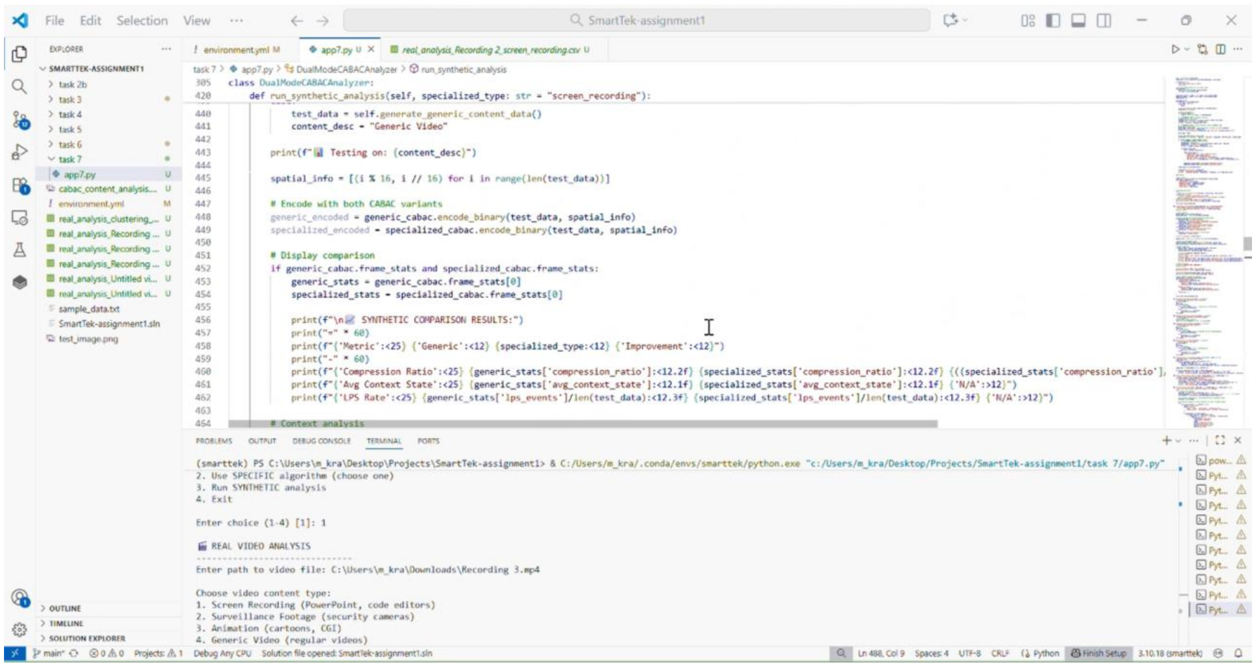
CABAC Coder for Video

In this project we implemented a simplified CABAC encoder in Python. The encoder takes frame differences (residuals), turns them into bits and encodes them using several context models whose probabilities are updated after each bit. We used four different context setups: a generic model, a model tuned for screen recordings, one for surveillance-style video and one for animation. To compare generic and specialized content we used two clips: an AI-generated walking video in a city square (generic) that we got from youtube **[1]** and a screen recording of Visual Studio Code while we scroll through code (specialized). For runtime reasons we analysed only the first 40 frames of the generic clip and the first 50 frames of the screen-recording clip, keeping the context states between frames. For each clip and model we measured the compression ratio and how the average context state changed over time to see how the different CABAC variants adapt to these two types of video.

