Experimental Design for Mobile-VideoGPT QVED Fine-tuning

Overview

This document outlines the experimental plan to evaluate 4 vision-language models on the QVED physiotherapy dataset:

- **Mobile-VideoGPT-0.5B** (current codebase support)
- **Mobile-VideoGPT-1.5B** (current codebase support)
- **VideoLLaMA3-2B** (requires adapter)
- **NVILA-Lite-2B** (requires adapter)

Current Codebase Capabilities

The Mobile-VideoGPT codebase currently supports:

- ▯ **LoRA fine-tuning** (configured in `scripts/finetune_qved.sh`)
- ▯ **Learning rate warmup** (0.05 ratio) and **cosine annealing** scheduler
- ▯ **Frame sampling** (NUM_FRAMES=16, configurable via environment variable)
- ▯ **Batch processing** with gradient accumulation
- ▯ **Mixed precision** (bf16, fp16, tf32)
- ▯ **Performance metrics** (throughput, latency already captured in `test_inference.py`)
- ▯ **LLM-as-Judge evaluation** (Mixtral-8x7B for quality assessment)
- ▯ **Exercise-specific metrics** (exercise ID, per-class evaluation)
- ⚠ **Full fine-tuning** (supported but not configured - requires disabling LoRA)
- ▯ **INT8 quantization** (not currently implemented)
- ▯ **Memory profiling** (not systematically captured)
- ▯ **Visual degradation testing** (no synthetic perturbation pipeline)

Experiment Organization

```
experiments/
├── README.md                           # Experiment overview and
instructions
├── shared/
│   ├── metrics.py                      # Common evaluation metrics
│   ├── data_utils.py                   # Dataset manipulation utilities
│   ├── visualization.py                # Plotting and result visualization
│   └── config.py                       # Shared configuration and
constants
├── rq1_training_efficiency/
│   ├── rq1_1_1_full_vs_lora.sh         # RQ1.1.1: Full vs LoRA fine-tuning
│   ├── rq1_1_2_lr_schedules.sh         # RQ1.1.2: Learning rate schedules
│   ├── rq1_1_3_data_efficiency.sh      # RQ1.1.3: Dataset size sensitivity
│   ├── analyze_parameters.py           # Count trainable parameters
│   ├── analyze_convergence.py          # Convergence speed analysis
│   └── analyze_data_efficiency.py      # Data efficiency curves
├── rq1_temporal_modeling/
│   ├── rq1_2_1_sequence_length.sh      # RQ1.2.1: Temporal pattern capture
│   ├── rq1_2_3_frame_sampling.sh       # RQ1.2.3: Frame sampling
degradation
│   ├── prepare_frame_variants.py       # Generate videos with different
frame counts
│   └── analyze_temporal.py             # Temporal reasoning analysis
├── rq1_robustness/
│   ├── rq1_3_3_visual_degradation.sh   # RQ1.3.3: Visual robustness
│   ├── generate_degradations.py        # Synthetic degradation pipeline
│   └── analyze_robustness.py           # Robustness curves
├── rq2_error_localization/
│   ├── rq2_1_1_error_detection.sh      # RQ2.1.1: Form error detection
│   ├── annotate_errors.py              # Create annotated test set
│   └── analyze_localization.py         # Error detection accuracy
├── rq3_efficiency/
│   ├── rq3_1_1_baseline_profile.sh     # RQ3.1.1: Baseline latency/memory
│   ├── rq3_1_2_batch_scaling.sh        # RQ3.1.2: Batch size effects
│   ├── rq3_1_3_resolution_sweep.sh     # RQ3.1.3: Resolution/sequence
sweep
│   ├── profile_memory.py               # Memory profiling utility
│   └── generate_pareto.py              # Pareto frontier visualization
├── rq3_compression/
│   ├── rq3_2_1_accuracy_headroom.sh    # RQ3.2.1: Pre-quantization margin
│   ├── rq3_2_2_precision_sweep.sh      # RQ3.2.2: Precision reduction
│   ├── quantize_model.py               # Quantization utilities
│   └── analyze_compression.py          # Compression resilience analysis
├── rq4_failure_analysis/
│   ├── rq4_1_2_confusion_matrix.sh     # RQ4.1.2: Exercise-specific
failures (COMBINED)
│   ├── rq4_1_3_class_imbalance.sh      # RQ4.1.3: Data distribution
effects (COMBINED)
│   ├── create_balanced_split.py        # Generate balanced dataset
│   └── analyze_failures.py             # Confusion matrix and failure
modes
├── rq5_clinical_guidance/
│   ├── rq5_2_1_feedback_quality.sh     # RQ5.2.1: Corrective guidance
quality
│   ├── evaluate_feedback.py            # LLM-as-judge for feedback quality
│   └── analyze_guidance.py             # Clinical accuracy analysis
└── results/
    ├── rq1_training_efficiency/
    ├── rq1_temporal_modeling/
    ├── rq1_robustness/
    ├── rq2_error_localization/
    ├── rq3_efficiency/
    ├── rq3_compression/
    ├── rq4_failure_analysis/
    └── rq5_clinical_guidance/
```

Research Questions and Implementation Plan

RQ1: Training Efficiency

RQ1.1.1: Full Model Fine-tuning vs. LoRA

Question: How do full fine-tuning and LoRA compare in terms of trainable parameters and accuracy?

