ApproxBit: Efficient Video Analytics through Latency-Aware Offloading with Learned Binary Codes

Hyunseung Kim
kim4061@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Sheetal Prasanna
sprasan@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Yin Li
yin.li@wisc.edu
University of Wisconsin-Madison
Madison, Wisconsin, USA

Somali Chaterji
schaterji@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Saurabh Bagchi
sbagchi@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Abstract

With the growing ubiquity of video content, efficient video analytics has become essential for applications such as surveillance, autonomous driving, and augmented reality. Yet, deploying video analytics models on resource-constrained edge devices and in low-bandwidth environments remains challenging. A dominant method for handling demanding video analytics tasks on edge devices has been to offload computation strategically from the edge device to servers. However, all prior solutions fail to offload under severely constrained, real-world network conditions (such as, a few-Mbps satellite network) due to the much higher data rates associated with video tasks. We introduce ApproxBit, a system to optimize shared edge-to-cloud processing for video analytics tasks; the two that we experiment with are video action recognition and video question answering. ApproxBit integrates an encoder within the video model, uses learned binary codes to effectively compress and offload data, and adaptively decides on the offloading point depending on the network bandwidth. ApproxBit's adaptive and efficient data compression, which reduces the original feature map size by up to 2142.4×, makes it an ideal solution for video analytics on edge devices, especially with constrained networks. We evaluate ApproxBit on the two video tasks, across different model architectures (e.g., convolution- and Transformer-based) and multiple datasets (e.g., Something-Something-v2, Kinetics, and MSVD). Our results of latency and accuracy are superior over baselines: edge-only processing, server-only processing, DNN Surgery [ToCC '23], full offloading of H.264-encoded videos, DeepCOD [SenSys '20], neural video compression DCVC-FM [CVPR '24], and LimitNet [MobiSys '24]. We also demonstrate ApproxBit's adaptivity to changing network conditions, and generalization in a real-world user study.

CCS Concepts

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Distributed computing methodologies**; **Machine learning**; **Computer vision**.

Keywords

Edge-Cloud Collaborative Processing, Video Action Recognition, Video Question Answering, Offloading Optimization, Learned Binary Codes

ACM Reference Format:
Hyunseung Kim, Sheetal Prasanna, Yin Li, Somali Chaterji, and Saurabh Bagchi. 2026. ApproxBit: Efficient Video Analytics through Latency-Aware Offloading with Learned Binary Codes. In *ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems (SenSys '26), May 11–14, 2026, Saint Malo, France*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3774906.3802757

SenSys '26, Saint Malo, France

ACM ISBN 979-8-4007-2309-4/2026/05
https://doi.org/10.1145/3774906.3802757

1 Introduction

The rapid growth of digital video content has driven an increasing demand for robust video analytics at the edge that can operate effectively under diverse network conditions. In many real-world scenarios, network connectivity is limited or unreliable [46, 50], requiring edge devices to perform most processing locally. However, video analytics tasks such as video action recognition (VAR) and video question-answering (VQA) rely on deep neural networks that achieve impressive accuracy but demand heavyweight computation. These models far exceed the processing capacity of today's mobile edge devices. Consider a flagship mobile SoC like the Apple A19, which despite being one of the most powerful mobile processors offers only 2.074 TFLOPS for FP32 inference. Modern video analytics models demand far greater compute. For example, a recent VAR model, VideoMAE-V2 [65], requires 38.2 TFLOPs per a 64-frame clip. A recent VQA model, LLaVA-OV-7B [34], demands 40.8 TFLOPs at its prefill stage to infer over a 32-frame video. This gap between required and available compute capacity is an order of magnitude and makes real-time video analytics infeasible in today's edge platforms.

To bridge this gap, prior research has made significant progress in *offloading* DNNs from edge devices. Existing works have explored two major directions for reducing *offloading latency*: (1) compressing the input video before transmission, and (2) compressing intermediate features after partial processing on the edge device. However, solutions from both directions fall short for video

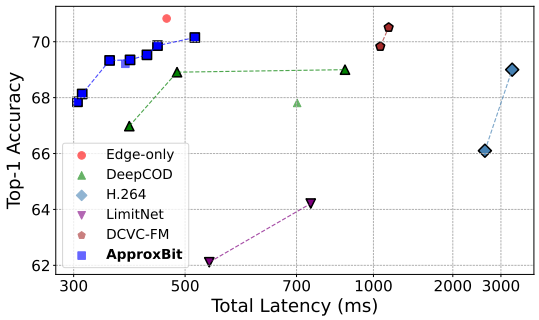

Figure 1: Accuracy-Latency Pareto Curves for Video Action Recognition (VAR) task, on SSv2 dataset with MViTv2 model at 3 Mbps wireless bandwidth. ApproxBit **yields the best Pareto frontier among baseline methods with Top-1 accuracy plotted against total latency. Additionally,** ApproxBit **offers adjustable parameters for optimal Service Level Agreements (SLA)-based tuning.**

applications on resource-constrained devices that demand real-time operation and/or operate under constrained bandwidth.

Video-level compression reduces communication cost by compressing the raw video prior to transmission. Both conventional video codecs [49, 55] and neural video compression [36–38] have been applied, generating compact bitstreams through content-aware compression or learned encoding. However, content-aware approaches such as LimitNet [19] process each frame independently, ignoring temporal consistency and producing suboptimal compression for video data. In contrast, neural video compression methods [36–38] learn temporal structure through complex encoders, but this introduces significant computational overhead on resource-limited edge devices. In practice, such overhead often dominates end-to-end latency, wiping out the benefits of reduced transmission size. Furthermore, because these approaches minimize perceptual redundancy rather than preserving task-relevant semantics, the compressed representations can severely degrade the accuracy of the downstream task.

At the feature level, DNN partitioning, a key offloading technique, reduces end-to-end latency by splitting model computation between the edge and the cloud and transmitting intermediate feature maps. Neurosurgeon [28] pioneered this concept, and later studies enhanced it through optimized workload distribution [11, 13, 41] and early-exit mechanisms for collaborative inference [32, 35]. To further reduce transmission latency, compression-based extensions such as DeepCOD [69] and DynO [2] compress or quantize intermediate features while preserving task semantics. However, these image-based methods face fundamental challenges in video models, whose feature maps include an additional temporal dimension that greatly increases bandwidth demand. For example, VideoMAE-V2 [65] processes 64 frames per clip, causing data volume to scale with frame count; even with a 4× temporal reduction, its offloading size exceeds that of a 2D CNN by over 16×. Lightweight models such as MoViNet-A2 [29] lower spatial resolution (e.g., 7×7 per frame) but still generate large cumulative features due to high channel counts. Thus, even compact video models produce massive intermediate representations, making direct offloading impractical under limited bandwidth. *This exposes a fundamental dilemma in video offloading: constrained networks demand aggressive compression, yet excessive compression harms task accuracy.* Fluctuating real-world bandwidth exacerbates the situation, underscoring the need for adaptive, learning-based compression that balances accuracy and efficiency.

To address these challenges, we propose ApproxBit, an *adaptive offloading system* that optimizes *content-dependent* compression of video features for devices with limited computational power and constrained network bandwidth. Specifically, ApproxBit is built on three key design ideas. *First*, ApproxBit leverages binary quantization to learn compact yet informative feature representations that preserve spatio-temporal dependencies while reducing transmission volume. *Second*, it adapts to varying bandwidth, latency, and accuracy requirements through three configuration knobs: *codebook size*, *presence or absence of an encoder*, and *offloading point*. At runtime, ApproxBit dynamically tunes these knobs to maintain optimal trade-offs across different network environments. *Third*, ApproxBit is designed to support various video tasks such as VAR and VQA by effectively compressing and transmitting diverse intermediate representations. To the best of our knowledge, ApproxBit is the first solution with adaptive offloading for video analytics, achieving high accuracy under stringent latency and bandwidth constraints.

We evaluate ApproxBit on VAR and VQA. VAR experiments use two datasets (SSv2 [14], Kinetics 600 [4]), two model architectures (MoViNet [29], MViTv2 [40]), and two edge devices (Jetson AGX Orin, AGX Xavier). VQA is evaluated using two models (Video-LLaVA [42] and LLaVA-OneVision [34]) on the MSVD dataset [5]. All experiments offload to an A100 GPU server under 3, 6, and 50 Mbps network conditions, representing scenarios spanning highly constrained satellite communication, 3G speeds, and urban broadband. Baselines include edge-only, server-only, DNN Surgery [41], H.264 offloading [55], DeepCOD [69], LimitNet [19], and DCVC-FM [38]. As shown in Figure 1, ApproxBit achieves the best accuracy-latency Pareto frontier for the VAR task at 3 Mbps, outperforming all baselines. Beyond this setting, we demonstrate that the same trend holds across all evaluated bandwidth settings and extends to other models and tasks. In addition, our binary encoding reduces the offloaded feature map size by up to 2142.4× with minimal accuracy drop.

Contributions. Our core contributions are as follows.

(1) **End-to-End Solution for Edge-Cloud Collaborative Processing of Video:** In contrast to previous solutions for offloading image recognition tasks, ApproxBit is the first to handle video analytics tasks, addressing the growing ubiquity and complexity of video content.
(2) **Adaptive Offloading with Learned Binary Codes:** ApproxBit provides two key technical innovations. It allows offloading at multiple points, so as to adapt to varying network conditions and meet user-specified accuracy or latency SLAs. It compresses features for offloading using learned binary codes tailored for videos, leading to compact offloaded features with reduced communication cost.
(3) **Insights**: Our results highlight that: (1) adaptive offloading is crucial to achieve accuracy-latency trade-offs across a broad spectrum of network conditions; (2) binary encoding for video

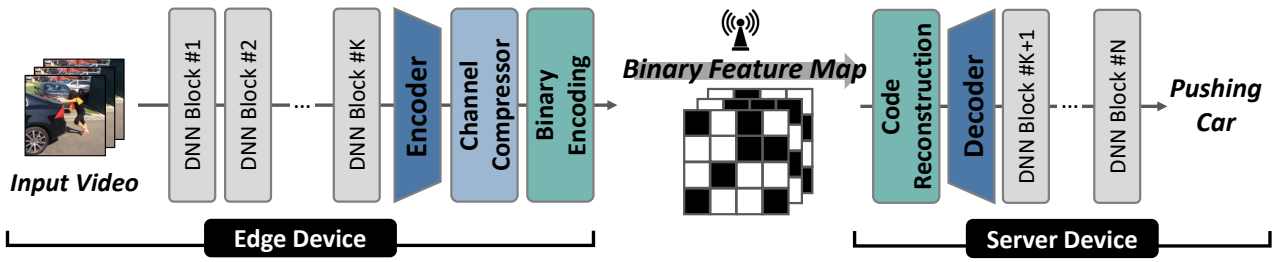

Figure 2: ApproxBit's overview: The workflow of ApproxBit starts with an adaptive offloading module on the edge device that selects configurations based on network conditions and accuracy-latency needs. Here, the video data is processed through initial DNN layers up to the chosen offloading point, then encoded into binary vectors. The resulting binary tensor is transmitted to the server for reconstruction and completion of DNN processing.

features, and further learned codes, are important to reduce latency while maintaining accuracy.

