE-ENGINE. The FORGE-BENCH encompasses 10 NP-hard optimization tasks across five categories (e.g., subset selection, path planning), designed to assess reasoning capabilities. An automated pipeline consisting of a Data Generator, Solution Validator, and Heuristic Solver ensures controllable data synthesis, rigorous evaluation, and scalable training. A case study on the Hamiltonian Circuit problem shows the model's capacity to find optimal solutions. Furthermore, evaluations on OOD benchmarks demonstrate that our training enhances general reasoning capabilities.

Abstract

Large Language Models (LLMs) have achieved remarkable success on reasoning benchmarks through Reinforcement Learning with Verifiable Rewards (RLVR), excelling at tasks such as math, coding, logic and puzzles. However, existing benchmarks evaluate only correctness, overlooking optimality—the ability to find the best solutions under constraints. We propose FORGE-ENGINE, the first comprehensive framework for training and evaluating LLMs on NP-hard optimization problems through quality-aware RLVR. FORGE-ENGINE provides three key components: a scalable training infrastructure with instance generators, quality verifiers, and optimal baselines across 10 tasks; a rigorous benchmark with 1,000 instances evaluating both feasibility (Success Rate) and quality (Quality Ratio); and quality-aware rewards enabling continuous improvement beyond binary correctness. Training on Qwen2.5-7B-Instruct-1M with 15K examples achieves 93.1% SR and 46.6% QR, significantly outperforming GPT-4o (29.6% SR, 14.6% QR). Beyond optimization, training on FORGE-ENGINE transfers to diverse tasks: mathematics (+2.2%), logic (+1.2%), knowledge (+4.1%), and instruction-following (+6.1%). Our analysis reveals quality-aware rewards improve solutions by 28.8% over binary rewards, and task diversity drives generalization more than data quantity—offering insights into RLVR scaling for complex reasoning.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities on reasoning benchmarks, with recent models achieving near-human performance on mathematics (He et al., 2025), coding (Liu and Zhang, 2025), logic (Xie et al., 2025), and puzzle solving (Chen et al., 2025; Ma et al., 2024). These advances have been driven largely by Reinforcement Learning with Verifiable Rewards

(RLVR) (Xu et al., 2025b; Albalak et al., 2025), which leverages objective, automatically verifiable outcomes to guide model optimization (OpenAI, 2024; Guo et al., 2025; Google, 2025; Anthropic, 2025). However, existing reasoning benchmarks focus exclusively on *correctness*—whether answers are right or wrong, true or false, pass or fail. This binary evaluation overlooks a critical dimension: **optimality**—the ability to find not just valid solutions, but solutions that maximize or minimize an objective function.

kIn many practical scenarios, finding a *feasible* solution is merely the baseline; the true intelligence lies in finding the *optimal* one. Consider a logistics agent planning a route for 20 cities: while there are 2.4×10^{18} valid paths, a route that is merely "valid" but $3\times$ longer than necessary is functionally useless. This distinction between *correctness* (feasibility) and *optimality* (efficiency) represents a significant gap in current LLM capabilities. Even state-of-the-art models like GPT-4o often settle for the first valid solution they find, struggling to engage in the iterative optimization search required to refine a solution towards optimality.

To bridge this gap, we identify **NP-hard combinatorial optimization problems** as the ideal testbed for advancing LLM reasoning. These problems offer a unique trifecta of properties perfectly suited for RLVR training. **First**, they are computationally intractable, forcing the model to develop strategic heuristics rather than relying on rote memorization or brute force. **Second**, unlike standard logic tasks with sparse binary rewards (0 or 1), NP-hard problems possess an *objective quality score* (e.g., total tour length, subset value). This allows us to construct **dense, continuous reward signals**—rewarding a solution that is 90% optimal more than one that is 50% optimal—thereby guiding the model through a smoother optimization landscape. **Third**, verification is efficient (polynomial time), enabling scalable, automated supervision without human labeling.

Despite this potential, utilizing NP-hard problems for RLVR remains unexplored. Existing benchmarks like NPHardEval (Fan et al., 2024) and NPPC (Yang et al., 2025b) focus primarily on feasibility (e.g., "Is there a solution?") or rely on coarse evaluation metrics. Crucially, they lack a mechanism to generate the *ground-truth optimal values* needed to compute fine-grained rewards. Without the optimal solution, an RL system cannot judge the quality of the model's output, reverting the training process back to inefficient binary feedback.

To address this, we propose FORGE-RLVR, the first comprehensive framework designed to empower optimization reasoning in LLMs via RLVR. FORGE-RLVR encompasses 10 distinct tasks across five domains (e.g., Scheduling, Routing, Clustering). The core innovation lies in our **Generator-Verifier-Solver pipeline**: (i) **Controllable Generators** produce infinite training instances with graded difficulty (Easy/Medium/Hard) to facilitate curriculum learning; (ii) **Rule-based Verifiers** ensure strict constraint satisfaction; and most importantly, (iii) **Heuristic Solvers** rapidly compute near-optimal baselines for every generated instance. This pipeline transforms NP-hard problems from simple evaluation tasks into a rich training environment where models receive precise feedback on their *optimality gap*.

We further introduce FORGE-BENCH, a rigorous evaluation suite of 1,000 high-complexity instances. Using this framework, we train FORGE (based on Qwen2.5-7B) using multi-stage RLVR. The results are striking: our 7B model achieves a 93.1% Success Rate and 46.6% Quality Ratio, significantly outperforming GPT-4o (62.1%SR, 36.2% QR) on in-domain combinatorial optimization tasks.

Beyond optimization tasks, we investigate whether optimization training transfers to general reasoning. Specifically, we ask: *Does learning to solve the Traveling Salesman Problem improve a model's ability on general math or logic tasks?* The answer is affirmative. FORGE shows consistent gains across diverse reasoning benchmarks, including Math (+2.2%), Logic (+1.2%), Knowledge (+4.1%), and Instruction Following (+6.1%). These results suggest that optimization training fosters a transferable **"optimization mindset"**—the ability to generate candidates, and self-refine—benefiting broader reasoning capabilities.

Our key contributions are:

- **FORGE-ENGINE Framework:** We establish an end-to-end infrastructure for optimization reasoning, featuring the novel integration of heuristic solvers to provide fine-grained optimality rewards. This enables effective RLVR training that is superior to standard SFT.
- **FORGE-BENCH Benchmark:** We propose a dual-metric benchmark (Success Rate & Quality Ratio) that rigorously evaluates the depth of optimization reasoning, moving beyond the binary correctness paradigm of prior works.

- **OOD Generalization:** We provide empirical evidence that optimization training transfers to general reasoning capabilities. Our analysis reveals that *task diversity* and *quality-aware rewards* are the critical factors driving this generalization, offering new insights into the scaling behavior of RLVR-based training.

2 FORGE-ENGINE: Data Construction

FORGE-ENGINE provides comprehensive infrastructure for training and evaluating LLMs on NP-hard optimization. The framework consists of: (i) scalable training infrastructure with generators, verifiers, and optimal baselines across 10 tasks, and (ii) a rigorous benchmark with 1,000 high-complexity instances. We describe the task categories, data construction pipeline, and evaluation protocol below.

2.1 Task Categories and Problem Domains

FORGE-ENGINE covers 10 NP-hard optimization tasks spanning five fundamental categories (Figure 1). Each category represents a distinct class of combinatorial optimization problems with different constraint structures and reasoning requirements.

Graph Clustering. These tasks require identifying vertex subsets that satisfy strict adjacency constraints while optimizing structural objectives. We include three canonical problems: `Maximum Clique` (finding the largest fully-connected subgraph), `Maximum Independent Set` (finding the largest set of non-adjacent vertices), and `Graph Coloring` (minimizing the number of colors needed such that no adjacent vertices share a color). These tasks test graph structural understanding, conflict resolution, and chromatic reasoning.

Resource Scheduling. These tasks involve assigning activities to limited resources while avoiding conflicts and maximizing a global objective. The `Meeting Scheduling` problem requires allocating meetings to rooms and time slots while respecting attendee availability, room capacity, and temporal constraints. This task tests multi-constraint optimization and resource allocation strategies.

Graph Partitioning. These tasks require dividing graphs into balanced subsets while minimizing edge cuts between partitions. `Balanced Minimum Bisection` partitions a graph into two nearly equal-sized sets while minimizing the total weight of edges crossing the partition. This task tests the ability to balance competing objectives: partition balance versus cut minimization.

