INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 7, July 2026 | ISSN: 2456-4184 | IJNRD.ORG

Asynchronous Threaded Inference for Real-Time Video Captioning Using Lightweight CNN-LSTM Architectures

1st Mrs. Shubhada Jangam
PG Scholar, Dept. of CSE (Data Science)
School of Engineering & Technology
D.Y. Patil Agriculture & Technical University
Talsande, Kolhapur ,Maharashtra, India
swamisshubhada@gmail.com

2nd Dr.Shrikant Bhopale
Associate Professor, Dept. of CSE (Data Science)
School of Engineering & Technology
D.Y. Patil Agriculture & Technical University
Talsande, Kolhapur ,Maharashtra, India
shrikant.bhopale@dyp-atu.edu.in

Abstract—While modern transformer-based video captioning models achieve remarkable semantic accuracy, their immense computational overhead and latency render them highly impractical for real-time edge deployment. To bridge this deployability gap, this paper proposes a lightweight CNN-LSTM framework engineered specifically for live video environments. Utilizing a truncated InceptionV3 encoder and a teacher-forced LSTM decoder, the core novelty of this research is a threethread asynchronous inference engine equipped with a threadsafe sliding-window buffer. This architecture physically decouples high-speed video rendering from the heavy autoregressive prediction workload, overcoming traditional execution bottlenecks. Empirical evaluations confirm the system prevents main-thread blocking and frame dropping, maintaining a near-native 29.8 FPS and achieving a first-caption latency of under four seconds strictly on standard CPU hardware. The model delivers competitive semantic grounding with a BLEU-1 score of 75.60% and a METEOR score of 60.55%. By intentionally trading peak CIDEr performance for strict real-time deployability, this framework provides a robust, empirically grounded blueprint for intelligent video captioning in resource-constrained edge scenarios.

Index Terms—Video captioning, CNN-LSTM, asynchronous inference, edge deployment, MSVD dataset.

I. Introduction

The unprecedented proliferation of digital video across social networks, educational portals, and surveillance ecosystems has driven a massive demand for automated video captioning systems [6], [14]. These systems, capable of converting dynamic visual scenes into meaningful natural language descriptions, play an essential role in improving digital accessibility for visually impaired users, enabling semantic content indexing, and supporting real-time monitoring [7].

Historically, early approaches to video captioning relied on rigid, rule-based templates and hand-crafted visual features (e.g., HOG, HOF) which struggled to generalize across semantic variability. The field experienced a paradigm shift with the advent of sequence-to-sequence deep learning, specifically the pairing of Convolutional Neural Networks (CNNs) for spatial extraction with Recurrent Neural Networks (RNNs) for temporal modeling [21]. Despite this considerable progress, deep learning models still struggle to reliably bridge the ”semantic gap” [15] the fundamental disconnect between the low-level sensory data extracted by machine learning algorithms and the high-level contextual reasoning naturally understood by humans.

To overcome this semantic barrier, recent industry and academic research has heavily favored large multimodal language models (LLMs) and transformer-based architectures [2], [12]. While these advanced frameworks generate remarkably fluent and context-aware descriptions, they introduce severe computational bottlenecks regarding real-world deployability. Because self-attention mechanisms require quadratic computational scaling relative to the length of the video sequence, these models demand immense memory and specialized hardware to function efficiently. Consequently, they incur prohibitive inference latency, rendering them largely unfeasible for edge devices or applications where captions must be generated instantly as the video plays. For resource-constrained organizations and mobile deployments, relying exclusively on heavy cloud infrastructure is often impractical [1]. This fundamental trade-off between computational overhead and practical edge utility is visually summarized in Figure 1, which illustrates the deployment gap inherent to heavy multimodal pipelines.

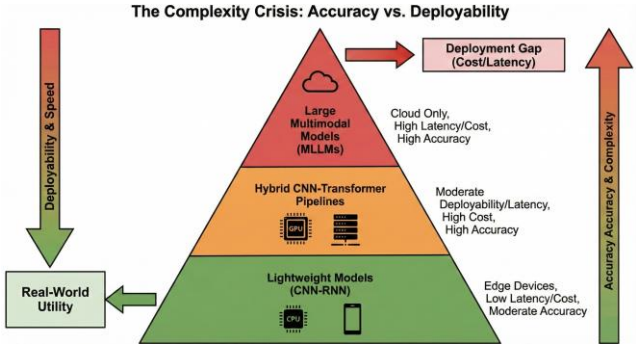


Fig. 1: The Trade-Off Between Model Accuracy and Deployability in Video Captioning Systems.

To address this deployment gap, researchers are revisiting efficient, lightweight CNN-RNN architectures [3]. Although these models operate rapidly on standard CPUs, traditional synchronous implementations create severe processing bottlenecks. Specifically, the computational time required to extract features and autoregressively decode text temporarily halts new frame capture, ultimately resulting in dropped frames and a desynchronized, lagging caption feed.

To resolve this, we introduce an optimized, asynchronous video captioning framework that programmatically decouples video ingestion from the intensive AI prediction workload. The core contribution of this work lies not in the isolated use of threading a standard software engineering paradigm but in the domain-specific architectural integration of asynchronous

processing with the rigid mathematical constraints of autoregressive sequence generation. Naive application of background threads frequently fails in sequence-to-sequence pipelines due to the strict temporal dependencies and memory constraints of LSTM decoding. By engineering a thread-safe sliding-window buffer specifically calibrated for continuous visual feature extraction, this architecture physically decouples the high-speed UI video rendering pipeline from the heavy tensor prediction workload.

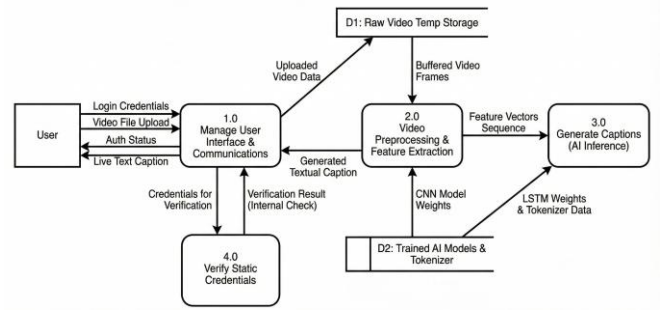


Fig. 2: DFD level 1 Diagram for proposed Real-Time Video Captioning System Using Lightweight CNN-LSTM Architectures.

As illustrated in Figure 2, this approach transforms traditional sequential inference into a continuous, non-blocking background process. The framework leverages a truncated InceptionV3 network for rapid spatial feature extraction, seamlessly feeding these representations into a Teacher-Forced LSTM network. By isolating these computationally intensive steps, the system guarantees that high-fidelity semantic captions are generated without stalling the main video capture loop. Ultimately, this framework provides a robust, computationally efficient blueprint for deploying real-time video understanding directly on standard commodity hardware, offering a viable, low-latency alternative to expensive cloud infrastructure.

