Unsupervised Video Anomaly Detection Using Convolutional Autoencoders: Design, Deployment, and Comparative Evaluation

Mr. S. Sundeep Kumar
Assistant Professor
Computer Science and Engineering
Sree Dattha Institute of Engineering and Science
Hyderabad, India
sundeep.sp2032@gmail.com

Mr. Chalavadi Jaswanth Kumar
PG Student
Computer Science and Engineering
Sree Dattha Institute of Engineering and Science
Hyderabad, India
nameisjashu@gmail.com

***Abstract*—Automated anomaly-detection methods are required because the fast expansion of surveillance systems has rendered continuous manual monitoring unfeasible. A video anomaly detection system based on an unsupervised convolutional autoencoder trained on typical surveillance footage from the UCF-Crime dataset is presented in this study. The frames were resized to 64×64, converted to grayscale, and normalized before being passed through an encoder–decoder convolutional architecture. Anomalies were identified using the mean squared reconstruction error (MSE) between an input frame and its reconstruction, with frames exceeding a fixed threshold being flagged as anomalous. Alongside the autoencoder, three supervised architectures, a 2D convolutional neural network (CNN), a recurrent architecture combining time-distributed convolutions with an LSTM layer, and a 3D convolutional neural network were independently designed and trained on labeled normal and anomalous segments spanning categories such as arson, explosion, fighting, road accidents, robbery, and shooting. A convolutional autoencoder was chosen for deployment because it learns only from normal video samples and therefore does not rely on labeled anomaly examples during inference, better matching the practical rarity and diversity of real anomalous events. The complete system is deployed as a Flask web application that scores uploaded videos, groups and merges temporally adjacent anomalous frames into discrete events, discards short noise-like detections, and generates an annotated output video, a CSV report, and an email alert. This study describes the full architecture, training pipeline, and deployment workflow, along with a qualitative comparison of four architectures. Quantitative evaluation was not conducted in this phase and was identified as the primary direction for future research.**

Keywords—Video Anomaly Detection; Convolutional Autoencoder; Unsupervised Learning; Video Surveillance; Reconstruction Error; UCF-Crime Dataset

I. INTRODUCTION

Modern surveillance systems play an essential role in ensuring security at transportation hubs, commercial buildings, residential complexes, and public spaces. However, the proliferation of surveillance cameras has created a data volume problem that outpaces the human monitoring capacity. A single facility may generate hundreds of hours of footage daily, the overwhelming majority of which depicts routine and non-anomalous activities. Human monitoring requires considerable time and effort and is vulnerable to fatigue, leading to missed anomalous events, particularly given that the occurrences that surveillance is meant to catch are rare, unpredictable, and often brief. This gap between data volume and human monitoring capacity motivates automated video anomaly detection as a practical necessity rather than a purely academic exercise.

Video anomaly detection is inherently more challenging than standard video classification tasks for two reasons: First, anomalous events are heterogeneous by nature; an anomaly may be a fight, a fire, a road accident, or an act of theft, and these categories share little visual or motion-based structure with one another. A system trained to recognize specific anomaly categories may fail when confronted with a genuinely novel type of anomalous behavior not represented in its training data. Second, labeled anomalous data are inherently scarce relative to labeled normal data because anomalies are infrequent occurrences. These two constraints make purely supervised classification approaches, which require representative labeled examples of every anomaly type to be detected, a poor structural fit for real-world deployment, even though such approaches are straightforward to design and often achieve strong performance when evaluated on the specific anomaly categories present in their training data.

A more practical solution is unsupervised anomaly detection, in which the model learns normal behavior and identifies deviations without requiring labeled anomaly examples. Rather than learning to discriminate between predefined categories, an unsupervised model learns only the distribution of normal behavior and flags deviations from that learned distribution as anomalous, without requiring labeled anomaly examples during training. Autoencoders are a natural architectural choice for this formulation. A convolutional autoencoder trained exclusively on normal footage learns to compress and reconstruct normal frames with low error. When presented with an anomalous frame that falls outside the distribution the network has learned, the reconstruction error

tends to increase, providing a usable anomaly signal without ever needing to see an example of the anomaly itself during training.

This study presents the design, training, and deployment of a video anomaly detection system centered on a convolutional autoencoder trained on normal-class footage drawn from the UCF-Crime dataset. To contextualize this design choice, three supervised alternatives—a 2D convolutional neural network (CNN), a recurrent architecture combining time-distributed convolutions with a long short-term memory (LSTM) layer, and a 3D convolutional neural network—were independently designed and trained on labeled normal and anomalous video segments spanning categories including arson, explosion, fighting, road accidents, robbery, and shooting. These three architectures represent progressively richer treatments of temporal information, from none (CNN, frame-independent) to sequence modeling via recurrence (RNN/LSTM) to joint spatiotemporal convolution (3D CNN). All four architectures were implemented and trained; the convolutional autoencoder was ultimately selected for deployment because its unsupervised formulation aligned with the practical deployment constraint of limited labeled anomaly data, whereas the supervised architectures were retained and discussed as comparative design alternatives.

