Iterative Training of a Minesweeper-Playing LLM

Through Failure-Driven Reward Engineering

A Multi-Stage SFT and GRPO Pipeline with Systematic Failure Mode Resolution

AMD AI Reinforcement Learning Hackathon

Jointly hosted by AMD, Yardi School of AI, IIT Delhi, and Unsloth

Ritvik Shrivastava

J Bharath Reddy

Prashasth Immanuel

Kamal Enoch

15th February 2026

Abstract

We present an iterative, failure-driven approach to training Llama-3.1-8B-Instruct to play Minesweeper by outputting JSON actions given JSON board states. Our pipeline evolved through seven major iterations (v1–v7), each targeting specific failure modes discovered empirically. In Stage 1 (SFT), we generate 8,000 expert demonstrations using a logical deduction oracle that prioritizes provably safe reveals, achieving 100% valid JSON output. In Stage 2 (GRPO), we apply reveal-biased reward functions with harsh flag-toggle penalties (−15) and strong safe-reveal bonuses (+20). We document the complete progression through five critical failure modes: GRPO reward variance collapse, KL divergence explosion, deterministic output convergence, recursive flag-toggle loops, and random cell selection. Training uses LoRA (rank 32, α=64) on AMD Instinct MI300X with Unsloth and Dr. GRPO loss normalization. Our methodology demonstrates that systematic failure analysis outperforms top-down design when teaching LLMs combinatorial games.

1. Introduction

Minesweeper is a single-player puzzle under incomplete information where a player reveals safe cells while avoiding hidden mines. Kaye (2000) proved that the Minesweeper consistency problem is NP-complete. Traditional solvers use CSP formulations (Studholme, 2001; Bayer et al., 2006) or deep RL (Sinha et al., 2021).

GRPO (Shao et al., 2024) improves LLM reasoning by computing advantages relative to a group of sampled outputs, eliminating the critic network. The TiG framework (Yang et al., 2025) showed that SFT followed by GRPO enables LLMs to acquire procedural game knowledge. DeepSeek-R1 (Guo et al., 2025) established SFT→RL as the standard approach for teaching complex reasoning.

We frame Minesweeper as a language modeling task and document our complete iterative development through seven versions. Our key contributions:

- An iterative, failure-driven methodology spanning v1–v7 with complete failure mode documentation.

- A reveal-focused expert oracle generating 8,000 demonstrations biased toward logically deducible safe moves.
- A GRPO stability analysis showing the critical role of learning rate (5×10^{-6}), gradient clipping (0.1), and Dr. GRPO loss.
- Resolution of five distinct failure modes: reward variance collapse, KL explosion, deterministic convergence, flag-toggle loops, and random cell selection.
- Practical solutions for infrastructure challenges: read-only caches, save failures, and checkpoint safety.

2. Problem Formulation

2.1 Game Environment

We implement a configurable Minesweeper environment supporting 5×5 to 50×50 boards with 10–20% mine density. Two actions are supported: Reveal (uncover a cell with flood-fill on zeros) and Flag (toggle flag marker). The game is won when all safe cells are revealed.

2.2 LLM Interface

The model receives a JSON state and outputs a JSON action within 128 tokens. System prompt: "You output JSON actions for Minesweeper. No text, only JSON."

```
{"type": "reveal", "row": 2, "col": 3}
```

2.3 Competition Scoring

Table 1: Competition scoring rules

Action	Points
Reveal safe (logical)	+15
Reveal safe (random)	+10
Flag correct mine	+15
Win game	+100
Reveal mine	−25
Flag already-flagged	−12
Out of bounds	−15
Invalid JSON	−50

3. Methodology

Our two-stage pipeline was refined through seven iterations. We present the final architecture (v7).

Table 2: Training pipeline overview

Stage	Component	Details
Base	Llama-3.1-8B-Instruct	4-bit quantized, LoRA rank=32, α=64
Stage 1	SFT	8,000 oracle reveals, 2 epochs, LR=1×10^{-4}
Stage 2	GRPO	2,000 states, 100 steps, LR=5×10^{-6}, Dr. GRPO
Output	Merged Model	16-bit merged LoRA for competition

3.1 Stage 1: Supervised Fine-Tuning

3.1.1 Expert Design

The oracle uses full mine knowledge to generate optimal moves with a strict priority hierarchy:

Table 3: Oracle priority hierarchy

Priority	Move Type	Description
1	Logically deducible safe	Provably safe via constraint propagation
2	Near-number strategic	Safe cell adjacent to numbered cells
3	Corner/edge opening	Periphery cell (statistically safer)
4	Any safe cell	Fallback safe reveal

3.1.2 Dataset Composition

8,000 examples across 12 board configs (5×5 to 10×10 including non-square). Each created by: random board initialization, 1–14 random safe moves for mid-game states, oracle action, chat-template formatting.

Table 4: SFT dataset composition

Move Type	~Count	~%
Logically deducible safe	3,800	48%
Near-number strategic	2,200	28%
Corner/edge opening	1,000	12%

Other safe reveals	1,000	12%

3.2 Stage 2: GRPO

3.2.1 Reward Functions

Two complementary functions: R_total = R_validity + R_gameplay.

