

COPUS Video Evaluation System

Technical Reference Manual

Andy Franck, Brendan Ng, Zane Derrod, Ben Fitzgerald

November 18, 2025

Abstract

Technical documentation for the COPUS (Classroom Observation Protocol for Undergraduate STEM) Video Evaluation System. Automates classroom video analysis using vision-language models to identify teaching and learning behaviors in 2-minute intervals. We assume strong Python fundamentals, some machine learning background, and experience developing with PyTorch, Huggingface, and other similar deep learning libraries.

Contents

1	Background and Prerequisites	3
1.1	Understanding the Problem	3
1.1.1	The COPUS Protocol	3
1.1.2	Challenges	3
1.1.3	Sliding Window Approach	3
1.1.4	Temporal Aggregation	4
1.2	Technical Prerequisites	4
1.2.1	Python and ML Libraries	4
1.2.2	Input/Output Formats	4
2	System Overview	5
2.1	Introduction	5
2.2	COPUS Protocol	5
3	Application Layer	6
3.1	copus_evaluation_app.py	6
3.1.1	Class: COPUSEvaluationApp	6
3.1.2	evaluate_video Method	6
3.1.3	Excel Conversion	7
3.1.4	CLI Interface	8
3.2	copus_gui.py	8
3.2.1	Class: COPUSEvaluationGUI	8
3.3	Python API Usage	8
4	Core Evaluation Engine	10
4.1	full_lecture_evaluation.py	10
4.1.1	Class: FullLectureEvaluator	10
4.1.2	evaluate_full_lecture Method	10
4.1.3	predict_window Method	11
4.1.4	Aggregation Algorithm	12
5	Data Formats	13
5.1	JSON Output	13
6	Algorithms	14
6.1	Sliding Window	14
6.2	Temporal Aggregation	14

7	Getting Started with the Codebase	15
7.1	Key Files and Their Purposes	15
7.1.1	copus_evaluation_app.py	15
7.1.2	full_lecture_evaluation.py	15
7.1.3	copus_gui.py	16
7.2	Development Workflow	16
7.2.1	Setting Up Development Environment	16
7.2.2	Code Style and Formatting	17
7.3	Current System State and Limitations	17
7.3.1	What Works Well	17
7.3.2	Known Limitations	17
7.4	Debugging and Troubleshooting	17
7.4.1	Common Development Issues	17
7.4.2	Useful Debugging Commands	18
8	Future Works	19
8.1	Improvements	19
8.1.1	Active Learning for Low-Frequency Behaviors	19
8.1.2	Visual Saliency Maps	19
8.1.3	Hierarchical Behavior Modeling	19
8.2	Research Ideas	20
8.2.1	Multi-Classroom Comparison	20
9	Deployment	21
9.1	Installation	21
9.2	Usage Examples	21
10	Troubleshooting	22
10.1	Common Issues	22
10.2	FAQ	22
A	COPUS Code Reference	24
B	Model Architecture	25

Chapter 1

Background and Prerequisites

1.1 Understanding the Problem

1.1.1 The COPUS Protocol

COPUS (Classroom Observation Protocol for Undergraduate STEM) is an tool to characterize instructional practices in STEM classrooms. Traditional COPUS coding requires:

- Trained human observers watching classroom videos
- Manual coding of behaviors every 2 minutes
- Tracking 24 distinct behavioral codes (13 student, 11 instructor)
- Reliability concerns with multiple observers having variability

1.1.2 Challenges

Multi-label Classification: Multiple behaviors can occur simultaneously within a 2-minute interval (e.g., students listening while instructor is lecturing and writing).

Temporal Reasoning: The system must understand actions over time. A student raising their hand only becomes "Answering Question" (AnQ) when contextualized with instructor acknowledgment.

Fine-grained Activity Recognition: Distinguishing between similar activities requires understanding subtle cues (e.g., "Clicker Group" vs. "Worksheet Group").

