From Trainee to Trainer: LLM-Designed Training Environment for RL with Multi-Agent Reasoning

Chao Chen[1], **Chengzu Li**[†2], **Zhiwei Li**[1], **Yinhong Liu**[†2] **and Zhijiang Guo**[†1,3]
[1]LARK, HKUST (GZ), [2]University of Cambridge, [3]HKUST

[†]Corresponding Author

Github Page: https://github.com/LARK-AI-Lab/Trainee-to-Trainer
Model Weight: https://huggingface.co/LARK-Lab/Trainee2Trainer/
Website: https://lark-ai-lab.github.io/trainee-to-trainer.github.io

Abstract

Reinforcement learning pipelines for Large Language Model (LLM) training often rely on manually redesigned environments between stages, requiring practitioners to heuristically infer which configuration will best improve the current policy. To automate this process, we propose the LLM-as-Environment-Engineer framework in which the current policy model analyzes failure trajectories together with contextual information and proposes modifications to the next-stage training environment configuration. We also introduce MAPF-FrozenLake, a controllable testbed whose generator exposes multi-dimensional environment configurations, making it suitable for studying and benchmarking environment redesign. On this testbed, we condition the environment engineer on structured summaries of policy behavior, failure cases, and environment statistics, from which it produces the configuration for the next training stage. With Qwen3-4B as the backbone, our framework achieves the strongest aggregate performance on our benchmarks, outperforming larger proprietary LLMs (e.g., GPT, Gemini) and fixed-environment training baselines. We further analyze which forms of context are most effective, finding that successful environment updates rely on failure evidence and preserve configurations that already work. Interestingly, the current RL checkpoint serves as a better environment engineer than the original base model, suggesting that policy learning improves the model's ability to diagnose its remaining weaknesses.

1. Introduction

In Reinforcement Learning (RL) pipelines for training Large Language Models (LLMs; OpenAI 2026; Team 2026), the interaction environment determines which behaviors, failures, and exploration signals the policy will encounter during training. As a result, environment design strongly affects optimization efficiency, generalization, and the emergence of new capabilities (Zhang et al., 2025). However, in current practice, improving the training environment is still largely a manual process (Xie et al., 2025). Practitioners repeatedly inspect rollout logs and validation failures, form hypotheses about the model's current weaknesses, and *manually* redesign the next-stage training environment. This workflow requires substantial expert effort and becomes increasingly difficult as RL training pipelines scale in complexity, highlighting the need for an automatically adaptive framework.

Recent work has automated related aspects of training adaptation. Curriculum learning (Azad et al., 2022; Bae et al., 2026; Xie et al., 2025) adjusts training difficulty over time, while self-play and multi-agent training frameworks (Fang et al., 2025; Shi et al., 2025; Yuan et al., 2024) adapt the training signal through interacting agents or opponents. More broadly, recent LLM-based data generation

Corresponding author(s): Chengzu Li (cl917@cam.ac.uk), Yinhong Liu (yl535@cam.ac.uk), Zhijiang Guo (zhijiangguo@hkust-gz.edu.cn),

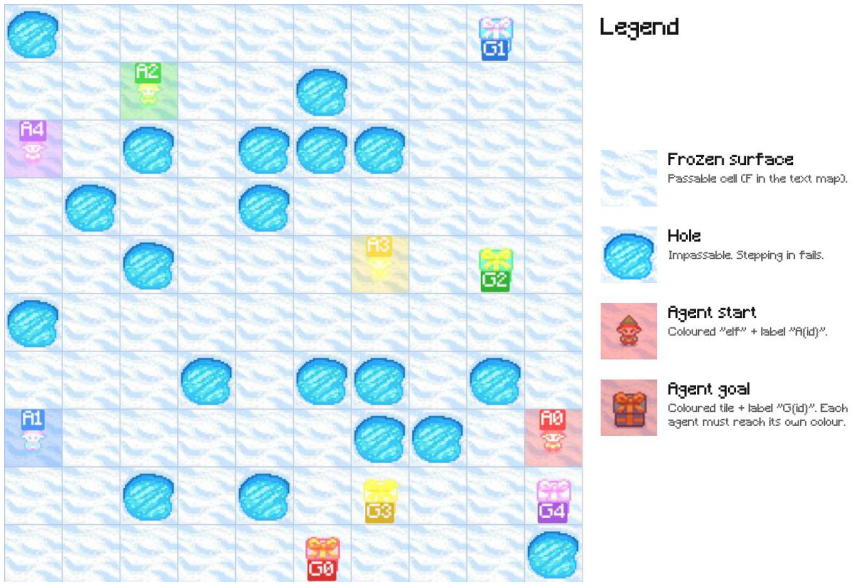

Figure 1 | A 5-agent MAPF-FrozenLake instance on a 10×10 grid. Agents A0/A1/A2/A3/A4 start at their colored cells and must reach goals G0/G1/G2/G3/G4 without colliding with each other or stepping into a hole (blue patches); a *wait* action is available for resolving conflicts.

methods automatically construct synthetic training examples for downstream optimization (Liang et al., 2025). However, these approaches mainly operate through training example selection, difficulty scheduling, or data synthesis within a fixed environment family. In contrast, we study a different problem: can the policy model itself proactively redesign the environment generator that defines its future RL training distribution?

To study this question, we propose a closed-loop framework for *policy-conditioned environment redesign*, which we refer to as an **LLM-as-Environment-Engineer**. After each RL stage, the current policy model receives structured summaries of its training behavior, validation failures, and environment statistics, and proposes a new environment configuration for the next training stage. The model does not directly synthesize or select individual training examples. Instead, it modifies the parameters of an environment generator, thereby reshaping the future distribution from which training instances will be sampled. The objective of the environment engineer is therefore not to solve the task itself, but to propose training distributions that maximize future policy improvement.

This setting introduces several challenges. Validation signals are often sparse and highly dependent on environment configuration, so aggregated reward can hide distinct failure modes across different regions of the training distribution. Naively maximizing difficulty is also insufficient: overly difficult environments may collapse the learning signal, while overly easy environments can lead to premature saturation and weak exploration. Effective environment redesign therefore requires evidence-driven adaptation rather than shallow heuristics or monotonic difficulty scaling.

Studying these dynamics in open-ended embodied (Shridhar et al., 2020) or web environments (Zhou et al., 2023) is difficult because the training distribution is only weakly controllable. Therefore, we design MAPF-FrozenLake, a controllable Multi-Agent Path Finding version of FrozenLake (Wu et al., 2025) with grid-based environments, as a testbed for environment redesign. Figure 1 shows a 5-agent example on a 10×10 grid. Our goal is not to maximize environmental realism, but to isolate and study whether an LLM can make structured decisions about future training distributions under controlled conditions. Each instance is generated from a parameterized configuration that controls properties such as grid-size distribution, conflict density, and obstacle density. The environment also provides

deterministic multi-dimensional evaluation signals, including path validity, optimality, and total trajectory length. These properties make MAPF-FrozenLake suitable for analyzing how environment redesign decisions affect downstream RL learning dynamics.