Subset Selection. These tasks involve choosing subsets of items under combinatorial constraints to optimize coverage or value-weight trade-offs. We include three representative problems: `Subset Sum` (selecting elements that sum to a target value while maximizing subset size), `Set Cover` (covering all elements with minimum number of sets), and `Knapsack` (maximizing total value subject to weight capacity). These tasks test discrete optimization and constraint satisfaction.

Path Planning. These tasks require finding optimal tours or cycles through all nodes in a graph. `Traveling Salesman Problem` (TSP) seeks the shortest tour visiting each city exactly once, while `Hamiltonian Cycle` finds the longest cycle visiting all vertices. These tasks test global optimization, permutation reasoning, and spatial planning.

2.2 Data Construction Pipeline

We construct FORGE-ENGINE through a systematic four-stage pipeline ensuring scalability, verifiability, and controllable difficulty.

Stage I: Task Selection. We select 10 NP-hard tasks spanning diverse constraint types with well-defined objectives and real-world relevance. Each task is formalized with clear input/output specifications and optimization objectives (e.g., TSP: given n cities with pairwise distances, find the shortest tour visiting each city exactly once).

Stage II: Generators and Verifiers. For each task, we develop: (i) *Programmable generators* that produce diverse instances with controllable parameters (e.g., complexity levels, graph density). To ensure **solvability**, we employ a **construction-by-design** approach, where each instance is synthesized from a known valid solution state to guarantee the existence of at least one feasible solution. (ii) *Rule-based verifiers* that automatically validate constraint satisfaction and compute objective values. These verifiers are cross-referenced with ground-truth solutions and have been manually audited to ensure absolute correctness.

Stage III: Heuristic Solvers. We implement task-specific heuristic algorithms that generate high-quality approximate solutions, providing baselines for reward computation during training. Heuristics are designed to be efficient (polynomial time) and effective (near-optimal quality). For example, our TSP heuristic uses multi-start nearest-neighbor initialization followed by 2-opt local search until convergence or timeout.

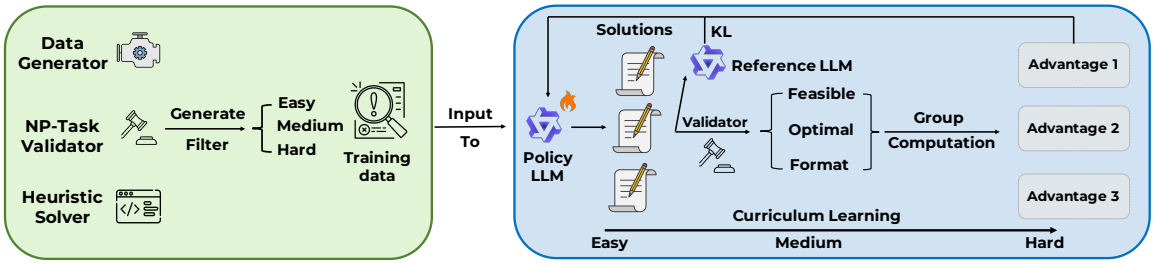

Figure 2: FORGE-RLVR training pipeline with quality-aware RLVR. The model generates solutions with step-by-step reasoning, which are evaluated through three components: (i) format verification checking output structure, (ii) feasibility verification ensuring constraint satisfaction, and (iii) quality assessment measuring optimality relative to heuristic baselines. The combined reward signal guides model optimization. Training progresses through curriculum learning across three stages (Easy → Medium → Hard), enabling progressive skill development from constraint satisfaction to sophisticated optimization.

Stage IV: Difficulty Calibration. We define three difficulty levels—*Easy*, *Medium*, and *Hard*—by systematically varying problem size and complexity parameters. Difficulty levels are calibrated empirically: we generate candidate instances across parameter ranges, evaluate baseline model performance, and select parameter settings that produce target success rates (Easy: 70-90%, Medium: 40-70%, Hard: 10-40%). For example, TSP difficulty is controlled by the number of cities: Easy (10-20 cities), Medium (20-30 cities), Hard (35-45 cities).

2.3 FORGE-BENCH: Evaluation Benchmark

Building on FORGE-ENGINE, we introduce **FORGE-BENCH**, a rigorous benchmark for evaluating LLM optimization capabilities.

Benchmark Design. FORGE-BENCH contains 1,000 test instances: 100 instances per task, with instances drawn from the high-complexity range to challenge model capabilities. For example, TSP instances contain 45-55 cities, significantly larger than training instances. All test instances are generated using the same generators as training data but with different complexity control variables, ensuring they are unseen during training while maintaining consistent problem structure.

Evaluation Metrics. Unlike existing benchmarks that measure only correctness, FORGE-BENCH assesses *optimization capability* through two complementary metrics:

Success Rate (SR): The percentage of instances for which the model generates a valid solution satisfying all constraints. This measures the model's ability to understand problem constraints and produce feasible solutions.

Quality Ratio (QR): The quality of model solutions compared to heuristic baselines. For minimization problems (e.g., TSP), QR $= \frac{\text{Heuristic Solution}}{\text{Model Solution}}$; for maximization problems (e.g., Knapsack), QR $= \frac{\text{Model Solution}}{\text{Heuristic Solution}}$. Invalid solutions receive QR = 0. This metric measures solution quality: QR = 1.0 indicates matching the heuristic baseline, while QR < 1.0 indicates suboptimal solutions.

Together, SR and QR provide a comprehensive assessment: SR measures whether models can generate valid solutions, while QR measures how close these solutions are to optimal. This dual-metric approach distinguishes FORGE-BENCH from prior benchmarks that evaluate only feasibility.

Key Advantages. Table 1 compares FORGE-BENCH with prior benchmarks in the NP-hard and reasoning domains. FORGE-BENCH offers several advantages over existing benchmarks: (i) *Scalability*—generators produce unlimited test instances at any difficulty level; (ii) *Quality-awareness*—QR metric enables fine-grained assessment of solution optimality; (iii) *Automatic evaluation*—verifiers provide instant feedback without human annotation; (iv) *Contamination-free*—all instances are generated procedurally, guaranteeing they do not appear in any training corpus; and (v) *Comprehensive coverage*—10 tasks across 5 categories test diverse optimization capabilities.

3 FORGE-RLVR: Quality-Aware RL

Standard reasoning RLVR relies on binary feedback (correct/incorrect), which is insufficient for combinatorial optimization where the goal is to maximize solution quality, not just validity. To address this, we propose a **Quality-Aware RLVR** framework. This approach replaces sparse binary signals with fine-grained, continuous feedback derived from heuristic baselines and employs a difficulty-based curriculum to guide the model

Table 1: Comparison of FORGE-BENCH with prior reasoning and optimization benchmarks. FORGE-BENCH is the first benchmark that combines scalability (unlimited instance generation), quality-aware evaluation (measuring solution optimality, not just correctness), and RLVR-readiness (fine-grained rewards for training). Existing benchmarks either lack scalable generation, measure only correctness, or are not suitable for RLVR training.

Benchmark	Task Type	Tasks	Scalable	Verifier	Trainable	Quality
KOR-Bench (Ma et al., 2024)	Knowledge	125	✗	✓	✗	✗
NPR (Wu et al., 2025)	Knowledge	1	✗	✗	✗	✗
Logic-RL (Xie et al., 2025)	Logic	1	✓	✓	✓	✗
ZebraLogic (Lin et al., 2025)	Logic	1	✓	✓	✓	✗
SearchBench (Borazjanizadeh et al., 2024)	Puzzle	11	✓	✓	✗	✗
Enigmata (Chen et al., 2025)	Puzzle	36	✓	✓	✓	✗
NPHardEval (Fan et al., 2024)	NP	9	✗	✗	✗	✗
FORGE-BENCH	NP	10	✓	✓	✓	✓

from basic constraint satisfaction to high-level optimization strategies.

3.1 Quality-Aware Reward Function

As illustrated in Figure 2, our reward function evaluates outputs sequentially across three dimensions: format, feasibility, and optimality. This hierarchical design ensures the model first learns to structure its reasoning, then to satisfy constraints, and finally to refine its solutions toward optimality.