(4) **Generalizable Performance Benefit**: We observe performance gains of ApproxBit over prior offloading methods across a wide variety of experimental configurations on two classes of edge GPUs, different model classes (Transformer-based and CNN-based), various datasets, and different video analytics tasks (VAR and VQA).

2 Background

Offloading Computation. The concept of offloading is crucial for ML processing on edge devices for latency-critical applications like autonomous driving, robotics, and smart manufacturing. With the increasing data volumes from advanced cameras, LiDAR, and ultrasonic sensors, handling all the processing at the edge becomes increasingly challenging. Many existing solutions [2, 11, 20, 28, 32, 35, 41, 69] follow a strategy that balances high accuracy with reasonable latency. To achieve this, they focus on (1) minimizing computation at the edge device or (2) reducing the volume of data to be transferred. Further, the server has significantly higher computational power than the edge device. This nudges us toward the design of an asymmetric encoder-decoder for the feature maps, diverging from the conventional symmetric autoencoder structure, as we explain in Section 3.3.

Video Analytics Tasks. VAR and VQA are two representative video analytics tasks that require understanding temporal information across multiple frames. Although the final objectives differ, with VAR predicting an action label and VQA generating a natural language answer, both tasks rely on a common pipeline of extracting spatiotemporal features from raw video. In VAR, frames pass through a vision backbone (i.e., a pre-trained DNN) to produce video features, and a final classifier predicts the action label. VQA uses similar vision backbones, but the resulting video features are further tokenized and passed to a large language model (LLM) that performs reasoning and generates textual answer. As a result, the output of vision backbone serves as a shared intermediate representation.

In VQA, splitting the LLM across the edge and the server may appear to support adaptive offloading, but this is impractical for two key reasons. First, video queries generate long token sequences, and the KV cache grows accordingly, so partitioning inference would require transferring this cache. This results in substantial communication overhead, which eliminates any potential latency benefit. In addition, recent optimizations such as quantization [9] and token reduction [6, 43, 59, 77] do not reduce resource demands enough to make partial LLM execution feasible on typical edge devices. These limitations suggest that the LLM is not a suitable partition boundary. The intermediate feature map from the vision backbone is compact, fixed in size, and independent of the answer length. This property directly motivates our design choice: the backbone boundary becomes a more practical offloading point than any LLM stage.

Temporal Complexity in Video Models. Modern video deep models are designed for various video analytic tasks, including video action recognition and video question answering. These models typically take an input of multiple video frames, each with a moderate spatial resolution. A key design choice is that the spatial dimension can be drastically downsampled (*e.g.,* to 14x14), while the temporal dimension has minor to no downsampling, *e.g.,* MViTv2-B [40] and MoViNet-A2 [29] preserve a temporal dimension of 16 and 50, respectively, before its final classification layer. Similarly, VQA models [34, 42] process multiple frames as input, which increases the number of tokens handled by the large language models. This choice is due to the fact that temporal motion dynamics are critical to decode an action. However, such design brings major challenges to existing offloading approaches, as compressing the temporal axis aggressively can lead to a considerable loss in accuracy.

Vector Quantization for Offloading. Vector quantization (VQ), a classic technique in signal processing [15], offers an appealing solution to compress video features for offloading. VQ assigns a floating-point vector in a feature map to its closest code vector in a codebook, allowing a high-dimensional vector to be represented by a single integer index. However, one major drawback of VQ for video data is that a very large codebook (> 10K) [16, 73] is required to compress high dimensional vectors. For example, Guo *et al.* [16] utilized a 16,384-sized codebook for video compression. Handling large codebooks in video analytics models remains challenging. To address this issue, we identified a more suitable approach for our use case: *learned binary codes*, for representing the video intermediate features.

Binary Codes for Offloading. ApproxBit builds on binary feature representations for offloading. Recently, Wang *et al.* [67] introduce binary latent representations for an entirely different task of image generation via the diffusion process. Their model uses an autoencoder where the encoder maps the input image to a compact binary latent space using a Bernoulli distribution. This binary latent space provides a discrete representation of the image, allowing for efficient modeling of the prior distribution of the latent representations. While we are inspired by the binary encoding in [67], our approach fundamentally differs in both concept and implementation. Unlike [67], which emphasizes stochastic settings for generating images, our method focuses on deterministic binary codes tailored for compressing high-dimensional video data. Moreover, it used 2D convolutions for static image processing, whereas we leverage 3D convolutions that are better suited for modeling the temporal dynamics for video analytics. This enables effective video compression while preserving crucial temporal information. To the best of our knowledge, ApproxBit is the first solution to leverage a binary autoencoder for offloading.

3 ApproxBit Design

ApproxBit's high-level design is depicted in Figure 2. ApproxBit processes video data through a DNN model (adaptively partitioned between edge and server execution), compresses it with an encoder, and sends it to the server for decoding. We now describe each of its three major components: the adaptive offloading module, the asymmetric encoder/decoder, and the binary encoding module.

3.1 Adaptive Offloading Module

We design a configurable system architecture with network-adaptive parameters by leveraging multiple knobs. The end-to-end latency of our framework is expressed as follows:

$$t(o,\mathbb{I},C) = \underbrace{t_e(o) + t_{enc}(o,\mathbb{I}_s,C)}_{\text{edge}} + \underbrace{d/B}_{\text{network}} + \underbrace{t_{dec}(o,\mathbb{I}_s,C) + t_s(o)}_{\text{server}}. \quad (1)$$

Specifically, the DNN is partitioned at an offloading point o. At the edge, an encoder compresses the intermediate features at o by combining optional spatial encoding (denoted by the indicator function $\mathbb{I}_s$) and a binary encoder with codebook size C. The compressed binary features of size d are further transmitted over the network with bandwidth B. At the server, a decoder converts binary features back to floating points using the same codebook, followed by optional spatial decoding (if presented in the encoder). The decoded features are processed for the video task. $t_e(o)$ and $t_s(o)$ are the execution time on the edge and the server for processing the portion of DNN before and after o, respectively. t_{enc} and t_{dec} are the encoder and decoder execution times respectively.

In ApproxBit, the configuration space for optimizing offloading is vast, given the multiple dimensions of variability. We streamline this space to focus on the most impactful knobs, balancing the trade-off between accuracy and latency.

Offloading Point Selection o. The system allows offloading at multiple points in the DNN that are selected based on it architecture (see Section 4). To prune this space, we concentrate on two key offloading points at approximately 1/3 and 2/3 of the total FLOPs of the vision backbone, ensuring an appropriate balance between the edge device and server's workload.

Presence of Spatial Encoding and Decoding $\mathbb{I}_s$. Spatial encoding at earlier offloading points is critical for managing large feature maps produced in the earlier stages of any DNN model. By halving both temporal and (the two) spatial dimensions, as widely adopted in modern DNNs, it reduces transmission data size to 1/8. However, this reduction comes with a reduction in accuracy and hence, we make the presence/absence of encoder a binary configuration knob.

Encoder-Decoder Codebook Size C. Larger codebook sizes can increase accuracy by offering more representation patterns for nuanced feature encoding but also increase communication latency and memory overhead. Through empirical experimentation, we selected a set of possible codebook sizes to provide a good balance, capturing sufficient feature complexity while maintaining low transmission delays.

3.2 Dependencies and Network Adaptivity of Configuration Knobs

In ApproxBit, the knob selection is formulated as meeting either the user latency or the accuracy requirement.

$$\begin{aligned} &\arg\max_{o,\mathbb{I}_s,C} acc(o,\mathbb{I}_s,C) \quad \text{s.t.} \quad t(o,\mathbb{I}_s,C) < t_{user}, \quad \textbf{OR} \\ &\arg\min_{o,\mathbb{I}_s,C} t(o,\mathbb{I}_s,C) \quad \text{s.t.} \quad acc(o,\mathbb{I}_s,C) > acc_{user}. \end{aligned} \quad (2)$$

We consider the user's maximum latency (t_{user}) or minimum accuracy (acc_{user}) requirements, and optimizes over o (the offloading point), $\mathbb{I}_s$ (the presence of spatial encoding and decoding), and C (the codebook size). Further, some knob values are dependent and the interplay between various configuration knobs plays a critical role in determining the system's performance and adaptability. For example, selecting an offloading point early in the DNN model leads to larger feature maps, necessitating the use of the encoder for efficient data compression. To simplify this search space (note, each configuration requires independent training of our encoder-decoder), we pruned the configurations to a small set of key configurations. (specific values are in Section 4). These configurations were carefully chosen to represent a balanced spectrum of trade-off between accuracy and latency.

Finally, these knobs are dynamically adjustable, allowing ApproxBit to adapt to varying network conditions and user requirements in real-time. ApproxBit's network speed adaptation module leverages an accuracy-latency lookup table (LUT) to determine the optimal parameters (o, C and $\mathbb{I}$) for the current network condition, allowing for continuous adaptation to changing network conditions measured using any standard real-time network bandwidth monitoring tool; we use `iftop`. By preloading all potential configurations into the edge device's memory, we reduce the switching overhead. This is feasible because the backbone model is frozen during training and only the ApproxBit's module are updated, resulting in all configurations sharing the same backbone parameters. This incurs only a minor memory overhead because the backbone model parameters (such as, MViTv2's) are shared across all configurations and only parameters of the encoders and decoders need to be switched in.

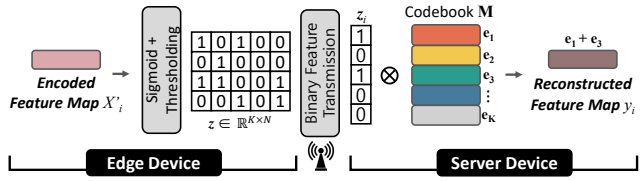

Figure 3: Binary encoding for offloading. The initial encoded feature map X', represented by floating-point numbers, are first normalized via a sigmoid function, constraining their values between 0 and 1. Subsequently, a thresholding operation converts each pixel value to binary format. This transformation compresses feature maps for offloading, significantly reducing communication cost.

3.3 Asymmetric Encoder/Decoder Network

Our architecture uses a computationally lightweight encoder with a single 3D convolutional layer, reducing intermediate feature map dimensions by a factor of 2. In contrast, the more complex decoder, operating on the server side, includes two ResBlocks, a self-attention layer, and a deconvolution layer for upsampling, designed for accurate reconstruction. For example, the encoder and decoder we used in MViTv2 have 573.42 MFLOPs and 2.99 GFLOPs, respectively. When compared to MViTv2, which operates at 52 GFLOPs, the computational costs are significantly lower.

