PacketGame: Multi-Stream Packet Gating for Concurrent Video Inference at Scale

Mu Yuan
ym0813@mail.ustc.edu.cn
University of Science and Technology of China
Hefei, China

Lan Zhang*
zhanglan@ustc.edu.cn
University of Science and Technology of China
Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
Hefei, China

Xuanke You
yxkyong@mail.ustc.edu.cn
University of Science and Technology of China
Hefei, China

Xiang-Yang Li
xiangyangli@ustc.edu.cn
University of Science and Technology of China
Hefei, China

ABSTRACT

The resource efficiency of video analytics workloads is critical for large-scale deployments on edge nodes and cloud clusters. Recent advanced systems have benefited from techniques including video compression, frame filtering, and deep model acceleration. However, based on our year-long experience of operating a real-time video analytics system on more than 1000 cameras, we identified a previously overlooked bottleneck of end-to-end concurrency: video decoding. To support concurrent video inference at scale, in this work, we investigate a new task, named video packet gating, which selectively filters packets before running a decoder. We propose a novel multi-view embedding approach for video packets and present PacketGame that has both theoretical performance guarantee and practical system designs. Experiments on both public datasets and a real system show PacketGame saves 52.0-79.3% decoding costs and achieves 2.1-4.8× concurrency compared to original workloads. Comparisons with four state-of-the-art complementary methods show the superiority of PacketGame in end-to-end concurrency.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**; • **Networks → Packet classification**; • **Computing methodologies → Concurrent algorithms**.

ACM Reference Format:
Mu Yuan, Lan Zhang, Xuanke You, and Xiang-Yang Li. 2023. PacketGame: Multi-Stream Packet Gating for Concurrent Video Inference at Scale. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3603269.3604825

*Lan Zhang is the corresponding author.

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3604825

1 INTRODUCTION

The demand for *video inference* (AI-powered video analytics) on various sources (IP cameras [37], drones [47], mobile live streams [52], and user-generated content [54]) has been growing rapidly. For example, smart city systems apply computer vision models to videos from tens of thousands of cameras for emergency response and environmental protection [55]; Twitch reports over 100,000 concurrent live streams at any given time [9] and recent works propose to improve video quality using neural super-resolution [26].

A typical video inference pipeline [19, 20, 31, 34, 53, 55] (see Fig. 1) first parses videos from real-time network streams (e.g., RTSP) or the local file system (e.g., MP4), then decodes packets and runs the AI model on RGB frames. Great efforts have been made to optimize the efficiency of the video inference pipeline and we divide them into four categories: (1) On-camera frame filtering [37] filters frames at the beginning of network stream analytics. On each camera, it selects frames based on the feature difference of successive frames and only encodes selected frames for transmission to servers. (2) Video compression [49]. Unlike common video encoding approaches [5] (e.g., H.264 and VP9) that are designed for human visual perception, video compression aims to minimize perception loss of inference models. Therefore it can effectively improve video transmission efficiency. (3) On-server frame filtering [19, 53] shares the same idea with on-camera frame filtering but moves the filter to the server. After decoding videos, this series of methods decide whether to perform inference on each frame, based on the neural network classifier. (4) Model acceleration [8, 29] focuses on the final inference phase of the pipeline. It improves the computational efficiency of inference models by pruning and fusing operators in deep neural networks.

System observation: concurrency bottleneck. At our university, we have developed a real-time video analytics system that processes more than 1000 cameras installed in the public area. To support mobile modeling and emergency response functionalities, we deployed state-of-the-art vision models [2, 28] and applied on-server frame filtering (InFi [53]) and model acceleration (TensorRT [8]) techniques to improve resource efficiency. On our edge GPU servers, these approaches effectively improve the system throughput from 27 FPS to 3,500 FPS. However, during a one-year operation of this system, we identified a previously overlooked

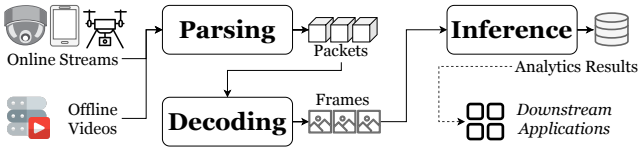


Figure 1: Video inference pipeline.

bottleneck: *concurrency* level, i.e., the number of streams that can be processed simultaneously. Experimental results show that the end-to-end concurrency is limited by the *video decoding* module (which takes encoded packets as inputs and outputs decoded RGB frames). Using 12 CPUs, the decoding module can only support 35 streams (18 streams using one GPU), while the concurrency level of the frame filtering and inference modules is orders of magnitude higher, 143 and 3015, respectively. The reason is that the decoder and the frame filter need to process all frames, while the inference module only needs to process a much smaller fraction (<2%) of frames passed through the filter.