Combining the LLM-as-Environment-Engineer framework and MAPF-FrozenLake, we perform controlled investigations of environment redesign in RL-based LLM training. In particular, we study how different feedback signals influence redesign quality, whether RL training improves the model's ability to diagnose its own weaknesses, and which redesign behaviors are associated with successful downstream adaptation. Overall, our contributions are summarized as follows:

- **Framework.** We propose a closed-loop framework where the policy LLM iteratively redesigns its own environment generator, and introduce MAPF-FrozenLake, a controllable testbed for studying this process.
- **Empirical Findings.** A 4B Qwen3 model under our framework outperforms both carefully-designed curricula and much larger proprietary LLMs (GPT, Gemini) as environment designers.
- **Mechanistic Insights.** RL training markedly improves the model's self-diagnostic ability: successful redesign is driven by evidence-based, failure-mode-targeted adaptation rather than naive difficulty scaling.

2. Related Work

Curriculum learning improves training efficiency by controlling the difficulty or ordering of training experience, and has been widely applied across domains (Bengio et al., 2009; Graves et al., 2017). In reinforcement learning, many approaches assume a predefined set of tasks and learn or specify a curriculum over them (Huang et al., 2022; Li et al., 2023). When such task sets are unavailable, curricula are generated automatically by identifying tasks near the current policy's capability boundary using signals such as value estimates (Kim et al., 2023; Zhang et al., 2020) or episodic rewards (Florensa et al., 2018). Recent LLM-based work has adopted curriculum-style RL to improve reasoning and generalization (Bae et al., 2026; Zeng et al., 2025), while other methods manually progress from easier to harder tasks after fixed training stages (Team, 2025a; Xie et al., 2025). Overall, existing methods typically rely on auxiliary heuristics or fixed schedules to select training tasks, limiting their ability to adapt the next-stage environment directly to the model's observed failure patterns.

Self-improvement methods reduce reliance on external supervision by allowing models to generate their own training signals, such as tasks, responses, critiques, rewards, or opponents (Gao et al., 2025). Existing approaches mainly fall into two paradigms. Multi-model methods (Huang et al., 2025; Shi et al., 2025) train several models jointly so that one model provides challenges or feedback for another, while single-model methods (Chen et al., 2025; Liang et al., 2025; Yuan et al., 2024; Zhou et al., 2026) let a single model play multiple roles, including task proposal, solving, and evaluation. Across both paradigms, the goal is to create additional learning pressure without fully human-labeled data, typically within the task, response, or reward space. In contrast, our work focuses on the interaction environment itself: the model adapts the generator configuration that defines the situations encountered in the next RL stage.

3. Method

We study whether an LLM can iteratively redesign its own future training environment from detailed context. After each training round, the model reviews the current state, decides how the next training environment should change, and uses the resulting data for the next RL stage. The central question is what context the model needs in order to make these self-designed configurations useful.

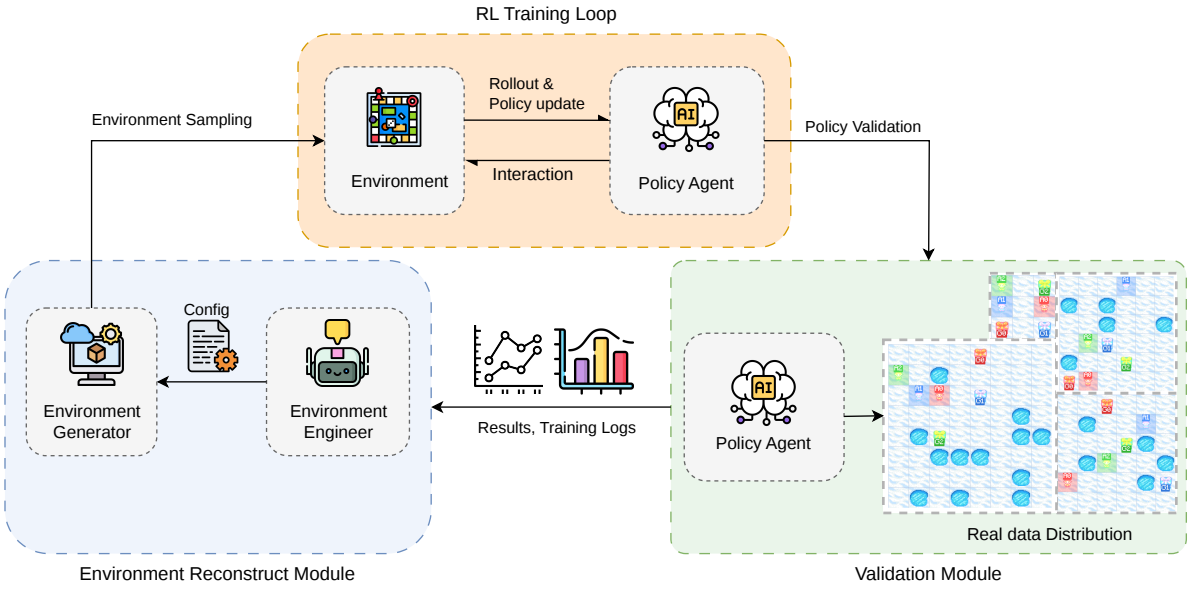

Figure 2 | Overview of environment-engineering framework. The model is trained, validated, and then used as an environment engineer to design the next-round training configuration.

3.1. Task Formulation

3.1.1. MAPF-FrozenLake Environment

MAPF-FrozenLake gives us a controlled but challenging setting for studying environment design. Each instance places multiple agents on a grid with holes; agents must reach their goals without collisions or falling into holes, and may use *wait* actions to resolve conflicts. The environment can generate new instances from a configuration, include map-size distribution, conflict ratio, and hole density. Evaluation is also multi-dimensional, covering validity and total steps.

We build a environment generator on top of the Conflict-Based Search algorithm. A generator configuration is denoted by C. For each map size m, C specifies a data ratio $C_{\text{data}}(m)$, a hole ratio $C_{\text{hole}}(m)$, and a wait ratio $C_{\text{wait}}(m)$; these values determine how many instances of each type are generated and how difficult those instances are. To simulate a realistic cold start, the initial training data is produced from a randomly sampled configuration. Let $\mathcal{S} = \{3\times3, \ldots, 10\times10\}$ denotes the set of map sizes. For each map size s, a *configuration*

$$C = \{\, (r_s,\, h_s,\, w_s) \,\}_{s\in\mathcal{S}}$$

specifies, three components: r_s, the *data ratio* – the share of training instances sampled at size s, with $\sum_s r_s = 1$; h_s, the *hole ratio* – the fraction of cells turned into holes in maps of size s; and w_s, the *wait ratio* – the fraction of generated instances at size s that require at least one wait action to resolve agent conflicts. At round R_n, the configuration is denoted C_n and the generator emits the corresponding training set $\mathcal{D}_n$.

3.1.2. RL Reward Design

The total reward combines an accuracy term and a length term with adaptive weights:

$$R = w_{\text{acc}} \cdot R_{\text{acc}} + w_{\text{len}} \cdot R_{\text{len}}. \tag{1}$$

Algorithm 1 Environment-engineering train loop.

