1. GRPO Demo - Gemma3 1B (Baseline)

File: grpo-demo-gemma3-1b.ipynb

Model Setup & Architecture

- **Model**: Gemma 3 1B-IT
- **Method**: Group Relative Policy Optimization (GRPO)
- **LoRA Configuration**: Rank=64, Alpha=64.0
- **Framework**: JAX/Tunix/Flax, TPU v5e-8
- **Sharding**: FSDP + Tensor Parallelism across 4 devices `[(1, 4), ("fsdp", "tp")]`

Training Pipeline

- **Dataset**: GSM8K (grade school math) via HuggingFace
- **Data Format**: Structured XML-style tags `<reasoning>`, `</reasoning>`, `<answer>`, `</answer>`
- **Preprocessing**: Extracts answers from `####` delimiter format
- **Batch Size**: 4 (micro-batch)
- **Total Batches**: 3,738
- **Epochs**: 1

Reward Function & RL Components

Multi-Component Reward System (4 reward functions):

1. `match_format_exactly`: +3.0 points for perfect XML structure
2. `match_format_approximately`: ±0.5 per tag (penalizes duplicates)
3. `check_answer`:
    - +3.0 for exact match
    - +1.5 for whitespace-normalized match
    - +0.5/+0.25 for answers within 10%/20% ratio
    - -1.0 penalty for wrong answers
4. `check_numbers`: +1.5 for extracted numerical correctness

GRPO Hyperparameters:

- Generations per prompt (G): 4
- KL penalty (β): 0.08
- PPO-style clipping (ε): 0.2
- Iterations per batch (μ): 1

Configurations

- **Optimizer**: AdamW (LR=3e-6, β1=0.9, β2=0.99)
- **LR Schedule**: 10% warmup → Cosine decay to 0
- **Gradient Clipping**: 0.1 (critical for KL stability)

- **Weight Decay**: 0.1
- **Generation**: Temperature=0.9, top_k=50, top_p=1.0 (high diversity during training)
- **Checkpoint**: Every 500 steps, keep 4 latest

What's Unique

- **Comprehensive reward shaping** with tolerance for partial correctness
- **Format enforcement** through dual exact/approximate matching
- **Ratio-based scoring** (0.9-1.1 range) for near-correct numerical answers
- Standard GRPO implementation following DeepMind's algorithm

Strengths

✅ Well-documented baseline
✅ Robust multi-reward system with graceful degradation
✅ Reproducible HuggingFace dataset pipeline
✅ Memory-efficient GRPO (no separate value model)

Weaknesses

❌ No trajectory-level rewards (only token-level)
❌ High temperature (0.9) may hurt convergence
❌ Single epoch limits learning potential
❌ No advanced techniques (e.g., curriculum learning, self-consistency)

---

2. "Efficiency Build" - Gemma3 1B (Memory-Optimized)

Files: start-with-gemma3-1b-it-tutorial.ipynb, `start-with-gemma3-1b-it-tutorial (1).ipynb`

Model Setup & Architecture

- **Model**: Gemma 3 1B-IT
- **Key Difference**: **Resource-constrained adaptation for Kaggle TPU v3-8**
- **LoRA Configuration**: **Rank=32, Alpha=32.0** (halved from baseline!)
- **Batch Size**: **2** (reduced from 4)

Training Pipeline

- Identical dataset/preprocessing to baseline
- **Custom auto-login logic** for Kaggle secrets (UX improvement)
- Same XML tag structure

Reward Function & RL Components

- **Identical** 4-reward system to baseline
- Same GRPO hyperparameters (β=0.08, ε=0.2, G=4)

Configurations

Memory Optimizations:

- LoRA rank: 64 → **32** (-50% trainable params)
- Batch size: 4 → **2** (-50% memory)
- Maintains same LR/optimizer settings

What's Unique

- **Explicitly designed for OOM prevention** on lower-tier TPUs
- **Documented trade-off philosophy**: "Efficiency over capacity"
- Auto-login helper for Kaggle environment
- Claims "significant performance gains even with limited compute budgets"

