

COPUS Video Evaluation System

Technical Reference Manual

Andy Franck, Brendan Ng, Zane Derrod, Ben Fitzgerald

November 4, 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.

Contents

1	System Overview	3
1.1	Introduction	3
1.2	COPUS Protocol	3
2	Application Layer	4
2.1	copus_evaluation_app.py	4
2.1.1	Class: COPUSEvaluationApp	4
2.1.2	evaluate_video Method	4
2.1.3	Excel Conversion	5
2.1.4	CLI Interface	6
2.2	copus_gui.py	6
2.2.1	Class: COPUSEvaluationGUI	6
2.2.2	Key Features	6
3	Core Evaluation Engine	8
3.1	full_lecture_evaluation.py	8
3.1.1	Constants	8
3.1.2	Action Definitions	8
3.1.3	Class: FullLectureEvaluator	8
3.1.4	Window Evaluation	9
3.1.5	Full Lecture Evaluation	9
3.1.6	Temporal Aggregation	10
4	Data Formats	12
4.1	JSON Output	12
4.2	Excel Output	12
5	Algorithms	14
5.1	Sliding Window	14
5.2	Temporal Aggregation	14
6	Deployment	15
6.1	Installation	15
6.2	Usage Examples	15
7	Troubleshooting	16
7.1	Common Issues	16
7.2	FAQ	16

A	COPUS Code Reference	18
B	Model Architecture	19

Chapter 1

System Overview

1.1 Introduction

The COPUS system automates behavioral analysis of STEM classroom videos using deep learning. Traditional COPUS requires trained observers to manually code behaviors every 2 minutes.

Key Features:

- Automated full-lecture processing (50-90 min videos)
- COPUS-compliant 2-minute interval coding
- Multiple interfaces: CLI, GUI, Python API
- Excel, JSON, and text report outputs
- GPU acceleration with CPU fallback

1.2 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 2

Application Layer

2.1 copus_evaluation_app.py

Main CLI application providing programmatic interface and orchestration.

2.1.1 Class: COPUSEvaluationApp

Listing 2.1: 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"""
```

2.1.2 evaluate_video Method

Listing 2.2: 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

```

2.1.3 Excel Conversion

Converts JSON to COPUS matrix with formulas:

Listing 2.3: 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.)

2.1.4 CLI Interface

Listing 2.4: 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/
```

2.2 copus_gui.py

Tkinter-based GUI for non-technical users.

2.2.1 Class: COPUSEvaluationGUI

Listing 2.5: 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"""
```

2.2.2 Key Features

Layout:

- Title and description section
- Input configuration (video file, output directory)

- Optional settings (model checkpoint, device selection)
- Action buttons (Start, Cancel, Clear Log, Exit)
- Progress bar (indeterminate during processing)
- Scrollable log display (real-time feedback)

Threading:

Listing 2.6: Non-blocking Evaluation

```

1 def start_evaluation(self):
2     # Validate inputs
3     # Update UI state (disable start, enable cancel)
4     # Launch evaluation in thread
5     eval_thread = threading.Thread(
6         target=self.run_evaluation,
7         args=(video_file, output_dir),
8         daemon=True
9     )
10    eval_thread.start()
11
12 def run_evaluation(self, video_file, output_dir):
13     try:
14         app = COPUSEvaluationApp(...)
15         json_path, excel_path = app.evaluate_video(...)
16         # Show success dialog
17         # Offer to open output directory
18     except Exception as e:
19         # Show error dialog
20     finally:
21         # Re-enable UI

```

Platform-Specific Directory Opening:

Listing 2.7: Cross-Platform File Browser

```

1 import platform, subprocess
2
3 system = platform.system()
4 if system == "Windows":
5     os.startfile(directory)
6 elif system == "Darwin": # macOS
7     subprocess.Popen(["open", directory])
8 else: # Linux
9     subprocess.Popen(["xdg-open", directory])

```

Installation:

```

1 # Install PyTorch (adjust CUDA version for your gpu, system)
2 pip install torch==2.0.0+cu118 -f \
3     https://download.pytorch.org/whl/torch_stable.html
4
5 # Install dependencies
6 pip install -r requirements.txt
7
8 # optional FFMPEG install if you want to handle MTS files