Require: base learner M_0, generator G, eval set $\mathcal{V}$, initial config C_0, modules $\mathcal{M} \subseteq \{F, G, H, T\}$, training meta $\mathcal{T}$, rounds N

history $\mathcal{B} \leftarrow \emptyset$
for $n = 0$ **to** $N-1$ **do**
  $\mathcal{D}_n \leftarrow G(C_n)$ ▷ generate env
  $M_{n+1} \leftarrow \mathrm{GRPO}(M_n, \mathcal{D}_n; R)$
  $E_n \leftarrow \textsc{Validate}(M_{n+1}, \mathcal{V})$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(C_n, E_n)\}$
  $ctx_n \leftarrow \textsc{Compose}(\mathcal{M}; \mathcal{B}, \mathcal{T})$
  $\tilde{C}_{n+1} \leftarrow M_{n+1}(ctx_n)$
  $C_{n+1} \leftarrow \textsc{Project}(\tilde{C}_{n+1})$ ▷ enforce constraints
end for
return M_N, $\{C_n\}_{n=1}^{N}$

Accuracy reward. The response must pass eight strict validity checks; failing any one sets $R_{\text{acc}} = 0$. Details in Appendix A. If all eight pass, R_{acc} is shaped by the cost gap to the ground truth. Let $c = model\_cost - gt\_cost$ and let $c_{\max} = 2\,gt\_cost$ be the saturation threshold; we set

$$R_{\text{acc}}(c) = \begin{cases} 1.0, & c = 0, \\ 1.0 - 0.7\,\frac{c}{c_{\max}}, & c \in (0, c_{\max}], \\ 0.3, & c > c_{\max}, \end{cases} \tag{2}$$

which maps the cost-gap interval $[0, c_{\max}]$ linearly to the reward interval $[1.0, 0.3]$ and saturates at 0.3 beyond. When gt_cost cannot be parsed we fall back to $R_{\text{acc}} = 0.5$.

Length reward. We discourage verbose outputs. Let ℓ denote the response length in tokens, and let $L_1 = 1500$ and $L_2 = 4096$ be the soft and hard length thresholds; we set

$$R_{\text{len}}(\ell) = \begin{cases} 0, & \ell \le L_1, \\ -\frac{\ell - L_1}{L_2 - L_1}, & L_1 < \ell < L_2, \\ -1, & \ell \ge L_2, \end{cases} \tag{3}$$

R_{len} maps the length interval $[L_1, L_2]$ linearly to reward interval $[0, -1]$ and saturates at -1 beyond.

Adaptive weights. The two weights are scheduled to shift emphasis from brevity to correctness as the model learns to produce concise outputs. Let s denote the EMA of the short-response ratio (fraction of responses with $\ell \le L_1$) across batches. We set $(w_{\text{acc}}, w_{\text{len}}) = (0.5, 0.5)$ when $s < 0.5$, linearly interpolate to $(0.8, 0.2)$ on $s \in [0.5, 0.9]$, and stay at $(0.8, 0.2)$ once $s \ge 0.9$.

3.2. Environment-Engineering Framework

Unlike static curriculum design, our framework forms a closed feedback loop in which the learner modifies the distribution of its future training environments as its weaknesses change. As shown in Figure 2, each round consists of TRAIN → EVAL → DESIGN. The model used in DESIGN is the current learner checkpoint: it reads the latest validation result, proposes the next configuration C_{n+1}, and the generator produces the corresponding training data.

The design context determines what evidence the environment engineer can use when producing C_{n+1}. We study five modules. (1) **Failure breakdown (F)** reports the latest validation outcome, including aggregate valid and optimal rates and per-map-size counts for parse errors, illegal moves,

Variant	Failure breakdown	Guideline	History (w/ default config)	History (w/o default config)	Summary	Training details
V1	✓	✗	✗	✗	✗	✗
V2	✓	✓	✗	✗	✗	✗
V3	✓	✓	✓	✗	✗	✗
V4	✓	✓	✗	✓	✗	✗
V5	✓	✓	✗	✓	✓	✗
V6	✓	✓	✗	✓	✗	✓

Table 1 | Modules included in each of the six context settings.

conflicts, hole collisions, out-of-bound moves, and goal failures. (2) **Guideline (G)** provides task-level design heuristics that are independent of a particular round. (3) **History (H)** gives a short record of previous $\{failure, config\}$ pairs; we compare versions with and without the randomly sampled round-0 configuration. (4) **Summary (S)** is a model-generated explanation of the current configuration choice that is carried into later rounds. (5) **Training details (T)** describes the RL objective already used during training, including the reward design and adaptive-weight schedule. It does not expose held-out instances or evaluation labels. Because the environment engineer can only modify generator configurations, this information supports training-aware environment design rather than reward hacking.

Context variants. We build up the context incrementally with 6 variants as summarized in Table 1. V1–V3 add modules incrementally on top of the failure breakdown; V4 differs from V3 only in that the round-0 default configuration is removed from the history, so that the model does not treat the random default as a recommended baseline; V5 and V6 then add the model-generated summary (**S**) and the training-details module (**T**) on top of V4. Implementation details of the prompts are provided in Appendix D.

Final framework. Based on the per-setting results, we adopt **V6** as the final implementation of our framework. See Figure 8 in Appendix B for the detailed results. We analyze why this setting works best in Section 4.3.

4. Experiments and Analysis

4.1. Setup

Data Construction and Evaluation. All training and validation data are produced by the generator described in §3.1.1. For each map size, the generator configuration specifies the data ratio, the hole ratio, and the wait ratio. Both the training set and the validation set consist exclusively of 2-agent instances, on grids ranging from 3×3 to 10×10. We use a fixed total of 4000 training instances in each round. Round 0 uses a randomly sampled configuration to produce both the initial training data and the validation set; from round 1 on, the configuration is emitted by the environment engineer to adjust the training data, while the validation set is held fixed across rounds.

To probe generalization, we use a separate *evaluation benchmark* composed of 3-, 4- and 5-agent instances. The benchmark is divided into three subsets by wait ratio: `wr_025`, `wr_050`, and `wr_075`, corresponding to wait ratios 0.25, 0.50, and 0.75. For each map size within each subset, we use 50 samples, evenly distributed across the five hole ratios $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. We use *valid rate* and *optimal rate* as the evaluation metrics. The valid rate measures whether a response parses to legal, conflict-free paths that reach the goals without hitting holes or leaving the grid, matching the criteria