Current Support: ✅ LoRA implemented, ⚠ Full fine-tuning needs configuration

Implementation:

```
# experiments/rq1_training_efficiency/rq1_1_1_full_vs_lora.sh
```

Approach:

1. **LoRA training** (current setup):
    - Use existing `scripts/finetune_qved.sh` with `--lora_enable True`
    - Vary LoRA rank: r={8, 16, 32, 64, 128}
    - Keep alpha=2*r (standard practice)
2. **Full fine-tuning**:
    - Modify training script: `--lora_enable False`
    - May need smaller batch size due to memory constraints
    - Train only final layers (e.g., last 2-4 transformer blocks) for memory efficiency
3. **Metrics to collect:**
    - Total parameters vs. trainable parameters (use `analyze_parameters.py`)
    - Final accuracy (BERT, METEOR, ROUGE-L, LLM Accuracy)
    - Training time per epoch
    - Memory usage during training
    - Model size on disk

Expected Output:

- Table: Model size vs. trainable parameters vs. accuracy
- Plot: Accuracy vs. trainable parameters (LoRA rank sweep)
- Recommendation: Optimal LoRA rank for acceptable accuracy

How to run:

```
# Calls scripts/finetune_qved.sh with different LoRA configs
# Then calls utils/test_inference.py and utils/generate_test_report.py
for LORA_R in 8 16 32 64 128; do
    bash experiments/rq1_training_efficiency/rq1_1_1_full_vs_lora.sh \
        --model Mobile-VideoGPT-0.5B \
        --lora_r $LORA_R \
        --output_dir experiments/results/rq1_1_1/lora_r${LORA_R}
done

# Full fine-tuning
bash experiments/rq1_training_efficiency/rq1_1_1_full_vs_lora.sh \
    --model Mobile-VideoGPT-0.5B \
    --full_finetune \
    --output_dir experiments/results/rq1_1_1/full_finetune
```

RQ1.1.2: Learning Rate Schedules

Question: How do different LR schedules affect convergence speed and final accuracy?

Current Support: □ Warmup (0.05) + cosine annealing already implemented

Implementation:

```
# experiments/rq1_training_efficiency/rq1_1_2_lr_schedules.sh
```

Approach:

1. **Fixed LR** (no warmup, constant):
   - Set `--warmup_ratio 0.0` and `--lr_scheduler_type constant`
2. **Warmup only** (linear warmup to target LR, then constant):
   - Set `--warmup_ratio 0.1` and `--lr_scheduler_type constant_with_warmup`
3. **Cosine annealing** (current setup):
   - Keep `--warmup_ratio 0.05` and `--lr_scheduler_type cosine`
4. **Cosine with warmup**:
   - Vary warmup ratio: {0.03, 0.05, 0.1, 0.2}

Metrics to collect:

- Convergence speed: steps/epochs to reach 90%, 95%, 99% of final accuracy
- Final accuracy on all metrics
- Training loss curve
- Validation loss curve
- LR schedule visualization

Expected Output:

- Plot: Training/validation curves for each schedule
- Plot: Convergence speed comparison (steps to target accuracy)
- Table: Final accuracy per schedule

- Recommendation: Optimal schedule for fast convergence

How to run:

```
# Test different schedules
for SCHEDULE in "constant" "constant_with_warmup" "cosine" "linear"; do
    for WARMUP in 0.0 0.03 0.05 0.1; do
        bash experiments/rq1_training_efficiency/rq1_1_2_lr_schedules.sh \
            --model Mobile-VideoGPT-0.5B \
            --lr_scheduler $SCHEDULE \
            --warmup_ratio $WARMUP \
            --output_dir
experiments/results/rq1_1_2/${SCHEDULE}_warmup${WARMUP}
    done
done

# Analyze convergence
python experiments/rq1_training_efficiency/analyze_convergence.py \
    --results_dir experiments/results/rq1_1_2 \
    --output convergence_analysis.pdf
```

RQ1.1.3: Dataset Size Sensitivity

Question: How much training data is needed for acceptable accuracy?