```

Chapter 3

Core Evaluation Engine

3.1 full_lecture_evaluation.py

3.1.1 Constants

Listing 3.1: Processing Parameters

```
1 LECTURE_FPS = 3           # Target video FPS
2 WINDOW_DURATION = 10      # Window size (seconds)
3 WINDOW_FRAMES = 30        # Frames per window
4 WINDOW_STEP = 5           # Step size (50% overlap)
5 AGGREGATION_INTERVAL = 120 # 2-minute COPUS intervals
6
7 MAX_NUM_FRAMES = 15        # Model frame limit
8 MAX_NUM_PACKING = 2        # Frame packing factor
9 TIME_SCALE = 0.1           # Temporal resolution
```

3.1.2 Action Definitions

Listing 3.2: COPUS Action Mappings

```
1 COPUS_ACTIONS = {
2     "student_listening": 0, "student_individual_thinking": 1,
3     "student_clicker_group": 2, "student_worksheet_group": 3,
4     "student_other_group": 4, "student_answer_question": 5,
5     "student_ask_question": 6, "student_whole_class_discussion": 7,
6     "student_prediction": 8, "student_presentation": 9,
7     "student_test_quiz": 10, "student_waiting": 11,
8     "student_other": 12,
9     "instructor_lecturing": 13, "instructor_real_time_writing": 14,
10    "instructor_follow_up": 15, "instructor_posing_question": 16,
11    "instructor_clicker_question": 17,
12    "instructor_answering_question": 18,
13    "instructor_moving_guiding": 19, "instructor_one_on_one": 20,
14    "instructor_demo_video": 21, "instructor_administration": 22,
15    "instructor_waiting": 23,
16 }
17
18 COPUS_LABELS = {
19     "student_listening": "L-Listening/Taking Notes",
20     # ... full mappings
21 }
```

3.1.3 Class: FullLectureEvaluator

Listing 3.3: Evaluator Structure

```

1 class FullLectureEvaluator:
2     def __init__(self, checkpoint_path=None,
3                 base_model="openbmb/MiniCPM-V-4_5", device="cuda"):
4         """Load VLM and optional fine-tuned classifier"""
5         self.device = device
6         self.model = AutoModel.from_pretrained(...)
7         self.tokenizer = AutoTokenizer.from_pretrained(...)
8         if checkpoint_path:
9             self.load_classifier(checkpoint_path)
10
11     def evaluate_window(self, frames, temporal_ids):
12         """Analyze 10-second window -> action predictions"""
13
14     def evaluate_full_lecture(self, video_path, output_path):
15         """Process full video -> 2-minute intervals"""
16
17     def aggregate_results(self, window_results, duration):
18         """Aggregate windows into COPUS intervals"""
19
20     def parse_action_classifications(self, model_response):
21         """Extract actions from VLM text response"""

```

3.1.4 Window Evaluation

Listing 3.4: Window Processing

```

1 def evaluate_window(self, frames, temporal_ids):
2     # Create prompt with all 24 COPUS actions
3     prompt = f"""Analyze this classroom video and identify
4     ALL COPUS actions occurring.
5
6     COPUS Actions:
7     {list of all actions with descriptions}
8
9     Format response as:
10    DETECTED ACTIONS:
11    action_code: confidence_level - justification
12    ...
13    """
14
15    msgs = [{"role": "user", "content": frames + [prompt]}]
16
17    with torch.no_grad():
18        answer = self.model.chat(
19            msgs=msgs, tokenizer=self.tokenizer,
20            temporal_ids=temporal_ids
21        )
22
23    predictions = self.parse_action_classifications(answer)
24    return {"description": answer, "predictions": predictions}

```

3.1.5 Full Lecture Evaluation

Listing 3.5: Complete Processing Pipeline

```

1 def evaluate_full_lecture(self, video_path, output_path):
2     # 1. Load video
3     vr = VideoReader(str(video_path), ctx=cpu(0))
4     fps, total_frames = vr.get_avg_fps(), len(vr)
5     duration = total_frames / fps
6
7     # 2. Calculate sliding windows
8     num_windows = int((duration - WINDOW_DURATION) / WINDOW_STEP) + 1