3.4 Learning Binary Codes for Offloading

Learning binary codes rather than vector quantization (VQ) is particularly suited to our use case, as it handles the two system constraints well. *First*, binary codes are computationally simpler for edge devices with limited computation power, as VQ involves the additional step of searching for the nearest code in a codebook, while binary codes are created by a simple sigmoid thresholding. *Second*, binary codes create denser representation thus allowing offload through bandwidth-constrained networks, while preserving as much of the accuracy as possible. By directly learning binary codes, ApproxBit bypasses the overhead of searching and matching in a codebook, making it a more efficient approach.

To reduce communication costs during offloading, we introduce a binary encoding network designed to compress the data representation size of feature maps. A video feature map X at the edge, which is the output of (optional) spatial encoding, has a size of $D \times T \times H \times W$, i.e., $N = T \times H \times W$ feature vectors of size D, with spatial dimensions (height × width) $H \times W$ and time T. When spatial encoding is absent, X is simply a feature map from DNN layer without any spatio-temporal encoding. A lightweight channel compressor at the edge, realized using a single 3D convolution, is employed to reduce the feature dimension into K, producing a transformed feature map $X' \in \mathbb{R}^{K \times T \times H \times W}$. Next, the transformed feature map X' is normalized using Sigmoid function σ. Following this, the normalized feature map $\sigma(X')$ undergoes binary thresholding, and is then rearranged into binary codes $z \in \mathbb{R}^{K \times N}$ where $N = T \times H \times W$, expressed as $z = 1$, if $\sigma(X')$>0.5, and $z = 0$, otherwise. Since the binary code z is expressed using a 1-bit representation, whereas X is using 32-bit floating-point values, this results in a 32× reduction in data size. Binary code z is communicated to the server and further decoded on the server side using a codebook $\mathbf{M} \in \mathbb{R}^{D \times K}$ and a decoder. This decoding follows two steps. First, binary codes z are decoded using the codebook $\mathbf{M}$ by computing $\mathbf{M}z = y \in \mathbb{R}^{D \times N}$ and rearranging its layout into the original size $D \times T \times H \times W$. Second, the resulting feature map is further decoded using a heavy decoder (*e.g.,* using multiple 3D convolutions and self-attention layer).

A subtle yet important distinction is that learning binary codes is different from *binary quantization*, although both produce binary vectors. Binary quantization applies a fixed threshold to the input floating-point vectors, often leading to significant loss of information. In contrast, we create the codebook $\mathbf{M}$ within the context of an end-to-end network training setting, enabling a learning-based approach that constructs a complex nonlinear mapping from floating-point vectors to their binary codes, while preserving meaningful information of the input. This learned binary encoding is a novel aspect of our design and supports our goal of adapting to varying network conditions and content characteristics.

3.5 Encoder and Decoder Training

In training ApproxBit, the goal of the encoder/decoder is to accurately reconstruct feature maps, guided by the reconstruction loss $\mathcal{L}_{recon}$. However, focusing solely on reconstruction may not guarantee the model's accuracy. Therefore, our final training objective is formulated as:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_1 \mathcal{L}_{\text{recon}} + \lambda_2 \mathcal{L}_{\text{binary}}, \tag{3}$$

where $\mathcal{L}_{\text{task}}$, $\mathcal{L}_{\text{recon}}$, and $\mathcal{L}_{\text{binary}}$ are task loss, reconstruction loss, and binary loss, respectively. λ_1 and λ_2 are weight parameters for balancing the losses.

First, we preserve the loss $\mathcal{L}_{\text{task}}$ used for individual video analytics tasks. For VAR, $\mathcal{L}_{\text{task}}$ is cross-entropy loss that compares a predicted label and a ground-truth one. For VQA, $\mathcal{L}_{task}$ is still cross-entropy loss, yet compares each predicted token to ground-truth one in a decoded sequence of text tokens. Next, the reconstruction loss $\mathcal{L}_{\text{recon}}$ computes mean squared errors between the input of the encoder (original features) and output of the decoder (which decompresses the compressed features). Finally, the binary loss $\mathcal{L}_{\text{binary}}$ is simply binary cross entropy defined on the transformed feature map X' (see Section 3.4), given by

$$\mathcal{L}_{binary} = BCE(X', round(\sigma(X'))), \tag{4}$$

where σ is the sigmoid function. This loss encourages the binary vector before the sigmoid operation to have a value close to either zero or one. Both $\mathcal{L}_{\text{recon}}$ and $\mathcal{L}_{\text{binary}}$ are jointly optimized with $\mathcal{L}_{\text{task}}$ to train the encoder/decoder module. Empirically, we observe more stable training and better task performance when these losses are included. The importance of each loss term is empirically analyzed in Section 5.8. While $\lambda_1, \lambda_2 \in \mathbb{R}$ are used to balance the contributions of each loss term, for this analysis we set $\lambda_1, \lambda_2 \in \{0, 1\}$ to examine the effect of including or excluding each loss component.

An important design choice is that only parameters within the encoder and decoder are updated during training, but parameters within the base video model remain fixed. Thus, since only a single model is needed for all the different configurations, we can fit all the configurations within the limited device memory and switch among them at runtime.

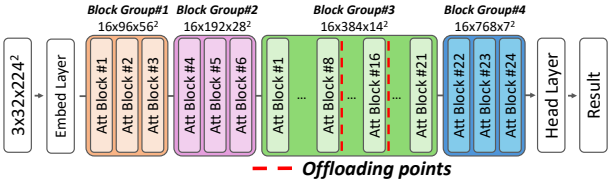

Figure 4: MViTv2-B model consists of a patch embedding layer, 24 attention blocks, and a head layer. After each attention block serves as a potential offloading point for this model. We choose two offloading points, after Block 8 and 16, to demonstrate the trade-off between accuracy and latency.

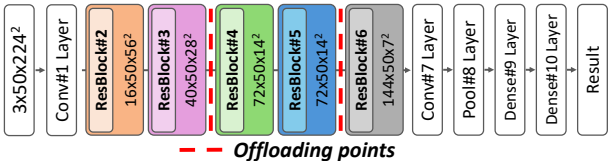

Figure 5: Offloading points in MoViNet-A2 at Block 3 and 5. Block 3 captures crucial features early on, ideal for constrained bandwidth scenarios, while Block 5, with its reduced feature map size, allows for more complex edge processing.

4 Implementation

We conducted our evaluations using an NVIDIA Jetson AGX Orin 32GB (more powerful) and Jetson AGX Xavier 32GB (less powerful) for the edge computing platform and an NVIDIA A100 GPU-equipped server. The AGX Orin demonstrates 6.25× higher TOPS (Tera operations per second) than the AGX Xavier and is the latest and most powerful embedded/mobile GPU. The eight model configurations for MViTv2 model that we show experimental results with together occupy only 4.6GB, well within the limits of Jetson's GPU memory (32GB).

Backbone Model Implementation. We consider MViTv2-B and MoViNet-A2 models for VAR and Video-LLaVA-7B and LLaVA-OneVision-0.5B models for VQA. MViTv2-B as shown in Figure 4, consists of multiple attention blocks, a foundational component of transformer-based model. Similarly, the vision encoders of Video-LLaVA and LLaVA-OneVision are also composed of multiple attention blocks. On the other hand, MoViNet-A2, as shown in Figure 5, consists of multiple blocks, each containing several convolutional layers and a residual link.

ApproxBit includes three configuration knobs: offloading point, encoder/no-encoder, and codebook size. The choices of the values of these knobs are not arbitrary but based on the structural properties of model backbones (such as attention block in MViTv2 and ResBlock in MoViNet). We make pragmatic choices to bound the possible values of these parameters that we use as each configuration of ApproxBit necessitates a new training. For the offloading point, we select two key spots that provide a balanced trade-off between early-stage ($\approx$ 1/3) and late-stage ($\approx$ 2/3) partitioning. For the encoder/no-encoder, it is simply a binary choice. For the codebook size, a large number of channels of the intermediate feature map at the offloading point guides us toward large codebook sizes and vice-versa. For example, for the Transformer-based model (MViTv2), we use larger codebook size compared to the CNN-based MoViNet model. Making these pragmatic choices, we prune the configurations for VAR to a final set of eight for MViTv2 and six for MoViNet-A2; for VQA, we have four configurations.

Training Overhead. We do not need to train the backbone model (such as MViTv2), but instead only train the lightweight encoder-decoder, resulting in a significantly smaller overall training cost. For example, the number of parameters in the the MViTv2 backbone model is 51.4M. In contrast, ApproxBit's encoder-decoder, which corresponds to the one used in MViTv2, has only 5.8M parameters. Although we train for each possible configuration of ApproxBit, such as with different offloading points and codebook sizes, the total training overhead remains far lower than training the full backbone, as only the small encoder-decoder moduels are updated while the backbone remains frozen.

Training Details. For network training, we use pre-trained weights of the MoViNet, MViTv2, Video-LLaVA and LLaVA-OneVision models as provided by the original authors. We adjust the weights of the encoder and the decoder during training. This method refines offloading efficiency without impacting the base model's integrity. For the VAR task, training parameters include setting λ_1 and λ_2 to 1 in Equation 3, the Adam optimizer with a 1×10^{-3} learning rate, and a batch size of 16 for 60 epochs. For the VQA models, λ_1 and λ_2 remain the same, but training is performed using the AdamW optimizer with a 2×10^{-4} learning rate, and a batch size of 16 for one epoch.

5 Evaluation

We evaluate ApproxBit on both VAR and VQA tasks, reporting metrics of accuracy, latency, and resource consumption against a wide range of baselines.

5.1 Dataset and Evaluation Protocol

For VAR, we conduct training and evaluation of all models on the Something-Something-v2 (SSv2) [14] and the Kinetics 600 (K600) [4] datasets, both leading benchmarks in action recognition. SSv2 consists of 174 human action classes, captured by 3-5 seconds videos with a frame rate of 12Hz. Kinetics 600 consists of 600 human action classes, with each video lasting 10 seconds at a frame rate of 25Hz. For MViTv2-B and MoViNet-A2 model processing, we uniformly select 32 and 50 frames respectively as input from an input video. The SSv2 dataset consists of 168,913 training (87%) and 24,777 test (13%) videos. The Kinetics 600 dataset consists of 357,771 training (93%) and 27,533 test (7%) videos. Training involves random cropping to 224 × 224 pixels for generalization, with no further augmentation. Inference employs center cropping of the same size. Further, we employ single-clip evaluation, which assesses a model's performance on a single, continuous segment within each video.

For VQA, we train ApproxBit using each model's corresponding fine-tuning dataset. For evaluation, we test Video-LLaVA-7B and LLaVA-OneVision-0.5B on the MSVD [5] dataset. This dataset contains 504 videos with 13,157 question-answer pairs, spanning diverse video durations and frame rates. We validate a model's performance via zero-shot QA evaluation [47], where accuracy is measured using a GPT-assisted scoring protocol [71].