Format Reward (R_{format}). To ensure verifiable and interpretable reasoning, we require outputs to follow a strict structure: step-by-step reasoning within `<think>` tags followed by a clearly delimited answer.

$$R_{\text{format}} = \begin{cases} +1, & \text{if format is valid} \\ -1, & \text{if format is invalid} \end{cases}$$

Feasibility Reward ($R_{\text{feasibility}}$). This component enforces problem-specific hard constraints (e.g., unique city visits in TSP, capacity limits in Knapsack). Invalid solutions receive a penalty, while valid solutions inherit the optimality score:

$$R_{\text{feasibility}} = \begin{cases} R_{\text{optimal}}, & \text{if solution is feasible} \\ -1.5, & \text{if solution is infeasible} \end{cases}$$

Optimality Reward (R_{optimal}). For feasible solutions, we compute a continuous quality score by comparing the model's objective value (M_s) against a heuristic baseline (M_h). This provides the dense signal necessary for optimization:

$$R_{\text{optimal}} = \begin{cases} M_s/M_h, & \text{maximization tasks} \\ M_h/M_s, & \text{minimization tasks} \end{cases}$$

$R_{\text{optimal}} \in (0, 1]$, where a value closer to 1.0 indicates near-optimal performance. The final total reward is computed as the sum of the structural and feasibility components: $R = R_{\text{format}} + R_{\text{feasibility}}$.

3.2 Difficulty-Based Curriculum

Training directly on complex NP-hard instances leads to the "cold start" problem, where rewards are too sparse for effective learning. We mitigate this via a progressive curriculum strategy (Easy → Medium → Hard). In the early phase (Easy), the model focuses on mastering *feasibility*—learning to generate valid structures and satisfy basic constraints. As difficulty increases (Medium/Hard), the focus shifts to *optimality*, challenging the model to refine its strategies within larger search spaces. This structured progression ensures a stable learning trajectory, preventing the model from collapsing under the complexity of hard instances.

3.3 Multi-Stage Curriculum Replay

Standard linear curriculum learning often suffers from catastrophic forgetting, where the model's proficiency on foundational tasks degrades as it adapts to higher complexities. To mitigate this, we propose **Multi-Stage Curriculum Replay**, a cyclic training strategy that iterates through the difficulty hierarchy (Easy → Medium → Hard) multiple times. By dividing the training process into distinct stages that each revisit the full difficulty spectrum using fresh instances, this approach continuously reinforces basic optimization reasoning while progressively stabilizing performance on harder problems, ensuring robust generalization across all difficulty levels.

4 Experiments

4.1 Experimental Setup

Evaluation Benchmarks. We evaluate on FORGE-BENCH (1,000 instances across 10 NP-hard tasks) and five OOD benchmarks: (1) **Logic**: KORBench; (2) **Mathematics**:

Math500 and OlympiadBench; (3) **Knowledge**: GPQA_Diamond; (4) **Instruction-Following**: IFEval. All experiments use OpenCompass (Contributors, 2023).

Training Configuration. We train FORGE on Qwen2.5-7B-Instruct-1M (Yang et al., 2025a) using curriculum replay with three stages (5K instances per stage, 15K total). We use Qwen2.5-7B-Instruct-1M rather than standard Qwen2.5-7B-Instruct as it provides better instruction adherence during RLVR training. We constructed a 10K SFT dataset via distillation from Qwen3-235B-Thinking, applying RFT to select solutions with a Quality Ratio > 0.5. Additionally, we utilized DAPO-17K (Yu et al., 2025) for math tasks. Full training details are listed in the Appendix A.2.

4.2 Main Results

As shown in Tables 2 and 3, FORGE achieves 93.1% SR and 46.6% QR on FORGE-BENCH, tripling the base model's performance and significantly outperforming GPT-4o (62.1% SR). Crucially, this optimization capability transfers to general reasoning: FORGE demonstrates consistent gains across logic, math, knowledge, and instruction-following, raising the overall OOD score from 50.4 to 53.6 (+3.2 points).

Performance by Task Category. Performance gains correlate strongly with task complexity. The largest improvements occur in Graph and Planning domains, where rigid structural constraints (e.g., Hamiltonian cycles) cause base models to fail almost entirely. RLVR closes this gap, achieving near-perfect performance (e.g., 98.2% SR in Planning). Selection tasks also show notable gains. These results highlight RLVR's effectiveness in strengthening LLM reasoning under high-dimensional, tightly constrained search spaces where base models often fail. For OOD generalization, we observe strong transfer to Knowledge Reasoning (+4.1%) and Instruction Following (+6.1%), suggesting that optimization training fosters broadly applicable reasoning skills.

Joint Training Synergy. Combining FORGE-ENGINE with mathematical reasoning data (DAPO-17K) yields further benefits. The joint model (**FORGE-7B-NP-MATH**) maintains high in-domain success (91.3% SR) while enhancing solution quality (51.3% QR). Moreover, it achieves the highest OOD average (54.2), surpassing both NP-only (53.6) and math-only (53.9) baselines. This indicates that optimization and mathematical reasoning are mutually reinforcing: optimization training likely sharpens constraint satisfaction useful for math, while mathematical training strengthens the logical rigor required for high-quality optimization.

Transfer Mechanisms. We attribute this cross-domain generalization to three cognitive capabilities fostered by optimization training: (1) *Constraint Reasoning*, which aids instruction following; (2) *Quality Evaluation*, which improves error detection; and (3) *Iterative Refinement*, which enhances self-correction. The pronounced gains in instruction-following (+6.1%) and knowledge reasoning (+4.1%) provide strong empirical support for this hypothesis.

RLVR vs. SFT. To isolate the contribution of RLVR, we compare FORGE against a SFT baseline (**Qwen2.5-7B-SFT**) trained on the same optimal solutions. While SFT improves in-domain performance (56.8% SR) over the base model, it lags significantly behind FORGE (93.1% SR). Crucially, SFT suffers from catastrophic out-of-domain degradation (dropping 12.0 points to an average score of 38.4), whereas FORGE improves generalization (53.6). We attribute this disparity to SFT's **lack of exploration**: by strictly mimicking optimal solutions, SFT overfits to specific solution patterns without mastering the underlying construction logic. In contrast, RLVR forces the model to explore diverse trajectories—including suboptimal and invalid attempts—enabling the acquisition of robust, transferable constraint satisfaction strategies rather than rote memorization.

4.3 Ablation Studies

4.3.1 Impact of Quality-Aware Rewards

Table 4 demonstrates the critical impact of reward design. Compared with binary rewards (Qwen2.5-7B (binary)-GRPO) and coarse rewards (Qwen2.5-7B (w/o GT)-GRPO), our fine-grained quality-aware rewards yield substantial improvements in both solution quality and success rate. Notably, reward schemes that focus solely on feasibility and assign a fixed reward of 1.0 tend to induce reward hacking in the early training stage, since finding a feasible solution is relatively easy for LLMs. These results confirm that explicit optimality feedback not only guides the model toward better solutions, but also strengthens its underlying mastery of feasibility constraints.

Table 2: Performance of reasoning LLMs, general LLMs, and our trained LLMs on FORGE-BENCH.