Current Support: □ Can subset dataset, existing scripts work with JSON

Implementation:

```
# experiments/rq1_training_efficiency/rq1_1_3_data_efficiency.sh
```

Approach:

1. **Create stratified subsets:**
   - Use `experiments/shared/data_utils.py` to create balanced subsets
   - Sizes: 25%, 50%, 75%, 100% of `qved_train.json`
   - Maintain class distribution (equal samples per exercise)
2. **Train on each subset:**
   - Keep all other hyperparameters constant
   - Use same validation set for fair comparison
3. **Metrics to collect:**
   - Accuracy vs. dataset size (all 5 quality metrics)
   - Per-exercise accuracy at each size

    - Data efficiency curve: accuracy gain per 100 samples

Expected Output:

- Plot: Accuracy vs. dataset size (learning curves)
- Plot: Per-exercise accuracy at each size
- Table: Data efficiency (samples needed for 90%, 95% of max accuracy)
- Recommendation: Minimum dataset size per model

How to run:

```
# Generate subsets
python experiments/shared/data_utils.py \
    --action create_subsets \
    --input dataset/qved_train.json \
    --output_dir experiments/data/subsets \
    --ratios 0.25 0.5 0.75 1.0

# Train on each subset
for RATIO in 0.25 0.5 0.75 1.0; do
    bash experiments/rq1_training_efficiency/rq1_1_3_data_efficiency.sh \
        --model Mobile-VideoGPT-0.5B \
        --train_data experiments/data/subsets/qved_train_${RATIO}.json \
        --output_dir experiments/results/rq1_1_3/data_${RATIO}
done

# Analyze data efficiency
python experiments/rq1_training_efficiency/analyze_data_efficiency.py \
    --results_dir experiments/results/rq1_1_3 \
    --output data_efficiency_curves.pdf
```

RQ1.2: Temporal Modeling (COMBINED)

RQ1.2.1 + RQ1.2.3: Temporal Pattern Capture and Frame Sampling

Questions:

- RQ1.2.1: How effectively does each model capture temporal patterns?
- RQ1.2.3: How does temporal reasoning degrade with frame sampling rate?

Current Support: □ NUM_FRAMES configurable via environment variable

Implementation:

```
# experiments/rq1_temporal_modeling/rq1_2_1_sequence_length.sh  (COMBINED
SCRIPT)
```

Approach - COMBINED:

1. **Vary frame count** (RQ1.2.1):
   - Test: NUM_FRAMES = {2, 4, 8, 16, 32}
   - Keep FPS constant (sample at regular intervals)
2. **Vary FPS/sampling rate** (RQ1.2.3):
   - Original FPS (30fps): sample every 1 frame
   - Half FPS (15fps): sample every 2 frames
   - Quarter FPS (7.5fps): sample every 4 frames
   - Keep NUM_FRAMES constant (e.g., 16 frames)
3. **Generate test variants:**
   - Use `prepare_frame_variants.py` to create resampled videos
   - Store in `experiments/data/temporal_variants/`
4. **Run inference only** (no retraining):
   - Use existing finetuned checkpoint
   - Test on all variants

Metrics to collect:

- Accuracy vs. frame count (RQ1.2.1)
- Accuracy vs. FPS (RQ1.2.3)
- Degradation curves for both dimensions
- Per-exercise sensitivity to temporal resolution

Expected Output:

- Plot: Accuracy vs. sequence length (2D: all models)
- Plot: Accuracy vs. FPS degradation curve
- Heatmap: Accuracy vs. (frame_count, FPS)
- Table: Minimum frames/FPS needed for acceptable accuracy
- Recommendation: Optimal temporal configuration per model

How to run:

```
# Generate temporal variants
python experiments/rq1_temporal_modeling/prepare_frame_variants.py \
    --input dataset/qved_test.json \
    --output_dir experiments/data/temporal_variants \
    --frame_counts 2 4 8 16 32 \
    --fps_ratios 1.0 0.5 0.25

# Test each variant (no training, just inference)
for FRAMES in 2 4 8 16 32; do
    for FPS_RATIO in 1.0 0.5 0.25; do
        export NUM_FRAMES=$FRAMES
        bash scripts/run_inference.sh \
            --model_path results/checkpoint-70 \
            --test_json
experiments/data/temporal_variants/fps${FPS_RATIO}_frames${FRAMES}.json \
            --output_dir
experiments/results/rq1_2/frames${FRAMES}_fps${FPS_RATIO}
    done
done

# Analyze temporal sensitivity
python experiments/rq1_temporal_modeling/analyze_temporal.py \
    --results_dir experiments/results/rq1_2 \
    --output temporal_sensitivity.pdf
```

RQ1.3: Robustness

RQ1.3.3: Visual Degradation Robustness

Question: How robust is each model to visual degradation?