Beyond model training, this study addresses the deployment gap that separates a trained model from a usable system. The autoencoder is embedded within a complete Flask-based web application that accepts an uploaded surveillance video, performs frame-level anomaly scoring via reconstruction error, and, because raw frame-level predictions are noisy, applies a temporal post-processing pipeline that groups nearby anomalous frames into discrete events, merges events separated by short gaps, and discards events below a minimum duration to suppress spurious single-frame detections. The system further generates a browser-viewable annotated output video, a downloadable CSV report summarizing detected events with timestamps and peak reconstruction error, and an automated email alert when anomalies are detected, forming an end-to-end pipeline from the raw video input to the actionable output.

The remainder of this paper is organized as follows. Section II reviews the related work on video anomaly detection. Section III describes the proposed methodology, including data preprocessing and the four implemented architectures used in this study. Section IV presents the system architecture and deployment designs. Section V details the implementation, including the training configuration, hardware, and deployment settings. Section VI presents the results based on the functional outputs of the system. Section VII discusses the findings, Section VIII addresses the limitations, and Section IX outlines directions for future work, followed by the conclusion in Section X.

II. Literature Review

Over the past several years, researchers have proposed numerous techniques for video anomaly detection, including handcrafted approaches, supervised learning methods, and reconstruction-based deep learning models, which broadly fall into three categories: handcrafted-feature methods, supervised deep learning methods, and unsupervised/semi-supervised deep learning methods based on reconstruction or prediction.

Datasets and Benchmarks. Sultani et al. introduced the UCF-Crime dataset, comprising 1,900 untrimmed real-world surveillance videos spanning 13 anomaly categories (including arson, explosion, fighting, road accidents, robbery, and shooting) alongside normal footage, and proposed a weakly supervised deep multiple-instance-ranking framework that learns from video-level rather than frame-level labels [1]. This dataset and the specific subset of categories used in it form the data foundation for the present work.

Unsupervised Reconstruction-Based Approaches. Hasan et al. proposed learning temporal regularity in video sequences using autoencoders trained exclusively on normal footage, framing anomalies as patterns the network fails to reconstruct accurately, establishing reconstruction error as a viable unsupervised anomaly signal without requiring labeled anomalous examples [2]. Chong and Tay extended this idea with a spatiotemporal autoencoder by combining convolutional layers with convolutional LSTM units to jointly capture appearance and short-term motion regularity [3]. Luo, Liu, and Gao similarly incorporated a convolutional LSTM into an encoder-decoder structure to model the temporal context for anomaly scoring, an approach conceptually related to the RNN architecture implemented in this project, although applied within a reconstruction rather than a purely discriminative framework [4]. The convolutional autoencoder used in the present work is architecturally simpler than these ConvLSTM-based variants, as it operates on individual frames without an explicit temporal memory component, reflecting a deliberate trade-off toward lower model complexity for practical deployment.

Prediction-Based and Generative Approaches. Liu et al. reframed anomaly detection as a future-frame-prediction problem, training a generative network to predict the next frame from preceding frames and using prediction error (combined with adversarial and optical-flow constraints) as the anomaly signal, achieving strong results on standard benchmarks [5]. This line of work builds on the foundational generative adversarial network (GAN) formulation of Goodfellow et al., in which a generator and discriminator are trained adversarially [6]. The same generator/discriminator structure is implemented, but not trained or deployed, in the present project's gan_model.py file.

Supervised and Spatio-Temporal Discriminative Approaches. Beyond unsupervised reconstruction, supervised classification remains common when labeled anomaly data are available. Tran et al. introduced 3D convolutional networks (C3D) that learn spatiotemporal features by convolving jointly across spatial and temporal dimensions, demonstrating that 3D convolutions capture motion information more effectively than 2D convolutions applied frame-by-frame [7], which is the architectural basis for the 3D CNN implemented in this project. Recurrent architectures combining convolutional feature extraction with long short-term memory (LSTM) units, as originally formulated by Hochreiter and Schmidhuber [8], represent an alternative approach to temporal modeling in which per-frame spatial features are extracted independently

before being passed to a recurrent layer for sequence-level reasoning, which is the approach adopted by the RNN architecture in this project.

Autoencoder Foundations. The general principle of using autoencoders to learn compressed representations of normal data, with the reconstruction error serving as a proxy for how well new data conform to that representation, traces back to the foundational work by Hinton and Salakhutdinov on dimensionality reduction with neural networks [9], which underlies the reconstruction-error-based anomaly scoring mechanism used throughout the unsupervised anomaly detection literature, including the present work.