In both tasks, to demonstrate the utility of ApproxBit across all bandwidths, with emphasis on its strengths in bandwidth-constraint networks, we consider three different network speeds: satellite network (3Mbps), 3G network (6Mbps), and 4G network (50Mbps).

5.2 Baselines

We compare ApproxBit against seven baselines, which can be categorized into video-level and feature-level offloading. The video-level baselines process or compress the original video before transmitting it to the server, where the full DNN executes. **Edge-only** executes the entire DNN model on the edge without offloading, while **Server-only** transmits the full video stream to the server for DNN processing. Video-level compression-based methods include **H.264** [55], the most widely used standard for video compression, and **DCVC-FM** [38], a neural video compression method that supports a wide quality range using learnable quantization scaling and periodic temporal feature refreshing. We also evaluate **LimitNet** [19], which incorporates a saliency detection mechanism and employs a gradual scoring algorithm to prioritize the transmission of important features. For video-level offloading baselines, we consider both low-compression and high-compression configurations. Although LimitNet was originally designed for image offloading, we adapt it for video offloading. For this, we apply its image compression method to each frame individually.

In contrast, feature-level offloading methods execute early layers of the network on the edge and transfer intermediate feature maps to the server. **DNN Surgery** [41] involves offloading feature maps directly to the server without any downsampling or encoding. **DeepCOD** [69] was originally designed for image tasks, so we replace its 2D operators with 3D counterparts to support the temporal dimension. While the original model reduces the intermediate feature map to 1/4 of its size in channel and spatial dimensions, video tensors require stronger reduction; therefore, we reduce the feature map to 1/32 of the original size (1/4 in channel and 1/2 in each of time/height/width), and to 1/128 under extreme bandwidth constraints (1/8 in channel, 1/4 in time, and 1/2 in height/width).

5.3 Key Findings

We summarize four key findings from ApproxBit's evaluation: (1) ApproxBit has a higher Pareto-optimal curve for VAR than all baselines for both models across all three network speeds (Figure 6). (2) ApproxBit dramatically reduces communication latency through far stronger feature-map compression (up to 2142.4× compared to the original feature map size of LLaVA-OneVision model) (Figures 7 and 8). (3) ApproxBit dynamically provides optimal configurations for varying network conditions as shown for both synthetic and real-world traces (Figures 9 and 10). (4) ApproxBit is task-agnostic and can generalize to two very different video analytics workloads (VAR and VQA), suggesting broad applicability in collaborative edge-server inference (Section 5.4).

5.4 Macro Benchmarks

In this macro evaluation, we measure the end-to-end latency and accuracy of ApproxBit and baselines, under the three different network speeds. Table 1 presents this result for the VAR task on the AGX Orin device with the MViTv2-B model. Table 2 shows the corresponding result for the VQA task on the MSVD with Video-LLaVA-7B and LLaVA-OneVision-0.5B models. The first row of Figure 6 complements Table 1 by illustrating the Pareto-optimal trad-off for VAR task, showing that ApproxBit's configurations lie on the Pareto frontier across all bandwidth settings. The results show that ApproxBit consistently provides the most balanced trade-off between end-to-end latency and accuracy, outperforming all baselines.

Table 1: MViTv2 latency and accuracy on AGX Orin for SSv2 dataset. Server GPU throughput (67.1ms) is 7.0× faster than edge (466.9ms). ApproxBit minimizes communication costs and leveraging computational differences between edge and server to achieve lower latency than baselines. DeepCOD(X)-BY denotes offloading with DeepCOD at a compression ratio of X at block Y. Similarly, ApproxBit-X-(No)Enc-BY refers to the ApproxBit method with a codebook size of X applied at block Y for offloading.

	Methods	Total latency (ms)			Accuracy	
		3Mbps	6Mbps	50Mbps	Top-1	Top-5
	MViTv2-B					
	Edge-only	466.9			70.8	92.7
Video-level	Server-only	6927.2	3497.2	478.7	70.8	92.7
Video-level	H.264-Low	3328.9	1778.5	414.2	69.0	92.1
Video-level	H.264-High	2608.8	1420.3	374.4	66.1	90.6
Video-level	DCVC-FM-Low	1068.9	1047.3	1028.2	69.8	92.5
Video-level	DCVC-FM-High	1156.8	1092.3	1035.5	70.5	92.7
Video-level	LimitNet-Low	725.5	440.8	190.3	64.2	90.3
Video-level	LimitNet-High	543.4	343.1	166.8	62.1	89.1
Feature-level	DNN Surgery-B8	13140.5	6718.0	1066.2	70.8	92.7
Feature-level	DNN Surgery-B16	13226.4	6803.9	1152.1	70.8	92.7
Feature-level	DeepCOD(1/128)-B8	399.9	349.7	305.6	67.0	90.8
Feature-level	DeepCOD(1/32)-B16	787.0	586.3	409.7	68.9	91.7
Feature-level	DeepCOD(1/128)-B16	485.5	435.4	391.2	68.9	91.9
Feature-level	ApproxBit-64-Enc-B8	306.9	302.7	299.0	67.9	91.2
Feature-level	ApproxBit-64-NoEnc-B8	364.5	331.1	301.6	69.3	92.0
Feature-level	ApproxBit-128-Enc-B8	315.3	307.0	299.6	68.1	91.4
Feature-level	ApproxBit-128-NoEnc-B8	431.5	364.6	305.8	69.5	92.2
Feature-level	ApproxBit-64-Enc-B16	392.8	388.6	385.0	69.2	92.1
Feature-level	ApproxBit-64-NoEnc-B16	450.5	417.0	387.6	69.9	92.3
Feature-level	ApproxBit-128-Enc-B16	401.2	392.8	385.5	69.4	92.3
Feature-level	ApproxBit-128-NoEnc-B16	517.4	450.5	391.7	70.2	92.4

Edge-only / Server-only: The **Edge-only** setting removes the communication overhead but suffers from high edge-side computation, resulting in substantial end-to-end latency. On MViTv2, the latency reaches 466.9 ms, and on Video-LLaVA-7B, it is 7791.0 ms, showing the heavy computation cost of Transformer-based models. In contrast, for both tasks, **Server-only** execution shows some reduction in latency under high-bandwidth conditions, but latency increases sharply as bandwidth decreases due to the large volume of raw video data. This makes communication overhead the dominant factor in the end-to-end latency.

Video-level Offloading: To address this communication bottleneck, video-level approaches compress the input video before transmission. For example, **H.264** compression shows comparable accuracy to Server-only while reducing video size by up to 65.3%. Even with significantly reduced offloading size, however, it still incurs

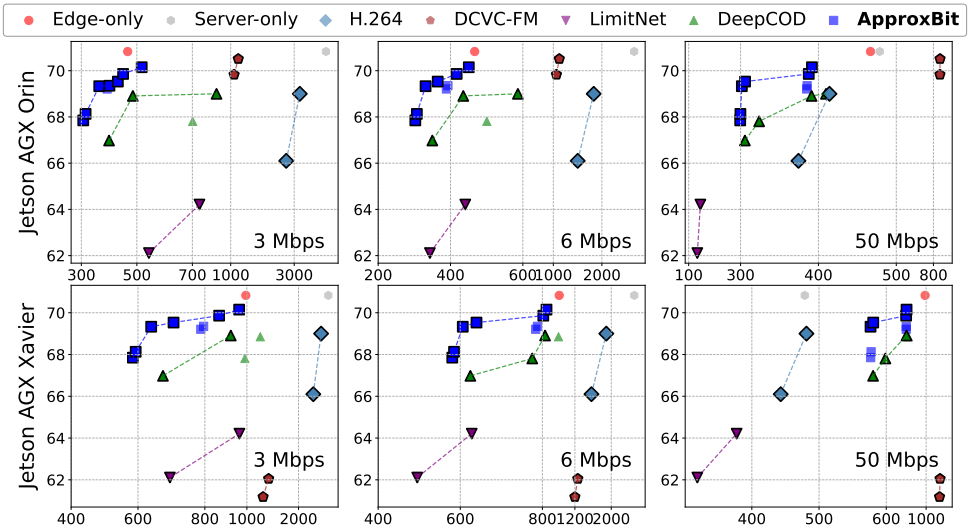

Figure 6: A comparison of Pareto curves (Y-axis: accuracy, X-axis: latency (ms)) for MViTv2 model on the SSv2 dataset between ApproxBit and baselines: ApproxBit consistently achieves a superior Pareto curve, outperforming baselines across all bandwidths. In particular, many baselines exhibit higher latency than Edge-only at 3 Mbps bandwidth setting, which diminishes the practical value of offloading. Notably, this trend generalizes across heterogeneous devices: the powerful Jetson Orin and the weaker Xavier platforms.

high communication costs on slow networks, and on fast networks, encoding time at the edge device comprises a large portion of the total latency. **LimitNet** further compresses video data by prioritizing important features, reducing total latency by further reducing the size of the encoded video. However, despite these compression efforts, the encoded feature size remains large due to the number of frames, which continues to impact overall latency. In addition, as LimitNet sends compressed information for each frame individually, it loses important temporal information, resulting in lower accuracy compared to ApproxBit in the VAR task. In the VQA task, this effect is mitigated because Video-LLaVA processes fewer frames than LLaVA-OneVision, allowing LimitNet to achieve slightly higher accuracy than ApproxBit. However, for LLaVA-OneVision, which takes in longer clips, the disrupted temporal structure degrades accuracy, while the encoding overhead and still-large offloaded size lead to higher end-to-end latency than ApproxBit. **DCVC-FM** maintains strong accuracy through learnable quantization scaling and periodic temporal feature refreshing, which preserve temporal consistency across frames. Nevertheless, although DCVC-FM can compress a video by up to 158.8×, its encoding and decoding stages are very heavyweight and constitute a large portion of the pipeline (even though the decoding happens on the server), so the end-to-end speedup remains limited.