Current Support: □ No synthetic degradation pipeline (needs implementation)

Implementation:

```
# experiments/rq1_robustness/rq1_3_3_visual_degradation.sh
```

Approach:

1. **Generate degraded test videos:**
    - **Motion blur:** Simulate camera shake (OpenCV: cv2.GaussianBlur with motion kernel)
    - **Compression artifacts:** Re-encode at lower bitrates (ffmpeg: crf=35, 40, 45)
    - **Lighting variations:** Adjust brightness/contrast (±20%, ±40%, ±60%)
    - **Occlusion:** Add synthetic black/random boxes (10%, 20%, 30% of frame area)
2. **Create degradation pipeline:**
    - Use `generate_degradations.py` to apply transformations
    - Store in `experiments/data/degraded/`
3. **Run inference** (no retraining):
    - Test on original + all degraded variants
    - Use existing finetuned checkpoint

Metrics to collect:

- Accuracy vs. degradation level (per type)
- Robustness score: (degraded_accuracy / clean_accuracy)
- Per-exercise robustness (some exercises may be more sensitive)

Expected Output:

- Plot: Accuracy degradation curves (4 types x 3-4 levels)
- Table: Robustness scores per model per degradation
- Heatmap: Model × Degradation type × Level
- Recommendation: Most robust model

How to run:

```
# Generate degraded videos
python experiments/rq1_robustness/generate_degradations.py \
    --input dataset/qved_test.json \
    --output_dir experiments/data/degraded \
    --degradations motion_blur compression lighting occlusion \
    --levels 0.1 0.2 0.3

# Test on degraded videos
for DEG in motion_blur compression lighting occlusion; do
    for LEVEL in 0.1 0.2 0.3; do
        bash scripts/run_inference.sh \
            --model_path results/checkpoint-70 \
            --test_json experiments/data/degraded/${DEG}_${LEVEL}.json \
            --output_dir experiments/results/rq1_3_3/${DEG}_${LEVEL}
    done
done

# Analyze robustness
python experiments/rq1_robustness/analyze_robustness.py \
    --results_dir experiments/results/rq1_3_3 \
    --baseline_dir results/6 \
    --output robustness_curves.pdf
```

RQ2: Error Localization

RQ2.1.1: Form Error Detection and Localization

Question: Can each model identify and localize specific form errors?

Current Support: ⚠ Requires annotated test set with error labels

Implementation:

```
# experiments/rq2_error_localization/rq2_1_1_error_detection.sh
```

Approach:

1. **Create annotated test set:**

- Manually label test videos with specific errors:
  - Knee valgus (knees caving in)
  - Excessive forward lean
  - Incomplete range of motion
  - Poor alignment
- Store in `experiments/data/annotated_errors.json`
- Format:

```
{
  "video": "squats/00123.mp4",
  "exercise": "squats",
  "errors": ["knee_valgus", "forward_lean"],
  "error_frames": [12, 15, 18],
  "ground_truth": "Keep knees aligned..."
}
```

2. **Run inference:**
   - Use existing checkpoint
   - Check if model output mentions specific errors

3. **Error detection metrics:**
   - **Detection accuracy:** Does model mention the error type?
   - **Precision/Recall:** True positives / (TP + FP)
   - **Spatial accuracy:** Does model identify correct frame/timestamp? (if annotations include)

Metrics to collect:

- Error detection accuracy (per error type)
- Precision, recall, F1 per error type
- Confusion matrix (predicted vs. actual errors)
- Spatial localization accuracy (if temporal annotations available)

Expected Output:

- Table: Detection accuracy per error type per model
- Confusion matrix: Predicted errors vs. actual errors
- Plot: Precision-recall curves
- Recommendation: Best model for error detection

How to run:

```
# Annotate errors (manual or semi-automated)
python experiments/rq2_error_localization/annotate_errors.py \
    --input dataset/qved_test.json \
    --output experiments/data/annotated_errors.json \
    --interactive
```

```
# Run inference on annotated set
bash scripts/run_inference.sh \
    --model_path results/checkpoint-70 \
    --test_json experiments/data/annotated_errors.json \
    --output_dir experiments/results/rq2_1_1

# Analyze error detection
python experiments/rq2_error_localization/analyze_localization.py \
    --predictions experiments/results/rq2_1_1/test_predictions.json \
    --annotations experiments/data/annotated_errors.json \
    --output error_detection_analysis.pdf
```

RQ3: Computational Efficiency

RQ3.1.1: Baseline Latency and Memory Footprint

Question: What is the baseline inference latency and memory usage?

Current Support: ▯ Throughput (tok/s) already captured, ⚠ Memory profiling needs implementation

Implementation:

```
# experiments/rq3_efficiency/rq3_1_1_baseline_profile.sh
```

Approach:

1. **Add memory profiling:**
    - Use `experiments/rq3_efficiency/profile_memory.py`
    - Wrap inference with memory tracking:

    ```
    import torch
    torch.cuda.reset_peak_memory_stats()
    # Run inference
    peak_memory = torch.cuda.max_memory_allocated() / 1e9  # GB
    ```

2. **Measure latency components:**
    - Video preprocessing time
    - Model forward pass time
    - Token generation time
    - Total end-to-end time

3. **Run profiling:**
    - Single sample (batch_size=1)
    - Measure on 100 samples for statistical stability

Metrics to collect:

- **Latency:** preprocessing, forward, generation, total (mean, std, p50, p95, p99)
- **Memory:** peak memory, model size, activation memory
- **Throughput:** tokens/sec, frames/sec
- **Compute bottleneck:** Which stage takes longest?