Model	3×3		4×4		5×5		6×6		7×7		8×8		9×9		10×10		Sum	
	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.
GPT-5.4	64.67	41.33	44.67	29.33	42.67	31.33	28.00	16.00	20.67	12.67	26.67	15.33	20.00	11.31	12.67	7.33	32.50	20.58
Grok-4.2	36.00	25.33	50.00	37.33	29.33	18.67	35.33	21.33	29.33	12.00	36.67	21.33	36.00	22.00	14.67	10.00	33.42	21.00
Gemini-3.1-Pro	45.33	33.33	32.67	23.33	35.33	20.00	24.00	12.00	18.67	9.33	20.00	10.67	12.00	9.33	8.00	4.67	24.50	15.33
Kimi-K2.5	66.00	43.33	59.33	34.67	57.33	34.00	47.33	34.67	38.00	22.67	41.33	24.00	36.67	24.67	23.33	16.00	46.17	29.25
Qwen3-4B (base)	40.00	38.00	24.00	21.33	18.00	16.67	10.67	10.67	8.00	8.00	5.33	4.67	10.00	10.00	2.67	2.67	14.83	14.00
Qwen3-4B + GRPO (random)	54.67	41.33	54.00	40.67	50.67	29.33	42.67	26.00	38.67	21.33	30.67	18.00	32.67	18.67	19.33	13.33	40.42	26.08
Qwen3-4B + GRPO + Ours	**68.67**	**48.00**	**64.67**	**41.33**	**62.67**	**35.33**	**52.67**	**37.33**	**46.00**	**26.00**	**42.67**	**24.00**	**44.00**	**25.33**	**32.00**	**16.00**	**51.67**	**31.67**

Table 2 | Main results on the **3-agent** evaluation set across map sizes 3×3 to 10×10. `acc.` is the valid rate (%) and `opt.` is the optimal rate (%); the right-most **Sum** column reports the aggregate over all sizes.

Model	4×4		5×5		6×6		7×7		8×8		9×9		10×10		Sum	
	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.
GPT-5.4	35.33	22.00	24.00	16.00	16.00	10.00	14.00	9.33	16.67	10.67	9.33	5.33	4.00	2.67	17.05	10.86
Grok-4.2	48.00	**34.00**	30.00	21.33	32.00	18.00	25.33	16.67	5.33	3.33	8.67	6.67	14.67	7.33	23.43	15.33
Gemini-3.1-Pro	28.67	22.67	16.67	14.67	12.67	10.67	16.67	12.67	8.67	5.33	4.67	2.00	3.33	2.67	12.95	10.10
Kimi-K2.5	44.67	28.67	35.33	20.67	28.67	16.67	28.00	22.67	22.00	16.67	20.67	14.00	9.33	6.00	26.95	17.90
Qwen3-4B (base)	10.67	9.33	4.67	4.00	2.67	2.67	4.00	3.33	2.00	2.00	0.00	0.00	0.00	0.00	3.43	3.05
Qwen3-4B + GRPO (random)	42.67	27.33	33.33	23.33	31.33	18.67	26.67	15.33	19.33	12.00	19.33	10.67	14.00	5.33	26.67	16.10
Qwen3-4B + GRPO + Ours	**49.33**	32.00	**37.33**	**25.33**	**36.67**	**22.00**	**33.33**	**24.67**	**31.33**	**20.67**	**25.33**	**16.67**	**18.67**	**8.00**	**33.14**	**21.33**

Table 3 | Main results on the **4-agent** evaluation set across map sizes 4×4 to 10×10. `acc.` is the valid rate (%) and `opt.` is the optimal rate (%); the right-most **Sum** column reports the aggregate over all sizes.

Model	5×5		6×6		7×7		8×8		9×9		10×10		Sum	
	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.	acc.	opt.
GPT-5.4	17.33	14.00	10.00	6.00	10.00	4.00	6.00	4.67	5.33	4.00	6.00	3.33	9.11	6.00
Grok-4.2	26.67	16.00	13.33	9.33	16.67	10.67	6.67	6.00	7.33	**5.33**	6.67	5.33	12.89	8.78
Gemini-3.1-Pro	9.33	8.00	6.67	4.67	5.33	4.00	4.67	3.33	2.00	1.33	0.67	0.00	4.78	3.56
Kimi-K2.5	23.33	17.33	18.00	12.00	16.00	7.33	14.00	8.67	8.00	4.67	6.00	2.67	13.47	8.78
Qwen3-4B (base)	2.67	2.00	2.67	2.67	0.67	0.00	2.00	2.00	0.67	0.67	0.00	0.00	1.44	1.22
Qwen3-4B + GRPO (random)	24.00	16.67	21.33	14.00	16.00	10.00	13.33	8.00	8.67	2.67	7.33	3.33	15.11	9.11
Qwen3-4B + GRPO + Ours	**28.00**	**18.00**	**26.00**	**17.33**	**22.00**	**12.00**	**18.00**	**10.67**	**10.00**	2.67	**8.00**	**5.33**	**18.67**	**11.00**

Table 4 | Main results on the **5-agent** evaluation set across map sizes 5×5 to 10×10. `acc.` is the valid rate (%) and `opt.` is the optimal rate (%); the right-most **Sum** column reports the aggregate over all sizes.

used by the accuracy reward in §3.1.2. The optimal rate measures whether the model can find an optimal solution. These two metrics together capture both correctness and solution efficiency.

Implementation and Baselines.

We use Qwen3-4B (Team, 2025b) as the base model and train it with verl, using GRPO with the adaptive-weight reward described in §3.1.2. To balance training gains and computational efficiency, we run three training rounds, each consisting of two epochs. We compare against four frontier LLMs: GPT-5.4, Grok-4.2, Gemini-3.1-Pro and Kimi-K2.5, and two open-source references that share our backbone: the untrained Qwen3-4B base model and *Qwen3-4B + GRPO (random)*, a Qwen3-4B trained with the same GRPO procedure but on a single randomly sampled Round-0 configuration. The latter isolates the contribution of the training loop from the contribution of GRPO training itself. Detailed training hyperparameters are provided in Appendix C.

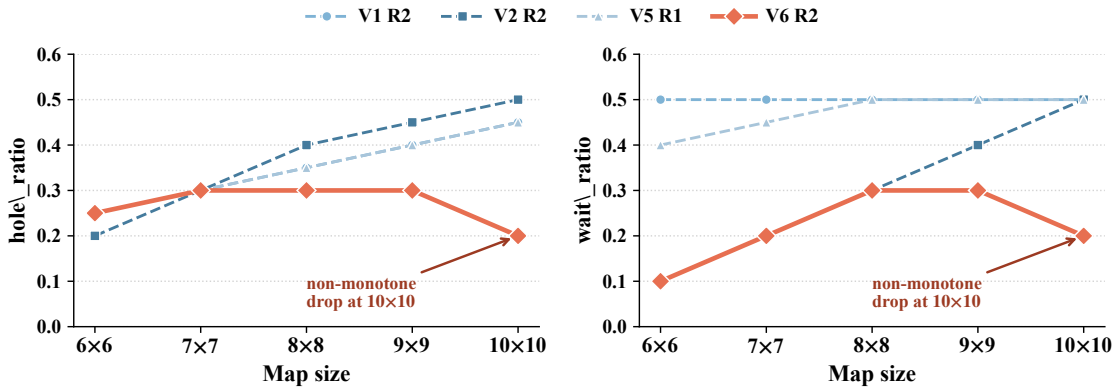

Figure 3 | `hole_ratio` (left) and `wait_ratio` (right) on the five largest map sizes (6×6–10×10), where the size-monotone vs. learner-aware contrast is sharpest. V1 R2, V2 R2, and V5 R1 keep both ratios monotone in map size (size-driven template), whereas V6 R2 plateaus at 7×7–9×9 and drops back at 10×10 on both variables, where the failure breakdown indicates the largest maps are no longer producing useful learning signal.