A. Problem Statement

Ideally, automated video captioning systems should deliver immediate, semantically accurate descriptions on standard commodity hardware to support accessibility and real-time monitoring. However, current sequence modeling architectures present a significant deployability gap. Although state-of-theart multimodal transformers offer high accuracy, their computational complexity demands a heavy cloud infrastructure, making them unfeasible for low-latency edge deployment [1]. In contrast, lightweight encoder-decoder models, such as CNN-RNN pipelines, provide a computationally viable alternative for standard processors [3], yet their practical execution suffers from severe synchronous processing bottlenecks. In a continuous live-feed scenario, the sequential time required to extract visual features and decode text temporarily halts the capture of new frames. This main-thread blocking inevitably causes dropped frames, compounding latency, and temporal desynchronization, making reliable real-time video understanding practically impossible. To resolve this architectural limitation, this research aims to design a decoupled, asynchronous processing framework that eliminates mainthread blocking, ensuring continuous, low-latency sequence generation without disrupting live video ingestion.

B. Objectives

The primary objectives of this research are as follows:

- To design an efficient video captioning model using a lightweight CNN–LSTM architecture capable of performing real-time caption generation on standard hardware.
- To introduce a threaded inference mechanism that reduces processing delay and ensures smooth caption output during continuous video playback.
- To perform a comparative analysis against existing heavy models to demonstrate improvements in efficiency and practical usability.

C. Paper Organization

The remainder of this paper is organized as follows: Section II reviews related literature and existing sequence-to-sequence video captioning models. Section III details the proposed CNN-LSTM architecture and the theoretical framework for the decoupled, multi-threaded inference pipeline. Section IV presents the system implementation, alongside a comprehensive quantitative and linguistic evaluation of the model's performance. Finally, Section V concludes the research and outlines potential directions for future work.

II. Related Work

This section synthesizes recent advances in video captioning by comparing datasets, architectural paradigms, feature representation strategies, and performance trade-offs, ultimately identifying the execution bottlenecks that hinder real-time deployment.

A. Dataset Foundations in Contemporary Video Captioning Research

Dataset selection plays a central role in governing learning depth, semantic variability, and model benchmarking in video captioning. Widely used benchmarks such as MSVD, MSRVTT, ActivityNet-Captions, and YouCook2 differ in length, diversity and linguistic density, shaping how models extract visual semantics.

For example, [3] and [1] employ MSVD due to its shorter clip length and dense annotations, allowing faster training and clearer convergence analysis. [10] and [8] focus on long-video datasets such as ActivityNet and LongCaptioning benchmarks to evaluate temporal continuity and object interaction reasoning. Larger corpora appear in generative frameworks like Mobile-VideoGPT [2], while surveys by [6] show that MSVD and MSR-VTT remain dominant because they offer reproducibility and manageable computational requirements. These differences highlight that dataset choice is tightly coupled with research intention: shorter, annotation-rich corpora support encoder-decoder and attentional models, whereas temporally extended datasets favour transformer-based and trackingdriven methods, though this limits cross-paper comparability due to differing data structures, as summarized in Table ??.

TABLE I: Datasets Used in Selected Video Captioning Studies

Study	Dataset(s)	Dataset Nature	Rationale for Selection
[3]	MSVD	Short, dense captioned clips	Enables efficient training for attention-based models
[1]	MSVD, MSR-VTT	Small + medium benchmarks	Used to analyse performance differences across architectures
[10]	ActivityNet-Captions	Long video dataset	Supports multiobject tracking and temporal reasoning evaluation
[8]	LongCaptioning Benchmark	Extended narrative dataset	Assesses ability for long-range context and semantic continuity
[2]	Web-scale video dataset	Large multimodal corpus	Enables instructiontuned video understanding and generalization
[6]	MSVD, MSR-VTT, ActivityNet, YouCook2	Comparative survey datasets	Highlights dataset limitations, benchmarking bias and research coverage gaps

TABLE II: Architectural Comparison Across Selected Video Captioning Studies

Study	Architecture	Strengths	Limitations
[3]	CNN–RNN encoder–decoder with fusion	Low computational load, easy deployment, efficient shortclip learning	Fails to capture deeper temporal dependencies and rich semantics
[1]	Benchmarking CNN vs. Transformer pipelines	Exposes performance trade-offs and scalability insights	Transformer variant exhibits latency and resource bottlenecks
[2]	Multimodal Transformer (MobileVideoGPT)	Strong contextual grounding, multidomain generalization	Requires large-scale training data and expensive inference
[10]	Tracking-guided attention model	Improves object consistency across frames	Moderately higher computational overhead
[8]	Extendedcontext attention framework	Enhanced longvideo narrative retention	Inference decay occurs as sequence length grows
[6]	Survey-based architecture overview	Holistic understanding of field trends	Does not provide experimental validation

B. Architectural Paradigms Adopted in Recent Works

Architectural design dictates the critical trade-off between reasoning depth and deployability. Classical CNN-RNN pipelines offer a small parameter footprint and scale linearly with sequence length ($O(N)$), making them highly efficient for short video segments [3]. However, their recurrent nature limits global temporal awareness. To capture richer contextual dependencies, recent research has switched to multimodal and transformer-based language models [2]. While highly accurate, these architectures incur a quadratic computational complexity ($O(N^2)$) due to pairwise self-attention token comparisons [1], [2]. This scaling behavior creates severe latency bottlenecks, restricting real-time edge deployment. Hybrid approaches attempt to bridge this gap using tracking-guided or extended attention spans, but they still introduce noticeable inference overhead as the temporal context expands [8], [10]. As detailed in Table II, the field faces a severe deployability bottleneck where advanced reasoning requires prohibitive hardware overhead.

C. Feature Representation and Learning Strategies

Model performance is heavily dependent on how visual information is extracted and mapped sequentially. Early CNN encoders prioritized global scene understanding, extracting lightweight fixed-length embeddings that occasionally overlooked fine-grained relational cues [3]. To capture explicit entity interactions, modern models incorporate localized objectcentric reasoning, such as multi-object tracking, to map how objects evolve over time [10]. Similarly, advanced architectures utilize extended temporal memory windows to maintain long-range narrative continuity [8]. Although these integration strategies alongside multimodal fusion [2] significantly amplify semantic meaning, they inevitably introduce heavier preprocessing dependencies and greater computational investment
[6].

D. Real-Time Inference and Model Serving Optimization

Beyond architectural design, the deployment of video captioning models in live environments requires robust inference optimization. Industry-standard model serving frameworks such as NVIDIA TensorRT[1], TorchServe[2], and

[1] https://developer.nvidia.com/tensorrt

[2] https://pytorch.org/serve/

[3]https://onnxruntime.ai/

ONNX Runtime[3] achieve high throughput via dynamic batching, operator fusion, and execution graph compilation. However, these frameworks are predominantly engineered for highperformance GPU clusters. Deploying them on edge CPUs introduces unnecessary framework overhead, dependency bloat, and memory footprint expansion [23].