In this work, we propose to add a selector module, named *packet gating* (i.e., selecting a subset of necessary video packets to decode from all streams), before the decoder in the video inference pipeline. Unlike prior frame filtering approaches that leverage low-level or learned features of RGB images [19, 37, 53], we attempt to make selection decisions based on packet metadata by parsing video streams. Packet gating can alleviate the computational overhead not only of the inference model but more importantly of the video decoder. Furthermore, packet gating does not require to modify video encoding and transmission protocols and thus can support commodity cameras and offline stored videos. The broad applicability and pluggability are lacking in on-camera frame filtering and video compression methods. Recall that packet gating is designed to improve the concurrency level. Therefore, cross-stream coordination (i.e., selecting packets across multiple streams based on limited decoding capabilities) is naturally an important consideration in our design, while prior work paid less attention to it.

Challenges. Building an effective framework for multi-stream packet gating involves two key challenges:

(1) Non-adaptive packet representation. It is challenging to develop representations of video packets that can adapt to various inference tasks and video content, especially before decoding, when only some metadata of the video packet is available, e.g., video codec, picture type, packet size, etc. Existing ideas we can refer to include packet classification [17, 43], network traffic classification [27, 41], and importance modeling of video frames [24, 50]. However, these representation approaches are not designed for the general pipeline of video inference. Experimental results show that they fail to effectively adapt to different video inference applications (§ 3.1).

(2) Inefficient cross-stream coordination. To improve the concurrency level, gating packets for a single video stream is not enough. For concurrent streams, the dynamic content and non-uniform decoding overheads (caused by video codec settings) make *stream-agnostic* resource schedulers largely sub-optimal. Experimental results show that the performance of the canonical round-robin policy significantly degrades as the number of concurrent streams increases (§ 3.2).

PacketGame. Overcoming challenges posed by packet gating requires multiple technical advances. First, we formalize the multi-stream packet gating problem and analyze the algorithmic structure. We present a general framework and figure out the main theory-practice gaps, namely packet sequence embedding and decoding dependency in GOP (group of pictures). Second, we design a sliding window-based temporal estimator that predicts the selection probability of each stream using online feedback and decision history. And we design a multi-view neural network that serves as the contextual predictor. The contextual predictor learns to embed packet sizes of independent and dependent frames as different feature views. It also fuses the probability returned by the temporal estimator for the final packet confidence. Third, taking the confidence of packets from multiple streams, we propose a combinatorial optimizer and prove its $1 - c/B$ approximation ratio, where B is the decoding budget and c is the maximal decoding cost of a packet. And based on the theory of multi-armed bandits [21, 58], we prove a $\tilde{O}(\sqrt{T})$ regret bound of our overall algorithm, where T is the number of decision rounds. We implement our theory-backed algorithm, named PacketGame , as a plug-in between the packet parser and decoder in the video inference pipeline.

We summarize three key contributions of this work as follows:

- We identify an overlooked system bottleneck of concurrency level in the video inference pipeline. And we propose a new idea: packet gating, which complements the decoding efficiency of existing methods.
- We present PacketGame , the first framework for multi-stream packet gating. PacketGame leverages a lightweight temporal estimator and contextual predictor to adaptively represent packets. We design a combinatorial optimizer with a proven approximation ratio for efficient cross-stream coordination. The overall performance of PacketGame is proved to have an online regret bound.
- We implement PacketGame and conduct evaluations of four inference tasks on public videos and a real system with 1108 cameras. Experimental results show that PacketGame saves 52.0-79.3% decoding costs and achieves 2.1-4.8× concurrency compared with original workloads. Comparisons with four state-of-the-art complementary methods [8, 37, 49, 53] show the superiority of PacketGame in end-to-end concurrency and broad applicability.

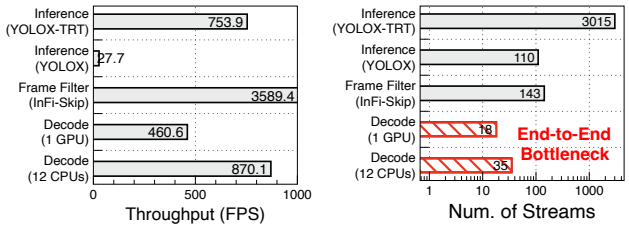
This work does not raise any ethical issues.