Feature-level Offloading: DNN Surgery transmits intermediate tensors *without compression*, preserving accuracy but incurring severe communication overhead. The large spatial-temporal dimensions of video features make this approach untenable, as communication quickly becomes the dominant source of latency. In LLaVA-OneVision, where both the number of frames and channel dimensions are large, the transmission cost remains infeasibly high, requiring more than 9 seconds even at 50 Mbps, which highlights

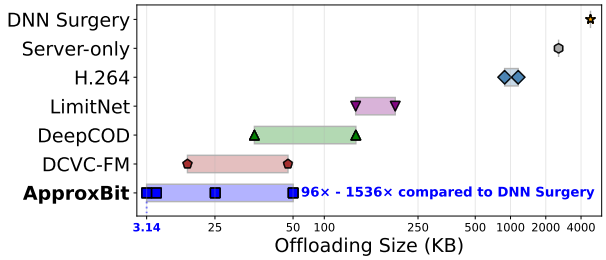

Figure 7: Offloading sizes of ApproxBit and baselines on MViTv2 using the SSv2 dataset. Because video models transmit feature maps or compressed content across multiple frames, other baselines typically require tens to thousand kilobytes. In contrast, ApproxBit reduces the offloading size by 96-1536× relative to feature maps, reaching sizes as small as 3.1KB.

the inefficiency of uncompressed feature offloading. On the other hand, **DeepCOD** reduces feature tensors across channel, spatial, and temporal dimensions. However, the achievable compression ratio is limited because aggressive compression leads to noticeable accuracy degradation. Also, the transmitted size increases linearly with the number of frames, so the communication cost remains substantial and limits the latency improvement under low network bandwidth. For instance, in the MViTv2 model, even with 1/128 compression at Block 16 (DeepCOD(1/128)-B16), the offloading size remains about 12.1× larger than ApproxBit-64-Enc-B16 configuration, while it yields lower top-1 accuracy.

Our Results: In contrast to prior feature-level offloading methods, ApproxBit achieves the lowest latency across most bandwidth settings by intelligently determining which configuration to use

Table 2: Latency and accuracy of Video-LLaVA-7B and LLaVA-OneVision-0.5B on MSVD dataset via AGX Orin: ApproxBit generalizes to VQA task and maintains consistently low latency across all bandwidth conditions. Also, multiple Pareto frontier configurations emerge, demonstrating that ApproxBit flexibly balances latency and accuracy while maintaining competitively high accuracy compared to other baselines.

	Methods	Total latency (ms) 3Mbps	6Mbps	50Mbps	Accuracy
Video-LLaVA-7B					
	Edge-only		7791.0		70.9
Video-level	Server-only	4121.4	2865.9	1761.0	70.9
Video-level	H.264-Low	4771.6	3862.4	3062.2	67.2
Video-level	H.264-High	3930.3	3415.1	2961.7	66.0
Video-level	LimitNet-Low	2855.0	2280.6	1775.1	68.7
Video-level	LimitNet-High	2472.1	2089.1	1752.2	67.5
Feature-level	DNN Surgery-B8	4437.4	3071.8	1870.1	70.9
Feature-level	ApproxBit-128-Enc-L8	1717.4	1711.8	1706.9	65.2
Feature-level	ApproxBit-128-NoEnc-L8	1792.1	1749.1	1711.4	68.0
Feature-level	ApproxBit-256-Enc-L8	1728.1	1717.1	1707.5	65.7
Feature-level	ApproxBit-256-NoEnc-L8	1877.4	1719.8	1716.5	68.4
LLaVA-OneVision-0.5B					
	Edge-only		7129.3		64.3
Video-level	Server-only	3373.0	2117.4	1012.5	64.3
Video-level	H.264-Low	4023.2	3113.9	2313.7	62.4
Video-level	H.264-High	3181.9	2666.7	2213.3	61.3
Video-level	LimitNet-Low	2621.8	1822.5	1119.1	63.0
Video-level	LimitNet-High	3439.8	2240.8	1185.7	62.2
Feature-level	DNN Surgery-B8	144404.8	72741.2	9677.2	64.3
Feature-level	ApproxBit-128-Enc-L8	1025.2	958.3	899.4	63.4
Feature-level	ApproxBit-128-NoEnc-L8	1899.6	1401.9	964.0	64.0
Feature-level	ApproxBit-256-Enc-L8	1152.9	1019.1	901.3	64.1
Feature-level	ApproxBit-256-NoEnc-L8	2894.5	1899.2	1023.3	64.5

based on bandwidth and latency or accuracy SLA. Across both VAR and VQA tasks, ApproxBit sustains accuracy close to the Edge-only / Server-only baselines while providing multiple Pareto-optimal configurations that flexibly balance latency and accuracy.

In the VAR task, as network speeds slow to 6 Mbps and 3 Mbps, DeepCOD performs worse in latency than Edge-only due to ineffective data compression. In contrast, ApproxBit minimizes offloading data size, achieving up to 34.2% lower latency than Edge-only at 3 Mbps network speed. Video data involves channels, temporal and spatial dimensions, with a specific data type, such as floating point or integer, impacting the overall data size. DeepCOD(1/128)-B16 compresses tensor data across channel, temporal, and spatial dimensions, considering floating point precision. In contrast, ApproxBit-64-NoEnc-Block16 compresses solely the channel dimension, employing binary representation, i.e., *a mere 1-bit encoding*, already achieving an impressive 1/32 compression ratio from the data representation. Among the 8 configurations from ApproxBit, when offloading the smallest size of data from Block16, which is only 3.1 KB, the top-1 and top-5 accuracy drops by 1.6% and 0.6%, respectively. In contrast, despite the offloading size being about 50 times larger, DeepCOD(1/128)-Block 16 experiences a 1.9% and 1.0% decrease in top-1 and top-5 accuracy, respectively.

For the VQA task, ApproxBit offers consistently lower latency across both Video-LLaVA-7B and LLaVA-OneVision-0.5B models

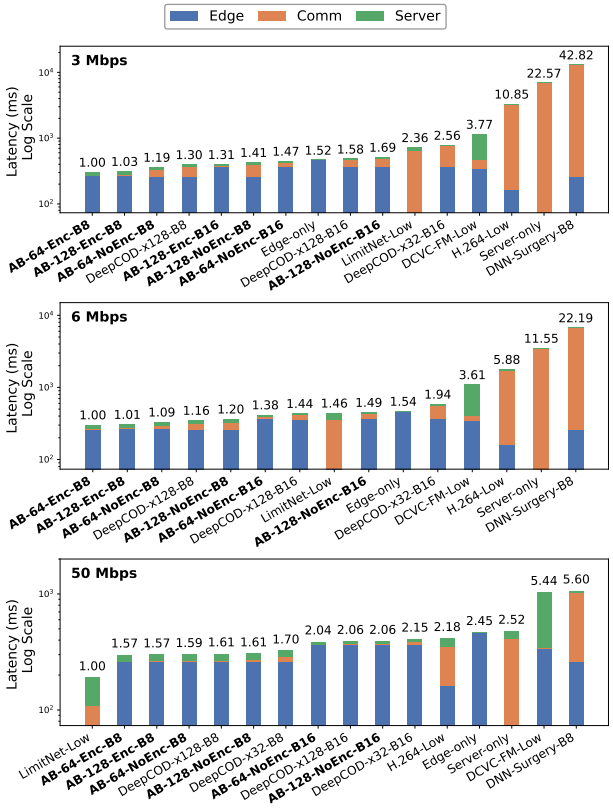

Figure 8: End-to-end latency breakdown of MViTv2 inference on SSv2 under 3, 6, and 50 Mbps network conditions, measured on a Jetson Orin device. Total latency is decomposed into edge processing, communication, and server processing on a log scale. The variants of ApproxBit (AB) (written in bold) maintain the lowest latency across all bandwidths while nearly eliminating communication overhead, whereas server-heavy and DNN-surgery baselines suffer from dominant communication delays.

while maintaining competitive accuracy. For Video-LLaVA-7B, the Server-only accuracy is 70.9%, and the ApproxBit configurations remain within a reasonable range, achieving 65.2-68.4%. For LLaVA-OneVision-0.5B, the Server-only accuracy is 64.3%, and ApproxBit preserves this almost exactly, achieving 63.4-64.5%. The difference in accuracy degradation between two models reflects the disparity in the size of their finetuning datasets: LLaVA-OneVision is trained on a substantially larger dataset (approximately 1M video-instruction pairs compared to 100K), which provides stronger robustness to small changes in intermediate feature representations. Consequently, although the two models exhibit different levels of accuracy degradation due to the size of finetuning data, ApproxBit consistently delivers higher accuracy and offers the lowest-latency configurations across all bandwidth conditions.

5.5 Latency Breakdown Analysis

To further understand the source of latency, Figure 8 decomposes the total latency into edge, communication, and server processing

under 3, 6, and 50 Mbps network speeds. Only the Pareto-optimal configurations of each method are displayed.

Server-only and DNN-Surgery approaches experience significant communication overhead, which dominates the total latency under all bandwidth settings. Video-level compression methods reduce the size of transmitted content and lower the communication cost compared to raw video transmission, but their encoding procedures significantly increase latency at the edge.

In contrast, our ApproxBit configurations consistently maintain the lowest latency under 3 Mbps and 6 Mbps by reducing communication overhead. At the 50 Mbps bandwidth setting, LimitNet achieves slightly lower end-to-end latency due to fast transmission, but this comes at the cost of reduced accuracy, showing 5.4% drop in top-1 accuracy compared to the ApproxBit-64-Enc-B8 configuration. Because ApproxBit preserves task-relevant semantics in the compressed feature representation, it avoids the accuracy degradation observed in video-level compression while maintaining competitive latency.

5.6 ApproxBit's Network Adaptability

ApproxBit's network speed adaptation module demonstrates its effectiveness in balancing accuracy and latency across diverse and changing network environments. The module's performance is illustrated in Figure 9, which presents the trade-off between top-1 accuracy and end-to-end latency under different network speeds. The network conditions are segmented into three parts: initially averaging 50Mbps (0-50s), followed by a period at 3Mbps, and concluding at 6Mbps, with some fluctuations within each period. These same three segments then recur. For the first three segments, the user specifies latency targets as SLAs, represented by the dotted lines, which increase from 300ms to 500ms. We see that ApproxBit is able to dynamically configure the pipeline so as to stay within the latency budget.

With slower network speeds, the offloading communication cost increases, leading to frequent shifts in the configuration even with minor changes in the network speed. The significant variation in top-1 accuracy between 50-100s in Figure 9 shows that ApproxBit effectively adapts to these network speed fluctuations. Note, however, that in the second segment, where the network speed is too low to support the low latency SLA, the accuracy exhibits an unstable pattern in the first half of the segment. In the final three segments, the user specifies accuracy SLAs changing from 68% to 71%. Again, ApproxBit is able to configure the pipeline such that the accuracy stays above the required SLA. Again, there is unstable latency behavior (2nd from last segment) when the network is too constrained to support the accuracy SLA.