Model	Graph		Schedule		Partition		Selection		Planning		Overall	
	SR	QR	SR	QR	SR	QR	SR	QR	SR	QR	SR	QR
Proprietary LLMs												
DS-V3.1-Thinking	86.0	78.2	**99.0**	91.4	98.0	**77.1**	**99.3**	**98.5**	61.9	54.3	88.8	**79.9**
gpt-o3	**97.0**	**86.4**	**99.0**	94.5	**100.0**	51.4	87.4	87.3	74.0	**65.1**	91.5	76.9
Qwen3-235B-Thinking	66.7	62.9	95.0	93.0	**100.0**	55.8	98.0	97.1	52.0	44.8	82.3	70.7
Qwen3-235B-Instruct	47.0	37.9	86.0	80.0	100.0	50.4	78.7	74.4	15.5	8.3	65.4	50.2
gpt-4o-2024-08-06	64.7	29.3	79.0	59.8	**100.0**	53.0	14.7	9.3	52.0	29.6	62.1	36.2
Open-Source LLMs												
Qwen3-30B-A3B	57.3	50.6	97.0	**97.5**	99.0	51.2	86.0	84.0	44.5	36.9	76.8	64.1
Qwen3-32B	44.7	39.3	94.0	93.9	99.0	52.6	94.1	91.4	21.6	11.2	70.7	57.6
Qwen3-8B	22.7	16.8	78.0	75.3	98.0	51.0	86.0	82.6	3.0	1.2	57.5	45.4
DS-R1-Qwen-32B	23.3	18.1	49.0	45.1	96.0	48.6	85.7	79.7	15.4	7.9	53.9	39.9
DS-R1-Qwen-14B	18.0	13.4	52.0	51.7	32.0	16.2	67.3	63.4	4.5	1.4	34.8	29.2
DS-R1-Qwen-7B	6.3	1.9	1.0	0.9	2.0	1.0	13.7	8.9	0.5	0.1	4.7	2.5
Qwen2.5-72B	34.7	15.2	59.0	58.5	90.0	39.5	27.0	17.4	6.5	2.1	43.4	26.5
Qwen2.5-32B	35.3	15.2	15.0	12.8	**100.0**	51.7	32.0	22.4	23.5	6.7	41.2	21.8
Qwen2.5-14B	30.0	11.5	21.0	15.8	89.0	44.3	23.3	12.7	17.5	4.9	36.2	17.8
Qwen2.5-3B	7.7	2.9	17.0	5.0	6.0	2.2	23.0	10.7	15.5	3.7	13.8	4.9
InternLM3-8b	15.0	3.6	20.0	9.5	86.0	43.3	41.3	23.7	16.0	4.1	35.7	16.8
LLama3.1-8B	23.0	8.0	9.0	7.8	0.0	0.0	28.7	11.4	15.0	1.8	15.1	5.8
Qwen2.5-7B	11.0	3.1	40.0	19.8	67.0	34.0	26.7	15.2	3.5	1.0	29.6	14.6
FORGE-7B-NP (Ours)	89.7	27.8	85.0	43.5	99.0	53.8	93.7	79.1	98.2	28.9	**93.1**	46.6
	+78.7	+24.7	+45.0	+23.7	+32.0	+19.8	+67.0	+63.9	+94.7	+27.9	+63.5	+32.0
QWEN2.5-7B-SFT(Ours)	35.0	29.3	63.0	57.2	96.0	49.7	68.0	62.0	22.0	16.8	56.8	43.0
	+24.0	+26.2	+23.0	+37.4	+29.0	+15.7	+41.3	+46.8	+18.5	+15.8	+27.2	+28.4
FORGE-7B-NP-MATH (Ours)	89.0	32.2	72.0	57.0	**100.0**	51.8	96.3	83.4	**99.0**	32.3	91.3	51.3
	+78.0	+29.1	+32.0	+37.2	+33.0	+17.8	+69.6	+68.2	+95.5	+31.3	+61.7	+36.7

Curriculum Design and Strategy. As shown in Table 5, we first analyze the impact of instance difficulty distribution by varying the ratio of Easy, Medium, and Hard examples. Results indicate that an *easy-heavy* configuration (5:4:1) yields the best in-domain performance (100% SR) and strongest out-of-distribution (OOD) transfer. This supports the hypothesis that mastering foundational skills on simpler tasks is essential for both advanced optimization and generalizable reasoning.

We evaluate *Curriculum Replay* against single-pass baselines, analyzing both overall performance (Figure 3) and training reward dynamics (Figure 4). Quantitatively, Replay breaks the performance ceiling of linear strategies, achieving 93.1% SR and a +15.1 boost in QR. The reward trajectories reveal the underlying mechanism: while the Linear strategy (purple) suffers from early saturation, Replay (green) exhibits higher volatility—a natural byproduct of cyclically revisiting difficulty levels. This dynamic re-calibration allows the policy to escape local optima, preventing stagnation and mitigating catastrophic forgetting to sustain a continuous upward trajectory.

5 Related Work

Reinforcement Learning with Verifiable Rewards (RLVR). As reinforcement learning (RL) becomes an increasingly important tool for enhancing the reasoning capabilities of LLMs, Reinforcement Learning with Verifiable Rewards (RLVR) has emerged as a compelling alternative to Reinforcement Learning with Human Feedback (RLHF). Unlike RLHF, which relies on pretrained reward models and subjective human annotations, RLVR utilizes objective, automatically verifiable outcomes to provide reliable supervision (Seed et al., 2025; Guo et al., 2025; Team et al., 2025). Recent models exemplify this paradigm shift: DeepSeek-R1 (Guo et al., 2025) improves long-chain reasoning and self-verification through RLVR, while Kimi K1.5 (Team et al., 2025) achieves strong performance with long-context training and streamlined policy optimization—without depending on complex value models. The ecosystem supporting RLVR is rapidly maturing. High-quality math corpora with verifiable solutions (He et al., 2025; Albalak et al., 2025), structured coding corpora with graded difficulty and reward pipelines (Liu and Zhang, 2025; Xu et al., 2025b), and procedurally generated puzzle-style datasets with algorithmic verification (Xie et al., 2025; Chen et al., 2025; Li et al., 2025) are now available. Notably, NP problems are inherently verifiable and offer controllable difficulty settings (Fan et al., 2024; Yang et al., 2025b), making them well-

Table 3: Performance on out-of-domain benchmarks, including both reasoning and non-reasoning tasks, demonstrates that RLVR training on FORGE-ENGINE generalizes effectively.

Model	Logic	Math		Knowledge	Instruction	Average
	KORBench	Math500	OlpBench	GPQA_diamond	IFEval	
Qwen2.5-72B	53.0	**84.2**	**49.1**	**46.0**	**84.3**	**63.3**
Qwen2.5-32B	**56.2**	82.8	48.8	43.9	79.5	62.3
Qwen2.5-14B	50.6	80.0	45.1	41.9	81.6	59.8
Qwen2.5-3B	36.6	67.2	29.7	30.3	59.1	44.6
InternLM3-8B	40.7	78.2	25.1	36.9	72.5	50.7
LLama3.1-8b	44.5	48.6	17.7	22.7	69.3	40.6
Qwen2.5-7B	42.9	72.4	30.0	33.3	73.5	50.4
FORGE-7B-NP	44.1 (+1.2)	74.6 (+2.2)	32.1 (+2.1)	37.4 (+4.1)	79.6 (+6.1)	53.6 (+3.2)
FORGE-7B-MATH	45.9 (+3.0)	77.4 (+5.0)	31.8 (+1.8)	33.8 (+0.5)	80.5 (+7.0)	53.9 (+3.5)
Qwen2.5-7B-SFT	30.1 (-12.8)	66.2 (-6.2)	21.4 (-8.6)	18.7 (-14.6)	55.8 (-17.7)	38.4 (-12.0)
FORGE-7B-NP-MATH	46.6 (+3.7)	75.0 (+2.6)	31.2 (+1.2)	38.9 (+5.6)	79.3 (+5.8)	54.2 (+3.8)

Table 4: Comparison of curriculum learning performance across different RL algorithms from FORGE-ENGINE during RLVR training. Additionally, the we compares the impact of fine-grained versus coarse reward designs.

Data Proportion	CL	In Domain		Out of Domain					
		SR	QR	KB	Math500	OB	GPQA	IF	Avg
Qwen2.5-7B (Base)	✗	29.6	14.6	42.9	72.4	30.0	33.3	73.5	50.4
Qwen2.5-7B (binary)-GRPO	✓	86.6	17.8	43.4	73.0	30.5	30.3	78.3	51.1
Qwen2.5-7B (w/o GT)-GRPO	✓	76.2	25.6	43.8	73.2	30.5	30.8	78.3	51.3
FORGE-7B-GPG (Chu et al., 2025)	✗	85.8	32.6	43.8	74.2	30.6	30.8	78.0	51.5
	✓	88.6	32.7	**44.9**	74.2	30.9	**38.9**	78.6	53.5
FORGE-7B-GSPO (Zheng et al., 2025)	✗	78.2	36.6	40.8	**75.6**	31.8	36.4	78.2	52.6
	✓	83.4	42.1	44.4	75.0	31.1	36.9	77.2	52.9
FORGE-7B-GRPO (Guo et al., 2025)	✗	80.8	37.4	43.8	75.2	31.7	34.3	77.9	52.6
	✓	**93.1**	**46.6**	44.1	74.6	**32.1**	37.4	**79.6**	**53.6**

suited for RLVR-based training. However, most prior RLVR efforts have focused on math, coding, logic, or puzzles, leaving the broader class of NP-hard problems underexplored.

Optimization Reasoning with LLMs. Various benchmarks have been proposed to evaluate LLMs' reasoning capabilities across different domains, including mathematical (Glazer et al., 2024), logical (Xie et al., 2025), puzzle (Chen et al., 2025), and programming reasoning (Xu et al., 2025a). These tasks typically involve binary answer validation (e.g., True or False), which primarily assess deductive or symbolic reasoning. In contrast, optimization reasoning presents a fundamentally different challenge: it requires models to generate not only feasible solutions but also solutions that are as optimal as possible. Previous work on NP tasks has faced challenges in trainability (Fan et al., 2024; Yang et al., 2025b). Despite its significance, optimization reasoning has been underexplored in RLVR-based LLM training. Our work addresses this gap by focusing on NP-class problems.

