Minesweeper LLM Agent: Fine-Tuning Qwen2.5-14B-Instruct for Competitive Minesweeper Play

Team 92

February 2026

Abstract

We present a fine-tuned Qwen2.5-14B-Instruct model that plays Minesweeper by outputting structured JSON actions given board states in a novel *frontier format*. Our approach combines: (1) a custom three-tier constraint satisfaction solver for high-quality training data, (2) a novel prompt representation that converts spatial grids into coordinate-based constraints, (3) supervised fine-tuning with LoRA on 37K curated examples, and (4) GRPO reinforcement learning with three custom reward functions. Key findings: frontier format achieves 100% valid move rate vs. 7–15% for compact ASCII grids, and system prompt alignment between training and inference is the single most important factor. The model achieves +34.8 average score per game with 100% valid JSON and 100% valid moves across all board sizes up to 50×50.

1 Introduction

Minesweeper is a constraint satisfaction problem where an agent must reveal safe cells and flag mines on a grid. In this competition, an LLM must output a single JSON action `{"type":"reveal"|"flag","r` per turn, scoring +15 for safe reveals, +15 for correct flags, −25 for mine hits, −10 for wrong flags, −12 for redundant moves, and +50 for winning. Boards range from 6×6 to 50×50 with 10–20% mine density.

The core challenge is teaching an LLM to: (a) parse spatial board information, (b) perform constraint reasoning over numbered cells and their hidden neighbors, and (c) output concise, valid JSON without verbose reasoning that would exceed the 128-token limit.

Our solution addresses all four competition pillars: **prompt engineering** (Section 2), **custom reward functions** (Section 3), **dataset curation** (Section 4), and **GRPO training** (Section 5).

2 LLM Prompt Engineering (Novel Design)

We designed our prompt system *entirely from scratch*, replacing the provided compact grid format with a novel **frontier format** representation.

2.1 The Problem: LLMs Cannot Read Grids

The default prompt represents the board as an ASCII grid:

```
MINESWEEPER 8x8 MINES:10 FLAGS:2 LEFT:8
00001...
00012...
00001F..
```

We discovered that LLMs **fundamentally cannot reason spatially** about ASCII grids. The model cannot reliably determine which cells are adjacent to which numbers, leading to only 7–15% valid moves (targeting cells that are actually hidden).

2.2 Our Solution: Frontier Format

We invented a **frontier format** that converts the spatial grid into an explicit constraint listing:

```
MINESWEEPER 8x8 MINES:10 FLAGS:2 LEFT:8
FRONTIER (numbered cells with hidden neighbors):
R0C4=1 flags:0 hidden:[(0,5)(0,6)(1,5)]
R1C3=2 flags:1 hidden:[(0,5)(1,5)(2,5)]
HIDDEN NEAR NUMBERS: (0,5)(0,6)(1,5)(2,5)...
TOTAL HIDDEN: 52 INTERIOR(no adj number): 40
RULES: .=hidden F=flag 0-8=adjacent mines
- If number N has N flags around it, remaining hidden
  neighbors are SAFE->reveal
- If number N needs (N-flags) more mines and has exactly
  that many hidden neighbors, all are MINES->flag
- Flag certain mines FIRST, then reveal safe cells
- NEVER act on already revealed or flagged cells
- Choose ONLY from HIDDEN NEAR NUMBERS cells listed above
Output ONLY: {"type":"reveal"|"flag","row":R,"col":C}
```

This format:

- Explicitly lists each numbered cell's constraint: its value, flag count, and hidden neighbor coordinates
- Provides a whitelist of valid target cells (`HIDDEN NEAR NUMBERS`)
- Embeds the constraint reasoning rules directly in the prompt
- Caps frontier info at 200 entries and hidden cells at 100 to control token count

Table 1: Impact of prompt format on model performance.

Format	Valid Moves	Valid JSON	Avg Score
Compact Grid (provided)	7–15%	85–95%	< 0
Frontier (ours)	**100%**	**100%**	**+34.8**

We set `FRONTIER_THRESHOLD = 0`, meaning *all* boards use frontier format, ensuring training and inference representations are identical.

2.3 System Prompt Design

We designed a concise system prompt optimized for structured output:

```
"You are an expert Minesweeper AI. Analyze constraints and output ONLY
a valid JSON action.  No explanation."
```

Critical finding: The system prompt at inference must *exactly match* the training prompt. We tested four configurations:

Table 2: System prompt alignment impact (same model, different prompts).

Model	System Prompt	Avg Score/Game
v1	Our training prompt (match)	**+34.8**
v1	Tournament prompt (mismatch)	+17.2
v2	v2 training prompt (match)	+37.1
v2	Tournament prompt (mismatch)	+4.7

Mismatched prompts cause up to **7.4× performance degradation**. This was the single most impactful discovery in our project.

3 Custom Reward Functions and Strategy

We designed **three custom reward functions** for GRPO training, each targeting a different aspect of Minesweeper play. These are combined with configurable weights.