1.1.3 Sliding Window Approach

To handle full-length lectures (50-90 minutes), we employ a sliding window strategy:

$$W_i = [t_i, t_i + \Delta t], \quad t_i = i \cdot s \quad (1.1)$$

Where:

- W_i = Window i
- Δt = Window duration (10 seconds)
- s = Step size (5 seconds)

- Overlap = $\Delta t - s = 5$ seconds (50%)

This creates dense coverage with overlapping predictions that are later aggregated.

1.1.4 Temporal Aggregation

For each 2-minute COPUS interval, we aggregate predictions from all overlapping windows using maximum confidence:

$$\hat{y}_{action,interval} = \begin{cases} 1 & \text{if } \max_{w \in W_{interval}} c_{action,w} \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \quad (1.2)$$

Where $c_{action,w}$ is the confidence score for an action in window w . This strategy favors recall (capturing brief activities) over precision.

1.2 Technical Prerequisites

1.2.1 Python and ML Libraries

- Python 3.9+
- PyTorch: Tensor operations, model loading, GPU management
- NumPy: Array operations, indexing
- Pandas: DataFrame manipulation, pivot tables
- Transformers (Hugging Face): Model APIs, tokenizers

1.2.2 Input/Output Formats

Video Formats:

- Primary: MP4 (H.264 codec)
- Auto-conversion: MTS
- Supported: AVI, MOV

Output Formats:

- JSON: Machine-readable with full confidence scores
- Excel: Human-readable COPUS matrix with formulas
- Text: Summary statistics and metadata

Chapter 2

System Overview

2.1 Introduction

2.2 COPUS Protocol

COPUS documents student and instructor behaviors in 2-minute intervals with 24 behavior codes:

Student Codes (13): L (Listening), Ind (Individual thinking), CG (Clicker group), WG (Worksheet group), OG (Other group), AnQ (Answering question), SQ (Student question), WC (Whole class discussion), Prd (Prediction), SP (Student presentation), TQ (Test/quiz), W (Waiting), O (Other)

Instructor Codes (11): Lec (Lecturing), RtW (Real-time writing), FUp (Follow-up), PQ (Posing question), CQ (Clicker question), AnQ (Answering question), MG (Moving/guiding), 1o1 (One-on-one), D/V (Demo/video), Adm (Administration), W (Waiting)

Processing Pipeline:

1. Video loaded at 3 FPS
2. Extract 10-second windows (50% overlap)
3. Model analyzes each window for all 24 COPUS actions
4. Window predictions aggregated into 2-minute intervals
5. Results exported to JSON, Excel, and text formats

Chapter 3

Application Layer

3.1 copus_evaluation_app.py

Main CLI application providing programmatic interface and orchestration.

3.1.1 Class: COPUSEvaluationApp

Listing 3.1: Class Structure

```
1 class COPUSEvaluationApp:
2     # COPUS code mappings
3     CODE_NAMES = {...}           # Code -> Description
4     JSON_KEY_TO_CODE = {...}     # Internal key -> Code
5     PREFERRED_ORDER = [...]      # Display order
6
7     def __init__(self, model_checkpoint=None, device="cuda"):
8         self.evaluator = FullLectureEvaluator(
9             checkpoint_path=model_checkpoint, device=device
10        )
11
12    def evaluate_video(self, video_path, output_dir):
13        """Main evaluation pipeline"""
14        # Returns: (json_path, excel_path)
15
16    def convert_json_to_excel(self, json_path, excel_path):
17        """Convert JSON results to COPUS matrix"""
18
19    def generate_summary_report(self, results, output_dir, ...):
20        """Generate text summary"""
```

3.1.2 evaluate_video Method

Listing 3.2: Main Evaluation Pipeline

```
1 def evaluate_video(self, video_path: str, output_dir: str):
2     # 1. Validate input
3     video_path = Path(video_path)
4     if not video_path.exists():
5         raise FileNotFoundError(...)
6
7     # 2. Setup output directory and filenames
8     output_dir = Path(output_dir)
9     output_dir.mkdir(parents=True, exist_ok=True)
10    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
11    json_path = output_dir / f"{video_path.stem}_{timestamp}_copus.json"
12    excel_path = output_dir / f"{video_path.stem}_{timestamp}_copus.xlsx"
```



```

13
14     # 3. Run evaluation (core processing)
15     evaluation_results = self.evaluator.evaluate_full_lecture(
16         str(video_path), str(json_path)
17     )
18
19     # 4. Convert to Excel format
20     self.convert_json_to_excel(json_path, excel_path)
21
22     # 5. Generate summary report
23     self.generate_summary_report(evaluation_results, output_dir, ...)
24
25     return json_path, excel_path