We propose FORGE-ENGINE, a unified framework for data generation, optimal solution annotation, RLVR training, and evaluation, which empowers LLMs with optimization reasoning capabilities.

6 Conclusion

We presented FORGE-ENGINE, a comprehensive framework that leverages scalable generation and heuristic-guided rewards to train LLMs on NP-hard optimization problems. Through this infrastructure and the accompanying FORGE-BENCH benchmark, we developed FORGE, which significantly outperforms GPT-4o in optimization reasoning. Our experiments validate the effectiveness of quality-aware reward design and multi-stage curriculum learning. Furthermore, we reveal a critical insight: task diversity in RLVR training is a key driver of out-of-domain generalization. We hope this work lays a foundation for future research on integrating LLMs with optimization-based reasoning, offering new insights to the community and advancing the frontier of LLM reasoning capabilities.

Limitations

Due to computational resource constraints, we have conducted experiments using the Qwen2.5-7B-Instruct-1M model. Larger models, such as those with 14B or 32B parameters, have not been trained, and the performance of these more powerful models, starting from a bigger model size, remains unexplored. Additionally, designing individual NP tasks for RLVR training requires meticulous attention to various aspects, including problem definition, validation script development, heuristic algorithm design, and difficulty level calibration. As a result, we have currently designed 10 tasks. Scaling to larger models and incorporating additional tasks remain avenues for future exploration.

References

Alon Albalak, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, and 1 others. 2025. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models. *arXiv preprint arXiv:2502.17387*.

Anthropic. 2025. Claude 3.7 sonnet and claude code.

Nasim Borazjanizadeh, Roei Herzig, Trevor Darrell, Rogerio Feris, and Leonid Karlinsky. 2024. Navigating the labyrinth: Evaluating and enhancing llms' ability to reason about search problems. *arXiv preprint arXiv:2406.12172*.

Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Qiying Yu, Xuefeng Li, Jiaze Chen, Hao Zhou, and Mingxuan Wang. 2025. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. *Preprint*, arXiv:2505.19914.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. 2025. Gpg: A simple and strong reinforcement learning baseline for model reasoning. *Preprint*, arXiv:2504.02546.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass.

Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. 2024. Nphardeval: Dynamic benchmark on reasoning ability of large language models via complexity classes. In *62nd Annual Meeting of the Association for Computational Linguistics, ACL 2024*, pages 4092–4114. Association for Computational Linguistics (ACL).

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, and 1 others. 2024. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*.

Google. 2025. Gemini 2.5: Our most intelligent ai model.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, and 1 others. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*.

Peiji Li, Jiasheng Ye, Yongkang Chen, Yichuan Ma, Zijie Yu, Kedi Chen, Ganqu Cui, Haozhan Li, Jiacheng Chen, Chengqi Lyu, Wenwei Zhang, Linyang Li, Qipeng Guo, Dahua Lin, Bowen Zhou, and Kai Chen. 2025. Internbootcamp technical report: Boosting llm reasoning with verifiable task scaling. *Preprint*, arXiv:2508.08636.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*.

Jiawei Liu and Lingming Zhang. 2025. Code-r1: Reproducing r1 for code with reliable rewards. *arXiv preprint arXiv:2503.18470*.

Kaijing Ma, Xinrun Du, Yunran Wang, Haoran Zhang, Zhoufutu Wen, Xingwei Qu, Jian Yang, Jiaheng Liu, Minghao Liu, Xiang Yue, and 1 others. 2024. Kor-bench: Benchmarking language models on knowledge-orthogonal reasoning tasks. *arXiv preprint arXiv:2410.06526*.

OpenAI. 2024. Learning to reason with llms.

ByteDance Seed, Yufeng Yuan, Yu Yue, Mingxuan Wang, Xiaochen Zuo, Jiaze Chen, Lin Yan, Wenyuan Xu, Chi Zhang, Xin Liu, and 1 others. 2025. Seed-thinking-v1. 5: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Zixuan Wu, Francesca Lucchetti, Aleksander Boruch-Gruszecki, Jingmiao Zhao, Carolyn Jane Anderson, Joydeep Biswas, Federico Cassano, Molly Q Feldman, and Arjun Guha. 2025. Phd knowledge not

required: A reasoning challenge for large language models. *arXiv preprint arXiv:2502.01584*.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. 2025. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*.

Shiyi Xu, Yiwen Hu, Yingqian Min, Zhipeng Chen, Wayne Xin Zhao, and Ji-Rong Wen. 2025a. Icpc-eval: Probing the frontiers of llm reasoning with competitive programming contests. *arXiv preprint arXiv:2506.04894*.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. 2025b. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv preprint arXiv:2503.02951*.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, and 9 others. 2025a. Qwen2.5-1m technical report. *Preprint*, arXiv:2501.15383.

Chang Yang, Ruiyu Wang, Junzhe Jiang, Qi Jiang, Qinggang Zhang, Yanchen Deng, Shuxin Li, Shuyue Hu, Bo Li, Florian T Pokorny, and 1 others. 2025b. Non-deterministic polynomial-time problem challenge: An ever-scaling reasoning benchmark for llms. *arXiv preprint arXiv:2504.11239*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

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. 2025. Group sequence policy optimization. *Preprint*, arXiv:2507.18071.

A Appendix

A.1 Use of Large Language Models

Large Language Models are used for grammar check and polishing in this paper.

A.2 Detailed Training Setting

A.2.1 RLVR Training Setting

The training experiment utilizes the verl framework, employing the GRPO algorithm for fine-tuning the Qwen2.5-7B-Instruct-1M model. Training is performed on 8 A800 GPUs with a batch size of 16 for both training and validation. The maximum prompt length is set to 20,000 tokens, and the response length is capped at 4,096 tokens. Key hyperparameters include a learning rate of 4×10^{-7}, with a mini-batch size of 16 and a micro-batch size of 16 for PPO updates. KL loss regularization, with a coefficient of 0.001 (KL_{coef}), is applied to stabilize training.

A.2.2 SFT Training Implementation

Data Curation. We derived our SFT dataset through knowledge distillation from the `Qwen3-235B-Thinking` teacher model. To ensure data quality, we applied Rejection Sampling Fine-Tuning (RFT), filtering generated trajectories based on their Quality Ratio (QR). Specifically, we selected solutions with $QR > 0.5$, indicating that the solution quality is at least 50% of the heuristic optimal baseline. This process yielded a dataset of 10,000 samples, evenly distributed with 1,000 instances across each of the 10 NP tasks.

Training Details. We employed the ms-Swift framework for full-parameter fine-tuning, leveraging DeepSpeed ZeRO-2 and `bfloat16` precision to optimize computational efficiency. The model was trained for 2 epochs with a learning rate of 1×10^{-5} and a warmup ratio of 0.1. To accommodate the extensive context required for NP-hard reasoning, we set the maximum sequence length to 20,480 tokens. For memory management, we utilized a per-device batch size of 2 combined with 16 gradient accumulation steps.

A.3 Experiment Results

We conduct comprehensive experiments to answer three key questions: (1) Can LLMs learn to generate near-optimal solutions through quality-aware RLVR? (2) Does optimization training transfer to general reasoning? (3) What training strategies are most effective? Our experiments demonstrate

that FORGE achieves state-of-the-art optimization performance (93.1% SR, 46.6% AR), transfers to diverse reasoning tasks (+3.2 points OOD), and reveals that task diversity and quality-aware rewards are critical for effective learning.

A.3.1 Curriculum Learning and Data Proportion

As shown in Table 5, we investigate the impact of different data proportions—easy (E), medium (M), and hard (H)—on RLVR training, along with the role of curriculum learning (CL) strategies. The experiments focus on the TSP problem to better summarize the rules. We compare several configurations, including the baseline model and a variant without FORGE-HEURISTIC to provide accurate ground-truth (GT) signals in the reward signal, as well as different data ratios (E:M:H). In terms of in-domain performance, all RLVR configurations show improvements. Even the `Qwen2.5-7B (w/o GT)` model achieves noticeable gains, with SR rising from 4.0 to 90.0 and AR from 1.9 to 25.6. The introduction of curriculum learning (CL) results in further gains across all data proportions. The best performance is achieved with the *E:M:H=5:4:1* configuration, which achieves an average AR of 29.0, outperforming other configurations. For out-of-domain tasks, particularly in the *Math500* and *GPQA* benchmarks, the `E:M:H=5:4:1` configuration demonstrates superior generalization with an OOD average of 52.9. The inclusion of curriculum learning stabilizes performance and enhances the model's ability to generalize across both reasoning-heavy and instruction-following tasks.