Expected Output:

- Table: Latency breakdown per model
- Table: Memory footprint per model
- Plot: Latency distribution (violin plot)
- Plot: Memory profile over time
- Recommendation: Fastest/most memory-efficient model

How to run:

```
# Profile all 4 models
for MODEL in Mobile-VideoGPT-0.5B Mobile-VideoGPT-1.5B; do
    bash experiments/rq3_efficiency/rq3_1_1_baseline_profile.sh \
        --model $MODEL \
        --test_json dataset/qved_test.json \
        --output_dir experiments/results/rq3_1_1/$MODEL \
        --profile_memory
done

# Summarize results
python experiments/shared/visualization.py \
    --action summarize_profiling \
    --results_dir experiments/results/rq3_1_1 \
    --output baseline_profiling.pdf
```

RQ3.1.2: Batch Size Effects

Question: How does batch size affect latency and memory?

Current Support: □ Batch processing supported in inference

Implementation:

```
# experiments/rq3_efficiency/rq3_1_2_batch_scaling.sh
```

Approach:

1. **Vary batch size:** 1, 2, 4, 8, 16

2. **Measure for each batch size:**

    - Total latency (time for full test set)
    - Latency per sample (total / num_samples)
    - Throughput (samples/sec)
    - Peak memory usage

3. **Identify scaling regime:**
    - Memory-constrained: Latency increases sharply above certain batch size
    - Compute-constrained: Linear latency scaling with batch size

Metrics to collect:

- Latency vs. batch size
- Memory vs. batch size
- Throughput vs. batch size
- Efficiency (throughput / memory) vs. batch size

Expected Output:

- Plot: Latency vs. batch size (per model)
- Plot: Memory vs. batch size
- Plot: Throughput vs. batch size
- Table: Optimal batch size per model (max throughput before memory limit)
- Classification: Memory-bound vs. compute-bound models

How to run:

```
# Test batch scaling
for MODEL in Mobile-VideoGPT-0.5B Mobile-VideoGPT-1.5B; do
    for BATCH in 1 2 4 8 16; do
        bash experiments/rq3_efficiency/rq3_1_2_batch_scaling.sh \
            --model $MODEL \
            --batch_size $BATCH \
            --test_json dataset/qved_test.json \
            --output_dir experiments/results/rq3_1_2/${MODEL}_batch${BATCH}
    done
done

# Analyze scaling
python experiments/rq3_efficiency/analyze_batch_scaling.py \
    --results_dir experiments/results/rq3_1_2 \
    --output batch_scaling_analysis.pdf
```

RQ3.1.3: Resolution and Sequence Length Impact (COMBINED)

Question: How do input resolution and sequence length jointly impact computational cost?

Current Support: ✅ NUM_FRAMES configurable, ⚠ Resolution not parameterized (may need model code changes)

Implementation:

```
# experiments/rq3_efficiency/rq3_1_3_resolution_sweep.sh
```

Approach:

1. **Create input variants:**
   - Resolutions: 224×224, 336×336, 448×448, 560×560 (multiples of 112 for CLIP)
   - Sequence lengths: 4, 8, 16, 32 frames
   - Full factorial: 4 resolutions × 4 lengths = 16 combinations
2. **Measure for each variant:**
   - Latency
   - Memory
   - Accuracy (quality metrics)
3. **Build 2D heatmaps:**
   - Latency(resolution, seq_length)
   - Memory(resolution, seq_length)
   - Accuracy(resolution, seq_length)
4. **Pareto frontier:**
   - Find optimal (resolution, seq_length) pairs
   - Maximize accuracy / (latency × memory)

Metrics to collect:

- Latency vs. (resolution, sequence_length)
- Memory vs. (resolution, sequence_length)
- Accuracy vs. (resolution, sequence_length)
- Pareto-optimal configurations

Expected Output:

- Heatmap: Latency vs. (resolution, seq_length)
- Heatmap: Memory vs. (resolution, seq_length)
- Heatmap: Accuracy vs. (resolution, seq_length)
- Plot: Pareto frontiers (accuracy vs. cost)
- Table: Recommended configs for different scenarios (low-latency, high-accuracy, balanced)