```

3.1.3 Excel Conversion

Converts JSON to COPUS matrix with formulas:

Listing 3.3: Excel Generation (Simplified)

```

1 def convert_json_to_excel(self, json_path, excel_path):
2     # Load JSON and extract interval data
3     data = json.load(open(json_path))
4     intervals = data["intervals"]
5
6     # Create records (interval, code, value)
7     records = []
8     for interval in intervals:
9         for action, value in interval["actions"].items():
10             code = self.JSON_KEY_TO_CODE.get(action)
11             if code:
12                 records.append({
13                     "Interval": interval["interval_number"],
14                     "Action_Code": code,
15                     "Value": int(bool(value))
16                 })
17
18     # Pivot to matrix format
19     df = pd.DataFrame(records)
20     pivot_df = df.pivot_table(
21         index="Action_Code",
22         columns="Interval",
23         values="Value"
24     )
25
26     # Write to Excel with formulas
27     with pd.ExcelWriter(excel_path, engine="xlsxwriter") as writer:
28         # Add title, headers, data
29         # Add SUM formulas for Total column
30         # Add percentage formulas
31         # Add composite metric formulas

```

Excel Structure:

- Row 1: Title (merged)
- Row 2: Headers (2-min Intervals — 1 — 2 — ... — Total — Percent)
- Row 3+: Code rows (L - Listening — 1 — 0 — 1 — ... — 25 — 83%)
- Additional columns for composite metrics (Student Talk, Instructor Guide, etc.)

3.1.4 CLI Interface

Listing 3.4: Command-Line Arguments

```
1 parser = argparse.ArgumentParser(  
2     description="COPUS_Video_Evaluation_System"  
3 )  
4 parser.add_argument("video_file", help="Path_to_video_(MP4/MTS/AVI/MOV)")  
5 parser.add_argument("--output-dir", "-o", default="copus_results")  
6 parser.add_argument("--model-checkpoint", "-m", default=None)  
7 parser.add_argument("--device", choices=["cuda", "cpu"], default="cuda")  
8 parser.add_argument("--verbose", "-v", action="store_true")
```

Usage:

```
1 python copus_evaluation_app.py lecture.mp4  
2 python copus_evaluation_app.py lecture.mp4 -o results/ --device cpu  
3 python copus_evaluation_app.py lecture.mp4 -m models/copus_model_best/
```

3.2 copus_gui.py

Tkinter-based GUI for non-technical users.

3.2.1 Class: COPUSEvaluationGUI

Listing 3.5: GUI Class Structure

```
1 class COPUSEvaluationGUI:  
2     def __init__(self, root):  
3         self.root = root  
4         self.root.title("COPUS_Video_Evaluation_System")  
5         self.root.geometry("900x700")  
6  
7         # Variables  
8         self.video_path = tk.StringVar()  
9         self.output_dir = tk.StringVar(value="copus_results")  
10        self.model_checkpoint = tk.StringVar()  
11        self.device = tk.StringVar(value="cuda")  
12  
13        self.create_widgets()  
14  
15    def create_widgets(self):  
16        """Build GUI layout"""  
17  
18    def browse_video(self):  
19        """File dialog for video selection"""  
20  
21    def start_evaluation(self):  
22        """Launch evaluation in separate thread"""  
23  
24    def run_evaluation(self, video_file, output_dir):  
25        """Thread worker function"""
```

