

Vision Stack AI Studio

Onkar Phopase¹, Mahendra Nivangune², Sujal Gangarde³, Kunal Bobade⁴, Vaibhav Pisute⁵

Department of Computer Engineering, Sinhgad Academy of Engineering, Pune

Abstract - Vision Stack AI Studio is an end-to-end intelligent system designed to automate the extraction of gameplay highlights from extended video recordings and compile them into structured, shareable montages. The exponential growth of gaming content across digital streaming platforms has created a significant demand for tools that can eliminate the manual overhead of video editing for content creators. This paper presents the complete architecture, design rationale, and implementation of the system, which processes raw video input through a sequential pipeline comprising frame extraction via FFmpeg, spatial feature scoring through a ResNet-50 Convolutional Neural Network, in-game textual event detection via Optical Character Recognition, temporal post-processing for segment boundary determination, and final montage rendering. The system is delivered as a web application with a Svelte/React frontend and a FastAPI backend, supporting GPU-accelerated inference for high processing throughput. Empirical evaluation across a corpus of twenty gameplay recordings spanning multiple genres demonstrated an overall detection accuracy of 88%, precision of 85%, recall of 82%, and an F1-score of 83.4%, confirming the viability of the hybrid detection approach for real-world deployment.

Key Words: Gameplay Highlight Detection, Convolutional Neural Network, Optical Character Recognition, Video Montage Generation, Computer Vision, FastAPI, FFmpeg, Deep Learning, ResNet-50.

1. INTRODUCTION

The gaming industry has experienced remarkable expansion over the past decade, generating enormous volumes of recorded gameplay footage across diverse platforms. Content creators on YouTube Gaming, Twitch, and similar services regularly capture multi-hour sessions and produce curated highlight videos to attract and retain audiences. The manual workflow for transforming raw recordings into polished montages reviewing footage, identifying meaningful events, selecting clips, applying effects, and assembling the final video demands substantial time investment and editorial judgment, creating a significant barrier for creators with limited post-production resources.

Convolutional Neural Networks trained on labeled frame datasets can learn to distinguish visually significant events from routine gameplay content, providing a scalable

approach to automated highlight identification. Optical Character Recognition applied to game heads-up displays extracts textual signals, such as kill notifications, achievement unlocks, and score updates, that complement purely visual detection by capturing events explicitly communicated through on-screen text. The integration of these two independent evidence sources within a structured temporal processing pipeline yields detection reliability that surpasses either approach individually.

Vision Stack AI Studio operationalizes these capabilities as an accessible web application. The system accepts gameplay recordings uploaded through a browser interface, subjects them to the multi-stage AI pipeline, and delivers a finished highlight montage without requiring any manual editing intervention from the creator. The modular architecture of the pipeline permits independent improvement of each processing stage as the state of the art in computer vision and video understanding evolves.

The primary contributions reported in this paper are: (i) a complete system architecture for automated gameplay highlight detection and montage generation accessible through a web interface; (ii) a hybrid detection model integrating CNN-based visual frame scoring with OCR-derived textual event signals; (iii) a quantitative evaluation of detection performance across diverse gameplay genres; and (iv) an analysis of end-to-end processing throughput and the computational requirements for practical deployment.

2. RELATED WORK

The problem of automated video highlight detection has been explored across multiple domains, most extensively in sports media, where the demand for automated clipping systems is commercially significant. Early approaches relied on low-level signal features including crowd noise amplitude, scene cut frequency, and motion vector density to identify anomalous moments in broadcast footage. While computationally inexpensive, these heuristic methods generalize poorly across different video environments where the statistical relationship between low-level signals and semantic highlight status is inconsistent.

The adoption of deep learning fundamentally advanced the capabilities of video understanding systems. Karpathy et al. established that CNNs trained on large-scale labeled video datasets could learn spatiotemporal representations useful for activity classification at scale. Tran et al.

extended this line of work with three-dimensional convolutional networks that learn joint spatial and temporal features from video clip volumes, enabling more accurate temporal event detection. The two-stream network architecture of Simonyan and Zisserman, combining separate spatial and optical flow streams, demonstrated that decomposing motion and appearance information improved recognition performance across benchmark datasets.