Research Gap. Much of the existing literature evaluates a single architectural paradigm in isolation, either purely reconstruction-based, purely prediction-based, or purely supervised discriminative, typically benchmarked with full quantitative evaluation (precision, recall, AUC) against standard datasets. While many studies emphasize model accuracy, comparatively few describe the complete engineering workflow required to deploy these models in real-world applications from a trained model to a deployed, user-facing system, including practical concerns such as post-processing raw frame-level predictions into coherent temporal events, generating human-readable reports, and issuing automated alerts to users. The present work addresses this applied gap: rather than proposing a novel architecture, it documents the design and training of four established architectural paradigms (2D CNN, RNN with LSTM, 3D CNN, and convolutional autoencoder) applied to the UCF-Crime dataset, the deliberate selection of the unsupervised autoencoder for deployment based on practical labeling constraints, and a complete deployment pipeline including event-level temporal post-processing while explicitly acknowledging that formal quantitative benchmarking of the four architectures against one another was not conducted in this phase of the study.

III. Proposed Methodology

This section describes the complete methodology implemented in this study, covering dataset organization, preprocessing, the four trained architectures, the training procedure, and the deployment-time inference pipeline. Every step described here was drawn directly from the project's source code.

1. *Dataset Description and Class Organization*

The dataset used is the UCF-Crime dataset, organized locally into category-specific subfolders corresponding to the classes used in this project: normal, arson, explosion, fighting, road accidents, robbery, and shooting. As implemented in preprocessing.load_dataset(), each category is assigned a binary label: normal videos are labeled 0, and all six anomaly categories are labeled 1, collapsing the multi-class anomaly taxonomy into a binary normal/anomaly distinction for the supervised architectures (CNN, RNN, 3D CNN). In contrast, the autoencoder is trained exclusively on frames drawn from the normal class, consistent with its unsupervised formulation.

2. *Preprocessing Pipeline*

Preprocessing was implemented in preprocessing.py and applied uniformly across all four architectures.

- Frame Extraction: extract_frames() reads a video via OpenCV and extracts up to a maximum of 70 frames per video.
- Resizing: Each frame was resized to a fixed spatial resolution of 64×64.
- Grayscale Conversion: Frames were converted from BGR to single-channel grayscale.
- Normalization: Pixel intensities were scaled from the [0, 255] range to [0, 1] by dividing by 255.

For the temporal architectures (RNN and 3D CNN), prepare_sequences() further groups the extracted frames into overlapping sequences of 10 consecutive frames using a sliding window, producing the (10, 64, 64, 1) input tensors required by these models. The CNN and autoencoder operate on individual frames of shape (64, 64, 1).

3. *Model Architectures*

To investigate different learning strategies, four independent deep learning architectures were developed, implemented, and trained within the proposed framework, each targeting a different point in the design space between spatial-only modeling, temporal modeling, and supervision.

- Convolutional Neural Network (CNN): A frame-level binary classifier Conv2D(32) → MaxPooling2D → Conv2D(64) → MaxPooling2D → Flatten → Dense(128, relu) → Dense(1, sigmoid) trained with binary cross-entropy loss on individually labeled frames. This architecture serves as a spatial baseline with no temporal context.
- Recurrent Neural Network (RNN, TimeDistributed CNN + LSTM): TimeDistributed(Conv2D(32)) → TimeDistributed(MaxPooling2D) → TimeDistributed(Flatten) → LSTM(64) → Dense(1, sigmoid), operating on 10-frame sequences. Per-frame spatial features are extracted independently before being passed to an LSTM layer that models their evolution across the sequence, producing a single anomaly judgment per 10-frame window size.
- 3D Convolutional Neural Network (3D CNN): Conv3D(32, 3×3×3) → MaxPooling3D → Conv3D(64, 3×3×3) → MaxPooling3D → Flatten → Dense(128, relu) → Dense(1, sigmoid), also operating on 10-frame sequences. Unlike RNNs, spatial and temporal features are learned jointly through volumetric convolution rather than through a separate recurrent stage.
- Convolutional Autoencoder: An unsupervised encoder-decoder. Encoder: Conv2D(32) → MaxPooling2D → Conv2D(16) → MaxPooling2D; Decoder: Conv2D(16) → UpSampling2D → Conv2D(32) → UpSampling2D → Conv2D(1, sigmoid) trained with MSE reconstruction loss

exclusively on normal-class frames. No anomaly labels were used at any point during the training of this model.

4. *Model Selection for Deployment*

Among the four trained architectures, a convolutional autoencoder was selected for deployment in the Flask application. This selection was made based on its unsupervised training formulation: because it requires only normal footage during training, it does not depend on the availability of labeled anomalous examples, which is a practical constraint in real surveillance settings, where anomalous events are rare, diverse, and expensive to annotate exhaustively. The CNN, RNN, and 3D CNN architectures, while fully trained and saved, remain supervised classifiers whose reliability is bounded by the specific anomaly categories represented in their training labels. They are retained in this work as architectural comparison points rather than as deployed components. This project does not claim the quantitative superiority of the autoencoder over the other three architectures, as a comparison was not empirically conducted, but rather justifies its selection on structural/practical grounds appropriate to the deployment scenario.