Strengths

✅ **Practical for resource-constrained users**
✅ Preserves training stability with reduced capacity
✅ Better UX with auto-login logic
✅ Demonstrates LoRA compression effectiveness

Weaknesses

❌ **Lower model capacity** (rank 32 vs 64) may hurt complex reasoning
❌ Smaller batches increase gradient variance
❌ No unique algorithmic contributions
❌ Unverified claims about "performance gains" (no metrics provided)

---

3. Supervised Fine-Tuning (Full) - Gemma3 1B

File: supervised-fine-tuning-full.ipynb

Model Setup & Architecture

- **Model**: Gemma 3 1B
- **Method**: **Full fine-tuning** (NOT LoRA! All 1B parameters trainable)
- **Framework**: JAX/Tunix/Flax, TPU v5e-8
- **Sharding**: `(8, 1)` - **FSDP across all 8 TPU cores**

Training Pipeline

Dataset Cleaning (Critical differentiator):

```
def clean_gsm8k_content(text):

  # Removes GSM8K calculation artifacts like <<10+5=15>>
```

```
return text.replace("<<", "(").replace(">>", ")")
```

- **Applies cleaning BEFORE training** to reduce noise
- Uses "STRICT System Prompt" emphasizing format compliance

Data Processing:

- Max sequence length: 2048
- Loss mask: Only active for model response (not prompt)
- Batch size: 2 (micro), 2×8×4=64 (effective global batch with grad accumulation)

Reward Function & RL Components

N/A - This is supervised learning, not RL!

- Standard cross-entropy loss
- No reward functions
- Loss computed only on completion tokens (prompt is masked)

Configurations

- **Optimizer**: AdamW (LR=2e-5, β1=0.9, β2=0.999, ε=1e-8)
- **LR Schedule**: 50-step warmup → Cosine decay to 0.1×LR
- **Gradient Accumulation**: 4 steps (effective batch=64)
- **Gradient Clipping**: 1.0 (10× looser than GRPO)
- **Weight Decay**: 0.01 (10× less aggressive)
- **Epochs**: **10** (far more than GRPO's 1)
- **Max Steps**: 1,170 (117 batches × 10 epochs)

Post-Training Inference:

- **Majority Voting** with k=1 sample (self-consistency)
- Temperature=0.7 for diverse reasoning paths
- Robust answer extraction (XML → LaTeX boxed → text patterns → last number)

What's Unique

🔥 **Only full fine-tuning approach** (vs parameter-efficient LoRA)
🔥 **Data cleaning pipeline** to remove GSM8K artifacts
🔥 **Majority voting evaluation** (self-consistency)
🔥 **Multi-pattern answer extraction** (most robust)
🔥 **10 epochs** vs 1 for RL approaches

Strengths

✅ **Full model capacity** utilization
✅ **Data quality focus** (cleaning artifacts)
✅ **Proper evaluation methodology** (voting, robust extraction)
✅ **Reproducible** with clear TPU verification steps
✅ **Best for datasets with clean supervision** (GSM8K has gold reasoning traces)

Weaknesses

❌ **No parameter efficiency** (trains all 1B params)
❌ **Requires high-quality reasoning traces** in training data
❌ **Doesn't learn reward optimization** (just imitation)
❌ **10× slower** than single-epoch GRPO
❌ **Risk of overfitting** with 10 epochs on same 7.5k examples

---

4. TT-Tunix-on-TPU (Incomplete Attempt)

File: tt-tunix-on-tpu.ipynb

Model Setup & Architecture

- **Model**: Gemma2 2B-IT
- **Attempted Method**: GRPO (based on Windmaple's code)
- **LoRA Configuration**: Rank=64, Alpha=64.0

Training Pipeline

- **Data Source**: Kaggle CSV (The Devastator's GSM8K)
- Setup code for Gemma2 2B checkpoint loading

Reward Function & RL Components

- Copied baseline 4-reward system
- No modifications

Configurations

- Batch size: 2
- Standard GRPO hyperparameters

What's Unique

❌ **Incomplete submission** - Training failed at LoRA policy creation:

TypeError: set_metadata takes either 1 argument or 1 or more keyword arguments,

got args=('sharding_names', ('fsdp', None))

- Demonstrates **common pitfall**: Flax/JAX version incompatibility with Tunix

Strengths

✅ Attempted larger model (2B)
✅ Documented failure (useful for debugging)

Weaknesses

❌ **Non-functional** code
❌ No unique contributions
❌ Version mismatch issues
❌ Abandoned midway

---

5. Tunix-Hack-Winner: Trajectory Reward Training

File: tunix-hack-winner-trajectory-reward-training.ipynb

Model Setup & Architecture

- **Model**: Gemma2 2B-IT
- **Method**: GRPO with **trajectory-level rewards**
- **LoRA Configuration**: Rank=64, Alpha=64.0
- **Framework**: JAX/Tunix, TPU v4/v5

Training Pipeline

- **Dataset**: Kaggle CSV (GSM8K main_train.csv)
- **Batch Size**: 2 (safe for V4 TPUs)
- **Batches**: 5,000 (vs 3,738 in baseline)
- Standard preprocessing

Reward Function & RL Components

🔥 **COMPETITIVE ADVANTAGE**: **Trajectory Reward**

```
def trajectory_reward(prompts, completions, answer, **kwargs):
    # Reasoning quality (50% weight initially, tuned to 60%)
    reasoning = re.search(r'<reasoning>(.*?)</reasoning>', comp, re.DOTALL)
    r_score = 0.5 if reasoning and len(reasoning.group(1).split()) > 20 else 0.0

    # Answer correctness (50% weight initially, tuned to 40%)
    ans = re.search(r'<answer>(.*?)</answer>', comp, re.DOTALL)
    a_score = 1.0 if ans and ans.group(1).strip() == ref else 0.0

    # Weighted combination: 60% reasoning + 40% answer
```

reward = (0.6 * r_score + 0.4 * a_score) * 3.0

Key Innovation:

- **Holistic evaluation**: Entire completion scored as a unit
- **Reasoning length threshold**: 20+ words required
- **Weighted objectives**: 60% process, 40% outcome
- **Scaled rewards**: 3× multiplier for final score

Configurations

- **Optimizer**: AdamW (LR=5e-5, higher than baseline's 3e-6)
- **GRPO**: G=2 generations (vs 4 in baseline), β=0.08, ε=0.2
- **Rollout**: 768 max tokens (vs 512)
- **Batch Processing**: V4-safe settings

What's Unique

🏆 **Claimed winner approach**
🔥 **Trajectory-level reward** vs token-level
🔥 **Explicit reasoning quality metric** (word count threshold)
🔥 **60/40 weighting** between process and outcome
🔥 **Higher learning rate** (5e-5 vs 3e-6)
🔥 **Fewer generations** (2 vs 4) for efficiency

Strengths

✅ **Process-oriented reward design** (aligns with hackathon goals!)
✅ **Scalable to larger models** (2B vs 1B)
✅ **Efficient generation** (G=2 reduces compute)
✅ **Transparent reasoning evaluation** (simple word count)
✅ **Reward scaling** prevents vanishing signals

Weaknesses

❌ **Crude reasoning metric** (word count ≠ quality)
❌ **No citation counting** or logical step verification
❌ **Fixed threshold (20 words)** not adaptive
❌ **Higher LR** may cause instability
❌ **No reproducibility verification** (no training logs shown)

Cross-Project Comparison Table

Aspect	GRPO Baseline	Efficiency Build	SFT Full	TT-Tunix	Trajectory Winner
Model	Gemma3 1B	Gemma3 1B	Gemma3 1B	Gemma2 2B	Gemma2 2B
Method	GRPO	GRPO	Supervised	GRPO (failed)	GRPO
LoRA Rank	64	**32**	N/A (full)	64	64
Batch Size	4	**2**	2 (×8×4=64 eff.)	2	2
Epochs	1	1	**10**	-	1
Reward Type	Multi-component	Multi-component	N/A	-	**Trajectory**
Data Cleaning	❌	❌	✅	❌	❌
Reasoning Metric	Format only	Format only	N/A	-	**Word count**
Evaluation	Greedy	Greedy	**Majority vote**	-	Unknown
Reproducible	✅	✅	✅	❌	⚠️

