Zindi AI Telco Troubleshooting Challenge - Solution Report

1. Approach Overview

Our solution uses a two-stage fine-tuning pipeline: reasoning trace generation followed by Supervised Fine-Tuning (SFT) and Group Relative Policy Optimization (GRPO) on Qwen3-32B. The core insight is that 5G troubleshooting follows well-defined engineering rules - rather than relying on a general-purpose LLM to reason from raw data, we encode expert domain knowledge into structured reasoning traces, then train the LLM to reproduce and generalize that reasoning.

2. Reasoning Trace Generation

Question Taxonomy

The 863 Phase 2 questions are classified into three types: - **Type A** (681 questions): Cell parameter analysis - given drive test metrics, identify the root cause among 8 categories (C1-C8: excessive downtilt, overshooting, neighboring cell interference, co-frequency interference, frequent handovers, PCI collision, high speed, low resource blocks). - **Type B** (100 questions): Drive test root cause analysis - given throughput drop patterns, identify causes among 9 categories (overlap, inter-freq HO threshold, transport anomaly, missing neighbor, weak coverage, PDCCH congestion, ping-pong handover, intra-freq threshold). - **Generic** (82 questions): General math and history/reading comprehension.

Rule System Design

For Type A, we built a 5-tier classification system calibrated against the training set ground truth:

1. **Tier 1** (100% accuracy on train): Speed > 40 km/h (C7), distance > 1 km (C2), handovers >= 3 (C5), avg RBs < 170 (C8).
2. **Tier 2**: C1 (downtilt) detection via RSRP, neighbor count, and tilt thresholds.
3. **Tier 3**: C4 (interference) detection with ratio-based false positive filtering.
4. **Tier 4**: PCI collision (C6) detection using a `pci_collision_ratio` metric - the fraction of drive test rows with PCI mod 30 collision. This metric is a near-perfect separator (all true C6 have ratio=1.0).
5. **Tier 5**: C1 vs C3 tiebreaker using tilt, RSRP, SINR gates, and rescue rules for ambiguous cases.

For Type B, we built an 8-rule cascade parsing 4 data tables (drive test, parameter, configuration, signaling events) and computing PHY-layer health metrics (MCS, SINR, BLER during low-throughput rows). The Type B rules are embedded in the inference system prompt so the model can apply them at inference time.

Trace Generation

Traces are generated exclusively from `train.csv` (2,400 labeled Type A questions) using `generate_traces_final.py`. Each trace is a structured chain-of-thought showing the rule evaluation with actual metric values. For example:

> "Step 1: max_speed = 12.3 km/h (< 40, not C7). Step 2: max_distance = 0.45 km (< 1.0, not C2)…"

continuing through each tier until a match is found.

The rule system achieves 98.3% accuracy on the training set (2,359/2,400 correct). For the 41 cases where it disagrees with the ground truth, an "expert correction" branch is appended to the trace, showing additional domain reasoning that overrides the initial prediction. This teaches the model to recognize edge cases where the standard path is insufficient.

The table parser uses header-based column mapping (`telco_utils.py`) rather than hardcoded positional indices, making it robust to column reordering across different question formats.

3. SFT Training

We fine-tuned Qwen3-32B (4-bit quantized via bitsandbytes NF4) using Unsloth's QLoRA implementation: - **LoRA config**: rank=64, alpha=128, targeting all attention + MLP projections (q/k/v/o/gate/up/down_proj) - **Training**: 3 epochs, batch size 4, gradient accumulation 8 (effective batch 32), learning rate 2e-4 with linear schedule - **Context**: 8192 max sequence length, 4096 max completion tokens - **Output format**: `<think>reasoning</think>\boxed{ANSWER}` - **Training data**: 2,400 Type A reasoning traces generated from the labeled training set

The SFT model learns the structured reasoning patterns from the traces. At inference time, detailed system prompts provide the Type B and Generic rules, allowing the model to generalize its reasoning capability to unseen question types.