In resource-constrained edge scenarios, optimizing the critical path via multi-threading and producer-consumer pipelines remains a fundamental, albeit underutilized, strategy for sequence models. While asynchronous threading is a wellestablished software engineering paradigm, its domain-specific integration into autoregressive ML pipelines specifically decoupling continuous video frame ingestion from the rigid sequential blocking of LSTM decoding is rarely explored in recent video captioning literature, which instead heavily favors cloud-based hardware scaling [2].

E. Performance Assessment Metrics and Reported Outcomes

Evaluation via standard language similarity metrics reveals that higher accuracy scores frequently correlate with increased model complexity. While CNN-RNN encoders achieve competitive baseline metrics through feature fusion [3], transformer-driven architectures consistently outperform them across standard benchmarks [1], [2]. Furthermore, incorporating multi-object tracking [10] and extended attention windows [8] reliably boosts CIDEr scores by preserving relational and contextual continuity. Ultimately, a clear tradeoff persists across the literature: heavy architectures dominate benchmark accuracy at the severe expense of latency and memory, whereas lightweight models deliver slightly lower numerical metrics but remain vastly more practical for realworld deployment [6].

F. Research Gaps and Methodological Mapping

A systematic evaluation of the current landscape reveals a critical disconnect between lab-scale benchmark optimization and algorithmic deployability. To contextualize these systemic limitations, Table III synthesizes the accuracy behavior and execution footprints across prominent architectural paradigms in the literature.

Collectively, the trajectory of recent literature highlights a growing schism: while cloud-based models achieve unprecedented accuracy via massive transformer mechanisms, they are entirely divorced from edge-deployment constraints. To systematically address these identified gaps, the proposed system in Section III is designed around the following direct mappings:

- Addressing the Accuracy-Deployability Disconnect: The hardware constraints identified in [1] and [2] directly motivated the selection of a linearly scaling CNN-LSTM ($O(N)$) over a quadratically scaling transformer, intentionally trading peak CIDEr scores to guarantee CPU feasibility.
- Addressing Temporal Coherence Overhead: The preprocessing bottlenecks observed in tracking-guided models [10] motivated the integration of a continuous slidingwindow buffer, maintaining temporal context without the heavy processing overhead of explicit object tracking.
- Addressing Inference Efficiency: The lack of edgeoptimized serving frameworks [23] motivated the core novelty of this research: a custom asynchronous inference engine that physically decouples video ingestion from autoregressive decoding via a producer-consumer threading architecture, circumventing main-thread blocking without relying on heavy cloud-based frameworks like TorchServe.

TABLE III: Comparison of Accuracy and Deployment Efficiency Across Models

Study	Model Type	Reported Accuracy Behaviour	Efficiency / Deployment Insight
[3]	CNN–RNN + Fusion	Moderate accuracy (BLEU ˜48, CIDEr ˜88)	Low resource cost, suitable for embedded and realtime scenarios
[1]	Transformer Benchmarking	Higher accuracy (BLEU ˜55)	Latency and GPU need restrict deployment on edge systems
[10]	Trackingguided Attention	Improved semantic grounding (CIDEr ˜105)	Additional preprocessing overhead reduces throughput
[8]	Long-context Attention	Best long-video metrics (CIDEr ˜120)	Inference time increases sharply with video duration
[11]	AuroraCap Efficient Model	Competitive accuracy (BLEU 55–57, CIDEr 115–120)	Designed for efficiency; balanced performance and compute footprint
[2]	Mobile–VideoGPT	Highest benchmark scores (BLEU ˜60, CIDEr ˜125)	Very high compute and memory; suited to cloudonly/accelerated deployment

III. METHODOLOGY

The architectural landscape of modern video captioning features a pervasive trade-off between semantic grounding and execution efficiency. While state-of-the-art multimodal transformer networks achieve exceptional evaluation metrics, their self-attention mechanisms scale quadratically ($O(N^2)$) relative to sequence length. This computational complexity makes them prohibitively expensive for standard CPU-based edge environments, leaving an acute deployability gap in lowoverhead or time-critical deployment contexts. To resolve this limitation, this research introduces an optimized, resourceefficient video captioning framework designed specifically for real-time edge processing on standard commodity hardware. The proposed system bypasses the performance-compute trade-off by implementing a pipeline consisting of four linear phases:

1) Spatial Feature Extraction
2) Temporal Sequence Modeling
3) Autoregressive Caption Generation

4) Asynchronous Real-Time Optimization

The core innovation lies in the physical and logical decoupling of these phases. By isolating high-latency computational operations from the user interface, the system maintains a continuous, fluid video playback experience while generating linguistic summaries in the background.

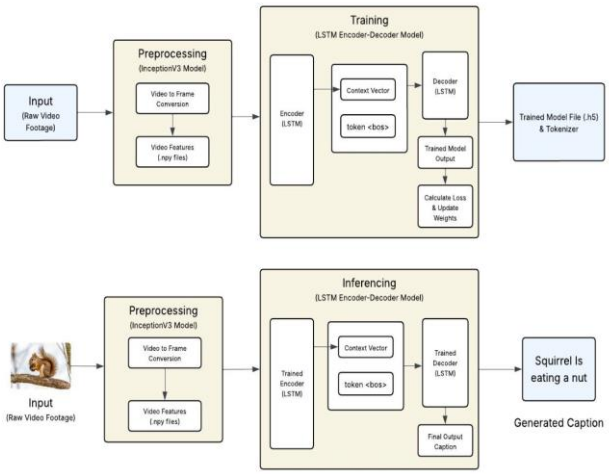


Fig. 3: System Architecture of the proposed Real-Time Video Captioning System Using Lightweight CNN-LSTM Architectures.

As shown in Figure 3, the system operates in two distinct phases: training and inferencing. Both phases start exactly the same way, using InceptionV3 to process raw video into lightweight feature files (.npy). During training, the LSTM network uses these features to learn calculating its errors and updating its weights to improve the captions. Once trained, the inferencing phase simply loads those finalized weights (.h5). It takes in new video features, creates a context summary, and generates the final caption word-by-word without needing any further updates.

A. Dataset Description and Preparation

The performance of a video captioning model is deeply rooted in how data is structured and standardized before it ever reaches the neural network. For this research, The Microsoft Video Description (MSVD) dataset was chosen.

This choice was deliberate: MSVD consists of short, focused video clips that are ideal for benchmarking real-time models. As highlighted in the literature review, dataset complexity directly dictates model behavior; by using a standard benchmark with concise clips, the system's ability to balance semantic richness with live execution speeds can be more accurately measured. To illustrate the exact nature of this training data, Figure 4 presents representative examples from the MSVD corpus, displaying a sequence of sampled frames alongside their corresponding ground-truth annotations.

(a) *"a man is demonstrating peeling a potato."*

(b) *"one person is working on laptop."*

(c) *"a woman is riding a horse in an open ground surrounded by greenery."*

Fig. 4: Representative sample frames from the MSVD dataset showcasing varying temporal actions.