5. *Deployment-Time Inference Pipeline*

At inference time, the deployed system operates as follows, as implemented in the app.py:

- An uploaded video is read frame-by-frame using OpenCV and processed in batches of 32 frames.
- Each frame was resized to 64×64, converted to grayscale, and normalized, mirroring the training-time preprocessing.
- Each batch was passed through the trained autoencoder, and the mean squared error between each input frame and its reconstruction was computed.
- A frame is flagged as anomalous if its MSE exceeds the fixed threshold of 0.12.
- The flagged frames were annotated with a colored label ("Anomaly"/"Normal") and written to an output video.

6. *Event-Level Post-Pipeline*

Because per-frame reconstruction error classification is inherently noisy, three post-processing stages are applied to raw frame-level detections to produce coherent anomaly *events* rather than isolated flagged frames:

- Event Grouping: Consecutive anomalous frames within a 5-frame gap of one another are grouped into a single candidate event.
- Event Merging: Candidate events separated by less than 1.0 s are merged into one event.
- Noise Discarding: Events with a duration shorter than 0.30 s are discarded as likely spurious single-frame detections.

Each surviving event is enriched with human-readable start/end timestamps, duration, and peak MSE, which populate the CSV report, email alert body, and web dashboard.

IV. System Architecture

1. *Offline Training Phase Architecture*

The complete system is organized into two distinct phases: an offline training phase, in which the four architectures are trained and saved to disk, and a deployment phase, in which the saved autoencoder is loaded into a Flask web application that serves the end users. This section describes both the phases and the components that connect them.

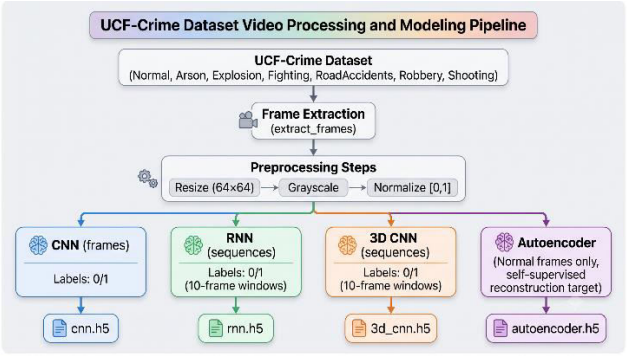

Fig. 1. Offline training phase

2. *Deployment Phase Architecture*

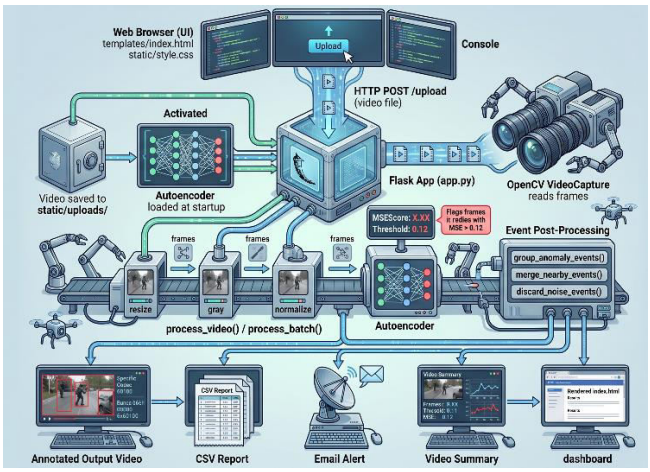

Fig. 2. Deployment phase architecture

3. *Component Responsibilities*

The developed framework comprises multiple interconnected modules that collectively perform video processing, anomaly detection, reporting, and user interaction, each of which is responsible for a specific functionality within the overall framework.

- Presentation Layer: The presentation layer consists of templates/index.html, static/style.css, and static/bg1.png files. It provides a user-friendly web interface that enables users to upload surveillance videos, view detection results, watch processed videos, download anomaly reports, and monitor the processing status using an animated loading interface.
- Application and Routing Layer: The application layer was implemented in app.py using the Flask framework. It manages HTTP requests, handles video uploads,

coordinates the anomaly detection workflow, invokes the required processing modules, and renders the final results on a web interface.