Further, we evaluate ApproxBit's adaptive module on real-world bandwidth data from the 2nd Bandwidth Estimation Challenge at ACM MMSys 2024, organized by Microsoft [48]. This result is shown in Figure 10. The network bandwidth fluctuates between 5 and 8 Mbps, averaging 7.8 Mbps, indicating a dynamic environment with unstable conditions. From 0 to 1500 seconds, when the user latency SLA stays between 300ms and 500ms, ApproxBit never violates the SLA while maintaining the maximum top-1 accuracy possible under that SLA. Subsequently, from 1500 to 3000 seconds, as the user accuracy SLA shifts from 68% to 70%, the system sustains

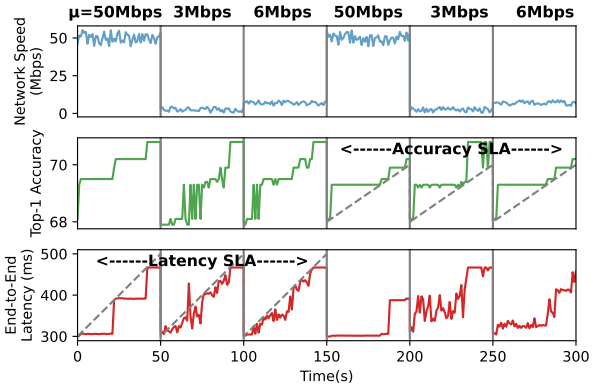

Figure 9: ApproxBit's network speed adaptation module dynamically responds to network speed variations. The top row shows the variation in the network speeds in 6 time windows. The top-1 accuracy and E2E latency, guided by the user-provided SLAs (the dotted lines, for latency in the first three segments and accuracy in the last three), stay within the requirements under all network conditions.

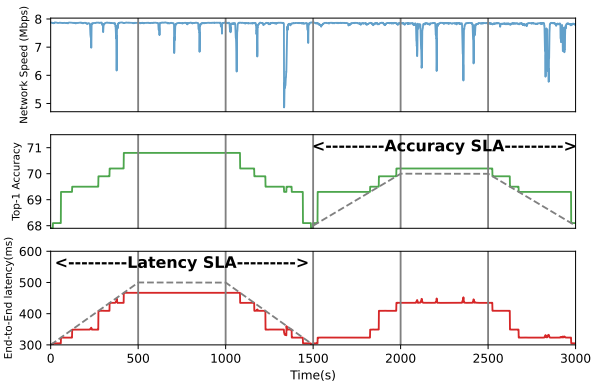

Figure 10: Dynamic adaptation of ApproxBit under bandwidth fluctuations from a real network trace. ApproxBit manages to consistently respect the latency or accuracy SLAs (the dashed lines), while maximizing accuracy (left half) or minimizing latency (right half).

minimum latency while respecting the SLA. This result shows that ApproxBit, integrated with a real-time network monitoring tool (`lftop`), effectively selects appropriate configurations in response to varying network conditions.

5.7 ApproxBit's Generalization

Model and Dataset. ApproxBit's generalizability is assessed by applying ApproxBit to a different vision backbone model and dataset. In addition to the Transformer-based MViTv2 model, we evaluate ApproxBit on MoViNet-A2, which has a lightweight CNN architecture. Table 3 shows only the Pareto-frontier configurations on SSv2 and Kinetics600. Figure 11 further visualizes the accuracy-latency trade-offs of ApproxBit and baselines under 3, 6, and 50 Mbps bandwidth settings. Despite changes in both the backbone and the dataset, ApproxBit continues to provide favorable accuracy-latency trade-offs and consistently outperforms the baselines. These

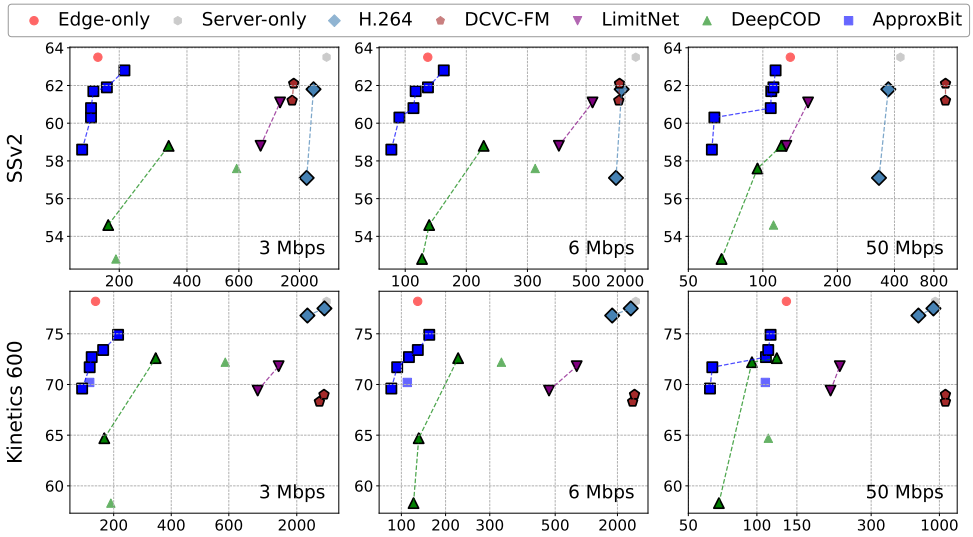

Figure 11: A comparison of Pareto curves (Y-axis: accuracy, X-axis: latency (ms)) for MoViNet model on the SSv2 and Kinetics 600 datasets between ApproxBit and baselines: ApproxBit remains on or near the Pareto frontier across all bandwidths and both datasets, outperforming other baselines. These results show that ApproxBit not only generalizes from Transformer-based models (MViTv2) to a CNN backbone (MoViNet) but also performs reliably across different video datasets.

Table 3: Model-level and dataset-level generalizability of ApproxBit at 3 Mbps network speed on AGX Orin: Beyond the models evaluated in Table 1, we additionally apply ApproxBit to MoViNet-A2 and report only the Pareto-frontier configurations on SSv2 and K600 datasets. Despite changing both the backbone and dataset, ApproxBit continues to outperform other baselines. This confirms that the proposed method generalizes across transformer and CNN architectures.

	Methods	Total Latency (ms) SSv2 / Kinetics600	SSv2 Top-1 Acc	Kinetics600 Top-1 Acc
	MoViNet-A2			
	Edge-only	136.9	63.5	78.2
Video-level	Server-only	6878.4 / 13738.5	63.5	78.2
Video-level	H.264-Low	3283.8 / 7208.2	61.8	77.5
Video-level	H.264-High	2556.5 / 2881.7	57.1	76.8
Video-level	DCVC-FM-Low	1648.4 / 6845.9	62.1	69.0
Video-level	DCVC-FM-High	1559.3 / 4833.9	61.2	68.3
Video-level	LimitNet-Low	997.2 / 1052.6	61.1	71.8
Video-level	LimitNet-High	720.1 / 788.0	58.8	69.4
Feature-level	DNN Surgery-B3	16777.7	63.5	78.2
Feature-level	DNN Surgery-B5	7639.5	63.5	78.2
Feature-level	DeepCOD(1/32)-B5	346.0	58.8	72.6
Feature-level	DeepCOD(1/128)-B5	167.4	54.6	64.7
Feature-level	ApproxBit-32-NoEnc-B5	215.3	62.8	74.9
Feature-level	ApproxBit-32-Enc-B5	123.6	61.7	72.7
Feature-level	ApproxBit-16-Enc-B5	116.7	60.8	70.2
Feature-level	ApproxBit-16-Enc-B3	90.4	58.6	69.6

results demonstrate that ApproxBit is not tailored to a specific model or data distribution. Instead, our approach remains stable across diverse vision architectures and datasets, highlighting its practicality for various scenarios.

Edge Device. Experiments are further conducted on another edge device, Jetson AGX Xavier, using the same model configuration, dataset, and bandwidth settings as the Orin evaluation. Orin is much more powerful than Xavier, with an 8.5× higher TOPS (at INT8). As shown in Figure 6, ApproxBit maintains a superior Pareto curve across 3 Mbps, 6 Mbps, and 50 Mbps network conditions on both devices. These observations indicate that ApproxBit's strategy generalizes across edge hardware with different performance capabilities. Although AGX Xavier provides lower computational power than AGX Orin, the relative gains remain consistent: baseline methods incur either high communication cost or heavy edge-side processing, whereas ApproxBit preserves efficient execution through its compact binary representation. Overall, the results demonstrate that ApproxBit remains effective even on weaker edge platforms.

5.8 Microbenchmark

Binary Codes vs. VQ. We discuss the representation size and the number of patterns of the vector quantizer and binary encoding module in Section 2. In this section, we comparatively evaluate these two methods for the MoViNet model. Setting the vector quantizer ApproxVQ's codebook size to 1024 and using an encoder/decoder network identical to ApproxBit's, we analyze both accuracy and offloading size, as shown in Table 5. Compared to ApproxBit-16-Encoder-Block5, which has the smallest offloading size among our models, ApproxVQ exhibits a 37.5% reduction in the offloading size. However, this reduced offloading size does not significantly lower the communication cost of ApproxVQ relative to ApproxBit. Even in a slow network like 3Mbps, this reduced offloading size translates to only a marginal 0.2ms difference in communication cost. However, in terms of accuracy, ApproxBit significantly outperforms ApproxVQ, with 5.9% and 2.6% improvement in Top-1 and Top-5 accuracy. This superior performance of ApproxBit, despite its larger

Table 4: Top-1 and Top-5 accuracy on SSv2 for the MoViNet ApproxBit-32-NoEnc-B5 configuration under different loss weight settings. The default setting is $\lambda_1 = 1$, $\lambda_2 = 1$. The reconstruction loss is important for maintaining accuracy, while the binary loss has a minor effect on accuracy but improves training stability.

λ_1	λ_2	Top-1 accuracy	Top-5 accuracy
0	0	62.4	87.3
0	1	62.2	87.3
1	0	62.8	87.5
1	1	**62.8**	**87.8**

Table 5: (MoViNet-A2 on Kinetics600) Comparison between ApproxVQ (quantizer from [64]) and ApproxBit-16-Encoder-Block5. Despite the somewhat smaller offloading size of ApproxVQ, this does not translate to a significant drop in communication latency. However, ApproxVQ decreases the top-1 accuracy by 5.9%, underscoring the advantage of binary encoding over vector quantization.

Methods	Offloading size	Top-1 accuracy	Top-5 accuracy
ApproxVQ	**1.53KB**	64.3%	87.8%
ApproxBit	2.45KB	**70.2%**	**90.4%**

offloading size relative to ApproxVQ, is due to a more compact representation by using binary codes. This enables the transmission of richer, more informative data, enhancing model accuracy and particularly suited for bandwidth-constrained environments.

Hyperparameter Sensitivity for Loss Function. To understand the contribution of each loss component, we perform an ablation study on the MoViNet model using the ApproxBit-32-NoEnc-B5 configuration trained on the SSv2 dataset by varying the loss weights with $\lambda_1, \lambda_2 \in \{0, 1\}$, where the default setting is $\lambda_1 = 1$ and $\lambda_2 = 1$. As shown in Table 4, removing the reconstruction loss (i.e., $\lambda_1 = 0$) leads to a clear drop in accuracy, indicating that preserving the structure of intermediate features after compression is important for maintaining task performance. The binary loss shows little impact on the final task performance, but we observe slightly improved training stability when it is included. When both losses are used together, the model achieves the best overall result, suggesting that they play complementary roles in guiding the encoder–decoder toward compact and task-relevant representations.