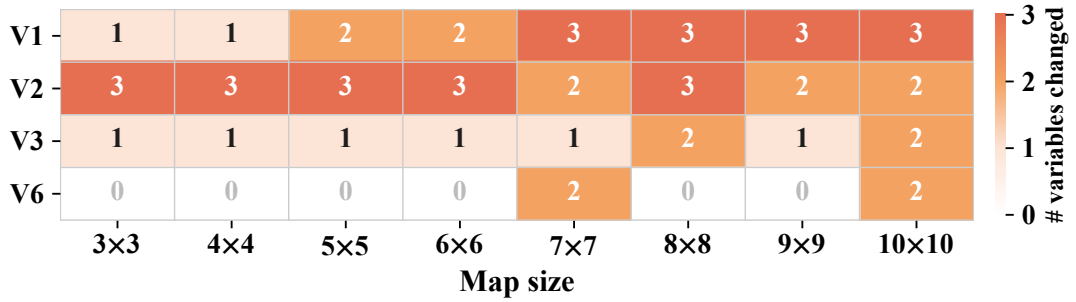

Figure 4 | Edit-granularity matrix. Each map size is controlled by three variables: `data_ratio`, `hole_ratio`, and `wait_ratio`. Each cell reports how many of these three variables the model changes between R1 and R2 for the given variant and map size (threshold $|\Delta|>0.01$; possible values 0–3).

4.2. Main Results

Tables 2, 3 and 4 report the per-map-size valid rate (`acc.`) and optimal rate (`opt.`) on the 3-, 4- and 5-agent evaluation sets. Across all three agent counts our framework, *Qwen3-4B + GRPO + Ours*, achieves the highest aggregate Sum on both metrics, surpassing every closed- and open-source baseline. Compared with the best commercial baseline (Kimi-K2.5) on each agent count, across the 3- to 5-agent benchmarks our framework improves the valid rate by +5.20 to +6.19 points and the optimal rate by +2.22 to +3.43 points. Compared with *Qwen3-4B + GRPO (random)* – which shares our backbone and training procedure but uses a fixed configuration – our framework adds +3.56 to +11.25 points of valid rate and +1.89 to +5.59 points of optimal rate across the three benchmarks, showing that the training loop both raises the model's overall capability and shapes the environment so that the model more often produces the optimal plan, not merely a valid one.

Rank	Setting	Salience	Granularity	Causal	Self-corr.	Task model
1	**V6**	✓	✓	✓	✓	✓
2	V4	✓	✓	×	✓	×
3	V3	×	△	×	△	×
4	V2	△	×	×	×	×
5	V5	××	×	×	×	×
6	V1	××	×	×	×	×

Table 5 | Behavioral profile of the six context settings along the five axes recovered from their reasoning traces. ✓ = passes, △ = partial, × = fails, ×× = fails on multiple patterns of the same axis. The left-most column gives the end-task ranking.

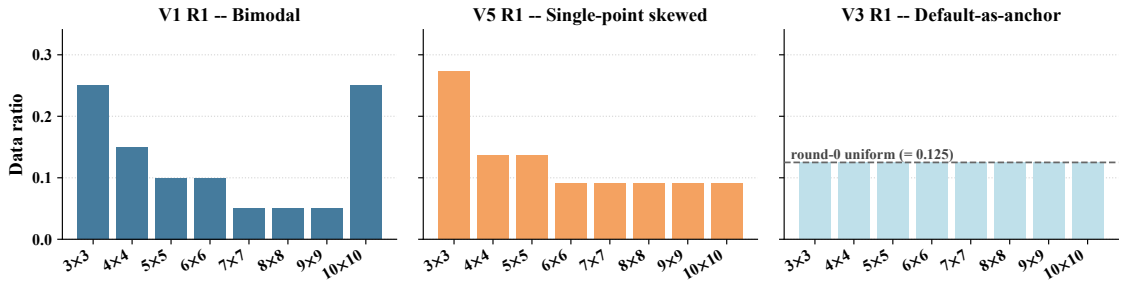

Figure 5 | Signal salience across context settings: each panel shows the `data_ratio` distribution emitted by a given (variant, round) configuration. V1 R1 and V5 R1 spend the budget on summing to 1; V3 R1 stays close to the round-0 uniform default; V5 R1→R2 lets its own self-summary override the raw failure breakdown.

4.3. Behavioral Analysis

We analyze the environment engineer's reasoning traces across the three training rounds (R0, R1, R2) and group the recurring patterns into five behavioral dimensions; Table 5 reports which dimensions each of the six context settings satisfies.

(1) Signal salience. As shown in Figure 5, the model is drawn to the most locally certain cue inside each module: V1 R1 and V5 R1 spend the budget on summing to 1, V3 R1 stays close to the round-0 uniform default, and V5 R1→R2 lets its own self-summary override the raw failure breakdown. The shared mechanism is that the most salient surface cue, rather than the most informative one, drives the decision. We therefore remove the round-0 default and the self-summary, and admit only modules whose dominant cue is task-grounded. **(2) Edit granularity.** From Figure 4 we see that V1 and V2 rewrite nearly every (size, knob) cell between rounds, while V3 and V6 leave most sizes untouched and edit only the sizes the failure breakdown points to. Full rewrites routinely degrade sizes that were already healthy (V1 R2 raises the small-map hole ratios and small-map valid rate drops), whereas selective edits do not. A good context should therefore give the model the confidence *not* to modify what is already working. **(3) Feature-based templates vs. learning-signal-based decisions.** Figure 3 shows that V1/V2/V5 keep both hole and wait ratios monotone in map size (the *"larger maps higher hole ratio"* template), whereas V6 plateaus at 7×7–9×9 and drops back at 10×10 on both variables. The training-details module helps V6 reason about which environments are likely to provide useful learning signal, rather than simply mapping difficulty to surface map size. Effective contexts therefore move the model from surface templates to updates that are conditioned on the observed failure pattern. **(4) Cross-round self-correction.** Figure 6 shows that V4 R2 and V6 R2 edit the configuration directly from the raw failure breakdown, whereas V5 R2 lets its own R1 self-

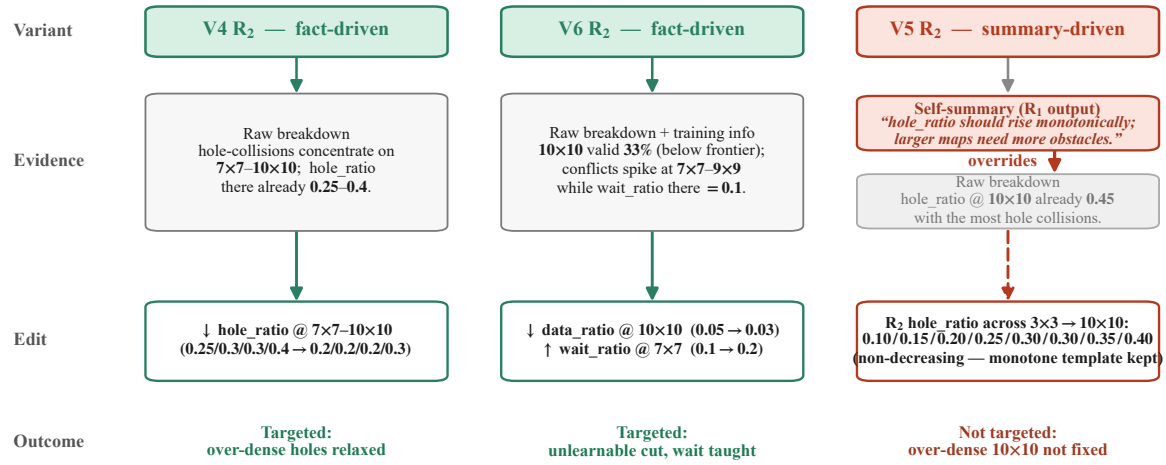

Figure 6 | Self-correction schematic. V4 R2 and V6 R2 edit the configuration based on the raw failure breakdown, while V5 R2 lets its own R1 self-summary override the breakdown and only nudges the existing monotone template.