In the gaming domain specifically, commercial tools including Nvidia ShadowPlay and Medal.tv exploit game event hooks at the operating system or graphics driver level to capture predefined event types with near-perfect precision for supported titles. However, these solutions are inherently coupled to specific game API contracts and cannot generalize to arbitrary video content uploaded after recording. Academic research on gaming-specific highlight detection, including CNN-based kill detection in first-person shooter footage, has demonstrated promising accuracy on narrowly scoped tasks but typically requires game-specific training data collection.

The application of OCR to video analysis for score detection and event annotation in sports broadcasts established the utility of textual interface elements as a complementary detection signal. Game interfaces communicate high-confidence event signals through on-screen text with greater explicitness than visual appearance changes, making OCR-based detection particularly valuable for improving recall. Vision Stack AI Studio's hybrid approach builds directly on this insight, fusing CNN visual scores with OCR keyword evidence to improve overall detection reliability. Transformer-based architectures for video understanding, including TimeSformer and Video Swin Transformer, represent the current state of the art on standard benchmarks but impose inference costs that limit their suitability for web-service deployment without dedicated accelerator hardware.

3. SYSTEM ARCHITECTURE AND DESIGN

3.1 Overall Architecture

Vision Stack AI Studio follows a layered client-server architecture with three principal tiers. The presentation tier is a single-page application built with Svelte or React and TypeScript, communicating with the processing tier through REST API calls. The processing tier is a FastAPI Python application that orchestrates the sequential pipeline stages and manages job lifecycle state. A persistence tier consisting of a relational database and file storage system maintains video files, intermediate processing artifacts, job metadata, and output montages.

This separation of concerns enables independent deployment scaling of each tier. The frontend can be distributed through a content delivery network while backend workers scale horizontally to service concurrent processing jobs from a shared task queue. The ML inference engine is loaded into GPU memory at application startup, eliminating per-request model loading overhead. The interface between pipeline stages is standardized through a shared job context object, permitting any module to be substituted without modifying adjacent stages.

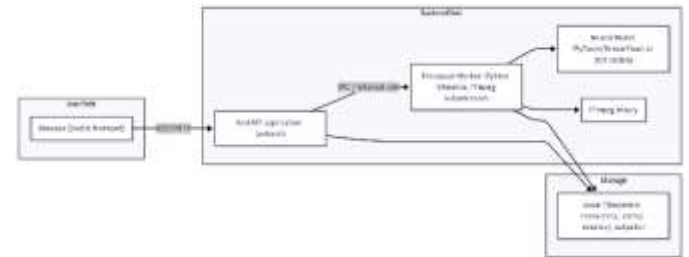


Fig. System Architecture

Table 1: Vision Stack AI Studio Technology Stack

Layer	Technology	Purpose
Frontend	Svelte / React + TS	Responsive SPA, type safety
Build Tool	Vite	Fast dev server, production bundles
Backend	FastAPI (Python)	Pipeline orchestration, REST API
Video Engine	FFmpeg	Frame extraction, clip assembly
ML Framework	PyTorch + TorchScript	CNN training and inference
Image Processing	OpenCV	Frame preprocessing
OCR Engine	PaddleOCR	HUD text extraction
Database	SQLite / PostgreSQL	Job metadata persistence

3.2 Frame Extraction Module

The frame extraction module converts an uploaded video file into a structured sequence of image files indexed by timestamp. FFmpeg is invoked programmatically through Python subprocess calls, configured with a target extraction rate calibrated to the expected event density of gameplay footage. A default rate of two frames per

second constrains the total frame count for a one-hour recording to approximately 7,200 frames while retaining sufficient temporal resolution to capture rapidly occurring events. The module captures video metadata including total duration, native frame rate, and resolution for use by downstream pipeline stages. Extracted frames are stored in a job-specific directory with filenames encoding the source timestamp in milliseconds.