3.1 Reward Function 1: Format Compliance (R_{format})

Ensures the model outputs valid, parseable JSON:

$$R_{\text{format}}(\text{response}) = \begin{cases} +1.0 & \text{if valid JSON with type, row, col fields} \\ -3.0 & \text{otherwise (heavy penalty for invalid output)} \end{cases} \tag{1}$$

The asymmetric penalty (-3.0 vs $+1.0$) strongly discourages format violations, which would cause immediate disqualification in the competition.

3.2 Reward Function 2: Gameplay Outcome (R_{game})

Simulates the actual game scoring with a full Minesweeper engine:

$$R_{\text{game}}(\text{action}) = \frac{1}{25} \times \begin{cases} +37.5 & \text{action wins the game (reveal/flag completes board)} \\ +15.0 & \text{correct flag on mine OR safe reveal (deducible)} \\ +10.0 & \text{safe reveal (non-deducible, lucky guess)} \\ -10.0 & \text{wrong flag OR over-flagging (flags} \geq \text{mines)} \\ -12.0 & \text{redundant (targeting revealed/flagged cell)} \\ -15.0 & \text{out of bounds} \\ -25.0 & \text{mine hit} \end{cases} \tag{2}$$

Each reward is normalized by dividing by 25 to keep gradients stable. The function reconstructs the full game state (mine positions, revealed cells, flagged cells) from dataset metadata and simulates the action's outcome.

3.3 Reward Function 3: Strategic Quality (R_{strat})

Rewards strategic thinking beyond just outcome:

$$R_{\text{strat}} = \sum \begin{cases} +0.20 & \text{if action targets cell adjacent to a number (frontier cell)} \\ +0.15 & \text{if correct deducible flag (logically certain mine)} \\ +0.15 & \text{if safe reveal opens a zero (cascading reveal)} \\ -0.30 & \text{if deducible move exists but model chose non-deducible} \\ -0.40 & \text{if flagging when already at mine count limit} \end{cases} \tag{3}$$

This reward encourages the model to:

- Prefer **deducible moves** (logically provable) over guesses (-0.3 penalty for ignoring available deductions)
- Target **frontier cells** (adjacent to numbers) rather than interior cells ($+0.2$)
- Flag only when **certain** ($+0.15$ for deducible flags, -0.4 for over-flagging)
- Prefer reveals that **cascade** (revealing zeros opens multiple cells, $+0.15$)

3.4 Combined Reward

The three functions are combined with learned weights:

$$R_{\text{total}} = 1.0 \cdot R_{\text{format}} + 2.0 \cdot R_{\text{game}} + 0.5 \cdot R_{\text{strat}} \tag{4}$$

Gameplay outcome is weighted highest ($2.0\times$) as it directly corresponds to competition scoring.

4 Dataset Curation

4.1 Three-Tier Constraint Satisfaction Solver

We built a custom solver (`solver.py`, 700+ lines) implementing hierarchical Minesweeper constraint satisfaction:

Tier 1 – Propagation: If numbered cell N has N flagged neighbors, remaining hidden neighbors are safe. If $N - F = |U|$, all hidden neighbors are mines. Covers $\sim$60–70% of deterministic moves.

Tier 2 – Set-Based: Coupled constraint analysis using subset reduction on paired cells sharing hidden neighbors. Covers $\sim$85–90% combined.

Tier 3 – Tank Solver: Backtracking enumeration over frontier components using Union-Find partitioning (35-cell component cap, 1s timeout). Computes per-cell mine probabilities weighted by:

$$\log w(m) = \text{lgamma}(Y+1) - \text{lgamma}(M-m+1) - \text{lgamma}(Y-M+m+1) \tag{5}$$

where Y = interior hidden cells, M = remaining mines, m = frontier mines. We use log-space (`lgamma`) instead of `math.comb` to avoid integer overflow on 50×50 boards where $Y > 1000$.

4.2 Board Size Distribution

We curated training data across **13 board configurations**, including 7 square and 6 rectangular sizes to match the competition's "NxM" specification:

Table 3: Training data distribution across board sizes (25K v2 dataset).

Board Size	Games	Mine Density	Type
6×6	1,000	10–20%	Square (small)
8×8	800	10–20%	Square
10×10	600	10–20%	Square
16×16	300	10–20%	Square
20×20	200	10–20%	Square
30×30	100	10–20%	Square (large)
50×50	80	10–20%	Square (max)
8×12	300	10–20%	Rectangular
10×16	200	10–20%	Rectangular
12×20	150	10–20%	Rectangular
16×30	100	10–20%	Rectangular
20×40	60	10–20%	Rectangular
30×50	40	10–20%	Rectangular

4.3 Data Quality Controls

- **Stage-balanced subsampling**: Prevents late-game/endgame domination. Each game contributes examples from early, mid, and late stages proportionally.
- **94% deducibility rate**: The vast majority of training examples have logically provable actions (Tier 1–3), teaching the model real constraint reasoning rather than guessing.
- **All frontier format**: Every example uses our novel frontier representation, ensuring zero train/inference mismatch.
- **Mine density matching**: All boards use 10–20% mine density, matching competition specification exactly.
- **Multiprocessing generation**: 32 parallel workers generate 25K+ examples in $\sim$90 seconds.