Figure 1: *Frame from generic (walking) video*



```python
class DualModeCABACAnalyzer:
    def run_synthetic_analysis(self, specialized_type: str = "screen_recording"):
            test_data = self.generate_generic_content_data()
            content_desc = "Generic Video"

        print(f"📊 Testing on: {content_desc}")

        spatial_info = [(i % 16, i // 16) for i in range(len(test_data))]

        # Encode with both CABAC variants
        generic_encoded = generic_cabac.encode_binary(test_data, spatial_info)
        specialized_encoded = specialized_cabac.encode_binary(test_data, spatial_info)

        # Display comparison
        if generic_cabac.frame_stats and specialized_cabac.frame_stats:
            generic_stats = generic_cabac.frame_stats[0]
            specialized_stats = specialized_cabac.frame_stats[0]

            print(f"\n📈 SYNTHETIC COMPARISON RESULTS:")
            print("=" * 60)
            print(f"{'Metric':<25} {'Generic':<12} {specialized_type:<12} {'Improvement':<12}")
            print("-" * 60)
            print(f"{'Compression Ratio':<25} {generic_stats['compression_ratio']:<12.2f} {specialized_stats['compression_ratio']:<12.2f} {((specialized_stats['compression_ratio'],
            print(f"{'Avg Context State':<25} {generic_stats['avg_context_state']:<12.1f} {specialized_stats['avg_context_state']:<12.1f} {'N/A':>12}")
            print(f"{'LPS Rate':<25} {generic_stats['lps_events']/len(test_data):<12.3f} {specialized_stats['lps_events']/len(test_data):<12.3f} {'N/A':>12}")

        # Context analysis
```

```
(smarttek) PS C:\Users\m_kra\Desktop\Projects\SmartTek-assignment1> & C:/Users/m_kra/.conda/envs/smarttek/python.exe "c:/Users/m_kra/Desktop/Projects/SmartTek-assignment1/task 7/app7.py"
2. Use SPECIFIC algorithm (choose one)
3. Run SYNTHETIC analysis
4. Exit

Enter choice (1-4) [1]: 1

🎬 REAL VIDEO ANALYSIS
------------------------------
Enter path to video file: C:\Users\m_kra\Downloads\Recording 3.mp4

Choose video content type:
1. Screen Recording (PowerPoint, code editors)
2. Surveillance Footage (security cameras)
3. Animation (cartoons, CGI)
4. Generic Video (regular videos)
```

SmartTek-assignment1
environment.yml
app7.py
real_analysis_Recording 2_screen_recording.csv
task 7 > app7.py > DualModeCABACAnalyzer > run_synthetic_analysis
EXPLORER
SMARTTEK-ASSIGNMENT1
task 2b
task 3
task 4
task 5
task 6
task 7
app7.py
cabac_content_analysis...
environment.yml
real_analysis_clustering_...
real_analysis_Recording ...
real_analysis_Recording ...
real_analysis_Recording ...
real_analysis_Untitled vi...
real_analysis_Untitled vi...
sample_data.txt
SmartTek-assignment1.sln
test_image.png
OUTLINE
TIMELINE
SOLUTION EXPLORER
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
main*
Debug Any CPU
Solution file opened: SmartTek-assignment1.sln
Ln 488, Col 9
Spaces: 4
UTF-8
CRLF
Python
Finish Setup
3.10.18 (smarttek)