3.3 Python API Usage

Listing 3.6: Programmatic Usage

```
1 from copus_evaluation_app import COPUSEvaluationApp  
2 import json  
3  
4 # Initialize application
```

```

5 app = COPUSEvaluationApp(device="cuda")
6
7 # Evaluate video
8 json_path, excel_path = app.evaluate_video(
9     "lecture.mp4",
10    "results/"
11 )
12
13 # Load and analyze results
14 with open(json_path, 'r') as f:
15     results = json.load(f)
16
17 # Access interval data
18 for interval in results['intervals']:
19     print(f"Interval_{interval['interval_number']}")
20     for action, present in interval['actions'].items():
21         if present:
22             confidence = interval['actions_with_confidence'][action]
23             print(f"_{action}:_{confidence:.2f}")

```

Chapter 4

Core Evaluation Engine

4.1 full_lecture_evaluation.py

4.1.1 Class: FullLectureEvaluator

Central component that orchestrates video processing and model inference.

Listing 4.1: Initialization

```
1 class FullLectureEvaluator:
2     def __init__(self, checkpoint_path=None, device="cuda"):
3         self.device = device
4         self.checkpoint_path = checkpoint_path
5
6         # Load MiniCPM-V model
7         if checkpoint_path:
8             self.model = load_from_checkpoint(checkpoint_path)
9         else:
10            self.model = AutoModel.from_pretrained(
11                "openbmb/MiniCPM-V-2_6",
12                trust_remote_code=True
13            ).to(device)
14
15        self.tokenizer = AutoTokenizer.from_pretrained(
16            "openbmb/MiniCPM-V-2_6",
17            trust_remote_code=True
18        )
```

4.1.2 evaluate_full_lecture Method

Listing 4.2: Main Evaluation Logic

```
1 def evaluate_full_lecture(self, video_path, output_json_path):
2     # 1. Convert MTS to MP4 if needed
3     if video_path.endswith('.MTS'):
4         video_path = convert_mts_to_mp4(video_path)
5
6     # 2. Load video and extract metadata
7     video_reader = VideoReader(video_path)
8     fps = video_reader.get_avg_fps()
9     total_frames = len(video_reader)
10    duration = total_frames / fps
11
12    # 3. Define sliding windows
13    WINDOW_DURATION = 10 # seconds
14    WINDOW_STEP = 5 # seconds
15    windows = create_sliding_windows(
```

```

16         duration, WINDOW_DURATION, WINDOW_STEP
17     )
18
19     # 4. Process each window
20     window_predictions = []
21     for i, (start, end) in enumerate(windows):
22         # Extract frames
23         frames = extract_frames_at_3fps(
24             video_reader, start, end
25         )
26
27         # Run model inference
28         predictions = self.predict_window(frames)
29
30         window_predictions.append({
31             "window_number": i,
32             "start_time": start,
33             "end_time": end,
34             "predictions": predictions
35         })
36
37     # 5. Aggregate into 2-minute intervals
38     intervals = aggregate_to_intervals(
39         window_predictions, duration
40     )
41
42     # 6. Save results
43     results = {
44         "video_path": video_path,
45         "video_info": {...},
46         "intervals": intervals,
47         ...
48     }
49     with open(output_json_path, 'w') as f:
50         json.dump(results, f, indent=2)
51
52     return results

```

4.1.3 predict_window Method

Listing 4.3: Window-Level Prediction

```

1 def predict_window(self, frames):
2     # Construct prompt for multi-label classification
3     prompt = """Analyze this classroom video segment.
4     For each behavior, respond YES or NO:
5
6     STUDENT BEHAVIORS:
7     - Listening/taking notes (L)
8     - Individual thinking/problem solving (Ind)
9     - Clicker group discussion (CG)
10    ...
11
12    INSTRUCTOR BEHAVIORS:
13    - Lecturing (Lec)
14    - Real-time writing (RtW)
15    - Posing questions to class (PQ)
16    ...
17    """
18
19    # Encode video frames
20    inputs = self.model.encode_video(
21        frames,
22        prompt=prompt,
23        fps=3
24    )

```

```

25
26     # Generate predictions
27     outputs = self.model.generate(
28         **inputs,
29         max_new_tokens=500,
30         temperature=0.7
31     )
32
33     # Parse structured output
34     predictions = parse_copus_output(outputs)
35
36     return predictions # Dict: action -> confidence