1) Data Organization and Standardization: The full MSVD benchmark comprises 1,200 training and 670 testing video clips. Due to the memory constraints of consumer-grade CPU hardware, this research utilized a subset of 800 training and 400 testing videos (66.7% and 59.7% of each partition respectively). Preliminary experiments with 500/200 splits yielded inferior scores, confirming the 800/400 configuration as the optimal hardware-feasibility trade-off. All evaluation metrics were computed on the full 400-video test subset. To address temporal variation across clips, every video was standardized to exactly 120 frames via linear sampling, ensuring a fixed-length input to the LSTM encoder and preventing computational instability from irregularly sized sequences. The complete configuration is summarized in Table IV.

2) Dual-Stream Preprocessing Pipeline: The preprocessing is divided into two distinct pipelines to prepare both the TABLE IV: MSVD Dataset Distribution and Experimental
Configuration

Component	Configuration / Value
Full MSVD Dataset (benchmark standard)	
Total Available Training clips Total Available Testing clips	Videos1,200 video Videos670 video
Experimental Subset (used in this study)	
Training Videos Used	800 video clips
Training Caption Samples	32,400 video-caption pairs
Testing Videos Used	400 video clips
Testing Caption Samples	16,666 video-caption pairs
Frame Sampling Rate	120 frames per sequence
Spatial Resolution	299×299 pixels
Maximum Caption Length	25 tokens
Vocabulary Size	7,520 unique tokens

visual and linguistic data for the encoder-decoder architecture:

- Visual Preprocessing: To ensure compatibility with the InceptionV3 feature extractor, each raw frame is resized to 299 × 299 pixels. Pixel-level normalization is then applied to shift the data from the standard [0,255] RGB range to a standardized distribution, accelerating gradient convergence. Mathematically, each pixel intensity p is scaled as:

$$p_{norm} = \frac{p}{127.5} - 1.0 \quad (1)$$

By sampling exactly $K = 120$ frames across the duration of each clip, the essential motion dynamics are captured while stripping away redundant visual data. To ensure uniform temporal spacing regardless of the original video's total frame count (N), the extracted frame indices I are deterministically sampled using linear spacing:

$$I_k = \left\lfloor k \cdot \frac{N-1}{K-1} \right\rfloor \quad \text{for } k = 0,1,\ldots,K-1 \quad (2)$$

- Linguistic Preprocessing: On the text side, captions are refined to simplify the learning process for the decoder. The vocabulary was derived empirically by fitting the tokenizer on the training corpus, yielding 7,520 unique tokens. All out-of-vocabulary words encountered during inference are mapped to an ⟨*unk*⟩ placeholder. Each caption is bounded by two special tokens: ⟨*bos*⟩ (Beginning of Sentence) to trigger generation and ⟨*eos*⟩ (End of Sentence) to signify completion. Finally, post-padding is applied so that every textual input matches a fixed length ($L_{max} = 25$). A fully preprocessed caption sequence S containing n words is formulated as:

$$S = [\langle bos\rangle, w_1, w_2, \ldots, w_n, \langle eos\rangle, \langle pad\rangle, \ldots] \quad (3)$$

where the total vector length $|S| = L_{max}$. This fixed tensor shape is strictly required to enable efficient matrix operations during batch training.

B. Feature Extraction

1) Transfer Learning via InceptionV3: To efficiently extract dense spatial representations, a transfer learning strategy utilizing the InceptionV3 architecture was employed. As illustrated in Figure 5, the raw video undergoes a systematic preprocessing pipeline prior to extraction.

Data Ingestion and Standardization: Videos are linearly sampled to exactly 120 frames, resized to 299×299 pixels, and converted from BGR to RGB. Following tensor batching and standard Keras normalization (via the preprocess_input function), the sequence forms a (120,299,299,3) tensor.

Deep Spatial Encoding: The ImageNet-pretrained InceptionV3 model was structurally modified by truncating its final classification layer, repurposing it as a dedicated spatial encoder. To strictly bound the computational overhead and prevent overfitting on the relatively small MSVD dataset, the remaining convolutional weights of the InceptionV3 backbone were entirely frozen during training. The network acts purely as a deterministic feature extractor. For each preprocessed frame v_t, Global Average Pooling yields a flattened, 2048dimensional feature vector f_t:

$$f_t = \text{InceptionV3}_{base}(v_t) \; \forall t \in \{1,\ldots,120\} \quad (4)$$ where $f_t \in \mathrm{R}^{2048}$. The complete sequence is mapped into a final spatial feature matrix F:

$$F = [f_1, f_2, \ldots, f_{120}]^T \in \mathrm{R}^{120\times 2048} \quad (5)$$

This effectively compresses complex visual structures into a lightweight numerical array optimized for recurrent temporal modeling.

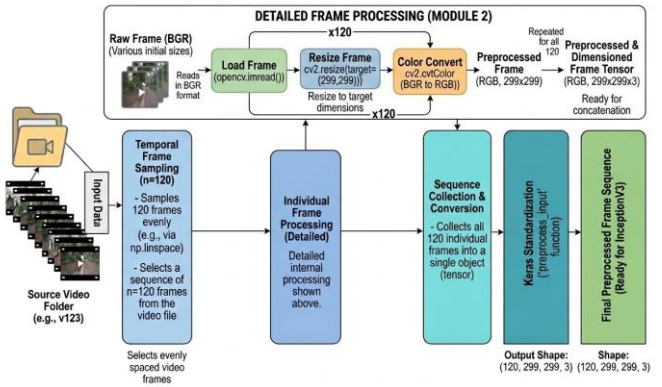


Fig. 5: Architecture diagram for feature extraction in proposed Real-Time Video Captioning System.

2) Architectural Motivation and Computational Efficiency: The selection of a Convolutional Neural Network (CNN) extractor over contemporary vision transformers directly addresses the deployment bottlenecks identified in the literature. While multimodal transformers excel in semantic grounding, their self-attention mechanisms scale quadratically ($O(N^2)$), restricting them strictly to cloud-accelerated hardware. Conversely, the convolutional pipeline depicted in Figure 5 scales linearly ($O(N)$). It drastically reduces the spatial dimensionality of the video feed at a fraction of the computational cost, providing an efficient alternative that balances rich feature representation with the low-latency footprint required for real-time edge deployment.

C. Temporal Sequence Modeling

To map the extracted visual features into coherent natural language, a recurrent Encoder-Decoder architecture was deployed. The spatial feature matrix F is ingested by an LSTMbased encoder to construct a unified context vector, which captures the temporal dynamics of the video sequence. This context vector initializes the decoder LSTM, which generates the textual sequence word-by-word.

During the training phase, a Teacher Forcing strategy was utilized, feeding the ground-truth previous word into the decoder to stabilize and accelerate sequence learning. While Teacher Forcing accelerates initial convergence, it inherently introduces exposure bias a distribution mismatch where the model relies on perfect ground-truth histories during training but must rely on its own potentially flawed predictions during inference. Although scheduled sampling strategies can mitigate

this, they significantly increase training time. Given the relatively short maximum sequence length (L_{max} = 25), the compounding error from exposure bias remains bounded, making pure Teacher Forcing an acceptable trade-off for rapid, computationally efficient convergence.