How to run:

```
# Generate resolution variants
python experiments/rq3_efficiency/prepare_resolution_variants.py \
    --input dataset/qved_test.json \
    --output_dir experiments/data/resolution_variants \
    --resolutions 224 336 448 560

# Run sweep
for MODEL in Mobile-VideoGPT-0.5B Mobile-VideoGPT-1.5B; do
    for RES in 224 336 448 560; do
        for FRAMES in 4 8 16 32; do
            export NUM_FRAMES=$FRAMES
            bash experiments/rq3_efficiency/rq3_1_3_resolution_sweep.sh \
                --model $MODEL \
                --resolution $RES \
                --frames $FRAMES \
                --test_json
experiments/data/resolution_variants/res${RES}.json \
                --output_dir
experiments/results/rq3_1_3/${MODEL}_res${RES}_f${FRAMES}
        done
    done
done

# Generate Pareto frontiers
python experiments/rq3_efficiency/generate_pareto.py \
    --results_dir experiments/results/rq3_1_3 \
    --output pareto_frontiers.pdf
```

RQ3.2: Compression Potential

RQ3.2.1: Accuracy Headroom for Quantization

Question: How much performance margin exists before quantization?

Current Support: ▯ Accuracy metrics available

Implementation:

```
# experiments/rq3_compression/rq3_2_1_accuracy_headroom.sh
```

Approach:

1. **Define minimum acceptable accuracy:**

    - Clinical requirement: e.g., 90% exercise ID accuracy, BERT ≥ 0.7
    - Set thresholds based on physiotherapist input

2. **Calculate headroom:**

    - Current accuracy (from baseline)
    - Minimum acceptable accuracy
    - Headroom = Current - Minimum

3. **Rank models by headroom:**
    - More headroom = more resilient to quantization accuracy drop

Metrics to collect:

- Current accuracy per model (all metrics)
- Minimum acceptable thresholds
- Headroom per metric per model
- Quantization budget (estimated allowable accuracy drop)

Expected Output:

- Table: Current vs. minimum vs. headroom per model
- Bar plot: Headroom per model
- Recommendation: Which model has most room for compression

How to run:

```
# Use existing baseline results
python experiments/rq3_compression/analyze_headroom.py \
    --baseline_results experiments/results/rq3_1_1 \
    --min_accuracy_config experiments/shared/min_thresholds.yaml \
    --output accuracy_headroom.pdf
```

RQ3.2.2: Precision Reduction Sweep

Question: What is the maximum compression feasible per model?

Current Support: □ Quantization not implemented (needs INT8, dynamic quantization)

Implementation:

```
# experiments/rq3_compression/rq3_2_2_precision_sweep.sh
```

Approach:

1. **Test precision levels:**
    - FP32 (baseline)
    - TF32 (already used in training)
    - BF16 (already used in training)
    - FP16 (already supported)
    - INT8 (need to implement: torch quantization or ONNX Runtime)
2. **Quantization strategies:**
    - Dynamic quantization (weights only)

    - Static quantization (weights + activations, requires calibration)
    - Quantization-aware training (retrain with simulated quantization)

3. **Measure sensitivity:**
    - Accuracy drop per precision level
    - Model size reduction
    - Latency speedup

4. **Compression ratio:**
    - FP32 → FP16: 2x compression
    - FP32 → INT8: 4x compression

Metrics to collect:

- Accuracy vs. precision
- Model size vs. precision
- Latency vs. precision
- Compression resilience score: (accuracy_drop / compression_ratio)

Expected Output:

- Plot: Accuracy vs. precision (per model)
- Plot: Latency speedup vs. precision
- Table: Compression ratios and accuracy tradeoffs
- Recommendation: Maximum compression per model before collapse

How to run:

```bash
# Quantize models
for MODEL in Mobile-VideoGPT-0.5B Mobile-VideoGPT-1.5B; do
    for PRECISION in fp32 bf16 fp16 int8_dynamic int8_static; do
        python experiments/rq3_compression/quantize_model.py \
            --model $MODEL \
            --precision $PRECISION \
            --output_dir experiments/models/quantized/${MODEL}_${PRECISION}

        # Test quantized model
        bash scripts/run_inference.sh \
            --model_path experiments/models/quantized/${MODEL}_${PRECISION}
\
            --test_json dataset/qved_test.json \
            --output_dir experiments/results/rq3_2_2/${MODEL}_${PRECISION}
    done
done

# Analyze compression
python experiments/rq3_compression/analyze_compression.py \
    --results_dir experiments/results/rq3_2_2 \
    --output compression_analysis.pdf
```

RQ4: Failure Analysis (COMBINED)

RQ4.1.2 + RQ4.1.3: Exercise-Specific Failures and Class Imbalance

Questions:

- RQ4.1.2: Are there exercise-specific failure modes per model?
- RQ4.1.3: How does fine-tuning data distribution affect performance?