4. GRPO Training

After SFT, we apply GRPO (Group Relative Policy Optimization) to improve answer accuracy through reinforcement learning. GRPO generates multiple completions per prompt, scores them with reward functions, and updates the policy to favor higher-reward outputs.

Configuration: 100 training steps, 8 generations per prompt, learning rate 5e-6, temperature 1.0, gradient accumulation 4, vLLM for fast inference generation. Sampling parameters: top_k=50, min_p=0.1, top_p=0.95, repetition_penalty=1.05.

Three reward functions guide learning: 1. **Boxed format** (+0.5/-0.5): Ensures the model outputs `\boxed{LABEL}` - a lightweight format check. 2. **Think tags with content** (+1.0/-0.5/-1.0): Requires `<think>...</think>` tags containing at least 200 characters of reasoning. This prevents mode collapse into degenerate short outputs. 3. **Answer accuracy** (+5.0/+3.0/-2.0/-3.0): The dominant reward signal. +5 for exact match, +3 for normalized match (e.g., "C1" matches "1"), -2 for wrong answer, -3 for no answer extracted.

The reward magnitudes are deliberately asymmetric: accuracy reward (up to +5) dominates format rewards (up to +1.5 combined), ensuring the model optimizes for correctness rather than format compliance.

Prompt format consistency: A critical design requirement is that the GRPO training prompt format exactly matches SFT training. Both stages present the model with pre-computed metrics (not raw data tables). The GRPO dataset preparation pipeline computes metrics from raw question text using the same functions as SFT (`compute_all_metrics`), strips raw tables, and formats the user message as `## Pre-computed Metrics\n\n{metrics}\n\n## Question\n\n{preamble}`. This ensures the model sees identical input structure during SFT, GRPO, and inference.

Training data: The 2,400 Type A traces use ground truth labels from the competition training set. GRPO reinforces the correct labels through the accuracy reward, tightening the model's adherence to the reasoning patterns learned during SFT.

5. Inference

Inference uses vLLM with the merged 16-bit GRPO model (`inference_grpo_final.py`). For each question, we generate 4 completions at temperature 0.6 and apply plurality voting: unanimous (4-0) and majority (3-1) cases consolidate to the winning answer, 2-1-1 splits use the plurality winner, and 2-2 ties preserve both raw answers across the 4 submission rows. Sampling parameters match GRPO training: top_k=50, min_p=0.1, top_p=0.95, repetition_penalty=1.05.

Question type detection uses prefix matching on the raw question text: "Analyze the 5G wireless network" maps to Type A, "Based on the following drive test" maps to Type B, and "Analyze the following question" maps to Generic. Each type receives its corresponding system prompt and pre-computed metrics, matching the format seen during training. The system prompt embeds the full rule set, and pre-computed metrics are injected into each user prompt so the model has all necessary data without parsing raw tables at inference time.

Answer remapping: The model was trained on Type A questions where options always use the C-prefix (C1-C8). At inference on Phase 2 test, option prefixes are shuffled per question (e.g., E1-E8, Z1-Z8, bare 1-8). The inference script automatically remaps model outputs to valid option labels: if the model outputs a bare digit (e.g., “3”) and options are letter-prefixed, it adds the correct prefix (e.g., “E3”); if it outputs a canonical C-label that is not a valid option, it maps to the corresponding shuffled label.

For Type B and Generic questions - which the model did not see during SFT/GRPO training - the detailed system prompts provide complete rules. The model’s structured reasoning capability, learned from Type A traces, generalizes well to these unseen types when guided by domain-specific system prompts.

This pipeline achieves a **leaderboard score of 0.9582** on the Phase 2 test set (863 questions).