Map size	V6 R1 `data_ratio`	V6 R2 `data_ratio`
3×3	0.05	0.05
4×4	0.10	0.10
5×5	0.15	0.15
6×6	0.15	0.15
7×7	0.15	**0.175**
8×8	**0.175**	0.175
9×9	**0.175**	0.175
10×10	**0.05**	**0.03**

Table 6 | The shift of V6 `data_ratio` from R1 to R2. The hardest map (10×10) is down-weighted in both rounds; in R2 the budget shifts from the 8×8/9×9 peak to 7×7, the new competence frontier.

summary override the breakdown and only nudges the existing monotone template. Self-correction therefore requires evidence that is independent of the model's previous narration. The context should provide facts as much as possible, rather than only interpretations of those facts. **(5) Task-grounded modeling.** Table 6 shows that only V6 down-weights the hardest map (10×10 set to 0.05 in R1) and concentrates budget just below the competence frontier, then shifts the frontier inward to 7×7 in R2. This behavior is contingent on the training-details module: V1–V5 treat every size as equally trainable and over-invest in the hardest size. Our framework therefore avoids simply maximizing difficulty and instead uses the observed failures to choose environments that remain useful for training. Since the environment engineer can only change generator configurations, not the reward function or evaluation set, this reflects training-aware environment design rather than reward hacking.

4.4. Ablation Study

What kind of training details matter (T). We ablate the training-details module to test whether V6 needs the full RL training details or only basic loop bookkeeping. Both settings include the current round index, epochs per round, and total epochs; only the full setting additionally includes the GRPO algorithm, adaptive-weight reward, and key hyperparameters. As shown in Table 7, the

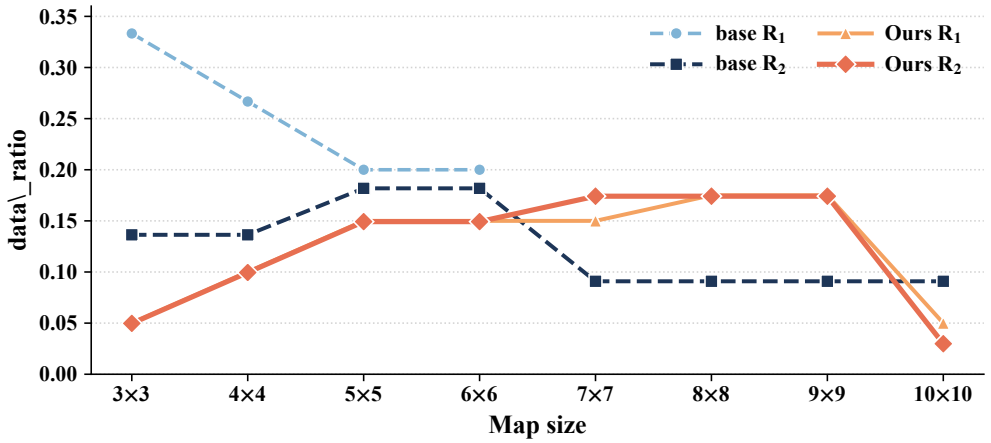

Figure 7 | `data_ratio` produced by the base model and by our current RL checkpoint when each plays the environment engineer, across rounds R1 and R2.

Setting	3-agent		4-agent		5-agent	
	acc.	opt.	acc.	opt.	acc.	opt.
Full RL details	38.83	26.50	24.76	18.19	14.22	10.11
Bookkeeping only (Ours)	**51.67**	**31.67**	**33.14**	**21.33**	**18.67**	**11.00**

Table 7 | Comparison between *Full RL details* and *Bookkeeping only* on the 3-, 4- and 5-agent benchmarks.

bookkeeping-only setting outperforms the full setting on every agent count, in both valid rate and optimal rate. This suggests that the model mainly needs to know where it is in the training loop, while detailed RL parameters can distract it from the current failure evidence. In our framework, training details are useful when they provide stage awareness, not when they overload the context with optimization details.

Who plays the environment engineer. We compare two choices for playingthe environment-engineer role: the current RL checkpoint of the learner, versus the untrained base model. As reported in Table 8, with every other component of the framework held fixed, the current checkpoint outperforms the base model on all 3-, 4- and 5-agent benchmarks in both valid rate and optimal rate. Figure 7 shows the underlying `data_ratio` allocations: in R1 the base engineer abandons 7×7–10×10 and pours the entire budget into 3×3–6×6, and in R2 – even after the larger sizes are added back – it still gives each of them only ~9% of the budget, while our checkpoint maintains a frontier-aware allocation across all eight sizes throughout both rounds. We interpret this as a form of *self-aware learning*: policy learning sharpens the model's ability to diagnose its own remaining weaknesses, so the trained checkpoint can target the next stage's data more precisely at those gaps than the untrained base model at the start.

5. Conclusion

We introduced a closed-loop framework where an LLM acts as an environment engineer, proactively redesigning the training configuration for its own RL learning. We developed MAPF-FrozenLake as a controllable testbed and showed that a 4B policy model, guided by structured feedback, can iteratively propose environment configurations that consistently outperform larger proprietary LLMs on Multi-Agent Path Finding tasks. Our mechanistic analysis further reveals that RL training enhances

Engineer	3-agent		4-agent		5-agent	
	acc.	opt.	acc.	opt.	acc.	opt.
Untrained base	45.21	30.00	27.62	19.62	16.00	10.89
Current checkpoint (Ours)	**51.67**	**31.67**	**33.14**	**21.33**	**18.67**	**11.00**

Table 8 | Comparison between the untrained base model and the current RL checkpoint playing the environment-engineer role, on the 3, 4, 5-agent benchmarks.

the model's ability to diagnose its own weaknesses, and that successful redesign depends on evidence-driven adaptation rather than naive difficulty maximization. These findings lay a foundation for studying self-improving learning systems via policy-conditioned environment engineering.

References

A. S. Azad, I. Gur, A. Faust, P. Abbeel, and I. Stoica. Clutr: Curriculum learning via unsupervised task representation learning. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:252992562.

S. Bae, J. Hong, M. Y. Lee, H. Kim, J. Nam, and D. Kwak. Online difficulty filtering for reasoning oriented reinforcement learning. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 700–719, 2026.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

L. Chen, M. Prabhudesai, K. Fragkiadaki, H. Liu, and D. Pathak. Self-questioning language models. *arXiv preprint arXiv:2508.03682*, 2025.