Overall, these experiments highlight the importance of data proportion in RLVR, particularly the need for a larger proportion of easy tasks to build a strong foundation before tackling more complex problems. Curriculum learning further enhances this process, improving both in-domain and out-of-domain generalization capabilities.

A.3.2 Task Diversity Drives Generalization

Table 3 and 6 reveals a striking finding: **task diversity drives generalization more than data quantity**. Training on 10 diverse tasks achieves 53.6 OOD score, outperforming training on 3 tasks (53.0 OOD score)—despite having 3.3× less data per task. This pattern holds across all OOD benchmarks, with largest gains on mathematics (+3.8% for 7 tasks vs. 3 tasks) and instruction-following (+6.1% for all tasks vs. baseline).

This finding has important implications for RLVR scaling: rather than collecting massive amounts of data for few tasks, practitioners should prioritize task diversity. The results suggest that exposure to varied problem structures and constraint types develops more robust and transferable reasoning capabilities than extensive practice on narrow task distributions.

A.3.3 Multi-Stage RL Recipe

As illustrated in Figure 3 and Figure 4, we analyze the efficacy of the **MultiStage-RL** strategy compared to a single-pass OneStage baseline. The results reveal that iterative training is decisive for mastering complex optimization constraints. In-domain, MultiStage-RL achieves a substantial performance leap, raising the overall Success Rate (SR) from 72.6% to 93.0%. The gains are most pronounced in structurally complex tasks—such as *Graph* (+27.7%) and *Selection* (+37.1%)—suggesting that a linear single epoch is insufficient for the model to internalize the intricate feasibility rules of these domains.

Regarding out-of-distribution (OOD) generalization, MultiStage-RL maintains and slightly improves performance (53.6 vs. 53.4 overall). Crucially, this demonstrates that the aggressive optimization of in-domain capabilities does not lead to overfitting or "alignment tax." Instead, the multi-stage adaptation fosters a robust reasoning policy that generalizes effectively, balancing specialized optimization skills with broad reasoning transfer.

A.4 NP-hard Tasks

A.4.1 Set Cover

The task is to solve the classical Set Cover Problem. Given a universal set U and a collection of subsets $S \subseteq 2^U$, the goal is to find the smallest possible sub-collection of S whose union equals U. In other words, we aim to select the minimum number of subsets such that every element in U is contained in at least one of the selected subsets. If no such selection exists, the answer should be "Impossible". The solution is represented as a list of subset indices corresponding to the chosen sub-collection.

For example, given $U = \{0, 1, 2, 3, 4, 5\}$ and

$$S = \{0 : \{0, 1, 2\},\ 1 : \{2, 3\},\ 2 : \{0, 4\},\ 3 : \{3, 4, 5\},\ 4 : \{1, 2, 5\}\},$$

a valid minimum cover is $[0, 3, 4]$, since the union of these subsets is equal to U.

Table 5: Comparison of different data proportions for easy (E), medium (M), and hard (H) from FORGE-ENGINE during RLVR, as well as curriculum learning (CL) strategies. Training on single TSP problem.

Data Proportion	CL	In Domain		Out of Domain					
		SR	AR	KB	Math500	OB	GPQA	IF	Avg
Qwen2.5-7B (baseline)	✗	4.0	1.9	42.9	72.4	30.0	33.3	73.5	50.4
E:M:H=1:4:5	✗	99.0	28.1	42.9	73.8	**31.1**	35.4	77.9	52.2
	✓	98.0	28.2	43.4	73.6	30.8	**37.9**	78.5	52.8
E:M:H=1:1:1	✗	97.0	26.9	42.3	74.2	29.6	32.3	78.8	51.5
	✓	98.0	27.1	**44.6**	**74.8**	29.4	32.8	**79.3**	52.2
E:M:H=5:4:1	✗	**100.0**	28.2	44.2	74.4	30.4	31.3	78.5	51.8
	✓	**100.0**	**29.0**	44.3	74.4	**31.1**	35.9	78.6	**52.9**

Task Number	Logic	Math		Knowledge	Instruction	Average
	KORBench	Math500	OlpBench	GPQA	IFEval	
Qwen2.5-7B	42.9	72.4	30.0	33.3	73.5	50.4
+3 Tasks	43.8 (+0.9)	74.8 (+2.4)	30.5 (+0.5)	37.4 (+4.1)	78.5 (+5.0)	53.0 (+2.6)
+5 Tasks	44.2 (+1.3)	73.2 (+0.8)	31.9 (+1.9)	37.4 (+4.1)	78.0 (+4.5)	52.9 (+2.5)
+7 Tasks	44.2 (+1.3)	76.2 (+3.8)	30.4 (+0.4)	35.9 (+2.6)	78.8 (+5.3)	53.1 (+2.7)
+ALL Tasks	44.1 (+1.2)	74.6 (+2.2)	32.1 (+2.1)	37.4 (+4.1)	79.6 (+6.1)	53.6 (+3.2)

Table 6: Performance on out-of-domain benchmarks, with increasing task scale from FORGE-ENGINE during RLVR training.

The difficulty of the problem instances is categorized based on the size of the universe $|U|$, the number of subsets $|S|$, and the relative subset size (controlled by the parameter subset_size_factor):

- **Easy**:
  - $|U| \in [10, 20]$, $|S| \in [5, 10]$, subset size factor $= 0.4$
  - Small universe and relatively large subsets, making coverage straightforward.
- **Medium**:
  - $|U| \in [20, 25]$, $|S| \in [10, 15]$, subset size factor $= 0.4$
  - Moderate universe size and subset count, requiring careful selection.
- **Hard**:
  - $|U| \in [25, 30]$, $|S| \in [15, 25]$, subset size factor $= 0.4$
  - Larger universe with more subsets, increasing combinatorial complexity.
- **Benchmark**:
  - $|U| \in [30, 40]$, $|S| \in [20, 30]$, subset size factor $= 0.4$
  - The most challenging setting, with the largest universes and dense subset collections.

A.4.2 Subset Sum

The Subset Sum Problem asks whether a subset of integers sums up to a given target value T. In this variation, the objective is not only to reach the target sum, but also to maximize the number of elements used in the subset.

Formally, given a set of integers $\{a_0, a_1, \ldots, a_{n-1}\}$ and a target T, the task is to find an index set $I \subseteq \{0, 1, \ldots, n-1\}$ such that

$$\sum_{i \in I} a_i = T,$$

and among all valid solutions, the chosen I maximizes $|I|$. If multiple such subsets exist, any of them is acceptable. The submission format requires returning the ordered list of indices, e.g., $[0, 1, 4]$.

For example, given $T = 10$ and

$$\text{numbers} = \{0:2,\ 1:3,\ 2:7,\ 3:8,\ 4:5\},$$

a valid solution is $[0, 1, 4]$, since $2+3+5 = 10$, and the subset uses three elements, which is maximal.

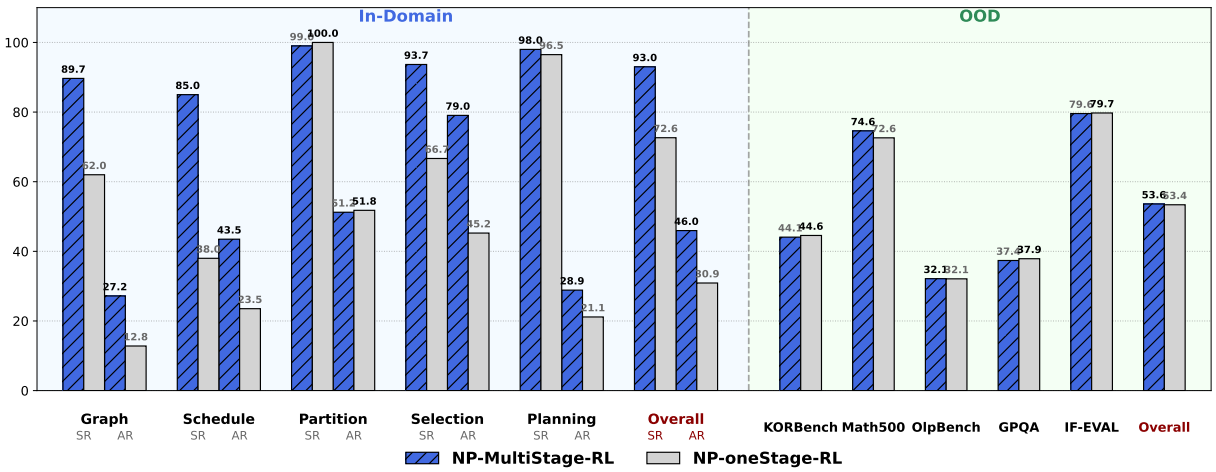

Figure 3: Comparison of RL training strategies during multi-task training, with performance evaluated on both in-domain and out-of-domain benchmarks.