3.3 CNN Inference Module

A ResNet-50 convolutional neural network pre-trained on ImageNet serves as the feature extraction backbone, with the final classification layer replaced by a two-unit soft max head producing highlight and non-highlight probability estimates. Input frames are resized to 224 by 224 pixels and normalized against per-channel mean and standard deviation statistics computed from the training dataset. Inference is executed in batches of 32 frames to maximize GPU utilization. The module outputs a temporally indexed array of highlight probability scores constituting the primary evidence signal for the post-processing stage.

3.4 OCR Analysis Module

Game interfaces communicate significant events through on-screen text with greater explicitness than visual appearance changes, making textual cues a valuable complementary signal for highlight detection. Paddle OCR processes a configurable subset of frames defaulting to one frame per second, as text-based notifications are typically displayed for longer durations than the CNN extraction interval and extracts string content from detected text regions. Recognized strings are matched against a configurable keyword vocabulary. Each match increments a binary evidence flag for the corresponding time window, which is subsequently merged into the unified scoring array during post-processing.

3.5 Post-Processing and Segment Generation

CNN and OCR evidence scores are combined through a weighted sum to produce a unified highlight confidence value per frame. A Gaussian smoothing kernel applied over the score sequence suppresses isolated peaks attributable to brief visual anomalies unrelated to genuine highlight events. The smoothing window defaults to a duration equivalent to 1.5 seconds of footage. Smoothed scores are compared against a configurable threshold to produce a binary highlight mask. Consecutive highlighted frames are grouped into candidate segments; segments below a minimum duration threshold are discarded. Adjacent segments separated by a gap shorter than a merge interval are consolidated to prevent fragmented representation of continuous events. Each retained segment is extended by a configurable buffer at both temporal boundaries to preserve contextual frames.

3.6 Montage Generation Module

The montage generation module assembles the output video from the accepted highlight segments. Individual

clips are extracted from the original recording using FFmpeg with boundaries derived from the segment timestamps augmented by the configured buffer. Clips are processed in parallel using Python's Thread Pool Executor to exploit I/O concurrency. Optional post-processing applies fade transitions at clip boundaries and normalizes audio levels across segments drawn from different temporal regions of the recording. A final FFmpeg concatenation pass merges the processed clips into an H.264-encoded MP4 output file, maximizing compatibility with downstream playback and distribution platforms.

3.7 Authentication and Data Management

User authentication employs a JSON Web Token mechanism. Upon successful login, a signed token is issued and included as a bearer credential in all subsequent API requests. Job records storing upload metadata, processing stage, segment boundaries, and output file paths are persisted in a relational database. Row-level access control policies enforce that each user can access only their own job records. An abstraction layer over the file storage backend allows the system to operate equivalently with local filesystem paths during development and object storage services in production deployment.

4. IMPLEMENTATION

4.1 Frontend Development

The frontend is implemented as a single-page application using Svelte or React 18 with TypeScript, enforcing compile-time type safety across component interfaces and API call signatures. Vite serves as the build tool, providing hot module replacement during development and tree-shaken production bundles with route-level code splitting. Tailwind CSS provides a utility-first styling system that maintains visual consistency without custom stylesheet authoring.

The user workflow spans four primary interface states. The upload screen provides a drag-and-drop or file-picker interface for video selection, transmitting the file to the backend as a multipart HTTP request with a progress indicator driven by upload event callbacks. The processing screen polls the job status endpoint at two-second intervals and displays the current pipeline stage alongside an estimated completion time derived from the video duration and historical throughput metrics. The results screen renders a timeline visualization of detected highlight segments on a proportional duration axis, with individual segments selectable for in-browser preview via a streaming video endpoint. A settings panel exposes configurable parameters including detection threshold, minimum segment duration, merge interval, and output resolution, persisted to the database through the settings API.