Table 5: Reward scoring (reveal-biased design)

Scenario	Score	Rationale
Logically deducible safe reveal	+20	Best move; constraint reasoning
Safe reveal near numbers	+13	Expands information frontier
Any safe reveal	+10	Correct but not strategic
Win the game	+100	Maximum completion bonus
Flag correct mine (deducible)	+17	Correct with logical backing
Flag already-flagged	−15	Harsh: prevents toggle loops
Wrong flag	−15	Harsh: discourages random flags
Reveal a mine	−25	Game-ending mistake

3.2.2 GRPO Configuration

Table 6: GRPO hyperparameters

Parameter	Value	Rationale
Learning rate	5×10^{-6}	Prevents KL explosion (20× lower than SFT)
Max gradient norm	0.1	Clips runaway gradients
Loss type	Dr. GRPO	Length-normalized loss
Generations (G)	8	Sufficient diversity
Temperature	1.0 train / 0.6 eval	Exploration vs. quality
Gradient accumulation	4	Effective batch = 32
Mask truncated	True	Ignores token-limit garbage

4. Infrastructure Challenges

4.1 Read-Only Cache Resolution

The competition environment mounts HuggingFace cache as read-only, causing Unsloth's save_pretrained_merged() to crash. Fix: copy cache to writable location:

```
os.environ["HF_HUB_CACHE"] = "/workspace/models"  # writable copy
```

4.2 Checkpoint Safety

LoRA checkpoints are saved immediately after each training phase. Three tiers: checkpoint_after_sft/ (LoRA + merged attempt), checkpoint_after_grpo/ (full model backup), and my_minesweeper_model_merged/ (final submission).

4.3 Hardware

Component	Specification
GPU	AMD Instinct MI300X (192 GB HBM3)
Platform	ROCm 7.0 + PyTorch 2.8.0
Framework	Unsloth 2025.10.6 + TRL
Model	Llama-3.1-8B-Instruct (4-bit)
Trainable params	83.9M / 8.1B (1.03%)

5. Iterative Failure Mode Analysis

This section documents our central contribution: systematic discovery and resolution of five failure modes across seven iterations.

5.1 Zero Training Loss (v1)

Symptom: loss = 0.0000 at every step, zero reward improvement. **Cause:** 4 generations produced identical outputs → reward_std = 0 → zero GRPO gradients. **Fix:** Increased to 8 generations, higher temperature.

5.2 KL Divergence Explosion (v2)

Symptom: KL spiked from 0.1 to 198.1 in 2 steps, then total collapse at step 89. All outputs became invalid JSON. **Cause:** LR=1×10^{-4} is 20× too high for GRPO. Without gradient clipping, one high-variance batch caused irreversible divergence. **Fix:** LR=5×10^{-6}, max_grad_norm=0.1, Dr. GRPO loss.

Table 7: v2 collapse timeline

Steps	KL	Behavior
1–20	0.0→0.1	Healthy training
26–27	0.1→198	KL EXPLOSION (1000× in 2 steps)
89–104	N/A	Death spiral: all outputs garbage

5.3 Deterministic Convergence (v3)

Symptom: Stable KL (<0.4) but 0% wins. Model output identical 19 tokens every step. reward_std=0 in 99% of steps 401–500. **Cause:** GRPO is a refinement tool, not a teaching tool. The instruct model quickly converged to one valid JSON and repeated it. **Key insight:** SFT is mandatory before GRPO.

5.4 Flag-Toggle Loops (v5)

Symptom: flag(3,2)→flag(3,2)→flag(3,2)... repeated 15×. Games use 80 moves doing nothing. **Cause:** SFT data included flag examples; flag is a toggle action creating oscillation. **Fix:** Reveal-only SFT data, −15 flag penalty, runtime loop detection.

5.5 Random Cell Selection (v6)

Symptom: First win (5% on 6×6) but avg 3.0 moves, hitting mines on move 2–3. **Cause:** Too many early-game examples; model learned positions not deduction. **Fix:** 1–14 pre-moves for deeper states, ~48% deducible examples, 8K samples.

6. Conclusion

We presented an iterative, failure-driven methodology for training an LLM to play Minesweeper. The progression from 0% win rate through five failure modes to functional gameplay validates that systematic failure analysis outperforms top-down design. Key findings: (1) SFT is mandatory before GRPO, (2) SFT data composition matters more than hyperparameters, (3) GRPO needs 20× lower LR and strict gradient clipping, (4) infrastructure challenges require proactive checkpoint strategies.

Acknowledgments

We thank the Yardi School of Artificial Intelligence, IIT Delhi for organizing this competition. We acknowledge AMD for Instinct MI300X access with 192 GB HBM3 and ROCm stack, the Unsloth team for efficient LoRA training, and Hugging Face TRL for GRPOTrainer and SFTTrainer implementations.

References

[1] D. Bayer et al. An interactive constraint-based approach to Minesweeper. AAAI, 2006.

[2] D. Guo et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via RL. arXiv:2501.12948, 2025.

[3] E. J. Hu et al. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685, 2021.

[4] R. Kaye. Minesweeper is NP-complete. Math. Intelligencer, 22(2):9–15, 2000.

[5] J. Schulman et al. Proximal Policy Optimization algorithms. arXiv:1707.06347, 2017.

[6] Z. Shao et al. DeepSeekMath. arXiv:2402.03300, 2024.

[7] Y. P. Sinha et al. CSP and learning for Minesweeper. arXiv:2105.04120, 2021.

[8] C. Studholme. Minesweeper as a CSP. U. Toronto, 2001.

[9] X. Yang et al. Think in Games: RL with LLMs. arXiv:2508.21365, 2025.