Current Support: □ Exercise IDs already captured, □ Can create balanced splits

Implementation:

```
# experiments/rq4_failure_analysis/rq4_1_2_confusion_matrix.sh  (COMBINED)
```

Approach - COMBINED:

1. **Analyze current dataset balance:**
   - Count samples per exercise in train/val/test
   - Identify class imbalance
2. **Create alternative splits:**
   - **Balanced subset:** Equal samples per exercise (undersample majority)
   - **Imbalanced full:** Use current distribution
   - **Weighted loss:** Assign class weights inversely proportional to frequency
3. **Train on each split:**
   - Use `scripts/finetune_qved.sh` with different train JSONs
4. **Generate confusion matrices:**
   - Per-exercise prediction accuracy
   - Which exercises are confused with each other?
   - Do all models struggle with the same exercises?
5. **Correlation analysis:**
   - Biomechanical similarity: Are confused exercises biomechanically similar?
   - Data size: Are low-accuracy exercises those with fewer training samples?

Metrics to collect:

- Confusion matrix per model per split
- Per-exercise accuracy
- Correlation: accuracy vs. training samples
- Failure mode patterns (which exercises confused)

Expected Output:

- Confusion matrices (per model, per split)
- Plot: Per-exercise accuracy (baseline vs. balanced vs. weighted)
- Correlation plot: Accuracy vs. training sample count
- Table: Most confused exercise pairs
- Analysis: Biomechanical similarity of confused exercises
- Recommendation: Best strategy for class imbalance

How to run:

```
# Create balanced split
python experiments/rq4_failure_analysis/create_balanced_split.py \
    --input dataset/qved_train.json \
    --output experiments/data/qved_train_balanced.json \
    --samples_per_class 200

# Train on balanced split
bash scripts/finetune_qved.sh \
    --train_json experiments/data/qved_train_balanced.json \
    --output_dir experiments/results/rq4_1/balanced

# Train with weighted loss (modify training script)
bash experiments/rq4_failure_analysis/rq4_1_3_class_imbalance.sh \
    --model Mobile-VideoGPT-0.5B \
    --strategy weighted_loss \
    --output_dir experiments/results/rq4_1/weighted

# Generate confusion matrices
python experiments/rq4_failure_analysis/analyze_failures.py \
    --predictions experiments/results/rq4_1/*/test_predictions.json \
    --ground_truth dataset/ground_truth.json \
    --output confusion_analysis.pdf
```

RQ5: Clinical Guidance Quality

RQ5.2.1: Corrective Guidance Generation

Question: How well can each model generate clinically accurate, actionable feedback?

Current Support: ✓ LLM-as-judge already implemented (Mixtral-8x7B)

Implementation:

```
# experiments/rq5_clinical_guidance/rq5_2_1_feedback_quality.sh
```

Approach:

1. **Extended LLM-as-judge evaluation:**
   - Current: Holistic 1-5 accuracy score
   - Add dimension-specific scores:
     - **Clinical accuracy:** Is the feedback physiologically correct?
     - **Actionability:** Is the guidance specific and implementable?
     - **Root cause:** Does it address root causes vs. symptoms?
     - **Completeness:** Does it cover all errors?
2. **LLM-as-judge prompt template:**

   ```
   Given:
   - Ground Truth: {ground_truth}
   - Model Prediction: {prediction}

   Rate on 1-5 scale:
   1. Clinical Accuracy: Is the feedback physiologically correct?
   2. Actionability: Is it specific (e.g., "keep knees aligned") vs.
   generic (e.g., "improve form")?
   3. Root Cause: Does it address why (e.g., "tight hip flexors causing
   lean") vs. just what?
   4. Completeness: Does it cover all errors present?

   Output JSON:
   {
     "clinical_accuracy": 4,
     "actionability": 3,
     "root_cause": 2,
     "completeness": 5,
     "overall": 3.5
   }
   ```
3. **Optional: Physiotherapist review:**
   - Select 50-100 samples
   - Get expert ratings on same dimensions
   - Compare LLM judge vs. human expert

Metrics to collect:

- Clinical accuracy score (1-5)
- Actionability score (1-5)
- Root cause analysis score (1-5)
- Completeness score (1-5)
- Overall quality score
- Agreement with physiotherapist (if available)

Expected Output:

- Table: Feedback quality scores per model

- Plot: Radar chart (4 dimensions per model)
- Examples: Best/worst feedback per model
- Correlation: Quality scores vs. baseline accuracy metrics
- Recommendation: Best model for clinical guidance

How to run:

```
# Run extended LLM-as-judge evaluation
python experiments/rq5_clinical_guidance/evaluate_feedback.py \
    --predictions experiments/results/rq3_1_1/Mobile-VideoGPT-
0.5B/test_predictions.json \
    --ground_truth dataset/ground_truth.json \
    --output_dir experiments/results/rq5_2_1 \
    --dimensions clinical_accuracy actionability root_cause completeness

# Compare across models
python experiments/rq5_clinical_guidance/analyze_guidance.py \
    --results_dir experiments/results/rq5_2_1 \
    --output feedback_quality_analysis.pdf

# Optional: Physiotherapist review interface
python experiments/rq5_clinical_guidance/expert_review_interface.py \
    --predictions experiments/results/rq5_2_1/Mobile-VideoGPT-
0.5B/test_predictions.json \
    --output expert_ratings.json
```