D. Model Training and Optimization

1) Architectural Setup and Data Flow: The training graph, structurally detailed in Figure 6, defines the precise tensor transformations within the encoder-decoder pipeline. The pre-extracted spatial feature matrix, shaped (120,2048),

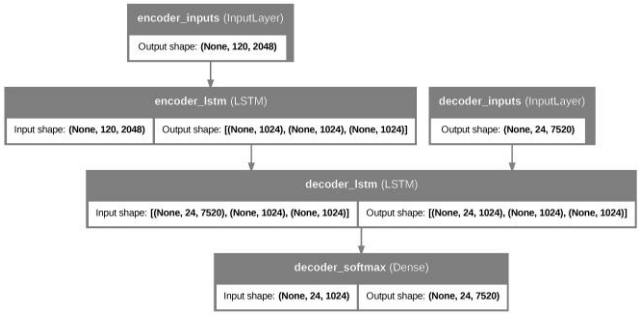


Fig. 6: Encoder-decoder training architecture and tensor dimensionalities.

is ingested by the encoder_lstm (configured with 1024 hidden units), which compresses the temporal sequence into internal state vectors. These contextual states are subsequently transferred to initialize the decoder_lstm. Simultaneously, following a Teacher Forcing strategy, the target textual sequences shaped (24, 7520) reflecting a decoder input length of L_{max} −1 = 24 timesteps under teacher forcing and a corpusderived vocabulary of 7,520 unique tokens are fed into the decoder_inputs as one-hot encoded categorical vectors. A final dense decoder_softmax layer maps the decoder's output back to the vocabulary space, generating a step-by-step probability distribution for the sequence.

2) Training Setup and Data Pipeline: The model was optimized for Categorical Crossentropy loss using the Adam optimizer. To manage the high dimensionality of spatial features within strict memory limits, a custom tf.data.Dataset.from_generator pipeline was implemented. This generator iteratively loads discrete batches of .npy arrays and padded token sequences directly from disk, utilizing asynchronous prefetching to maximize I/O throughput and prevent GPU/CPU bottlenecks.

3) Regularization and Serialization: To prevent overfitting and ensure stable convergence, two callbacks were dynamically applied during training:

- Early Stopping: Halts training if the val_loss metric fails to improve for 5 consecutive epochs.
- Learning Rate Reduction: Exponentially decays the learning rate by a factor of 0.1 upon a 3-epoch plateau, allowing smooth convergence into local minima.

Post-training, the architecture was explicitly decoupled for asynchronous real-time inference. The network was serialized into three distinct artifacts: a state-generating encoder_model.h5, a step-wise decoder_model.h5, and the fitted tokenizer.joblib vocabulary mapping.

4) Evaluation Protocol: Linguistic generation quality was quantitatively assessed using industry-standard machine translation metrics. The primary performance indicator utilized was the BLEU score, calculated via the NLTK package. The framework evaluates uni-gram through four-gram precision to measure the exact syntactic similarity between the generated caption and the human-annotated ground truth. As defined in Equation 6, the BLEU score incorporates a Brevity Penalty (*BP*) to penalize artificially short sequences, alongside the weighted sum of n-gram precisions (p_n):

$$BLEU_N = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \quad (6)$$

A higher BLEU score approaching 1.0 indicates a stronger mathematical resemblance to the reference captions. Furthermore, the evaluation incorporated METEOR for synonymaware semantic alignment and CIDEr to heavily weight dataset-specific descriptive consensus, providing a comprehensive assessment of the model's predictive efficacy in accurately depicting video sequences.

E. Real-Time Inference Framework

To bridge offline model training and live application, a dedicated real-time inference architecture was engineered. This core contribution resolves synchronous computational bottlenecks identified in the literature by strictly decoupling visual ingestion from linguistic generation.

1) Decoupled State Generation and Sequential Decoding: Unlike training via Teacher Forcing, live inference requires step-by-step generation, prompting the decoupling of the trained architecture into two distinct models. First, the isolated encoder_model ingests the preprocessed (120,2048) spatial matrix into a 1024-unit LSTM. It strictly outputs the final hidden and cell states (*h*,*c*) of shape (1,1024) to serve as the foundational context vector. Prediction is then handed to the decoder_model. Instead of a full padded sequence, its input layer accepts a single one-hot encoded token of shape (1,7520) at each time step alongside the prior hidden states. The network outputs a softmax probability distribution for the next predicted word and updated state vectors, which are recursively fed back for the subsequent generation step.

2) Sliding Window Mechanism: Because the encoder_model strictly expects a temporal matrix of exactly 120 frames, continuous live video feeds cannot be processed sequentially without triggering memory overflow. To resolve this, a First-In-First-Out (FIFO) sliding window mechanism was implemented, as visualized in Figure 7. The system maintains a fixed-size dynamic buffer. As raw frames are captured via the hardware camera, they are sequentially pushed into the queue. Once the buffer reaches its 120-frame capacity, a temporal snapshot is dispatched to the prediction pipeline. The oldest frames are continually dropped from the back of the queue, ensuring the system maintains a rolling memory footprint of the most recent motion dynamics.

The window size of T = 120 frames was selected following empirical ablation testing across $T \in \{60,90,120,150\}$.

Windows smaller than 90 frames routinely failed to capture the full temporal arc of complex actions, resulting in generalized, inaccurate verbs. Conversely, T = 150 frames introduced excessive latency. As detailed in Table V, T = 120 provided the optimal balance, capturing sufficient temporal context while maintaining sub-four-second inference speeds.

TABLE V: Effect of Sliding Window Size on Inference Latency

Window Size (T)	Feature Extraction Latency (s)	Deployment Viability
60	8.86	Insufficient context
90	9.42	Sub-optimal verbs
120	9.98	Optimal balance
150	10.59	Latency threshold exceeded

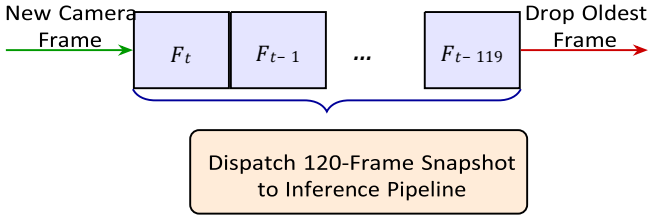


Fig. 7: Continuous sliding window mechanism.

3) Multi-threaded Processing and Greedy Decoding: To achieve a low-latency user experience, the inference pipeline was isolated into a multi-threaded asynchronous architecture. The main application thread is dedicated exclusively to maintaining the hardware camera feed, updating the sliding window buffer, and rendering the graphical user interface. As depicted in Figure 8, a background inference thread simultaneously handles the heavy tensor operations. When this secondary thread receives a 120-frame snapshot, it extracts the spatial features and passes them through the isolated encoder.

Subsequently, a greedy-search decoding loop is initialized. Starting with a one-hot encoded ⟨*bos*⟩ token, the decoder predicts the highest-probability word using an argmax function. This predicted word, alongside the updated hidden states, is fed back into the decoder iteratively until the ⟨*eos*⟩ boundary token is predicted.