- Model Layer: The model layer utilizes the trained autoencoder implemented in src/autoencoder_model.py. During application startup, the trained weights are stored in the models/autoencoder.h5 file, loaded into memory, and used to perform reconstruction-based anomaly detection on uploaded surveillance videos.
- Detection and Scoring Module: The detection module is implemented using the process_video() and process_batch() functions in app.py. It performs frame extraction, image resizing, grayscale conversion, normalization, batch processing, reconstruction using an autoencoder, computation of the Mean Squared Error (MSE), and threshold-based anomaly classification.
- Temporal Post-processing Module: To improve detection reliability, the system applies three post-processing operations: group_anomaly_events(), merge_nearby_events(), and discard_noise_events(). These functions combine consecutive anomalous frames into meaningful events, merge temporally adjacent events, and eliminate short-duration false alarms, thereby reducing the noise in the final detection results.
- Reporting Module: The reporting component generates structured anomaly reports using the generate_csv_report(), enrich_events_with_formatted_fields(), and format_email_body() functions. This module records anomaly timestamps, event durations, reconstruction errors, and summary information, which are stored as downloadable comma-separated value (CSV) reports.
- Notification Module: The notification module employs the send_anomaly_alert() function using Python's smtplib library with Gmail SMTP services. Whenever anomaly events are detected, an automated email notification containing the detection summary is generated and sent to the configured recipients.
- Storage Module: The storage component manages runtime data using three dedicated directories: static/uploads/ for uploaded videos, static/processed/ for annotated output videos, and static/reports/ for the generated anomaly reports. These directories facilitate the organized storage and retrieval of all application outputs.
- Offline Training Module: The offline training process was implemented using src/train.py, src/preprocessing.py, src/cnn_model.py, src/rnn_model.py, and src/cnn_3d_model.py. This module performs dataset preprocessing, model training, and weight generation for the CNN, RNN, 3D CNN, and Autoencoder architectures. The trained models were subsequently stored in the models/ directory for deployment.
- Exploratory Module: The project also contains src/gan_model.py, which defines the Generator and Discriminator architectures of a Generative Adversarial Network (GAN). However, this module was neither trained nor integrated into the deployed system and is therefore considered an exploratory component for future research.

4. *Design Rationale*

The architecture separates concerns cleanly along two axes: (1) training vs. deployment the four architectures are trained entirely offline, decoupling model development from the live application, and only the autoencoder's weights are loaded at Flask startup; and (2) detection vs. presentation the core detection logic (process_video, process_batch, event post-processing) is decoupled from the presentation layer (Jinja2 templates), with app.py acting as the orchestration point that passes structured results (video_summary, anomaly_events, avg_mse) to the template for rendering. This separation, while not indicative of a formally layered/microservice architecture, reflects a reasonable monolithic-but-modular design appropriate for a single-deployment academic prototype.

V. Implementation Details

1. *Development Environment*

The project was developed in Python, with core dependencies pinned in requirements.txt: TensorFlow 2.15.0, OpenCV (opencv-python) 4.9.0, NumPy 1.26.4, Matplotlib 3.8.3, and scikit-learn 1.4.1. Flask was used for the web application layer but was installed separately during development and was not pinned in requirements.txt. All four models were trained on a CPU only, with no GPU acceleration.

2. *Training Configuration*

During the model training phase, all the deep learning models were trained for 10 epochs using a batch size of 32 with the Adam optimizer. A validation split of 0.2 was employed using the Keras validation_split parameter to monitor the model performance during training. The CNN, RNN, and 3D CNN models were optimized using the Binary Cross-Entropy loss function for binary anomaly classification, whereas the autoencoder was trained using the Mean Squared Error (MSE) loss function to minimize the reconstruction error between the input and reconstructed frames. During training, accuracy was tracked as the evaluation metric for the CNN, RNN, and 3D CNN models; however, no separate post-training evaluation metrics, such as precision, recall, F1-score, or ROC-AUC, were computed as part of the current implementation.

3. *Data Preparation Recap*

Frames were extracted via OpenCV (extract_frames(), maximum 70 frames/video), resized to 64×64, converted to grayscale, and normalized to [0,1]. For the RNN and 3D CNN, the frames were grouped into overlapping 10-frame sequences via a sliding window (prepare_sequences()). No data augmentation was performed.

4. *Deployment Configuration*

The trained convolutional autoencoder was integrated into a Flask-based web application for deployment. During inference, the uploaded videos were processed frame by frame in batches of 32. Each frame was resized to 64×64 pixels, converted to grayscale, normalized, and passed through the trained autoencoder. An anomaly was detected when the

reconstruction error exceeded a predefined threshold of 0.12. The processed video, anomaly report, and event summary are then generated and presented to the user via a web interface.

The CNN, RNN, and 3D CNN models were formulated as binary classification models and optimized using a binary cross-entropy loss function. These models were trained to classify the input samples as normal or anomalous. During training, the accuracy was monitored as the primary evaluation metric to assess the learning performance of the supervised models.

In contrast, the autoencoder was trained using only normal surveillance frames in an unsupervised manner. The model was optimized using the Mean Squared Error (MSE) loss function, which minimizes the reconstruction error between the input and reconstructed frames. Unlike supervised models, autoencoders do not directly predict anomaly classes; instead, anomalies are identified during inference by comparing the reconstruction error against a predefined threshold. Although training accuracy was monitored for the supervised models, the current implementation did not include separate post-training evaluation metrics such as precision, recall, F1-score, ROC-AUC, or confusion matrix, which are considered potential enhancements for future work.