W. Fang, S. Liu, Y. Zhou, K. Zhang, T. Zheng, K. Chen, M. Song, and D. Tao. Serl: Self-play reinforcement learning for large language models with limited data. *ArXiv*, abs/2505.20347, 2025. URL https://api.semanticscholar.org/CorpusID:278910659.

C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR, 2018.

H. Gao, J. Geng, W. Hua, M. Hu, X. Juan, H. Liu, S. Liu, J. Qiu, X. Qi, Y. Wu, H. Wang, H. Xiao, Y. Zhou, S. Zhang, J. Zhang, J. Xiang, Y. Fang, Q. Zhao, D. Liu, Q. Ren, C. Qian, Z. Wang, M. Hu, H. Wang, Q. Wu, H. Ji, and M. Wang. A survey of self-evolving agents: On path to artificial super intelligence. *CoRR*, 2025.

A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. Pmlr, 2017.

C. Huang, W. Yu, X. Wang, H. Zhang, Z. Li, R. Li, J. Huang, H. Mi, and D. Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

P. Huang, M. Xu, J. Zhu, L. Shi, F. Fang, and D. Zhao. Curriculum reinforcement learning using optimal transport via gradual domain adaptation. *Advances in neural information processing systems*, 35:10656–10670, 2022.

S. Kim, K. Lee, and J. Choi. Variational curriculum reinforcement learning for unsupervised discovery of skills. In *International Conference on Machine Learning*, pages 16668–16695. PMLR, 2023.

Q. Li, Y. Zhai, Y. Ma, and S. Levine. Understanding the complexity gains of single-task rl with a curriculum. In *International Conference on Machine Learning*, pages 20412–20451. PMLR, 2023.

X. Liang, Z. Li, Y. Gong, Y. Shen, Y. N. Wu, Z. Guo, and W. Chen. Beyond pass@ 1: Self-play with variational problem synthesis sustains rlvr. *arXiv preprint arXiv:2508.14029*, 2025.

OpenAI. Openai GPT-5 system card. *CoRR*, 2026.

T. Shi, C. Huang, F. Wan, L. Zhong, Z. Yang, W. Shen, X. Quan, and M. Yan. Mutual-taught for co-adapting policy and reward models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16285–16298, 2025.

M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020. URL https://arxiv.org/abs/2010.03768.

K. Team. Kimi k1.5: Scaling reinforcement learning with llms. *CoRR*, 2025a.

K. Team. Kimi K2.5: visual agentic intelligence. *CoRR*, abs/2602.02276, 2026. URL https://doi.org/10.48550/arXiv.2602.02276.

Q. Team. Qwen3 technical report. *CoRR*, 2025b.

Q. Wu, H. Zhao, M. Saxon, T. Bui, W. Y. Wang, Y. Zhang, and S. Chang. Vsp: Diagnosing the dual challenges of perception and reasoning in spatial planning tasks for mllms. *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2270–2280, 2025. URL https://api.semanticscholar.org/CorpusID:286506971.

T. Xie, Z. Gao, Q. Ren, H. Luo, Y. Hong, B. Dai, J. Zhou, K. Qiu, Z. Wu, and C. Luo. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*, 2025.

W. Yuan, R. Y. Pang, K. Cho, X. Li, S. Sukhbaatar, J. Xu, and J. Weston. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning*, pages 57905–57923, 2024.

W. Zeng, Y. Huang, Q. Liu, W. Liu, K. He, Z. Ma, and J. He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *CoRR*, 2025.

J. Zhang, Y. Peng, F. Kong, C. Yang, Y. Wu, Z. Yu, J. Xiang, J. Ruan, J. Wang, M. Song, H. Liu, X. Tang, B. Liu, C. Wu, and Y. Luo. Autoenv: Automated environments for measuring cross-environment agent learning. *CoRR*, 2025.

Y. Zhang, P. Abbeel, and L. Pinto. Automatic curriculum learning through value disagreement. *Advances in Neural Information Processing Systems*, 33:7648–7659, 2020.

S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, Y. Bisk, D. Fried, U. Alon, and G. Neubig. Webarena: A realistic web environment for building autonomous agents. *ArXiv*, abs/2307.13854, 2023. URL https://api.semanticscholar.org/CorpusID:260164780.

Y. Zhou, S. Levine, J. Weston, X. Li, and S. Sukhbaatar. Self-challenging language model agents. *Advances in Neural Information Processing Systems*, 38:113959–113991, 2026.

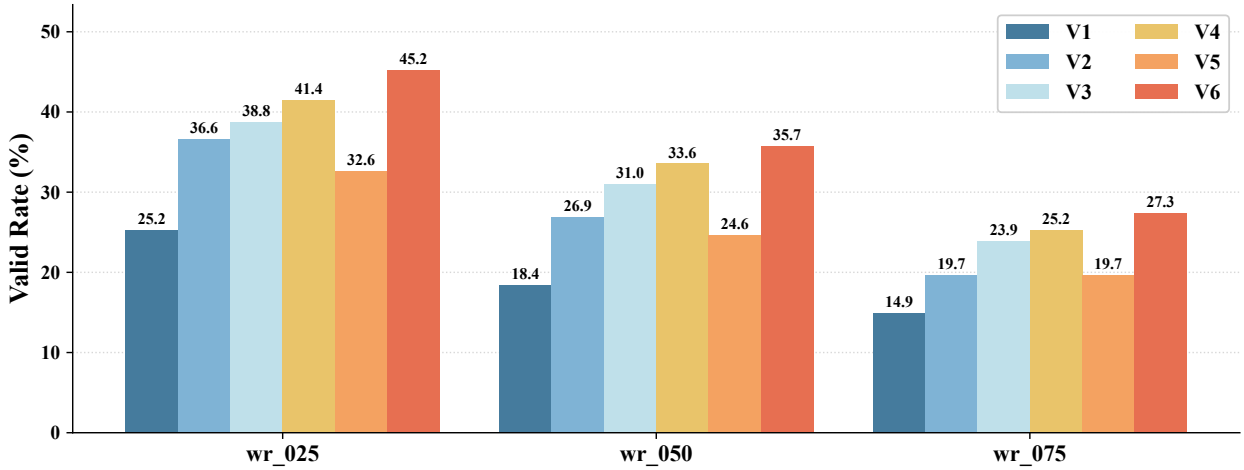

Figure 8 | Total valid rate of the six context settings on the three evaluation benchmarks (`wr_025`, `wr_050`, `wr_075`), aggregated across 3-, 4-, 5-agent instances and all map sizes. V6 achieves the highest valid rate on every benchmark.

A. Validity checks for the accuracy reward

This section lists the full set of validity checks that determine R_{acc}. A response is counted as a success only if it passes all eight of the following checks; failing any one sets $R_{\text{acc}} = 0$:

1. **Parsable** – paths can be extracted from the response;
2. **Legal-move** – each step has Manhattan distance ≤ 1;
3. **Conflict-free** – no vertex or edge conflict;
4. **Goal-reached** – every agent ends at its goal;
5. **Start-correct** – every agent starts at its start;
6. **Agent-count** – the parsed agent count matches the problem;
7. **Hole-free** – no path crosses a hole;
8. **In-bounds** – no path leaves the grid.