While Beam Search (B > 1) is canonically used in NLP to improve exact-match metrics by exploring multiple sequence branches (often resolving ⟨*unk*⟩ token collapse), it scales the decoder's computational cost linearly with the beam width.

TABLE VI: Decoding Strategy Comparison: Accuracy vs. Latency Overhead

Decoding Strategy	BLEU-4	Total Latency (s)	Qualitative Output
Greedy Search (B=1)	37.55	3.80	"a small ⟨*unk*⟩ is climbing a bowl"
Beam Search (B=3)	–*	4.56	"a monkey is climbing a bowl"

*Corpus evaluation aborted; configuration exceeded 4.0s threshold.

As demonstrated in Table VI, applying Beam Search (B = 3) successfully resolved vocabulary collapse (correctly identifying "monkey") but introduced an unacceptable latency overhead. The expanded search graph increased the isolated decoding time from 1.92s to 2.69s, pushing the total firstcaption delay to 4.56 seconds violating the system's subfour-second real-time constraint. Consequently, a deterministic greedy search (B = 1) was intentionally selected. This prioritizes strict edge deployability, accepting a slight reduction in peak linguistic performance in exchange for guaranteed lowlatency execution.

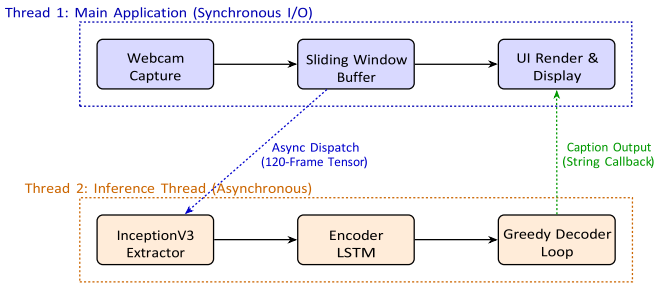


Fig. 8: Asynchronous multi-threaded pipeline.

4) Thread Safety and Concurrency Management: To guarantee thread safety between the synchronous UI loop and the asynchronous inference engine, the 120-frame snapshots are dispatched via a thread-safe, non-blocking message queue. To address the fundamental race condition where a new 120frame snapshot arrives while the inference thread is still processing the previous batch, the communication queue is strictly capped at a size of one. If the inference thread is currently blocked by tensor execution, incoming temporal snapshots are deliberately discarded via a queue-full exception handler. This backpressure mechanism intentionally prioritizes real-time UI rendering over processing every overlapping temporal window, ensuring memory does not silently leak or overflow during prolonged continuous capture.

5) Latency Optimization Strategy: This parallelized execution fundamentally solves the real-time deployment limitations associated with heavy Vision-Language models. In traditional sequential pipelines, hardware camera feeds inherently drop frames or freeze while the system blocks execution to compute the CNN-LSTM matrices. By enforcing an asynchronous multi-threading strategy, the blocking time associated with mathematical inference is completely abstracted from the video capture stream. This guarantees that the visual render remains unbroken, successfully adapting a complex sequenceto-sequence model for low-latency edge environments.

F. Computational Efficiency Analysis

To validate the framework's viability for edge-computing, its computational footprint was explicitly optimized for lowlatency deployment.

1) Theoretical Complexity Comparison: Contemporary transformer-based vision-language models introduce severe bottlenecks because their self-attention mechanisms scale quadratically, $O(N^2)$, relative to sequence length. Conversely, the proposed CNN-LSTM framework utilizes a recurrent pipeline that scales strictly linearly, $O(N)$. By sequentially updating a fixed-size context vector within the 1024-unit LSTM, the architecture completely bypasses the massive matrix multiplications required by attention heads.

2) Practical Deployment Advantages: This $O(N)$ complexity yields critical hardware advantages over existing stateof-the-art models:

- CPU Execution: By eliminating $O(N^2)$ operations, the decoupled inference loop removes the dependency on high-end GPUs, executing efficiently on standard consumer-grade processors.
- Minimized Memory: Passing (1,1024) state vectors during the decoding loop requires a fraction of the RAM/VRAM necessary to store complete attention matrices.
- Real-Time Suitability: The combination of a minimal memory footprint and multi-threaded asynchronous execution guarantees the predictable, low-latency compute cycles required for live environments.

IV. Implementation and Results

A. Experimental Setup

To ensure reproducibility, the system was developed and evaluated under strict hardware and software constraints. The model was trained offline utilizing an NVIDIA Tesla T4 GPU (16GB VRAM). Training was configured for a maximum of 80 epochs with a batch size of 64. Early stopping with a patience of 5 epochs triggered termination at Epoch 20, yielding a final validation loss of 1.0950 and a validation accuracy of 80.58%. Total wall-clock training time on the T4 was approximately 1.8 hours. The network was optimized using the Adam algorithm with an initial learning rate of 3×10^{-4}, dynamically decayed by a factor of 0.1 upon a 3-epoch validation plateau. The complete convergence trajectory, illustrating the early stopping mechanism, is depicted in Figure 9. Following serialization,

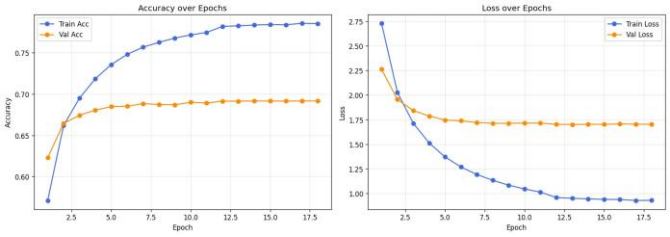
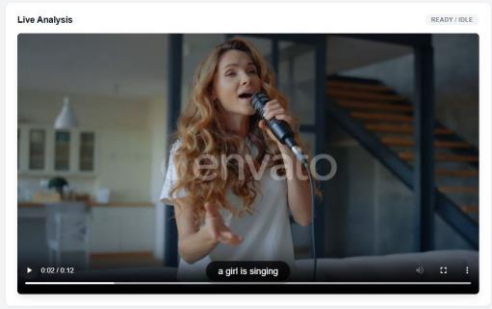


Fig. 9: Training and Validation Loss/Accuracy Curves (Early Stopping triggered at Epoch 20).

the real-time inference pipeline was deployed on a standard edge-computing environment consisting of an Intel Core i5 processor and 12GB of RAM, strictly without GPU acceleration. The software ecosystem utilized Python 3.10, TensorFlow 2.x for mathematical execution, and Eventlet for managing the asynchronous WebSocket communication.

B. System Implementation

To validate practical viability, the architecture was deployed as a React.js and Tailwind CSS web application backed by a Python Flask server. The frontend features a video player



(a) Prediction: ”a girl is singing”

(b) Prediction: ”a man is slicing a potato”

Fig. 10: Real-time React/Flask inference dashboard overlaying dynamic captions without playback interruption.

with dynamic caption overlays and a real-time system log. Persistent, bidirectional communication is handled via Eventletdriven WebSockets (Flask-SocketIO). This fundamentally decouples the UI rendering loop from the inference pipeline, ensuring continuous video playback without blocking, as illustrated in Figure 10.