5. *Web Application Implementation*

The Flask application exposes two routes: (renders the upload form via templates/index.html) and /upload (POST endpoint that accepts a video file, saves it to static/uploads/, runs the detection pipeline, and re-renders index.html with the results). The autoencoder is loaded once at the application startup (not per request), avoiding repeated model loading overhead across uploads. Processed videos are served via a dedicated static/processed/<filename> route in addition to Flask's default static file handling. The frontend (static/style.css) implements a dark terminal-themed UI with a JavaScript-driven loading overlay that cycles through stage messages (e.g., Extracting Frames, Running Autoencoder) during the synchronous upload/processing request, and a results dashboard showing a video summary card, a list of detected anomaly events with timestamps, an embedded video player for the annotated output, and download buttons for both the processed video and the CSV report.

6. *Email Notification*

When one or more anomaly events survive the post-processing pipeline, send_anomaly_alert() sends an email via Gmail's SMTP_SSL server (port 465) using smtplib, with the message body generated by format_email_body() listing the start/end time, duration, and peak MSE of each event. The sender/receiver credentials in the codebase are placeholder values (your_email@gmail.com, your_app_password_here) and were intentionally removed prior to sharing the project; the email functionality is implemented but requires valid credentials to be configured for actual delivery of the email.

VI. Results

Because quantitative performance evaluation was outside the scope of this implementation, this section focuses on demonstrating the successful execution and functional behavior of the developed system.

1. *Successful Environment Setup and Execution*

Figure 3 confirms the complete project structure, including the populated data/ directory with the seven UCF-Crime category subfolders and the four trained.h5 model files in models/. The integrated terminal shows that app.py launched successfully, with the autoencoder weights loading without error, confirming the correct environment setup and successful Flask application startup.

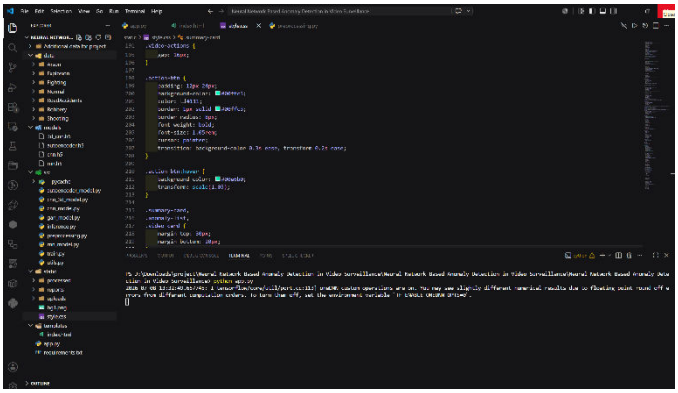

Fig. 3. Successful Environment Setup and Execution

2. *Web Interface*

Figure 4 shows the deployed application running locally, displaying the upload interface with the dark terminal-themed styling described in Section IV, confirming that the Flask routing, Jinja2 template rendering, and static asset serving function as implemented.

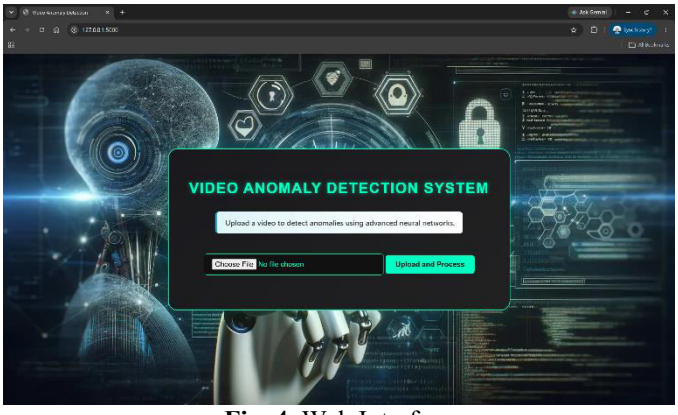

Fig. 4. Web Interface

3. *End-to-End Detection Workflow*

A recorded demonstration confirms the complete pipeline operating end-to-end on an uploaded surveillance video: file selection, submission triggering the animated multi-stage loading overlay, and rendering of the results dashboard upon completion. The dashboard displays the video summary card,

a list of detected anomaly events (each annotated with start time, end time, duration, and peak reconstruction MSE), an embedded player for the annotated output video, and download buttons for the processed video and the CSV report. This confirms that the frame batching, autoencoder inference, threshold-based flagging, event grouping/merging/noise-discarding chain, CSV report generation, and dashboard rendering all execute correctly together on a real, non-trivial video input, without runtime errors.