5.9 User Study

To evaluate ApproxBit's real-world robustness, we conduct a user study. We collect 30 videos in everyday environments, where participants we recruited perform actions that correspond to those defined by the SSv2 class labels; four samples are shown in Figure 12. These examples illustrate two successful cases and two failure cases. For evaluation on the "in-the-wild" dataset, we apply server-only processing on the MViTv2 model, which achieves the maximum accuracy, namely, top-1 accuracy of 66.6% and top-5 accuracy of 80.0%. When applying ApproxBit, the model maintains a top-1 accuracy of 63.3% and a top-5 accuracy of 80.0%, showing minimal performance degradation compared to the original model. This

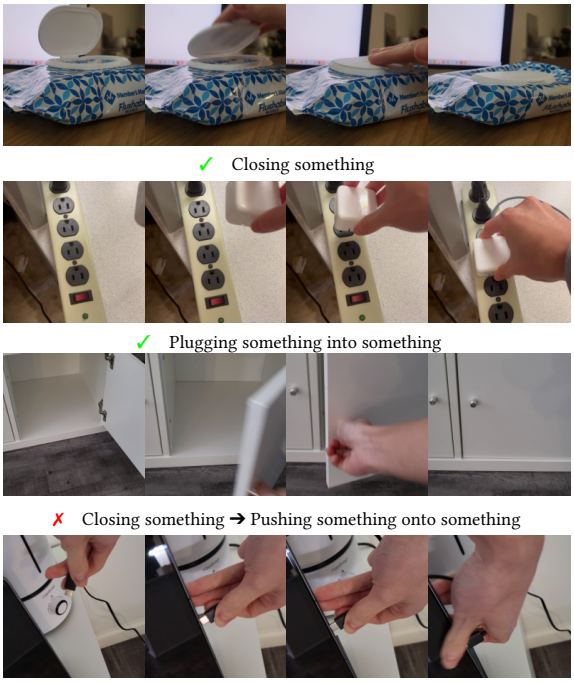

Figure 12: Examples of four real-world captured videos. The first and third videos share the same ground-truth label, as do the second and fourth. ApproxBit correctly infers the labels for the first two videos. In contrast, the bottom two examples illustrate cases where ApproxBit produces incorrect predictions, which appear to arise when key motion is not accurately captured during frame extraction.

shows that our approach maintains a high level of accuracy suitable for real-world applications. Some misclassifications occur due to challenging conditions, such as objects being overly zoomed in, actions being partially cut off by the camera, or subtle motion differences between ground truth and prediction. But the overall results indicate that ApproxBit can effectively process video-based tasks in the wild with reliability comparable to full-precision models.

6 Related Work

Efficient Video Analytics. Efficient inference is critical for deploying video analytic models on embedded or mobile devices with limited computational resources. To this end, several resource-efficient 3D convolutional models [18, 26, 30, 31, 52] have been developed, drawing inspiration from efficient 2D convolutional models [23, 58, 60, 76]. Recent video recognition models [29, 51, 57] incorporate the Neural Architecture Search (NAS) technique to automatically search for the most efficient architecture. NAS-driven models identify combinations of kernel sizes and layer configurations to achieve better efficiency-accuracy trade-offs. In parallel, efficient inference for Multimodal Large Language Models involves techniques such as token reduction [6, 43, 59, 77] and weight quantization [9] to reduce computational costs. This line of work is complementary to ours as we can partition these more efficient models between edge and server.

Video-level Offloading. Many studies [10, 75] aim to optimize video streaming for DNN consumption, but most rely on traditional control knobs such as resolution and frame rate that were originally designed for human perception. Task-driven approaches [45, 68, 74] improve accuracy while reducing the offloading size, but their effectiveness is not generalizable to new tasks. Although some video-level offloading methods [12] can be applied to multiple analytics workloads, they typically exhibit larger accuracy degradation compared to existing neural video compression techniques [25, 38]. Neural video compression achieves high reconstruction fidelity and compact bitstreams, but its encoding and decoding procedures impose substantial computational overhead. Motivated by these limitations, we shift the focus from video-level processing to feature-level offloading, resulting in a lightweight offloading mechanism that generalizes across diverse DNN architectures.

Feature-level Offloading. Edge-side data processing is crucial for applications like autonomous vehicles [8, 24, 62], healthcare monitoring [3, 39], and local device monitoring [56, 61], but deploying complex DNNs on constrained devices remains challenging. Optimizing DNNs includes pruning [17, 21, 44, 70], quantization [3, 17, 22, 63], layer-wise partitioning [11, 13, 28, 41, 66], early-exit DNN partitioning [32, 35], and input quantization [2], with cloud offloading to improve inference time [7, 53]. DeepCOD [69] uses an asymmetric autoencoder for data compression. Dyno [2] further advances this direction by quantizing the data to be offloaded based on runtime conditions. However, these methods focus on image data, but video offloading is hindered by the high communication costs of large feature maps and the added complexity from the temporal dimension. Our asymmetric autoencoder integrates a lightweight encoder and a binary quantization module to transform video representations into compact binary codes, a first in the field.

Vector Quantization and Binary Codes. Initially designed for data compression [15], vector quantization has recently been explored in deep learning, resulting in methods for clustering [27, 33], feature representation learning [64], and image generation [54, 72]. A key model is VQ-VAE, a CNN-based approach that encodes images into discrete latent codes and reconstructs them, using the latent space to model complex features. Recently, Wang *et al.* [67] explored binary encoding for image generation. Binary codes reduces data representation to a string of 0/1 bits, thus significantly reducing data size. We have the insight that this representation is ideal for offloading tasks in network-restricted environments by efficiently compressing data while preserving necessary information for accurate processing.

Our approach, distinct from [67], leverages *learned binary codes* for more efficient data transfer from edge to server device. Rather than using the index-based VQ-VAE approach, we achieve greater representational capacity and rely on a deterministic method for generating binary codes, ensuring predictable and reliable data compression.

7 Discussion

Training Overhead and Configuration Scalability. Training large video backbones typically requires substantial resources (*e.g.*, training MViTv2-B [40] requires 128 V100 32 GB GPUs with about 0.5 hours per epoch [1]), whereas ApproxBit optimizes only a lightweight encoder-decoder network whose parameter size is much smaller than that of the backbone. In our setup, each configuration was trained using a single NVIDIA A100 GPU with roughly 2 hours per epoch. Although these numbers are not directly comparable due to different training settings, they highlight the significantly lower resource requirement of our approach. Because the backbone weights are frozen and reused, new configurations or new backbone models can be supported by training only the small encoder-decoder network. This makes the framework efficient and easily extensible to other video backbone models.

Robustness under Various Network Conditions. Although we do not explicitly evaluate individual network factors such as RTT or packet loss, the real-world traces used in Figures 9 and 10 inherently capture their practical impact through time-varying network speeds. ApproxBit's adaptive configuration mechanism is designed to respond to the various network factors via the observed network speed. When network conditions degrade, the system naturally shifts toward more communication-efficient configurations, and in extreme cases it can fall back to fully edge-only execution, ensuring continued operation without relying on the network. These results suggest that ApproxBit can maintain stable behavior under practical and variable deployment environments.

8 Conclusion

Demanding video analytics tasks such as action recognition and question-answering pose significant challenges for edge devices. This has led to the exploration of offloading computation to more powerful servers. However, this strategy encounters bottlenecks in scenarios with constrained network bandwidth, where prior solutions either inadequately compress data for offloading or disproportionately rely on edge device computation. In our work, ApproxBit, we resolve this tension by balancing computational load and data transfer efficiency, introducing three key innovations to enhance offloading: (1) an integrated training strategy between the edge and the server that employs a customized joint loss function to optimize both tasks; (2) a significant reduction in data transmission latency through learning binary codes, leveraging a 1-bit representation for efficient data offloading; and (3) a network-adaptive system with multiple configuration knobs to adapt to varying network conditions. Looking ahead, we plan to extend ApproxBit's capabilities beyond VAR and VQA tasks to more complex video tasks such as augmented reality and explore collaborative inference across multiple edge devices. Additionally, we aim to optimize resource allocation *and* energy efficiency to enhance ApproxBit's effectiveness under diverse network conditions.

Acknowledgments

This material is based in part upon work supported by Adobe Research, Google, the Army Research Office funded Assured Autonomy Innovation Institute (A2I2) under Contract number W911NF-2020-221, and the National Science Foundation under Grant Numbers CNS-2333487/2333491 (CHORUS Center), IIS-2442739, and CNS-2140139. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Vaibhav Aggarwal, Mannat Singh, Anjali Sridhar, Yanghao Li, Shoubhik Debnath, Ronghang Hu, Will Feng, Xinlei Chen, Tingting Markstrum, Diana Liskovich, Anupam Bhatnagar, Chay Ryali, Haoqi Fan, Tete Xiao, Min Xu, Rahul Iyer, Christoph Feichtenhofer, Ross Girshick, Piotr Dollár, Aaron Adcock, Wan-Yen Lo, and CK Luk. 2022. Scaling Vision Model Training Platforms with PyTorch. https://pytorch.org/blog/scaling-vision-model-training-platforms-with-pytorch/. PyTorch Blog. Accessed: 2026-02-28.

[2] Mario Almeida, Stefanos Laskaridis, Stylianos I Venieris, Ilias Leontiadis, and Nicholas D Lane. 2022. Dyno: Dynamic onloading of deep neural networks from cloud to device. *ACM Transactions on Embedded Computing Systems* 21, 6 (2022), 1–24.

[3] Sourav Bhattacharya and Nicholas D Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. 176–189.

[4] Joao Carreira, Eric Noland, Andras Banki-Horvath, Chloe Hillier, and Andrew Zisserman. 2018. A short note about kinetics-600. *arXiv preprint arXiv:1808.01340* (2018).

[5] David Chen and William B Dolan. 2011. Collecting highly parallel data for paraphrase evaluation. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*. 190–200.

[6] Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. 2024. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*. Springer, 19–35.

[7] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 49–62.

[8] Mingyue Cui, Shipeng Zhong, Boyang Li, Xu Chen, and Kai Huang. 2020. Offloading Autonomous Driving Services via Edge Computing. *IEEE Internet of Things Journal* 7, 10 (2020), 10535–10547. https://doi.org/10.1109/JIOT.2020.3001218

[9] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. *Advances in neural information processing systems* 36 (2023), 10088–10115.

[10] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. 2020. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 557–570.

[11] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE transactions on mobile computing* 20, 2 (2019), 565–576.

[12] Xingtong Ge, Jixiang Luo, Xinjie Zhang, Tongda Xu, Guo Lu, Dailan He, Jing Geng, Yan Wang, Jun Zhang, and Hongwei Qin. 2024. Task-aware encoder control for deep video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 26036–26045.