Figure 2: *Frame from VS Code screen recording*

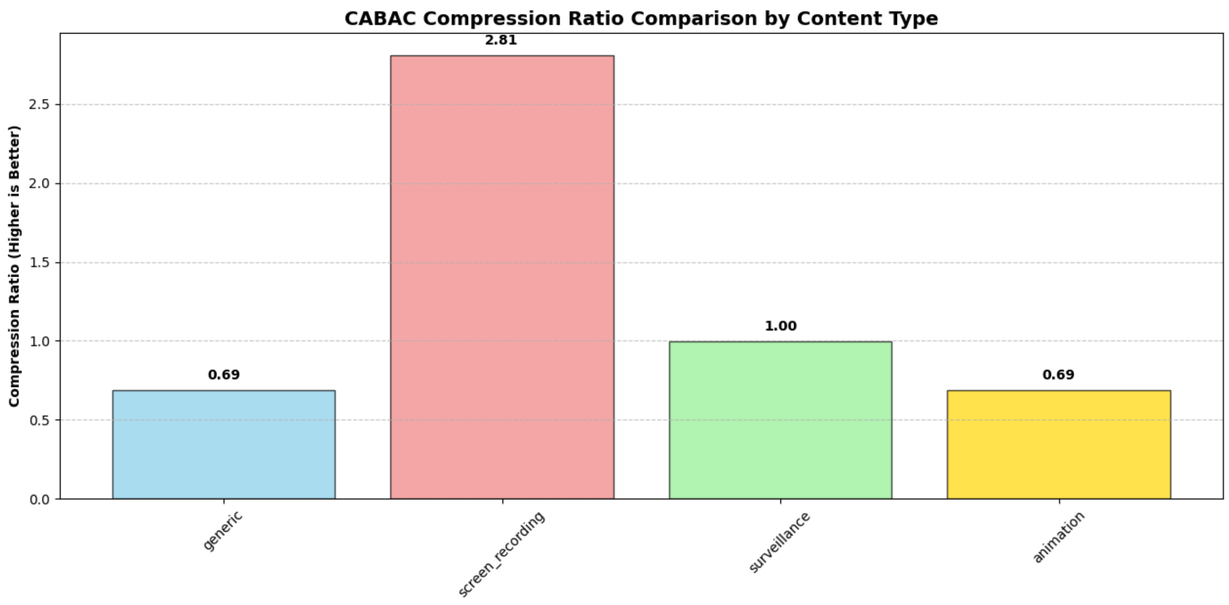


Figure 3: *Compression ratio chart – generic video*

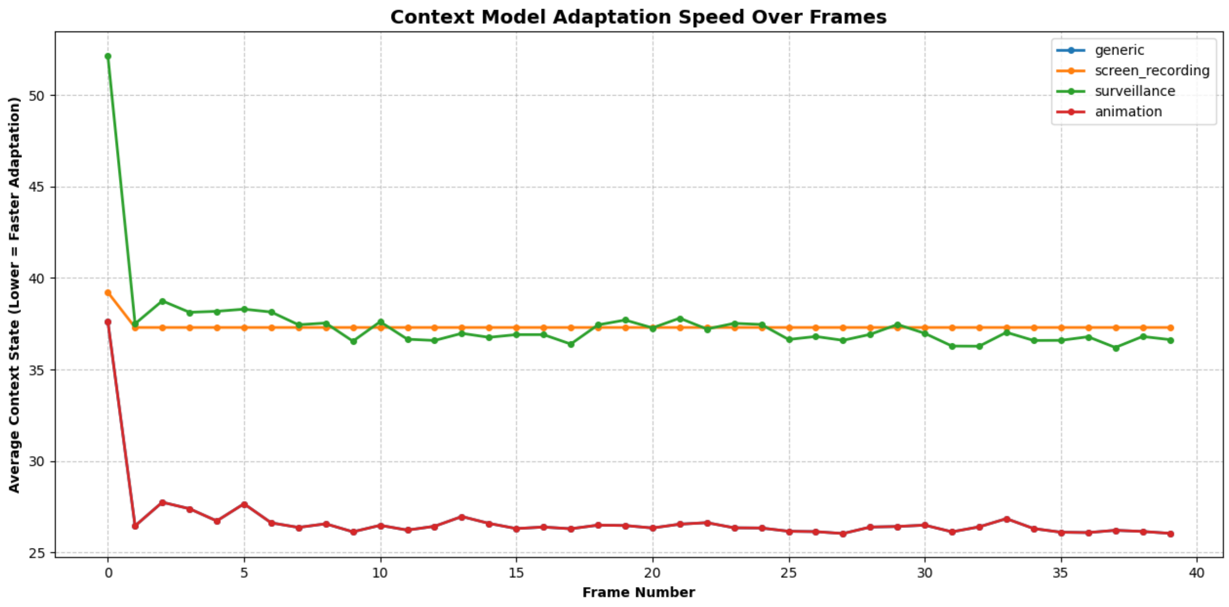


Figure 4: *Adaptation speed plot – generic video*

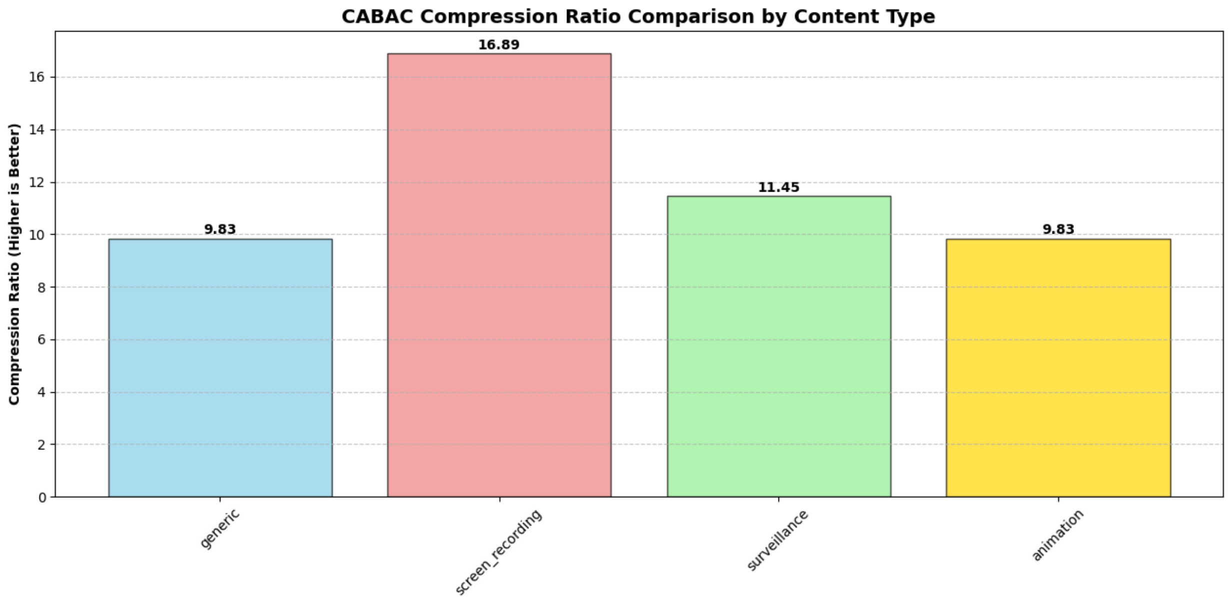


Figure 5: *Compression ratio chart – screen recording*

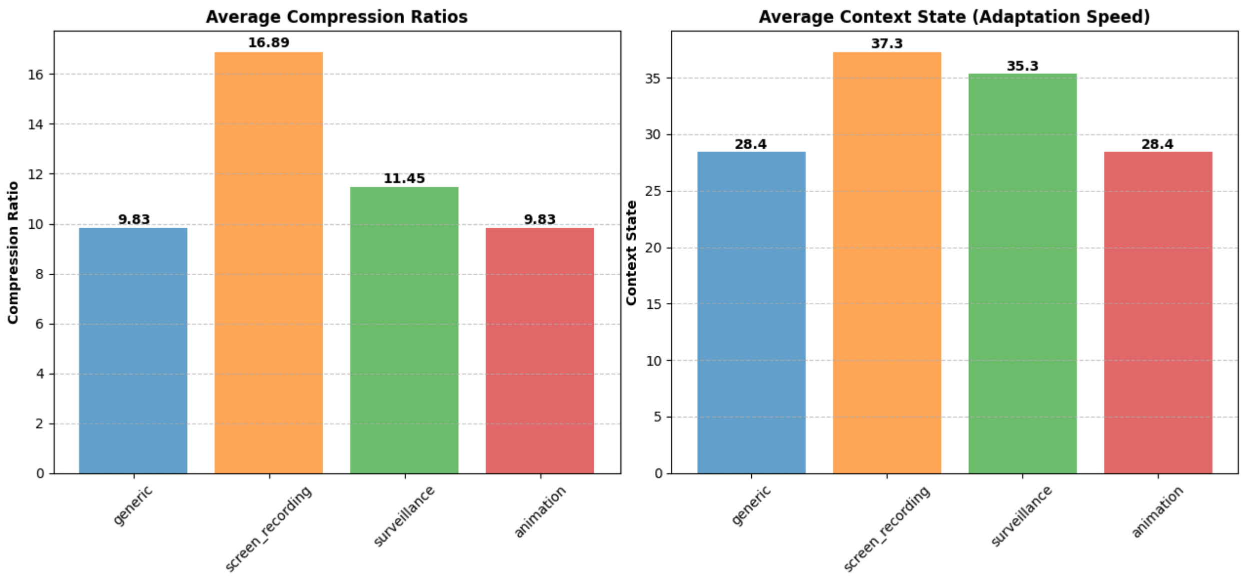


Figure 6: *Adaptation / average context state plot – screen recording*

Discussion

CABAC (Context-Adaptive Binary Arithmetic Coding) is an entropy coding method that works on bits instead of bigger symbols. The basic idea is: instead of always assuming 0 and 1 are equally likely, CABAC keeps a lot of small “context models.” Each context has a guess about which bit (0 or 1) is most probable, and a small state that tells how confident it is. When we encode bits, we update the state depending on whether our guess was right or wrong. Over time the contexts learn the statistics of the video.

In our project we wrote a simplified CABAC encoder in software and used it on two kinds of video: a “generic” clip (an AI-generated person walking in a city square) and a “specialized” clip (a screen recording of Visual Studio Code where we scroll through code). For each clip we tried four different CABAC setups: a generic model, a model tuned for screen recordings, one for surveillance, and one for animation. The difference between these models is mainly how they map pixels and neighbours to context IDs. We always kept the context states between frames so the models could adapt across the whole sequence, similar to a real video codec.