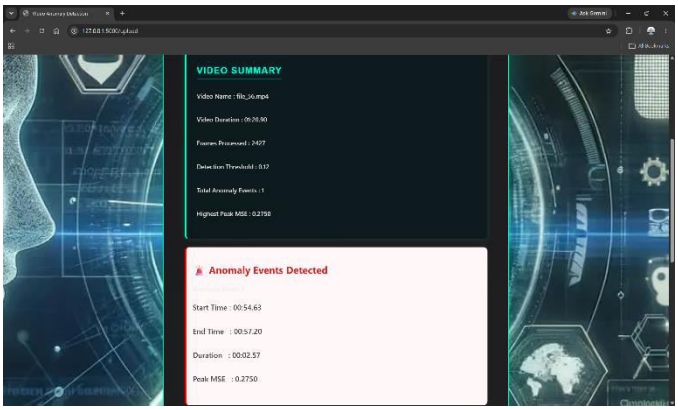

Fig. 5. Video Summary and Anomaly Event Detection Dashboard

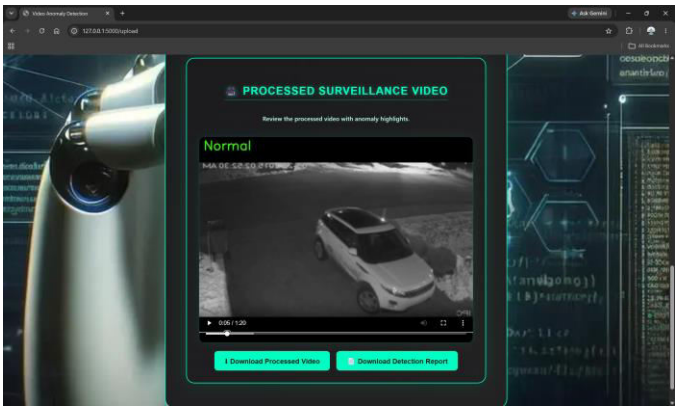

Fig. 6. Processed Surveillance Video During Normal Activity

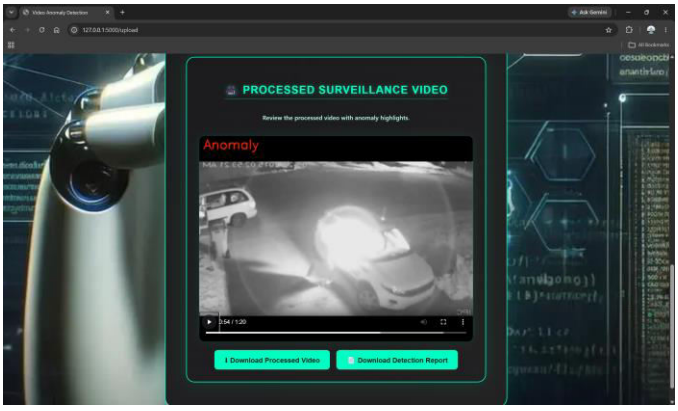

Fig. 7. Processed Surveillance Video During Anomaly Detection

4. Scope of Reported Results

Consistent with the evaluation protocol described in Implementation Details, this paper reports only that the system functions correctly end-to-end and produces its intended outputs (annotated video, event list, CSV report, and dashboard). No accuracy, precision, recall, F1-score, ROC-AUC, confusion matrix, inference-time/FPS measurement, or comparative performance figures between the four trained architectures are reported, as none were computed during this implementation phase. This is stated here as an explicit scope boundary rather than an oversight and is addressed further in Limitations and Future Work.

VII. Discussion

1. Unsupervised Approach Validated in Practice

The system confirms that a reconstruction-error-based autoencoder can be trained and deployed end-to-end without any labeled anomaly examples, unlike the CNN, RNN, and 3D CNN, which all required labeled anomaly categories. This directly supports the practical rationale given in Section III for selecting the autoencoder for deployment.

2. Value of Event-Level Post-Processing

Frame-level anomaly predictions often contain isolated false detections, making additional post-processing necessary to generate meaningful anomaly events. The grouping-merging–discarding pipeline converts this into coherent, timestamped events, meaningfully improving output usability without additional model training. However, since no event-level ground-truth evaluation was performed, its quantitative benefit (e.g., false-positive reduction) remains unverified.

3. Architectural Trade-offs, Not Benchmarked Performance

The four architectures represent different design points: the CNN is fast but temporally blind; the RNN and 3D CNN both model temporal context (via recurrence vs. joint 3D convolution, respectively) but require labeled data and a fixed window; the autoencoder sacrifices temporal modeling and supervision for label-free training. This is a design-level comparison only; no accuracy, precision, recall, or AUC data exists to support an empirical performance claim between them, and none is made.

4. Threshold Choice Is Functional but Unvalidated

The fixed MSE threshold (0.12) was set as a development-time constant, not derived via a validation sweep or ROC-based procedure. The demo shows it works on the cases exercised, but this qualitative observation doesn't substitute for systematic tuning.