5 Training Pipeline: SFT + GRPO

5.1 Phase 1: Supervised Fine-Tuning (SFT)

We use LoRA (Low-Rank Adaptation) for parameter-efficient fine-tuning:

Table 4: SFT hyperparameters.

Parameter	Value
LoRA rank (r)	64
LoRA alpha (α)	128 (scaling = $\alpha/r = 2.0$)
Target modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj
Trainable parameters	275M / 15B (1.83%)
Effective batch size	16 (2 per device $\times$ 8 accumulation)
Learning rate	2×10^{-5} (cosine decay)
Epochs	1
Training steps	2,298
Max sequence length	8,192 tokens
Precision	BF16
Loss trajectory	$0.91 \rightarrow 0.09$

SFT establishes strong format compliance (100% valid JSON) and basic constraint reasoning.

5.2 Phase 2: GRPO Reinforcement Learning

After SFT, we applied **Group Relative Policy Optimization (GRPO)** using TRL 0.23.0 with our three custom reward functions:

Table 5: GRPO configuration.

Parameter	Value
Loss type	DAPO (Dynamic Advantage Policy Optimization)
Epsilon (clip range)	0.2 (low) / 0.28 (high, asymmetric)
Beta (KL penalty)	0.0 (no KL constraint)
Num generations per prompt	4
Max completion length	128 tokens
Temperature	1.0 (sampling for diversity)
Reward scaling	Batch-level normalization
Reward weights	[1.0, 2.0, 0.5] (format, gameplay, strategic)
Learning rate	5×10^{-6}
Training steps	400
Optimizer	AdamW 8-bit

5.3 GRPO Analysis and Lessons Learned

GRPO produced mixed results. While the reward functions correctly scored model outputs, several factors limited GRPO's effectiveness:

1. **Insufficient generation diversity**: With only 4 generations per prompt (constrained by 14B model on 256GB VRAM; 8 generations caused OOM), reward variance was too low for meaningful policy gradients. `grad_norm` ≈ 0 for most steps.

2. **SFT already near-optimal**: The SFT model achieved >95% correct actions, leaving minimal room for RL improvement. The model was already in a local optimum for format and basic reasoning.

3. **Performance degradation**: After 400 steps, average score decreased from +33.6 to <+25, suggesting GRPO was "unlearning" SFT knowledge faster than learning new strategies.

Key insight: For structured output tasks with limited generation budget, SFT on high-quality solver-generated data outperforms GRPO. RL benefits require either (a) many more generations per prompt ($\geq$16) for sufficient reward signal diversity, or (b) a weaker SFT baseline with more room for improvement.

Our final submission uses the **SFT-only checkpoint** which scored +34.8 avg/game vs GRPO's <+25.

6 Continued SFT Experiments

We explored continued fine-tuning to improve specific weaknesses:

- **v2 (5K examples, different system prompt)**: Catastrophic forgetting. Score dropped to +4.7 when evaluated with the new prompt, proving system prompt changes during continued SFT are destructive.
- **v3 (2.9K targeted examples, same prompt)**: Improved 50×50 performance (+60 vs −10) but regressed on medium boards (10 × 10: +16.7 vs +42.0). Net worse overall (+27.4 vs +34.8).
- **Lesson**: The original v1 model trained on the full 37K dataset remains the most robust. Continued SFT on subsets causes forgetting of the broader distribution.

7 Results

Table 6: Final model (v1 SFT) performance across board sizes (66 games).

Board	Games	Avg Score	Valid JSON	Valid Moves	Safe Reveals	Flag Acc.
6 × 6	15	+2.0	100%	100%	31	34%
8 × 8	15	+32.0	100%	100%	38	63%
10 × 10	15	+42.0	100%	100%	38	65%
16 × 16	8	+23.1	100%	100%	10	57%
20 × 20	8	+125.6	100%	100%	46	56%
30 × 30	3	−5.0	100%	100%	4	40%
50 × 50	2	−10.0	100%	100%	1	43%
Overall	**66**	**+34.8**	**100%**	**100%**	**168**	**56%**

Key metrics: 100% valid JSON, 100% valid moves, 0% verbose output (∼20 tokens per response), +34.8 avg score/game, no recursion risk.

8 Competition Compliance

- ✓ Base model: Qwen/Qwen2.5-14B-Instruct (approved list)
- ✓ Model at `/workspace/your_finetuned_model` (28GB merged)
- ✓ `max_new_tokens: 128` in config

✓ No post-LLM processing (`SAFETY_NET_ENABLED = False`)

✓ No internet or third-party calls during inference

✓ Concise JSON output (∼20 tokens, well under 128 limit)

✓ Handles boards up to 50×50 with 10–20% mines

✓ Base model copied to `/workspace` before training