For the generic walking video, we analysed 40 frames. Here the compression results were: about 0.69× for the generic model, 2.81× for the screen-recording model, 1.00× for surveillance and 0.69× for animation. In our implementation, a value above 1 means real compression and below 1 means the coder actually expanded the data a bit. It was a bit surprising that the screen-recording model gave the best result, even though this is not a screen capture. A reasonable explanation is that the scene has big flat areas (pavement, sky) and strong horizontal/vertical edges from the buildings, which look structurally a bit like UI elements. So the “wrong” model accidentally matches the statistics of this clip better than the generic one.

The adaptation plots for the walking video show how the average context state changes over frames. Lower state in our code means the model is more confident and adapts faster. The generic and animation models quickly drop to states around 26–27, while the screen-recording and surveillance models sit higher, around 37–38. So generic and animation clearly adapt more aggressively, but still do not compress best. This is a nice lesson: fast adaptation alone is not enough; if the context mapping is not well aligned with the structure in the video, the probability estimates can still be sub-optimal.

For the screen-recording clip (12 seconds of VS Code, first 50 frames) the picture changes but in a good way for the specialized model. All four models now get high compression because screen content is very regular: lots of repeated text, clean edges and a static layout. The generic and animation models reach about 9.83× compression, surveillance is around 11.45×, and the screen-recording model is best with about 16.89×. Compared to the generic model this is roughly a 48.8% reduction in bits for the same data. This matches the idea of the assignment: a CABAC model that is designed for a specific content type (here: screen recordings) can clearly outperform a generic model on that content.

The adaptation behaviour on the screen recording is also interesting. The screen-recording model's average state stays almost flat around 37.3 during all 50 frames. That means it quickly finds a good probability configuration and then stays stable while the code scrolls. The generic and animation models work at lower states (around 28.4), i.e. they change their probabilities more aggressively, but they still do not reach the same compression ratio. The surveillance model is somewhere in the middle. This suggests that for highly structured content like an editor window, having context models that match the typical patterns (rows of text, clear borders, etc.) is more important than constantly "twitching" the probabilities.

Based on these experiments we can say that the screen-recording CABAC model is the most important and effective one for our target content. It gives the best compression on the VS Code recording and also does surprisingly well on the generic walking video. The results show that CABAC is very sensitive to how we design and choose context models: the same arithmetic coder with different context layouts can behave very differently. They also show that "good" context adaptation is not just about moving the state quickly, but about converging to a probability model that actually matches the statistics of the video sequence.

References

[1] *Happy walking side view reference – Animation Reference Videos*, YouTube. Available at: https://www.youtube.com/watch?v=Mol0lrRBy3g (accessed 15 November 2025).