```

4.1.4 Aggregation Algorithm

Listing 4.4: Temporal Aggregation

```

1 def aggregate_to_intervals(window_predictions, duration):
2     AGGREGATION_INTERVAL = 120 # 2 minutes
3     num_intervals = int(np.ceil(duration / AGGREGATION_INTERVAL))
4
5     intervals = []
6     for i in range(num_intervals):
7         interval_start = i * AGGREGATION_INTERVAL
8         interval_end = min(
9             (i + 1) * AGGREGATION_INTERVAL,
10            duration
11        )
12
13        # Find overlapping windows
14        overlapping = [
15            w for w in window_predictions
16            if w["start_time"] < interval_end and
17                w["end_time"] > interval_start
18        ]
19
20        # Compute max confidence per action
21        action_confidences = {}
22        for window in overlapping:
23            for action, conf in window["predictions"].items():
24                action_confidences[action] = max(
25                    action_confidences.get(action, 0), conf
26                )
27
28        # Binary thresholding (>= 0.5)
29        actions_binary = {
30            action: (action_confidences.get(action, 0) >= 0.5)
31            for action in COPUS_ACTIONS.keys()
32        }
33
34        intervals.append({
35            "interval_number": i + 1,
36            "start_time": interval_start,
37            "end_time": interval_end,
38            "actions": actions_binary,
39            "actions_with_confidence": action_confidences,
40            "num_windows": len(overlapping)
41        })
42
43     return intervals

```

Chapter 5

Data Formats

5.1 JSON Output

```
1 {
2   "video_path": "path/to/lecture.mp4",
3   "video_info": {
4     "duration_seconds": 3600.0,
5     "duration_minutes": 60.0,
6     "fps": 3.0,
7     "total_frames": 10800
8   },
9   "processing_info": {
10    "window_duration": 10,
11    "window_step": 5,
12    "total_windows": 719,
13    "aggregation_interval": 120
14  },
15  "intervals": [
16    {
17      "interval_number": 1,
18      "start_time": 0,
19      "end_time": 120,
20      "start_time_str": "0:00:00",
21      "end_time_str": "0:02:00",
22      "actions": {
23        "student_listening": true,
24        "instructor_lecturing": true,
25        ...
26      },
27      "actions_with_confidence": {
28        "student_listening": 0.85,
29        "instructor_lecturing": 1.0,
30        ...
31      },
32      "num_windows": 25
33    },
34    ...
35  ],
36  "timestamp": "2024-01-15T10:30:45"
37 }
```

Chapter 6

Algorithms

6.1 Sliding Window

Pseudocode:

```
For i = 0 to num_windows - 1:
    start_time = i * WINDOW_STEP
    end_time = start_time + WINDOW_DURATION

    frames = extract_frames(video, start_time, end_time)
    encoded = encode_video_window(frames, fps)
    predictions = model_inference(encoded)

    results.append(predictions)
```

Parameters for 60-min lecture:

- Duration: 3600s, Window: 10s, Step: 5s
- Number of windows: $(3600 - 10)/5 + 1 = 719$
- Processing time: 24-48 HR depending on GPU power. Ran on NVIDIA RTX 5080 in around 32 hours.

6.2 Temporal Aggregation

Strategy: Maximum confidence across overlapping windows

$$C_{action,interval} = \max_{w \in W_{overlapping}} C_{action,w} \quad (6.1)$$

Binary threshold: Action marked present if $C \geq 0.5$

Chapter 7

Getting Started with the Codebase

7.1 Key Files and Their Purposes

7.1.1 `copus_evaluation_app.py`

Purpose: Application layer and main entry point

Responsibilities:

- Command-line interface parsing
- COPUS code mapping and naming
- JSON to Excel conversion
- Summary report generation
- Orchestration of FullLectureEvaluator

When to modify:

- Adding new CLI arguments
- Changing output formats
- Modifying Excel layout or formulas
- Adding new COPUS codes (rare, protocol-dependent)

7.1.2 `full_lecture_evaluation.py`

Purpose: Core evaluation logic

Responsibilities:

- Model loading and initialization
- Video preprocessing (frame extraction, format conversion)
- Sliding window generation
- Window-level inference
- Temporal aggregation to 2-minute intervals

When to modify:

- Changing window parameters (duration, step size)
- Modifying aggregation strategy
- Adjusting confidence thresholds
- Implementing model ensembles
- Adding preprocessing steps

7.1.3 copus_gui.py

Purpose: Graphical user interface

Responsibilities:

- UI layout and widgets
- User input validation
- Progress reporting
- Threaded evaluation (prevent UI freezing)
- Error display and result opening

When to modify:

- Improving UI/UX
- Adding batch processing in GUI
- Implementing result preview
- Adding visualization features

7.2 Development Workflow

7.2.1 Setting Up Development Environment

```
1 # Clone repository
2 git clone <repo-url>
3 cd copus-video-evaluation
4
5 # Install dependencies
6 cd app
7 pip install -r requirements.txt
8
9 # Verify installation
10 python -c "import torch; print(f'CUDA: {torch.cuda.is_available()}')"
11 python -c "import decord; print('Decord OK')"
12 ffmpeg -version
```

7.2.2 Code Style and Formatting

The project formats all code via. the Black Python formatter. The plugin is freely available on most IDEs, and can be manually installed via pip:

```
1 pip install black
2 black .
```

7.3 Current System State and Limitations

7.3.1 What Works Well

- **High-frequency behaviors:** Achieves 85-95% accuracy on common actions like "Listening" and "Lecturing"
- **Long videos:** Reliably processes 50-90 minute lectures without crashes
- **Multiple formats:** Handles MP4, AVI, MOV, MTS without manual conversion
- **Temporal consistency:** Sliding window approach provides smooth transitions
- **User accessibility:** Both CLI and GUI interfaces work intuitively

7.3.2 Known Limitations

- **Low-frequency behaviors:** Actions like "Student Presentation" (SP) or "Prediction" (Prd) have lower recall due to limited training examples
- **Audio-dependent actions:** "Student Question" (SQ) detection relies heavily on visual cues (hand raising) and may miss verbal questions
- **Processing speed:** This system requires 24-48 hours per full lecture on high-end GPUs.
TLDR: PC EXPLODER

7.4 Debugging and Troubleshooting

7.4.1 Common Development Issues

Issue: Video loading fails with "codec not supported"

- **Cause:** FFmpeg missing required codecs
- **Fix:** Install full FFmpeg build (not minimal version): `sudo apt install ffmpeg libavcodec-extra`

Issue: GPU out of memory during processing

- **Cause:** Batch size too large or other GPU processes
- **Fix:** Reduce batch size in model config; clear GPU cache with `torch.cuda.empty_cache()`
- Consider a more powerful GPU or utilize a cluster

Issue: Temporal aggregation produces strange patterns (should be completely resolved)

- **Cause:** Window timestamps miscalculated or overlapping window identification bug
- **Fix:** Add logging to `aggregate_to_intervals()`; verify window start/end times align with intervals

7.4.2 Useful Debugging Commands

```

1 # Enable verbose logging
2 export COPUS_DEBUG=1
3 python copus_evaluation_app.py lecture.mp4 --verbose
4
5 # Profile model inference time
6 python -m cProfile -o profile.stats full_lecture_evaluation.py
7 python -c "import pstats; p = pstats.Stats('profile.stats'); \
8 p.sort_stats('cumulative').print_stats(20)"
9
10 # Monitor GPU usage during processing
11 watch -n 1 nvidia-smi
12
13 # Check video metadata without processing
14 ffprobe -show_format -show_streams lecture.mp4

```

Chapter 8

Future Works

8.1 Improvements

8.1.1 Active Learning for Low-Frequency Behaviors

Motivation: Rare COPUS codes (SP, Prd, TQ, D/V) have insufficient training examples, leading to poor recall.

Implementation Approach:

1. **Uncertainty Sampling:**

- Run model on large unlabeled video corpus
- Identify windows where model confidence is low for rare actions (e.g., $0.3 < P(SP) < 0.6$)
- Sample 200-500 uncertain windows for human annotation

2. **Iterative Fine-Tuning:**

- Annotate selected windows (faster than full videos)
- Add to training set and re-train model
- Repeat 3-5 times

8.1.2 Visual Saliency Maps

Motivation: Educators and researchers can then understand *why* the model made certain predictions.