B. Per-setting valid rate of the six context variants

Figure 8 shows the total valid rate of all six context settings on the three evaluation benchmarks, aggregated across 3-, 4-, 5-agent instances and all map sizes. V6 achieves the highest valid rate on every benchmark.

C. Training details

We train with GRPO on top of Qwen3-4B (base). Each GRPO update draws a batch of 128 prompts; for every prompt we sample n=8 trajectories from the current actor with vLLM (`tensor_parallel`=2, `gpu_memory_utilization`=0.55, `max_prompt_length`=1536, `max_response_length`=5120). The resulting 1024 samples form one mini-batch, processed with per-GPU micro-batch 4, yielding 8 steps of gradient accumulation across 4× H100 80 GB GPUs.

We optimize the actor with AdamW at a constant learning rate of 2×10^{-6} and apply a low-variance KL penalty (β=10^{-3}) directly in the actor loss; no KL is added to the reward and entropy regularization is disabled. GRPO requires no critic. The reference policy is the policy at the start of each round and

is kept frozen with FSDP parameter offload; the actor uses FSDP without offload but with gradient checkpointing.

Each training round performs three RL epochs over the round's 4000-sample training set, and the pipeline runs for three rounds in total. The software stack is VERL v0.6.1 with PyTorch 2.8 / CUDA 12.4 / vLLM 0.10.3.

D. Prompts used by the environment engineer

This section reproduces the two prompts used at every training round. The *analysis prompt* is given the evaluation breakdown and emits the next-round configuration as a fenced YAML block. The *summary prompt* condenses the just-completed round into a short note that is then fed back as **S**-module context.

Analysis prompt

```
You are analyzing the training progress of an AI model that learns to solve
↪  Multi-Agent Path Finding (MAPF) problems.

## Task Description
MAPF involves finding collision-free paths for multiple agents on a grid map with
↪  obstacles (holes). Each agent must move from start to goal. At each time step, an
↪  agent can move UP, DOWN, LEFT, RIGHT, or WAIT. Two agents cannot occupy the same
↪  cell (vertex conflict) or swap positions (edge conflict).

The model is trained exclusively on 2-agent data but evaluated on 3/4/5-agent tasks to
↪  test generalization. The goal is for the model to learn fundamental MAPF skills
↪  (spatial reasoning, obstacle avoidance, conflict resolution) from 2-agent
↪  scenarios and generalize to more agents. Your config should only control 2-agent
↪  training data -- do NOT add 3/4/5-agent entries.
{training_details}
## Training Data Configuration
The total number of training samples is FIXED at 4000 across all map sizes. You
↪  control how those 4000 samples are distributed by setting per-map-size **ratios**.

The training data is generated from a YAML config that controls:
- **Map sizes** (e.g. 3x3, 5x5, 10x10): The grid dimensions for the MAPF problem.
- **ratio** per size: Share of the 4000 samples allocated to that map size (range
↪  0-1). ...
- **hole_ratio** per size: Fraction of cells that are obstacles (range 0-0.5).
- **wait_ratio** per size: Fraction of samples that must contain WAIT actions for
↪  conflict resolution (range 0-1).

You can adjust any of these parameters to tailor the training curriculum. ...

## Training History
{history_section}

## Your Task
Analyze the evaluation results above. You are given:
1. **Overall failure breakdown**: failure counts across all map sizes (categories are
↪  NOT mutually exclusive).
2. **Per map-size summary**: Valid% and Optimal% for each map size, showing where the
↪  model excels vs. struggles.
3. **Per map-size failure breakdown**: failure type counts broken down by map size.

Failure type reference:
- **Conflict failures**: two agents collided (vertex or edge conflict).
```

```
- **Parse failures**: the model output could not be parsed into valid action
↪  sequences.
- **Illegal moves**: the model output an action that is not in the action space.
- **Goal failures**: one or more agents did not reach their goal position.
- **Hole collisions**: an agent moved onto an obstacle cell.
- **Out of bounds**: an agent moved outside the grid.

## Guidelines
- If training history is provided, pay close attention to how the failure distribution
↪  AND per-size performance changed after each config adjustment.
- Since training is continual, overall metrics will generally improve over rounds. Pay
↪  more attention to which map sizes or failure types are still lagging behind ...
- Prior config adjustments were made based on earlier training stages and may no
↪  longer be appropriate. Make your own judgment based on the current metrics ...
- You may also analyze the current training data distribution to identify what types
↪  of data are missing or underrepresented ...
- **IMPORTANT**: The model learns progressively from easy to hard -- ensure it has
↪  mastered simpler tasks before shifting focus to harder ones.
- **IMPORTANT**: Because the total budget is fixed at 4000 samples and all 8 sizes
↪  (3x3 to 10x10) are always included, increasing the ratio for one map size
↪  implicitly takes samples away from another. Think of the 8 ratios as a probability
↪  distribution that always sums to 1.0.

Based on your analysis (and the trend if history is available), recommend changes to
↪  the generation config for the next training round. You may change ratio,
↪  hole_ratio, and/or wait_ratio for any map sizes.

Output your recommendation as a fenced YAML block (```yaml ... ```). You do NOT need
↪  to output every field -- only the fields you want to change. ...

The YAML structure is:
```yaml
generation:
  2_agents:
    NxN: {{ratio: <float 0-1>, hole_ratio: <float 0-0.5>, wait_ratio: <float 0-1>}}
```
```

Summary prompt

```
You are reviewing the training progress of an AI model learning to solve Multi-Agent
↪  Path Finding (MAPF) problems.

## Background
MAPF involves finding collision-free paths for multiple agents on a grid map. The
↪  training data is generated from a YAML config controlling map sizes, sample
↪  counts, hole_ratio (obstacle density), and wait_ratio (fraction of
↪  conflict-resolution samples).

## Training History
{history_section}

## New Config Chosen
{new_config_summary}

## Your Task
Based on the evaluation results and the new config above, write a concise summary (3-5
↪  sentences) explaining:
1. What are the key observations from the current evaluation results?
2. Why was this config chosen -- what weaknesses is it trying to address?
```

```
3. What should be monitored in the next round to judge if this adjustment was
↪  effective?

Be specific about map sizes, failure types, and parameter changes. Do NOT output any
↪  YAML or config -- only your reasoning summary.
```

E. Limitations

Our study has several limitations that suggest directions for future work. First, while MAPF-FrozenLake provides fine-grained control over training distributions, it represents a single, self-contained task family. The redesign strategies learned by the environment engineer may not directly transfer to domains with qualitatively different failure modes or evaluation signals. Second, our experiments focus on a specific RL training pipeline; the interaction between environment redesign and other training paradigms, such as online imitation learning or reward-free exploration, remains unexplored. Finally, our current framework restricts the environment engineer to a fixed generator architecture. Extending the approach to allow structural modifications of the generator itself, such as introducing new environment mechanics, raises additional challenges that we do not address here.