5. Nature of the Contribution

This work's value lies in integrating an established technique (autoencoder-based reconstruction anomaly detection) into a complete, deployed pipeline training, inference, event aggregation, reporting, and alerting rather than in proposing a novel architecture or claiming state-of-the-art performance.

VIII. Limitations

- No quantitative evaluation. No accuracy, precision, recall, F1-score, or AUC was computed for any of the four models; results reported are functional, not benchmarked.
- Unvalidated detection threshold. The deployed MSE threshold (0.12) is a manually set constant, not derived from a threshold sweep or validation procedure.
- Frame-level, not video-level, data split. Validation splitting occurs at the frame/sequence level, creating a possible data-leakage risk between train and validation partitions.
- No event-level ground-truth evaluation. The event grouping/merging/noise-discarding pipeline has not been validated against annotated anomaly intervals.

IX. Future Work

1. *Quantitative Evaluation Framework*

Implement accuracy, precision, recall, F1-score, ROC-AUC, and confusion matrices for all four architectures, enabling a genuine empirical comparison rather than the architectural/design-level comparison presented in this paper.

2. *Video-Level Data Splitting*

Restructure the train/validation split to operate at the video level rather than the frame/sequence level, eliminating the potential data-leakage risk identified in Limitations, along with introducing a properly held-out test set.

3. *Threshold Optimization*

Replace the fixed 0.12 MSE threshold with a systematically validated value, selected via a threshold sweep or ROC-based analysis on a labeled validation set; explore adaptive, per-video thresholding as a further extension.

4. *Event-Level Evaluation*

Evaluate the event grouping/merging/noise-discarding pipeline against ground-truth temporal anomaly annotations (available for portions of UCF-Crime) to quantify its actual contribution to detection quality.

5. *GAN-Based Anomaly Detection*

Train and evaluate the existing generator/discriminator scaffolding (gan_model.py) as an alternative unsupervised approach, comparing adversarial-based detection against the reconstruction-error-based autoencoder.

X. Conclusion

This study presents the design, implementation, and deployment of an unsupervised video anomaly detection framework based on a convolutional autoencoder trained using normal surveillance videos from the UCF-Crime dataset. Alongside the autoencoder, three supervised architectures: a 2D-CNN, a TimeDistributed CNN with LSTM, and a 3D CNN were independently designed and trained on labeled normal and anomalous video segments, providing architectural points of comparison. The autoencoder was selected for deployment on the practical basis that its unsupervised formulation does not require labeled anomaly examples, better matching the real-world constraint that anomalous events are rare and diverse. The complete system was implemented as a Flask web application that performs frame-level reconstruction-error scoring, aggregates noisy frame-level detections into coherent anomaly events through a grouping-merging-noise-discarding pipeline, and produces an annotated output video, a downloadable CSV report, and an automated email alert. The system was demonstrated to function correctly end-to-end on real video input. This work's contribution lies primarily in this complete, integrated deployment pipeline rather than in a novel model architecture or a benchmarked performance claim; no quantitative evaluation (accuracy, precision, recall, F1-score, or AUC) was conducted in this phase, and this is stated explicitly as a scope boundary rather than an oversight. The identified limitations particularly the absence of quantitative evaluation, an unvalidated detection threshold, and a frame-level rather than video-level data split define clear, actionable directions for future work, including a systematic evaluation framework, threshold optimization, and event-level validation against ground-truth annotations. Taken together, this study demonstrates a practically deployable, unsupervised approach to video surveillance anomaly detection and establishes a foundation for a more rigorous empirical evaluation planned as the next step.

References

[1] W. Sultani, C. Chen, and M. Shah, "Real-World Anomaly Detection in Surveillance Videos," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6479–6488.

[2] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis, "Learning Temporal Regularity in Video Sequences," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 733–742.

[3] Y. S. Chong and Y. H. Tay, "Abnormal Event Detection in Videos Using Spatiotemporal Autoencoder," in *Proc. Int. Symp. Neural Networks (ISNN)*, Springer, 2017, pp. 189–196.

[4] W. Luo, W. Liu, and S. Gao, "Remembering History with Convolutional LSTM for Anomaly Detection," in *Proc. IEEE Int. Conf. Multimedia and Expo (ICME)*, 2017, pp. 439–444.

[5] W. Liu, W. Luo, D. Lian, and S. Gao, "Future Frame Prediction for Anomaly Detection – A New Baseline," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6536–6545.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[7] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning Spatiotemporal Features with 3D Convolutional Networks," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2015, pp. 4489–4497.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[10] P.G. R., M. HM, M. Shamil, and C. R. Mishra, "Anomaly Detection for Video Surveillance," *International Journal of Scientific Research in Engineering and Management (IJSREM)*, vol. 8, no. 5, May 2024, ISSN: 2582-3930, DOI: 10.55041/IJSREM34103.