Model Support Matrix

Model	Codebase Support	Adapter Needed	Notes
Mobile-VideoGPT-0.5B	□ Full	No	Current implementation
Mobile-VideoGPT-1.5B	□ Full	No	Same architecture, different size
VideoLLaMA3-2B	⚠ Partial	Yes	Need conversation template, may need model loader changes
NVILA-Lite-2B	⚠ Partial	Yes	Need conversation template, may need model loader changes

Adapter Development Needed

For VideoLLaMA3 and NVILA-Lite:

1. **Conversation template:** Add to `mobilevideogpt/conversation.py`
2. **Model configuration:** May need custom model class in `mobilevideogpt/model/`

3. **Tokenizer compatibility:** Ensure tokenizer handles special tokens
4. **Video preprocessing:** May need different frame sampling strategies

---

Experiment Execution Priority

Phase 1: Training Efficiency (Weeks 1-2)

- □ **Low effort:** RQ1.1.2 (LR schedules) - config changes only
- □ **Low effort:** RQ1.1.3 (data efficiency) - subset dataset
- ⚠ **Medium effort:** RQ1.1.1 (Full vs. LoRA) - needs full finetuning config

Phase 2: Temporal & Robustness (Weeks 3-4)

- □ **Medium effort:** RQ1.2.1/1.2.3 (temporal) - video resampling
- ⚠ **High effort:** RQ1.3.3 (degradation) - synthetic degradation pipeline

Phase 3: Efficiency Profiling (Week 5)

- □ **Low effort:** RQ3.1.1 (baseline profiling) - add memory tracking
- □ **Low effort:** RQ3.1.2 (batch scaling) - vary batch size
- ⚠ **Medium effort:** RQ3.1.3 (resolution sweep) - video resizing + full factorial

Phase 4: Compression (Week 6)

- □ **Low effort:** RQ3.2.1 (headroom) - analysis only
- ⚠ **High effort:** RQ3.2.2 (quantization) - implement INT8 quantization

Phase 5: Failure Analysis & Guidance (Week 7)

- □ **Medium effort:** RQ4.1.2/4.1.3 (confusion matrix) - retraining + analysis
- □ **Low effort:** RQ5.2.1 (feedback quality) - extend LLM judge
- ⚠ **High effort:** RQ2.1.1 (error localization) - requires manual annotation

---

Shared Utilities

`experiments/shared/metrics.py`

- All evaluation metrics (BERT, METEOR, ROUGE, LLM judge, Exercise ID)
- Confusion matrix generation
- Statistical tests (t-test, ANOVA for comparing models)

`experiments/shared/data_utils.py`

- Create stratified subsets
- Balance dataset
- Generate temporal/resolution variants
- Apply degradations

`experiments/shared/visualization.py`

- Plotting functions (learning curves, heatmaps, Pareto frontiers)
- Table generation (LaTeX-ready)
- Result summarization

`experiments/shared/config.py`

- Model paths
- Hyperparameters
- Evaluation thresholds
- Experiment naming conventions

Expected Deliverables

Per RQ:

- **Results folder:** `experiments/results/rqX_Y_Z/`
- **Plots:** PDF/PNG visualizations
- **Tables:** CSV/LaTeX tables
- **Summary:** Markdown report with findings

Final Deliverables:

1. **Comprehensive report:** `experiments/RESULTS_SUMMARY.md`
2. **Master comparison table:** All models × All RQs
3. **Recommendation matrix:** Use case → Best model
4. **Publication-ready figures:** High-res plots for paper

Getting Started

```
# Create experiments folder structure
bash experiments/setup.sh

# Run all RQ1.1 experiments for Mobile-VideoGPT-0.5B
bash experiments/run_rq1_1.sh --model Mobile-VideoGPT-0.5B

# Run full experimental suite (all RQs, all models)
bash experiments/run_all.sh

# Generate final report
python experiments/generate_report.py --output RESULTS_SUMMARY.md
```

Notes

- **Experiment reproducibility:** All scripts save hyperparameters and random seeds
- **Resource estimation:** Full suite ~200-300 GPU hours (depends on quantization experiments)
- **Parallelization:** Independent experiments can run in parallel (multi-GPU/multi-node)

- **Checkpointing:** Each experiment saves checkpoints for resumability
- **Version control:** Track experiment configs and results in Git

---

References

- QVED Dataset: [Paper link]
- Mobile-VideoGPT: Amshaker/Mobile-VideoGPT
- LoRA: Hu et al., 2021
- LLM-as-Judge: Zheng et al., 2023