[13] Soumendu Kumar Ghosh, Arnab Raha, Vijay Raghunathan, and Anand Raghunathan. 2024. Partnner: Platform-agnostic adaptive edge-cloud dnn partitioning for minimizing end-to-end latency. *ACM Transactions on Embedded Computing Systems* 23, 1 (2024), 1–38.

[14] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. 2017. The" something something" video database for learning and evaluating visual common sense. In *Proceedings of the IEEE international conference on computer vision*. 5842–5850.

[15] Robert Gray. 1984. Vector quantization. *IEEE Assp Magazine* 1, 2 (1984), 4–29.

[16] Haoyu Guo, Sida Peng, Yunzhi Yan, Linzhan Mou, Yujun Shen, Hujun Bao, and Xiaowei Zhou. 2024. Compact neural volumetric video representations with dynamic codebooks. *Advances in Neural Information Processing Systems* 36 (2024).

[17] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[18] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. 2018. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 6546–6555.

[19] Ali Hojjat, Janek Haberer, Tayyaba Zainab, and Olaf Landsiedel. 2024. LimitNet: Progressive, Content-Aware Image Offloading for Extremely Weak Devices & Networks. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*. 519–533.

[20] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.

[21] Yiming Hu, Siyang Sun, Jianquan Li, Xingang Wang, and Qingyi Gu. 2018. A novel channel pruning method for deep neural network compression. *arXiv preprint arXiv:1805.11394* (2018).

[22] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. 82–95.

[23] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[24] Hatem Ibn-Khedher, Mohammed Laroui, Mouna Ben Mabrouk, Hassine Moungla, Hossam Afifi, Alberto Nai Oleari, and Ahmed E Kamal. 2021. Edge computing assisted autonomous driving using artificial intelligence. In *2021 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 254–259.

[25] Zhaoyang Jia, Bin Li, Jiahao Li, Wenxuan Xie, Linfeng Qi, Houqiang Li, and Yan Lu. 2025. Towards practical real-time neural video compression. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 12543–12552.

[26] Boyuan Jiang, MengMeng Wang, Weihao Gan, Wei Wu, and Junjie Yan. 2019. Stm: Spatiotemporal and motion encoding for action recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2000–2009.

[27] Wenbin Jiang, Peilin Liu, and Fei Wen. 2017. An improved vector quantization method using deep neural network. *AEU-International Journal of Electronics and Communications* 72 (2017), 178–183.

[28] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices (ASPLOS)* 45, 1 (2017), 615–629.

[29] Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. 2021. Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16020–16030.

[30] Okan Köpüklü, Stefan Hörmann, Fabian Herzog, Hakan Cevikalp, and Gerhard Rigoll. 2022. Dissected 3D CNNs: Temporal skip connections for efficient online video processing. *Computer Vision and Image Understanding* 215 (2022), 103318.

[31] Okan Kopuklu, Neslihan Kose, Ahmet Gunduz, and Gerhard Rigoll. 2019. Resource efficient 3d convolutional neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*. 0–0.

[32] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–15.

[33] Dang-Khoa Le Tan, Huu Le, Tuan Hoang, Thanh-Toan Do, and Ngai-Man Cheung. 2018. DeepVQ: A deep network architecture for vector quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2579–2582.

[34] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. 2025. Llava-OneVision: Easy visual task transfer. *Transactions on Machine Learning Research* (2025).

[35] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 447–457.

[36] Jiahao Li, Bin Li, and Yan Lu. 2021. Deep contextual video compression. *Advances in Neural Information Processing Systems* 34 (2021), 18114–18125.

[37] Jiahao Li, Bin Li, and Yan Lu. 2022. Hybrid spatial-temporal entropy modelling for neural video compression. In *Proceedings of the 30th ACM International Conference on Multimedia*. 1503–1511.

[38] Jiahao Li, Bin Li, and Yan Lu. 2024. Neural video compression with feature modulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 26099–26108.

[39] Xiaohuan Li, Xumin Huang, Chunhai Li, Rong Yu, and Lei Shu. 2019. EdgeCare: Leveraging edge computing for collaborative data management in mobile healthcare systems. *IEEE Access* 7 (2019), 22011–22025.

[40] Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. 2022. Mvitv2: Improved multiscale vision transformers for classification and detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4804–4814.

[41] Huanghuang Liang, Qianlong Sang, Chuang Hu, Dazhao Cheng, Xiaobo Zhou, Dan Wang, Wei Bao, and Yu Wang. 2023. DNN surgery: Accelerating DNN inference on the edge through layer partitioning. *IEEE transactions on Cloud Computing* 11, 3 (2023), 3111–3125.

[42] Bin Lin, Yang Ye, Bin Zhu, Jiaxi Cui, Munan Ning, Peng Jin, and Li Yuan. 2024. Video-LLaVA: Learning United Visual Representation by Alignment Before Projection. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 5971–5984.

[43] Junyi Liu, Liangzhi Li, Tong Xiang, Bowen Wang, and Yiming Qian. 2023. TCRA-LLM: Token Compression Retrieval Augmented Large Language Model for Inference Cost Reduction. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 9796–9810.

[44] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the 16th annual international conference on mobile systems, applications, and services*. 389–400.

[45] Shengzhong Liu, Tianshi Wang, Jinyang Li, Dachun Sun, Mani Srivastava, and Tarek Abdelzaher. 2022. Adamask: Enabling machine-centric video streaming with adaptive frame masking for dnn inference offloading. In *Proceedings of the 30th ACM international conference on multimedia*. 3035–3044.

[46] Holger Lueschow and Roberto Pelaez. 2020. Satellite communication for security and defense. *Handbook of Space Security: Policies, Applications and Programs* (2020), 779–796.

[47] Muhammad Maaz, Hanoona Rasheed, Salman Khan, and Fahad Shahbaz Khan. 2023. Video-chatgpt: Towards detailed video understanding via large vision and language models. *arXiv preprint arXiv:2306.05424* (2023).

[48] Microsoft. 2024. 2nd Bandwidth Estimation Challenge at ACM MMSys 2024. https://www.microsoft.com/en-us/research/academic-program/bandwidth-estimation-challenge/. [accessed 1-July-2024].

[49] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. 2013. The latest open-source video codec VP9-an overview and preliminary results. In *2013 Picture Coding Symposium (PCS)*. IEEE, 390–393.

[50] Andri Prima Nugroho, Takashi Okayasu, Takehiko Hoshi, Eiji Inoue, Yasumaru Hirai, Muneshi Mitsuoka, and Lilik Sutiarso. 2016. Development of a remote environmental monitoring and control framework for tropical horticulture and verification of its validity under unstable network connection in rural area. *Computers and Electronics in Agriculture* 124 (2016), 325–339.

[51] AJ Piergiovanni, Anelia Angelova, and Michael Ryoo. 2020. Tiny video networks: Architecture search for efficient video models. (2020).

[52] Zhaofan Qiu, Ting Yao, and Tao Mei. 2017. Learning spatio-temporal representation with pseudo-3d residual networks. In *proceedings of the IEEE International Conference on Computer Vision*. 5533–5541.

[53] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. 2011. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*. 43–56.

[54] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. 2019. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems* 32 (2019).

[55] Iain E Richardson. 2004. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons.

[56] Michel Rottleuthner, Thomas C Schmidt, and Matthias Wählisch. 2021. Sense your power: The ECO approach to energy awareness for IoT devices. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 3 (2021), 1–25.

[57] Michael S Ryoo, AJ Piergiovanni, Mingxing Tan, and Anelia Angelova. 2019. Assemblenet: Searching for multi-stream neural connectivity in video architectures. *arXiv preprint arXiv:1905.13209* (2019).

[58] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[59] Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. 2024. Llava-prumerge: Adaptive token reduction for efficient large multimodal models. *arXiv preprint arXiv:2403.15388* (2024).

[60] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[61] Matthew Tancreti, Mohammad Sajjad Hossain, Saurabh Bagchi, and Vijay Raghunathan. 2011. Aveksha: A hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems. In *Proceedings of the 9th ACM conference on embedded networked sensor systems (SenSys)*. 288–301.

[62] Jie Tang, Shaoshan Liu, Liangkai Liu, Bo Yu, and Weisong Shi. 2020. LoPECS: A low-power edge computing system for real-time autonomous driving services. *IEEE Access* 8 (2020), 30467–30479.

[63] Frederick Tung and Greg Mori. 2018. CLIP-Q: Deep Network Compression Learning by In-parallel Pruning-Quantization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7873–7882. https://doi.org/10.1109/CVPR.2018.00821

[64] Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems* 30 (2017).

[65] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. 2023. Videomae v2: Scaling video masked autoencoders with dual masking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14549–14560.

[66] Li Wang, Xin Wu, Yi Zhang, Xinyun Zhang, Lianming Xu, Zhihua Wu, and Aiguo Fei. 2023. DeepAdaIn-Net: Deep adaptive device-edge collaborative inference for augmented reality. *IEEE Journal of Selected Topics in Signal Processing* 17, 5 (2023), 1052–1063.

[67] Ze Wang, Jiang Wang, Zicheng Liu, and Qiang Qiu. 2023. Binary Latent Diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 22576–22585.

[68] Xuedou Xiao, Yingying Zuo, Mingxuan Yan, Wei Wang, Jianhua He, and Qian Zhang. 2024. Task-oriented video compressive streaming for real-time semantic segmentation. *IEEE Transactions on Mobile Computing* (2024).

[69] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th conference on embedded networked sensor systems*. 476–488.

[70] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM conference on embedded network sensor systems*. 1–14.

[71] Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, et al. 2023. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. *arXiv preprint arXiv:2303.10420* (2023).

[72] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. 2021. Vector-quantized image modeling with improved vqgan. *arXiv preprint arXiv:2110.04627* (2021).

[73] Lijun Yu, José Lezama, Nitesh B. Gundavarapu, Luca Versari, Kihyuk Sohn, David Minnen, Yong Cheng, Agrim Gupta, Xiuye Gu, Alexander G. Hauptmann, Boqing Gong, Ming-Hsuan Yang, Irfan Essa, David A. Ross, and Lu Jiang. 2023. Language Model Beats Diffusion – Tokenizer is Key to Visual Generation. arXiv:2310.05737 [cs.CV]

[74] Shijing Yuan, Yuxin Liu, Song Guo, Jie Li, Hongyang Chen, Chentao Wu, and Yang Yang. 2024. Efficient online computing offloading for budget-constrained cloud-edge collaborative video streaming systems. *IEEE Transactions on Cloud Computing* (2024).

[75] Miao Zhang, Fangxin Wang, and Jiangchuan Liu. 2022. Casva: Configuration-adaptive streaming for live video analytics. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2168–2177.

[76] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6848–6856.

[77] Yiwu Zhong, Zhuoming Liu, Yin Li, and Liwei Wang. 2024. Aim: Adaptive inference of multi-modal llms via token merging and pruning. *arXiv preprint arXiv:2412.03248* (2024).