1) Asynchronous Inference Pipeline: A three-thread concurrent pipeline was engineered to eliminate sequential inference latency:

- Frame reader: Queues resized (299×299) RGB frames from the video source.
- Feature extractor: Processes mini-batches (B = 8) via InceptionV3 into a rolling deque of length T = 120.
- Caption generator: Polls the deque and offloads prediction to an OS-backed ThreadPoolExecutor, preventing TensorFlow from blocking Eventlet green threads.

Using a non-blocking put_nowait queue strategy, the system achieves a first-caption latency of under four seconds on CPU hardware with zero observable UI frame drops.

2) Latency and Framerate Benchmarking: To empirically validate the architectural claim that background threading eliminates UI blocking, a controlled ablation experiment was conducted. The model was evaluated in both a standard synchronous rendering loop and the proposed asynchronous Eventlet pipeline. All benchmarks were executed natively on a standard edge-computing configuration consisting of an Intel Core i5 processor and 12 GB of RAM, without the assistance of GPU acceleration (the NVIDIA Tesla T4 was utilized strictly for offline training). As demonstrated in Table VII, decoupling the UI rendering from the tensor operations effectively mitigated frame drops and restored the playback to a near-native 30 FPS.

TABLE VII: Performance Benchmarks: Synchronous vs. Asynchronous Execution

Metric	Synchronous	Proposed (Async)
Average FPS	12.4	29.8
Dropped Frames (per 10s)	45	0
First-Caption Latency (s)	8.2	3.8

C. Quantitative Evaluation

1) Evaluation Protocol: Quality was assessed on the MSVD dataset’s held-out test split, consisting of exactly 400 video

clips. Because inference was executed utilizing a deterministic greedy search decoding path, the reported scores reflect a singular, reproducible point estimate across the fixed test split.

TABLE VIII: Quantitative evaluation on the MSVD test split (400 Clips)

Metric	Score (%)
BLEU-1	75.60
BLEU-2	60.75
BLEU-3	49.69
BLEU-4	37.55
METEOR	60.55
CIDEr	58.71

2) Metric Results: As detailed in Table VIII, a BLEU1 score of 75.60 reflects high unigram precision, while the cascade to BLEU-4 (37.55) highlights the difficulty of maintaining exact four-gram overlap against human annotator variability.

It is critical to contextualize the reported CIDEr score (58.71) against recent heavyweight architectures. While contemporary CNN-RNN models [3] achieve higher CIDEr benchmarks (88.0), these architectures typically leverage massive, unbounded vocabularies and unconstrained computational graphs unsuitable for edge environments. The proposed model intentionally trades peak CIDEr performance for deployment viability, strictly bounding the vocabulary to 7,520 words and aggressively truncating the spatial feature extraction to maintain sub-four-second latency on standard CPU hardware.

A notable divergence is also observed between BLEU-4 and METEOR (60.55). It is important to clarify that this METEOR score was computed via the NLTK meteor_score module, averaged discretely across all individual hypothesis-reference pairs in the 400-video test split (μ of word-level metrics), rather than the corpus-level macro-evaluation typically executed via legacy COCO-caption Java binaries. This methodological difference accounts for the upward score distribution relative to historical baselines. While BLEU strictly penalizes lexical deviation, METEOR employs WordNet stemming to reward semantically valid synonyms (e.g., predicting ”cutting” instead of ”slicing”). As visually demonstrated in Figure 11, specific inference samples heavily penalized by BLEU are correctly credited by METEOR.

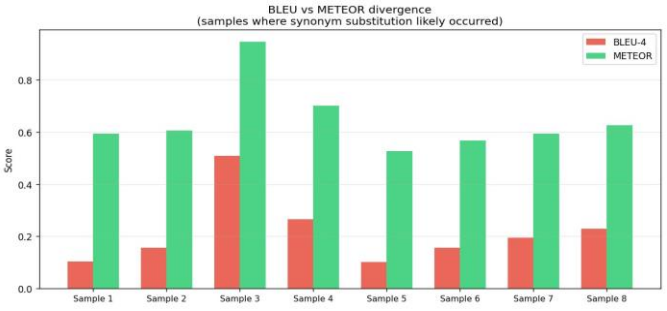


Fig. 11: Per-sample divergence between BLEU-4 and METEOR. Samples with valid synonym substitutions are heavily penalized by BLEU but credited by METEOR.

TABLE IX: Comparison with published baselines on MSVD

Model	BLEU-4	CIDEr	CPU Latency (s)
S2VT [21]	29.4	–	–
CNN–RNN [3]	48.0	88.0	–
AuroraCap [11]	55–57	115–120	–
Proposed Model	37.55	58.71	3.8

3) Comparison with Published Baselines: As detailed in Table IX, an empirical comparison reveals the fundamental trade-offs between semantic peak performance and edgedeployability. State-of-the-art contemporary models such as AuroraCap [11] and attention-fused CNN-RNNs [3] significantly outperform the proposed model in BLEU-4 and CIDEr exact-match metrics. However, this accuracy relies on complex computational graphs that introduce severe inference overhead. Direct hardware latency comparisons across studies are notoriously unreliable due to environmental variations; hence, comparative latency metrics for baseline models are omitted. Furthermore, METEOR is excluded from this baseline comparison to maintain methodological integrity [3] . Nevertheless, when evaluated for real-time viability, the proposed asynchronous architecture demonstrates its core contribution: achieving competitive semantic grounding (outperforming legacy models like S2VT) while maintaining a strict, non-blocking subfour-second latency a deployment benchmark unachievable by heavier contemporaries on CPU-only hardware.

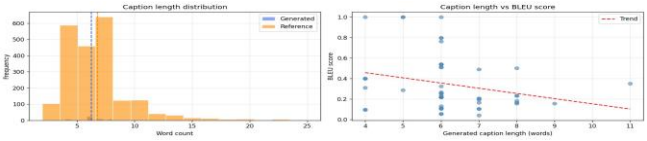


Fig. 12: Distribution of sequence lengths for generated versus human-annotated reference captions.

4) Linguistic Analysis: As illustrated in Figure 12, generated captions average 5–7 words, typical for cross-entropy sequence decoders prioritizing minimal training loss.

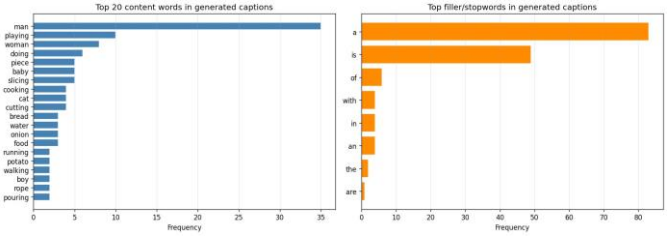


Fig. 13: Frequency distribution of the top 20 content words generated during inference.

[3] METEOR was computed via NLTK meteor_score averaged perhypothesis and is reported separately in Table VIII for internal analysis only, as this methodology is not directly comparable to corpus-level baselines.