Alignment with Hackathon Goals

Goal: "Teach model to show its work" (reasoning transparency + accuracy + reproducibility)

Rankings:

1. 🥇 **Trajectory Winner** - Explicitly rewards reasoning quality (60% weight), transparency-first
2. 🥈 **SFT Full** - Comprehensive pipeline with evaluation rigor, but relies on supervision
3. 🥉 **GRPO Baseline** - Solid foundation, format-enforcing, fully reproducible
4. **Efficiency Build** - Practical but no innovation
5. **TT-Tunix** - Non-functional

Strengths & Weaknesses Summary

Strongest Ideas:

1. **Trajectory-level rewards** (Winner) - Holistic reasoning evaluation
2. **Data cleaning** (SFT) - Removes artifacts for cleaner learning
3. **Majority voting** (SFT) - Self-consistency improves reliability
4. **Multi-reward shaping** (Baseline) - Graceful handling of partial correctness
5. **LoRA compression** (Efficiency) - Practical accessibility

Methods Likely to Fail:

1. **Word count as reasoning quality** - Easily gamed, no semantic understanding

2. **Single-epoch RL** - May underfit complex reasoning patterns
3. **High temperature (0.9)** - Reduces convergence stability
4. **No trajectory rewards in baseline** - Misses process-oriented learning

Most Reproducible:

1. **SFT Full** - Complete pipeline with TPU verification steps
2. **GRPO Baseline** - Standard Tunix tutorial format
3. **Efficiency Build** - Documented resource constraints

Easiest to Reproduce:

GRPO Baseline - Uses default Tunix APIs, HuggingFace datasets, no custom logic

Recommended Hybrid Approach

Combining the best ideas:

```
# Model: Gemma3 1B (SFT-pretrained)

# Phase 1: SFT with cleaned data (3 epochs)

# Phase 2: GRPO with hybrid rewards


def hybrid_reward(prompts, completions, answer, **kwargs):
    rewards = []
    for comp, ref in zip(completions, answer):
        # 1. Format enforcement (Baseline idea)
        format_score = 3.0 if match_format.search(comp) else 0.0

        # 2. Reasoning depth (Winner idea, improved)
        reasoning = re.search(r'<reasoning>(.*?)</reasoning>', comp, re.DOTALL)
        if reasoning:
            steps = len(re.findall(r'\n\d+\.|\n-', reasoning.group(1)))  # Count numbered steps
            words = len(reasoning.group(1).split())
```

```
            depth_score = min(steps * 0.3 + (words > 50) * 0.5, 2.0)  # Cap at 2.0
        else:
            depth_score = 0.0

        # 3. Answer accuracy (Baseline idea)
        ans = re.search(r'<answer>(.*?)</answer>', comp, re.DOTALL)
        if ans:
            try:
                acc_score = 3.0 if float(ans.group(1)) == float(ref) else 0.0
            except:
                acc_score = 0.0
        else:
            acc_score = 0.0

        # Weighted: 30% format + 40% reasoning + 30% answer
        total = 0.3*format_score + 0.4*depth_score + 0.3*acc_score
        rewards.append(total)
    return rewards

# Config: LoRA rank=64, batch=4, LR=3e-6, epochs=2
# Evaluation: Majority voting (k=5)
```

Why this works:

- SFT initializes reasoning structure
- GRPO refines quality with process-oriented rewards
- Step counting > word counting for depth
- Balanced weights across format/process/outcome
- Self-consistency for robust evaluation

Final Recommendation: The **Trajectory Winner** has the right philosophy (process over outcome), but implementation needs refinement. The **SFT Full** approach provides the most complete pipeline for reproducibility. A hybrid SFT→GRPO approach combining both would likely dominate.