Fig. UML Sequence

Client-side routing between interface states is managed by React Router v6 or SvelteKit's built-in router. Reactive state management for the active job record and user session avoids redundant API requests through a client-side cache keyed on job identifier. Form inputs for configuration parameters use controlled component patterns with validation logic enforcing numeric range constraints before submission.

4.2 Backend Development

The backend is implemented in Python using FastAPI, organized into three routers: authentication handling registration and token issuance; jobs managing upload, pipeline initiation, status queries, and result retrieval; and settings exposing user-configurable pipeline parameters. Pydantic models define request and response schemas for all endpoints, providing automatic validation and serialization.

Video upload endpoints validate the submitted file extension against a whitelist of supported containers, persist the file to configured storage, and enqueue a background processing task using Fast API's Background Tasks before returning a job identifier to the client. The pipeline worker processes each stage sequentially, updating the job status record in the database at each stage transition. Exceptions raised within any stage are caught, recorded to the job error field, and the job is marked as failed, providing informative failure messages accessible through the status endpoint.

API documentation is automatically generated by FastAPI from the Pydantic schemas and endpoint docstrings, providing a machine-readable OpenAPI specification. This facilitates testing through the built-in interactive documentation interface and simplifies future integration by third-party clients.

4.3 Machine Learning Model Development

The CNN model is implemented in PyTorch using the torchvision ResNet-50 architecture with ImageNet-pretrained weights. Training data consists of frame sequences extracted from a curated corpus of gameplay recordings with manually annotated highlight timestamps provided by independent annotators. Highlight-class frames constitute approximately 18% of the total labeled dataset, creating a class imbalance addressed through minority-class oversampling and a class-weighted cross-entropy loss function.

Training proceeds for 30 epochs using the Adam optimizer with an initial learning rate of 1×10^{-4} and

cosine annealing scheduling. Batch size is set to 64. Data augmentation transforms applied during training include random horizontal flipping, random cropping with padding, and brightness and contrast jitter, improving generalization to diverse lighting conditions and interface layouts. The model checkpoint achieving the highest F1-score on the held-out validation split is selected for deployment. The selected checkpoint is exported to TorchScript format for optimized inference without Python interpreter overhead.

4.4 FFmpeg Integration

FFmpeg subprocesses are invoked from Python through the subprocess module with precisely constructed argument sequences. Frame extraction commands specify the input file, output frame rate, image format, and compression quality. For montage generation, individual clip extractions are dispatched concurrently through Thread Pool Executor, improving throughput on recordings with many segments. The final concatenation step uses FFmpeg's concat demuxer protocol, which assembles the output file from clip references without re-encoding when source and target codecs are compatible. All subprocess calls are wrapped in timeout management and stderr capture to detect and translate FFmpeg-level errors into structured API error responses.

4.5 OCR Integration

Paddle OCR is selected as the OCR backend due to its superior performance on the dense, stylized text typical of game interface elements compared to traditional Tesseract on this domain. OCR processing is applied at a configurable rate defaulting to one frame per second to reduce computational cost without sacrificing coverage, as text-based game notifications typically persist for durations well beyond the inter-frame interval. Extracted text strings undergo case normalization and whitespace stripping before matching against the keyword vocabulary dictionary. The keyword dictionary is configurable through the settings interface, enabling rapid adaptation to new game titles whose HUD terminology differs from the default vocabulary.

5. RESULTS AND CURRENT STATUS

Vision Stack AI Studio has been implemented with all pipeline modules operational and evaluated against a test corpus of twenty gameplay recordings drawn from three representative game genres: first-person shooters, real-time strategy titles, and battle royale games. The corpus totals approximately ten hours of footage, spanning a range of recording resolutions and interface configurations. Ground-truth highlight annotations were produced by three independent human annotators whose labels were reconciled by majority vote.