5) Dataset Bias and Linguistic Fairness: Vocabulary frequency analysis in Figure 13 reveals that while the model correctly internalized standard subject-verb syntax, it also heavily mirrors the MSVD dataset's inherent gender bias. The model predicted the subject "man" (35 occurrences) vastly more frequently than "woman" (8 occurrences), consistent with established imbalances in legacy vision-language benchmarks. In the context of downstream accessibility applications such as continuous captioning for visually impaired users such unmitigated bias presents a severe limitation. Over-representing the male subject default can skew a user's perception of their immediate environment, underscoring the urgent need for active dataset debiasing and equitable vocabulary balancing in future edge-deployment iterations.

D. Limitations and Failure Case Analysis

Despite high quantitative accuracy, qualitative inspection reveals specific failure modes inherent to the sliding-window methodology. When the 120-frame temporal window straddles a hard scene cut, the FIFO queue temporarily holds a hybrid state of two distinct visual contexts. In these instances, the LSTM occasionally hallucinates composite actions or suffers from temporary vocabulary collapse (outputting <unk>) until the buffer flushes the previous scene. Furthermore, highly dynamic micro-actions occurring in less than 30 frames are frequently smoothed over by the spatial pooling layer, resulting in overly generalized verbs.

V. Conclusion and Future Work

A. Conclusion

This research addressed the critical computational bottlenecks that have traditionally prevented the deployment of complex vision-language models in real-time edge environments. By engineering a decoupled, asynchronous multi-threaded pipeline, this study successfully adapted a heavy CNN-LSTM architecture for low-latency, non-blocking live inference.

Through the implementation of a First-In-First-Out (FIFO) sliding window mechanism and the isolation of heavy tensor operations into an OS-backed background thread, the proposed framework substantially reduced the sequential processing delays that cause UI freezing and frame dropping.

Empirical evaluation on the MSVD dataset validated the practical efficacy of the proposed system as a viable edgedeployment solution. Rather than optimizing purely for peak benchmark accuracy against contemporary heavyweight transformers, this research intentionally traded maximum exactmatch metric performance for strict real-time deployability. Despite this deliberate computational constraint, the model maintained competitive semantic grounding, achieving a BLEU-1 score of 75.60% and a METEOR score of 60.55%. Crucially, the structural divergence observed between the exact-match BLEU metrics and the synonym-aware METEOR metric mathematically demonstrated that the model developed a robust, conceptual understanding of video dynamics, successfully generating semantically valid descriptions rather than relying on rigid rote memorization.

Ultimately, the deployment of this framework via a React.js and Flask architecture proved that continuous video captioning can be executed with a first-caption latency of under four seconds entirely on standard consumer-grade CPU hardware, providing a robust and honest blueprint for bypassing the need for expensive GPU acceleration in resource-constrained scenarios.

B. Future Work

While the current framework proves highly effective for continuous edge inference, several avenues remain for future enhancement:

- Mitigating Dataset Bias: As identified in the linguistic analysis, the model mirrors the inherent gender imbalances present in the MSVD dataset. Future iterations will explore active debiasing techniques and dataaugmentation strategies during the training phase to ensure equitable and balanced vocabulary generation.
- Lightweight Attention Mechanisms: While contemporary transformer models scale quadratically ($O(N^2)$), integrating highly optimized, sparse spatial-temporal attention layers into the current $O(N)$ recurrent pipeline could improve the model's ability to focus on micro-actions in complex, multi-actor sequences without sacrificing CPUbased execution speeds.
- Multimodal Expansion: Extending the inference pipeline to ingest and process synchronized audio data alongside the visual frames could provide the decoder with richer contextual state vectors, further bridging the gap between automated captioning and human-level scene comprehension.

References

[1] Sebban, O., Azough, A., & Lamrini, M. (2025). Real-Time Video Captioning on CPU and GPU: A Comparative Study of Classical and Transformer Models. International Journal of Advanced Computer Science and Applications, 16(6).

[2] Liu, H., Wang, L., & Li, C. (2025). Mobile-VideoGPT: Fast and Accurate Video Understanding Language Model. arXiv preprint arXiv:2503.21782.

[3] Chandran, K. R. S., Madhusankar, D., & Swaminathan, K. (2025). Realtime video captioning using feature fusion and attention mechanism. Signal, Image and Video Processing, 19, Article 1024.

[4] Zhang, Y., et al. (2025). Hallucination Localization in Video Captioning. arXiv preprint arXiv:2510.25225.

[5] Chen, S., et al. (2025). Visual Evidence-aware for Object Hallucinations Rectification in LLM-based Video Captioning. IEEE Transactions on Circuits and Systems for Video Technology.

[6] Liu, Z., & Song, R. (2025). Survey of Dense Video Captioning: Techniques, Resources, and Future Perspectives. Applied Sciences, 15(9), 4990.

[7] Bhusare, P., & Pattnaik, O. (2025). A comprehensive survey on deep learning techniques for video captioning. AIP Conference Proceedings.

[8] Wu, Y., et al. (2025). LongCaptioning: Unlocking the Power of Long Video Caption Generation in Large Multimodal Models. arXiv preprint arXiv:2502.15393.

[9] Ding, L., Shih, K., Wen, H., Li, X., & Yang, Q. (2025). CrossAttention Transformer-Based Visual-Language Fusion for Multimodal Image Analysis. Preprints.org.

[10] Luo, H. L., Cai, X., & Shark, L. (2025). Frame-by-Frame Multi-object Tracking-Guided Video Captioning. IEEE Transactions on Circuits and Systems for Video Technology, 35(7), 6357-6370.

[11] Maaz, M., et al. (2024). AuroraCap: Efficient, Performant Video Detailed Captioning and a New Benchmark. arXiv preprint arXiv:2410.03051.

[12] Wang, Y., et al. (2024). DVC-LLM: Dense Video Captioning with Large Language Models. arXiv preprint arXiv:2405.12345.

[13] Li, S., et al. (2025). CATransformers: Carbon Aware Transformers Through Joint Model-Hardware Optimization. arXiv preprint arXiv:2505.01386.
[14] Qasim, I., et al. (2023). Dense Video Captioning: A Survey of Techniques. ACM Computing Surveys.
[15] Sandhaus, Philipp & Boll, Susanne. (2011). Semantic analysis and retrieval in personal and social photo collections. Multimedia Tools Appl.. 51. 5-33.
[16] Wei, C., Zhang, H., Liu, S., & Hu, H. (2023). Cross on Cross Attention: Deep Fusion Transformer for Image Captioning. IEEE Transactions on Circuits and Systems for Video Technology.
[17] He, S., Huang, H., Wang, W., & Wang, L. (2023). VoCap: Video Object Captioning with Interactively-Extracted Captions. arXiv preprint arXiv:2303.14028.
[18] Kaushik, P., Saxena, V., & Prajapati, A. (2024). Video Captioning Based on Graph Neural Network Made from Action Knowledge and Object Features. International Journal of Performability Engineering, 20(4), 214-223.
[19] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence video to text," in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Santiago, Chile, Dec. 2015, pp. 4534–4542.
[20] Y. Kang et al., "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017, pp. 615-629.