The difficulty of generated problem instances is categorized according to the number of integers available (|numbers|), the typical size of the optimal solution ($|I|$), and the range of integer values:

- **Easy**:
  - Total numbers $\in [5, 10]$, solution size $\in [4, 8]$, values in $[1, 5]$.
  - Small input with low values, ensuring frequent feasible solutions.
- **Medium**:
  - Total numbers $\in [8, 12]$, solution size $\in [4, 8]$, values in $[1, 10]$.
  - Moderate instance size and range, requiring more careful subset selection.
- **Hard**:
  - Total numbers $\in [12, 15]$, solution size $\in [8, 12]$, values in $[1, 15]$.
  - Larger solution sizes and wider value ranges increase combinatorial difficulty.
- **Benchmark**:
  - Total numbers $\in [15, 20]$, solution size $\in [10, 15]$, values in $[1, 15]$.
  - The most challenging setting, with large search space and dense feasible solutions.

A.4.3 Knapsack

The Knapsack Problem requires selecting a subset of items to maximize the total value without exceeding a weight capacity. Formally, given a set of items $\{(w_i, v_i)\}_{i=0}^{n-1}$, each with weight w_i and value v_i, and a knapsack capacity W, the goal is to find an index set $I \subseteq \{0, 1, \ldots, n-1\}$ such that

$$\sum_{i \in I} w_i \leq W, \quad \text{and} \quad \sum_{i \in I} v_i$$

is maximized. The submission format requires returning the ordered list of chosen item IDs in square brackets, e.g., $[0, 2, 3]$.

For example, with $W = 20$ and

$$\begin{aligned} \text{items} = \{&0 : (3, 4),\ 1 : (4, 5), \\ &2 : (7, 10),\ 3 : (8, 11)\}, \end{aligned}$$

a valid optimal solution is $[0, 2, 3]$, achieving total weight $18 \leq 20$ and total value 25.

The problem instances are categorized into four difficulty levels, determined by the number of items, their weight/value ranges, and the relative knapsack capacity:

- **Easy**:
  - $6 \leq |I^*| \leq 10$ (solution items), total items $\approx$ 15–25.
  - Weights in $[5, 25]$, value-to-weight ratio in $[1.8, 2.5]$.
  - Capacity is 1.1–1.4 times the total weight of the solution items.
- **Medium**:
  - $8 \leq |I^*| \leq 12$, total items $\approx$ 25–35.
  - Weights in $[20, 80]$, value-to-weight ratio in $[1.5, 2.0]$.
  - Capacity is 1.05–1.25 times the total solution weight.

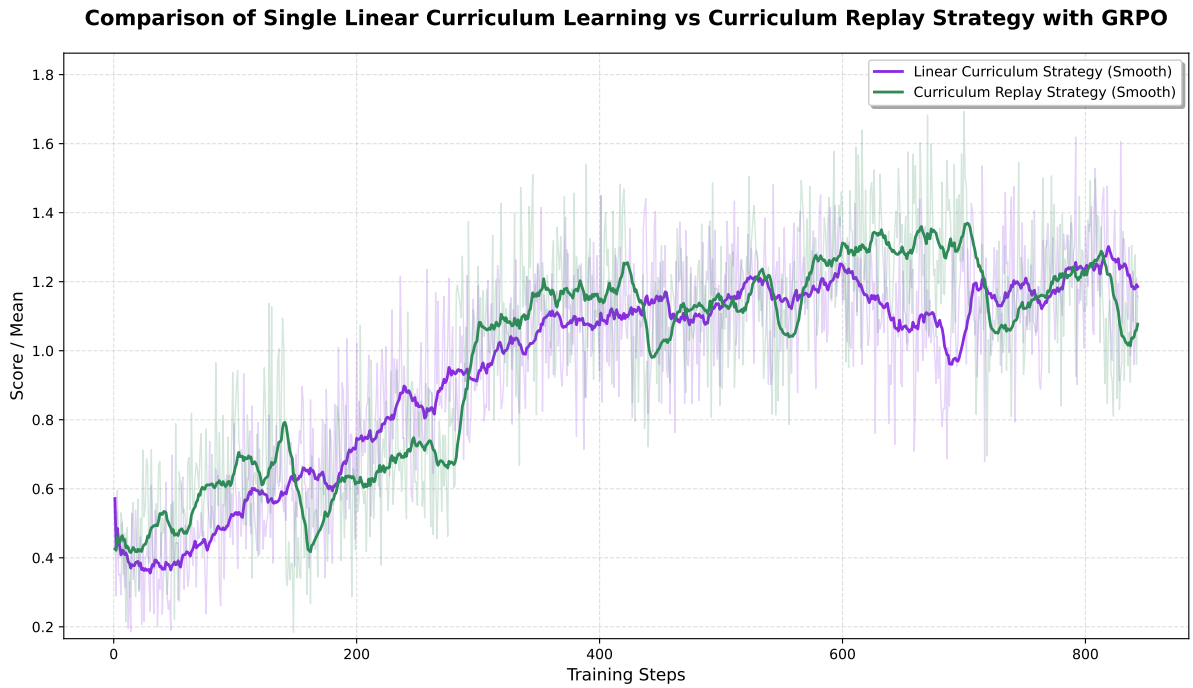

Figure 4: Comparison between single linear curriculum learning and curriculum replay strategy under GRPO training. The curriculum replay approach cyclically revisits samples across difficulty levels, whereas the linear strategy performs a single-pass curriculum in increasing order of difficulty.

- **Hard**:
  - $15 \leq |I^*| \leq 25$, total items $\approx$ 35–60.
  - Weights in $[50, 200]$, value-to-weight ratio in $[1.2, 1.6]$.
  - Capacity is 1.02–1.15 times the total solution weight.
- **Benchmark**:
  - $25 \leq |I^*| \leq 35$, total items $\approx$ 55–80.
  - Weights in $[50, 200]$, value-to-weight ratio in $[1.2, 1.6]$.
  - Capacity is 1.02–1.15 times the total solution weight.
  - The most challenging setting, with many items and tight capacity.

A.4.4 Balanced Minimum Bisection

The Balanced Minimum Bisection Problem requires partitioning a weighted undirected graph $G = (V, E)$ into two disjoint subsets of nearly equal size (differing by at most one vertex) such that the sum of the weights of edges crossing the cut is minimized. Unlike the classic Minimum Cut Problem, this task includes a balance constraint: both partitions must contain approximately the same number of vertices.

Formally, let V be divided into V_1 and V_2 such that $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$, and $\big||V_1| - |V_2|\big| \leq 1$. The objective is to minimize

$$\sum_{\substack{u \in V_1, v \in V_2 \\ (u,v) \in E}} w(u, v),$$

where $w(u, v)$ is the edge weight. The solution format specifies the two subsets explicitly, e.g., $[[0, 1, 2], [3, 4, 5]]$.

For example, consider the input graph:

$$0 : \{1 : 3,\ 2 : 1\}, \quad 1 : \{0 : 3,\ 2 : 2,\ 3 : 2\},$$
$$2 : \{0 : 1,\ 1 : 2,\ 3 : 3\}, \quad 3 : \{1 : 2,\ 2 : 3\}.$$

A valid optimal balanced bisection is $[[0, 1], [2, 3]]$.

The difficulty of generated instances is determined by the number of nodes, the structural complexity of the graph, and the noise level:

- **Easy**:
  - $|V| \approx 30$.
  - Graphs have clear community structures with dense intra-community edges and sparse inter-community connections, with small noise (≈ 0.1).
  - Balanced cuts are relatively easy to identify.