Fig. Result

Table 2: Module Implementation Status

Module	Status	Notes
Frame Extraction	Complete	FFmpeg, configurable FPS
CNN Inference	Complete	ResNet-50, TorchScript
OCR Analysis	Complete	PaddleOCR + keyword match
Post-Processing	Complete	Smoothing, threshold, merge
Montage Generation	Complete	Parallel clips + concat
FastAPI Backend	Complete	Async handlers, job queue
Frontend SPA	Complete	Upload, status, results
Authentication	Complete	JWT-based sessions

Quantitative detection performance metrics for both the CNN-only baseline and the full hybrid system are presented in Table 3. The hybrid approach achieves an accuracy of 88.0%, representing a 6.2 percentage point improvement over the CNN-only baseline. The improvement is most pronounced in recall, indicating that OCR-based keyword detection successfully recovers highlight instances that do not produce a distinctive visual signature strong enough to exceed the CNN threshold alone.

Table 3: Detection Performance Comparison

Metric	CNN Only	CNN + OCR
Accuracy	81.8%	88.0%
Precision	80.2%	85.0%
Recall	74.6%	82.0%
F1-Score	77.3%	83.4%

Processing throughput measured on an NVIDIA RTX 3060 GPU averages 14.2 frames per second inclusive of preprocessing overhead, yielding an end-to-end

processing time of approximately 9.4 minutes for a 30-minute recording. On CPU-only hardware, throughput decreases to 2.1 frames per second, resulting in processing times of approximately 58 minutes for equivalent input length. The pipeline is architecturally prepared for horizontal scaling through integration of a Celery task queue backed by Redis, permitting multiple backend worker processes to service concurrent jobs without modification to existing pipeline code.

6. CONCLUSIONS

This paper has described Vision Stack AI Studio, a web-based system that automates the generation of gameplay highlight montages through an integrated AI pipeline combining convolutional neural network-based visual frame scoring with OCR-derived textual event detection. The hybrid detection approach achieves an F1-score of 83.4% across a diverse evaluation corpus, representing a significant improvement over the CNN-only baseline attributable to the complementary coverage provided by textual signal detection. The modular pipeline architecture, FastAPI backend, and containerized deployment model together provide a practical foundation for real-world operation and iterative improvement.

Future development directions identified from evaluation analysis and user feedback include the substitution of transformer-based video encoders for the ResNet-50 backbone to improve detection on games with complex temporal event patterns; extension of the OCR keyword system with user-uploadable custom vocabularies for rapid adaptation to new titles; development of real-time highlight detection operating on live streams rather than post-hoc recordings; and integration of audio-based event detection as a third complementary evidence channel. Each of these extensions is compatible with the existing pipeline interface design, facilitating their incremental introduction without architectural disruption.

ACKNOWLEDGEMENT

The authors acknowledge the open-source communities behind FFmpeg, PyTorch, OpenCV, PaddleOCR, FastAPI, and Svelte for their foundational contributions to the tools employed in this project. The authors also thank the Department of Computer Engineering, Sinhgad Academy of Engineering, Pune, for providing computational resources and academic guidance throughout the development and evaluation of this work.

REFERENCES

1. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
2. A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video

classification with convolutional neural networks," in Proc. IEEE CVPR, pp. 1725–1732, 2014.

3. D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in Proc. IEEE ICCV, pp. 4489–4497, 2015.

4. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE CVPR, pp. 770–778, 2016.

5. R. Smith, "An overview of the Tesseract OCR engine," in Proc. 9th Int. Conf. Document Analysis and Recognition (ICDAR), pp. 629–633, 2007.

6. G. Bertasius, H. Wang, and L. Torresani, "Is space-time attention all you need for video understanding?" in Proc. ICML, 2021.

7. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in Proc. ICLR, 2015.

8. A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," in Proc. ICLR, 2021.

9. FFmpeg Project, "FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video." [Online]. Available: <https://ffmpeg.org>.

10. FastAPI Documentation, "FastAPI: Modern, fast web framework for building APIs with Python." [Online]. Available: <https://fastapi.tiangolo.com>.

11. PaddleOCR, "Rich multilingual OCR toolkits based on PaddlePaddle." [Online]. Available: <https://github.com/PaddlePaddle/PaddleOCR>.

12. A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," Advances in Neural Information Processing Systems, vol. 32, 2019.