Implementation Approach:

- Use Grad-CAM or attention visualization to highlight regions influencing predictions
- For each detected behavior, generate heatmap overlay showing attended areas
- Save visualizations alongside Excel outputs

8.1.3 Hierarchical Behavior Modeling

Motivation: Some COPUS codes have hierarchical relationships (e.g., "Group Work" subsumes CG, WG, OG).

Implementation Approach:

- Model predictions as tree: First predict "Group Work" vs. "Individual", then if "Group Work", predict CG/WG/OG

8.2 Research Ideas

8.2.1 Multi-Classroom Comparison

Research Question: What are typical COPUS profiles for effective vs. ineffective teaching?

Experimental Design:

1. Process videos across different instructors and courses
2. Correlate COPUS patterns with outcomes like exam scores, course evaluations
3. Build recommendation system: "Instructors with high student engagement tend to have X% more follow-up questions"

Chapter 9

Deployment

9.1 Installation

Requirements:

- Python 3.9+
- CUDA 11.8+ (for GPU)
- FFmpeg 4.0+
- 32GB+ RAM, 16GB+ VRAM (recommended)

9.2 Usage Examples

CLI:

```
1 python copus_evaluation_app.py lecture.mp4
2 python copus_evaluation_app.py lecture.mp4 -o results/ --device cpu
3 python copus_evaluation_app.py lecture.mp4 -m models/copus_model_best/
```

GUI:

```
1 python copus_gui.py
```

Python API:

```
1 from copus_evaluation_app import COPUSEvaluationApp
2
3 app = COPUSEvaluationApp(device="cuda")
4 json_path, excel_path = app.evaluate_video("lecture.mp4", "results/")
```

Chapter 10

Troubleshooting

10.1 Common Issues

CUDA Out of Memory:

- Close other GPU applications
- Reduce video resolution
- Switch computers, cluster, or as a last case resort (extremely slow) use `--device cpu`

FFmpeg Not Found:

- Ubuntu: `sudo apt install ffmpeg`
- macOS: `brew install ffmpeg`
- Windows: Download from ffmpeg.org, add to PATH

Model Loading Errors:

- Verify checkpoint directory exists
- Ensure internet connection for base model download
- Check CUDA compatibility

10.2 FAQ

Q: What video formats are supported?

A: MP4, AVI, MOV directly. MTS auto-converts to MP4.

Q: Can I modify the 2-minute interval?

A: Yes, change `AGGREGATION_INTERVAL` in `full_lecture_evaluation.py`.

Q: How to interpret confidence scores?

A: 1.0 = high confidence, 0.5-0.99 = medium, 0.25-0.49 = low (filtered out), 0-0.25 = "Houston, we have a problem".

Appendix A

COPUS Code Reference

Code	Category	Description
L	Student	Listening/taking notes
Ind	Student	Individual thinking
CG	Student	Clicker group discussion
WG	Student	Worksheet group work
OG	Student	Other group activity
AnQ	Student	Answering question
SQ	Student	Student asks question
WC	Student	Whole class discussion
Prd	Student	Making prediction
SP	Student	Student presentation
TQ	Student	Test/quiz
W	Student	Waiting
O	Student	Other
Lec	Instructor	Lecturing
RtW	Instructor	Real-time writing
FUp	Instructor	Follow-up/feedback
PQ	Instructor	Posing question
CQ	Instructor	Clicker question
AnQ	Instructor	Answering question
MG	Instructor	Moving/guiding
1o1	Instructor	One-on-one discussion
D/V	Instructor	Demo/video/simulation
Adm	Instructor	Administration
W	Instructor	Waiting

Appendix B

Model Architecture

MiniCPM-V-4.5:

- Parameters: 8.1B
- Vision Encoder: SigLIP-400M
- Language Model: Qwen2-7B
- Context: 32K tokens
- Temporal modeling: Native video understanding