- **Medium**:
  - $|V| \approx 42$.
  - Graphs exhibit fuzzier community boundaries and more inter-community edges, with moderate noise (≈ 0.15).
  - Increases difficulty by reducing the clarity of the optimal partition.
- **Hard**:
  - $|V| \approx 45$.
  - Graphs are generated with deceptive structures, including "traitor" nodes and reinforced communities.
  - Noise level around 0.1, making near-optimal but incorrect cuts more likely.
- **Benchmark**:
  - $|V| \approx 50$.
  - Graphs include reinforced "hell mode" structures, traitor nodes, and low noise (≈ 0.02).
  - The most challenging setting, with multiple plausible partitions and high combinatorial complexity.

A.4.5 Meeting Scheduling Problem

The Meeting Scheduling Problem (MSP) aims to assign meetings to rooms and times in order to maximize total attendee participation, subject to availability and capacity constraints. Each meeting requires a set of attendees and a duration, each attendee has availability intervals, and each room has a capacity. A feasible solution must assign to each scheduled meeting a start time and a room such that:

- All required attendees are available for the entire duration.
- The room has sufficient capacity for all attendees.
- No attendee or room is scheduled for overlapping meetings.

If a meeting cannot be scheduled under these constraints, it is omitted. The solution is expressed as an ordered list of tuples $(\text{meeting_id}, \text{room_id}, \text{start_time})$, sorted by start time.

For example, given the input:

$$\begin{aligned}
\text{meetings} &= \{0 : ([0,1,2], 60),\ 1 : ([1,3], 30), \\
&\qquad 2 : ([0,2,3], 90)\}, \\
\text{availability} &= \{0 : [(900, 1700)], \\
&\qquad 1 : [(900, 1200), (1300, 1700)], \\
&\qquad 2 : [(900, 1700)],\ 3 : [(1000, 1400)]\}, \\
\text{rooms} &= \{0 : 5,\ 1 : 3\},
\end{aligned}$$

a valid schedule is

$$[(0, 0, 900),\ (1, 1, 1000),\ (2, 0, 1020)],$$

which yields a total of 8 attendee participations.

The difficulty of generated MSP instances depends on the number of meetings, attendees, rooms, and fragmentation of availability:

- **Easy**:
  - 4–5 meetings, 3–5 attendees, 3–4 rooms.
  - At most 3 attendees per meeting.
  - Availability mostly continuous within the working day.
- **Medium**:
  - 5–6 meetings, 4–6 attendees, 4–5 rooms.
  - At most 4 attendees per meeting.
  - Some attendees have fragmented availability (e.g., lunch breaks).
- **Hard**:
  - 6–7 meetings, 5–7 attendees, 5–6 rooms.
  - At most 4 attendees per meeting.
  - Heavier overlap among meetings and tighter room capacities.
- **Benchmark**:
  - 8–10 meetings, 7–9 attendees, 6–7 rooms.
  - At most 5 attendees per meeting.
  - The most challenging setting, with dense scheduling conflicts and fragmented availability.

A.4.6 Hamiltonian cycle

The task is to find a Hamiltonian circuit in a given graph G, which is a path that visits every vertex exactly once and returns to the starting point. The goal is to maximize the number of vertices included in the Hamiltonian circuit. The process starts with

a random vertex and finds a small valid subgraph, then iteratively expands the subgraph while ensuring it remains valid, continuing until the largest possible Hamiltonian circuit is found.

The problem is categorized into four difficulty levels based on the number of vertices and edge density:

- **Easy**:
  - $|V| \in [15, 20]$, $\rho = 0.2$
  - Small graph with sparse edges.
- **Medium**:
  - $|V| \in [20, 30]$, $\rho = 0.3$
  - Moderate graph size with moderate connectivity.
- **Hard**:
  - $|V| \in [30, 40]$, $\rho = 0.4$
  - Larger graph with denser edges, increasing difficulty.
- **Benchmark**:
  - $|V| \in [40, 50]$, $\rho = 0.5$
  - The most challenging, with the largest and densest graph.

A.4.7 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classical combinatorial optimization problem. Given a set of cities and pairwise distances, the objective is to find the shortest possible tour that:

- Starts and ends at the same city.
- Visits each city exactly once in between.

The solution is expressed as a route $[c_0, c_1, \ldots, c_{n-1}, c_0]$, where c_0 is the starting city and each city appears exactly once except for the repetition of c_0 at the end.

For example, given the distance dictionary:

$$\begin{aligned} &0 : \{1 : 10,\ 2 : 15,\ 3 : 20\}, \\ &1 : \{0 : 10,\ 2 : 35,\ 3 : 25\}, \\ &2 : \{0 : 15,\ 1 : 35,\ 3 : 30\}, \\ &3 : \{0 : 20,\ 1 : 25,\ 2 : 30\}, \end{aligned}$$

a valid optimal solution is

$$[0, 1, 3, 2, 0].$$

The difficulty of generated TSP instances is determined primarily by the number of cities:

- **Easy**: 10–20 cities.
- **Medium**: 20–30 cities.
- **Hard**: 35–45 cities.
- **Benchmark**: 45–55 cities.

All instances are generated with symmetric distance matrices, with distances sampled uniformly within a predefined range.

A.4.8 Maximum Clique Problem

The Maximum Clique Problem (MCP) is defined on an undirected graph $G = (V, E)$. A clique is a subset of vertices $C \subseteq V$ such that every pair of distinct vertices in C is connected by an edge in E. The problem asks for the largest such subset, i.e., a clique of maximum cardinality.

The solution is expressed as a list of vertex IDs forming the clique. For example, given the adjacency lists:

$$\begin{aligned} &0 : [1, 2, 3, 4], \quad 1 : [0, 3, 4], \quad 2 : [0, 3], \\ &3 : [0, 1, 2, 4], \quad 4 : [0, 1, 3], \end{aligned}$$

a valid maximum clique is

$$[0, 1, 3, 4],$$

which has size 4.

The difficulty of generated MCP instances depends on the graph size and density:

- **Easy**: 4–8 vertices, cliques of size 2–4.
- **Medium**: 8–12 vertices, cliques of size 2–4.
- **Hard**: 12–16 vertices, cliques of size 2–6.
- **Benchmark**: 16–20 vertices, cliques of size 4–8.

Graphs are generated by first constructing a guaranteed clique and embedding it into a larger graph with random edges, ensuring the clique exists as the maximum solution.

A.4.9 Maximum Independent Set

The Maximum Independent Set (MIS) problem is defined on an undirected graph $G = (V, E)$. An independent set is a subset of vertices $I \subseteq V$ such that no two vertices in I are adjacent in G. The problem asks for the independent set of maximum cardinality.

The solution is expressed as a list of vertex IDs forming the set. For example, given the adjacency lists

$$0 : \{1, 2\}, \quad 1 : \{0, 2, 3\}, \quad 2 : \{0, 1, 3\}, \quad 3 : \{1, 2\},$$

a maximum independent set is

$$[0, 3],$$

which has size 2.

The difficulty of generated MIS instances depends mainly on the graph size and the planted independent set:

- **Easy**: 12–20 vertices, independent set size 4–8.
- **Medium**: 20–30 vertices, independent set size 8–12.
- **Hard**: 30–40 vertices, independent set size 12–16.
- **Benchmark**: 40–50 vertices, independent set size 16–20.

Graphs are generated by first selecting a guaranteed independent set and embedding it into a larger graph with randomly added edges, ensuring the independent set exists as the maximum solution.

A.4.10 Graph Coloring Problem

The Graph Coloring Problem (GCP) is defined on an undirected graph $G = (V, E)$. The task is to assign a color to each vertex such that no two adjacent vertices share the same color, while minimizing the total number of colors used.

The solution is expressed as a list of integers, where the i-th entry denotes the color assigned to vertex i. For example, given the adjacency lists

$$0 : [1, 2], \quad 1 : [0, 3], \quad 2 : [0, 3], \quad 3 : [1, 2],$$

a valid optimal coloring is

$$[1, 2, 1, 2],$$

which uses 2 colors.

The difficulty of generated GCP instances depends mainly on the number of vertices, the number of colors required, and the edge density:

- **Easy**: 8–12 vertices, 3–4 colors, edge density ≈ 0.2.
- **Medium**: 15–22 vertices, 4–6 colors, edge density ≈ 0.35.
- **Hard**: 25–32 vertices, 6–8 colors, edge density ≈ 0.5.
- **Benchmark**: 32–40 vertices, 6–8 colors, edge density ≈ 0.5.

Graphs are generated by partitioning vertices into color classes and adding random edges between different partitions, ensuring that the planted coloring remains a valid optimal solution.