6. Key Technical Decisions

- **Pre-computed metrics over raw parsing**: Rather than asking the LLM to parse tabular data (error-prone), we compute all metrics programmatically and inject them into the prompt. The LLM’s job is reasoning over numbers, not data extraction.
- **Header-based column mapping**: The table parser (`telco_utils.py`) maps header names to column indices rather than using hardcoded positional indices. This makes parsing robust to column reordering across different question formats, with fallback to positional indices if header detection fails.
- **Calibrated thresholds over option text**: In some cases, the correct classification threshold differs from what the question text implies (e.g., C8 threshold is 170 RBs, not the 160 mentioned in options). Thresholds were calibrated against the training set ground truth.
- **Rescue rules for ambiguous cases**: The C1/C3 tiebreaker has a “gray zone” (RSRP between -90 and -82 dB) where questions have low classification confidence. Instead of deferring to the LLM, we designed 3 rescue rules based on collision ratio, neighbor count, and interference level.
- **Train-only trace generation**: All reasoning traces are generated from the labeled training set (2,400 questions). No test set data was used for trace generation or threshold calibration.

7. Responsible AI Considerations

Data Privacy and Compliance

All training data comes exclusively from the competition dataset provided by Zindi. No external personal data, user information, or proprietary telecom subscriber data was used. The 5G network metrics in the dataset are synthetic/anonymized cell-level measurements (RSRP, SINR, throughput, handover counts) that contain no personally identifiable information. Our model processes only aggregated network performance metrics and cannot identify individual users or devices.

Model Security Risks

The model is specialized for 5G root cause classification - a narrow, well-defined task. It outputs only structured labels (C1-C8, A-I, or numeric answers) wrapped in a fixed format. The system prompt constrains outputs to the rule framework, limiting the attack surface for prompt injection. However, as with any LLM, adversarial inputs could potentially cause misclassification. In a production deployment, outputs should be validated against physically plausible ranges before acting on them.

Data and Model Access Control

The SFT and GRPO models are hosted on Hugging Face Hub (Phaedrus33/SFT_final_submission, Phaedrus33/GRPO_final_submission). The training code and classification pipeline are version-controlled on GitHub. No API keys, tokens, or credentials are stored in the codebase - authentication uses environment variables. The base model (Qwen3-32B) is open-source under Apache 2.0.

Edge Computing Considerations

The solution uses a 32B parameter model, which requires substantial GPU resources (1x A100/H200 80-140GB). For edge deployment in real telecom infrastructure, the rule-based classifier in `telco_utils.py` can run without any GPU - it achieves 98%+ accuracy on Type A questions using pure Python with no ML inference. This hybrid approach allows edge nodes to handle classification locally, with only ambiguous cases routed to a centralized LLM endpoint.

Data Governance

The training pipeline is fully reproducible: `run_final.sh --all` regenerates traces from the competition training CSV, trains SFT, trains GRPO, and produces the submission. Traces are generated exclusively from `train.csv` using `generate_traces_final.py`. No test set data is used in trace generation or model training. No data augmentation from external sources was performed.

8. Hardware and Reproducibility

- **SFT**: 1x NVIDIA A100 80GB, QLoRA 4-bit, Unsloth
- **GRPO**: 1x NVIDIA H200 NVL 141GB, vLLM 0.12.0 for generation
- **Inference**: 1x NVIDIA H200 NVL 141GB, vLLM 0.12.0, bfloat16, CUDA graphs enabled, batch size 32

Full pipeline: `bash run_final.sh --all` (requires HF_TOKEN environment variable).

Running Inference on a New Dataset

To run inference on a new test CSV (e.g., Phase 3) using the pre-trained GRPO model from HuggingFace Hub:

```bash
pip install -r requirements.txt
export HF_TOKEN=your_token
bash run_final.sh --infer --test-csv path/to/new_test.csv
```

Or directly via Python:

```bash
python inference_grpo_final.py \
    --model Phaedrus33/GRPO_final_submission \
    --test-csv path/to/new_test.csv \
    --output-dir outputs/inference \
    --num-gpus 1
```

The test CSV requires two columns: `ID` (unique question identifier) and `question` (full question text including data tables). Output submission CSVs are written to the output directory.