```

```

9
10 # 3. Process each window
11 window_results = []
12 for window_idx in range(num_windows):
13     start_sec = window_idx * WINDOW_STEP
14     end_sec = start_sec + WINDOW_DURATION
15
16     # Extract frames
17     frames_array = vr.get_batch(frame_indices).asnumpy()
18     frames, temporal_ids = encode_video_window(frames_array, fps)
19
20     # Evaluate
21     result = self.evaluate_window(frames, temporal_ids)
22     window_results.append({
23         "start_time": start_sec, "end_time": end_sec,
24         "predictions": result["predictions"]
25     })
26
27 # 4. Aggregate into 2-minute intervals
28 intervals = self.aggregate_results(window_results, duration)
29
30 # 5. Save results
31 output = {
32     "video_path": str(video_path),
33     "video_info": {...},
34     "intervals": intervals,
35     "timestamp": datetime.now().isoformat()
36 }
37
38 with open(output_path, "w") as f:
39     json.dump(output, f, indent=2)
40
41 return output

```

3.1.6 Temporal Aggregation

Listing 3.6: Window to Interval Aggregation

```

1 def aggregate_results(self, window_results, total_duration):
2     num_intervals = int(math.ceil(total_duration / 120))
3     intervals = []
4
5     for i in range(num_intervals):
6         interval_start = i * 120
7         interval_end = min((i + 1) * 120, total_duration)
8
9         # Find overlapping windows
10        overlapping = [w for w in window_results
11                       if w["start_time"] < interval_end and
12                          w["end_time"] > interval_start]
13
14        # Aggregate confidences (max across windows)
15        action_confidences = {}
16        for window in overlapping:
17            for action, conf in window["predictions"].items():
18                action_confidences[action] = max(
19                    action_confidences.get(action, 0), conf
20                )
21
22        # Binary thresholding (>= 0.5)
23        actions_binary = {
24            action: (action_confidences.get(action, 0) >= 0.5)
25            for action in COPUS_ACTIONS.keys()
26        }
27
28        intervals.append({

```

```
29         "interval_number": i + 1,
30         "start_time": interval_start,
31         "end_time": interval_end,
32         "actions": actions_binary,
33         "actions_with_confidence": action_confidences,
34         "num_windows": len(overlapping)
35     })
36
37     return intervals
```

Chapter 4

Data Formats

4.1 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 }
```

4.2 Excel Output

Structure:

- Row 1: Title (merged across columns)

- Row 2: Headers (2-min Intervals — 1 — 2 — ... — Total — Percent)
- Rows 3+: COPUS codes with binary values (0/1)
- Formulas: `Total = SUM(intervals)`, `Percent = Total/NumIntervals`
- Additional columns: Composite metrics (Student Talk, Instructor Guide, etc.)

Chapter 5

Algorithms

5.1 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: 12-15 min (GPU), 150-180 min (CPU)

5.2 Temporal Aggregation

Strategy: Maximum confidence across overlapping windows

$$C_{action,interval} = \max_{w \in W_{overlapping}} C_{action,w} \quad (5.1)$$

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

Rationale:

- Captures actions occurring for any significant portion of interval
- Aligns with human COPUS coding (mark if occurs during interval)
- Reduces false negatives for brief but important activities

Chapter 6

Deployment

6.1 Installation

Requirements:

- Python 3.9-3.11
- CUDA 11.8+ (for GPU)
- FFmpeg 4.0+
- 32GB+ RAM, 16GB+ VRAM (recommended)

Setup:

```
1 # Create environment
2 python3 -m venv venv
3 source venv/bin/activate
4
5 # Install PyTorch
6 pip install torch==2.0.0+cu118 -f \
7     https://download.pytorch.org/whl/torch_stable.html
8
9 # Install dependencies
10 cd app && pip install -r requirements.txt
11
12 # Verify
13 python -c "import torch; print(torch.cuda.is_available())"
14 ffmpeg -version
```

6.2 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 7

Troubleshooting

7.1 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

7.2 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