2 BACKGROUND

This section first introduces our motivating use cases: widespread video inference tasks with large-scale concurrency requirements (§ 2.1). Then we discuss prior efforts for efficient video inference (§ 2.2). Next, based on the experience of operating a real system and quantitative analysis, we identify the bottleneck of the concurrency level (§ 2.3). Then we propose a new idea, namely packet gating, and claim the design scope and uniqueness (§ 2.4).

2.1 Motivating Use Cases

(1) Surveillance video inference. Surveillance cameras are ubiquitous in today's society and are widely used for security in homes and public areas. AI models have empowered many analytics functionalities on videos from tens of thousands of IP cameras in cities, such as emergency response [55].



(a) Independent Throughput **(b) Potential Concurrency**

Figure 2: Performance benchmarks on a server using 25FPS 1080p video streams.

(2) Mobile video inference. Various mobile devices like phones, drones, and robots are equipped with cameras. Due to limited communication and computing resources, many applications offload video inference to the edge and cloud servers [38], e.g., construction site management based on the camera worn by workers [3].

(3) Offline video inference. Video-sharing platforms store a huge amount of videos (e.g., there are at least 800 million videos on YouTube [11]). To improve the quality of services, various AI models are developed for functionalities including activity-level advertising [54], content-based retrieval [45], and resolution enhancement [26].

High-concurrency demand. With the increasing number of hardware and users, regardless of the video source, these applications have a common demand for concurrent processing at scale.

2.2 Efficient Video Inference

Existing work explored mainly four types of approaches to improve the efficiency of video inference. We introduce four representative methods (also what we used for comparison experiments in § 6.5) as follows:

(1) Video compression. Grace [49] proposes a video compression algorithm that significantly saves the network bandwidth without degradation of inference performance. Grace optimizes the codec compression strategy for a target inference model by analyzing both spatial frequencies and colors.

(2) On-camera frame filtering. Reducto [37] filters frames at the camera side by adaptively setting a threshold on frame difference based on low-level visual features (e.g., pixel and area). Reducto only encodes and transmits selected frames, thus saving both network bandwidth and backend inference computation.

(3) On-server frame filtering. InFi [53] uses a lightweight convolutional neural network to learn to filter decoded frames. Its end-to-end learnability provides feature embedding with robust discriminability for difference inference tasks.

(4) Model acceleration. TensorRT [8] implements many inference acceleration techniques for NVIDIA GPUs, including weight quantization, layer fusion, parallel execution, etc.

2.3 Concurrency Bottleneck

Surprisingly, we find that most existing work focuses on latency and throughput metrics for video inference, but the end-to-end bottleneck of the concurrency level is under-explored.

Real-system experience. When developing a video analytics system that is fed with more than 1000 concurrent video streams from IP cameras, we applied TenosrRT [8] and InFi [53] to improve the efficiency. As shown in Fig. 2a, TensorRT greatly improves the throughput of the inference model (YOLOX [28]) from 27.7 to 753.9 FPS. And InFi achieves a 99% filtering rate while preserving over 90% accuracy. However, when it comes to real-time processing, the video decoder becomes the end-to-end bottleneck. As shown in Fig. 2b, using 12 CPUs / one TITAN X GPU on the edge can only support 35 / 18 concurrent streams. The potential concurrency level of the downstream filter and inference model is 143 and 3015, respectively, orders of magnitude higher than that of decoding. The reason is quite obvious: the decoder and filter need to process every packet, while the inference model only runs on a small fraction of the frames passed through the filter.

Expensive hardware solution. Deploying more hardware to decode is a direct but expensive way to alleviate the bottleneck. Considering 1080p 25FPS streams, dedicated decoder hardware (e.g., Kiloview DC230 [4]) costs about $62.5 per stream. The most advanced NVIDIA A100 [7] GPU costs $144 per stream per year on Azure. And using Azure vCores CPU costs $132 per stream per year. In addition to the cost of the hardware, deploying dedicated decoders or more machines with CPUs / GPUs brings additional communication overhead and considerable engineering work. Taking our system as an example, additionally spending more than $100,000 per year for analyzing 1000 cameras is prohibitive for most organizations. So we seek a pure software solution.

Quantitative condition. Beyond the above specific case, we now give a quantitative condition that decoding is the concurrency bottleneck: $T_{inference} > (1-r)T_{decode}$, where $r \in [0,1]$ is the filtering rate and $T_{inference}, T_{decode}$ are throughput of inference model and decoder, respectively. Note that, for different analytics frequencies of downstream applications, we can directly configure the decoder to decode according to a fixed frequency [5]. The filtering rate depends on the video content and inference tasks. As an empirical reference, previous works [19, 37, 53] report that the potential filtering rate for various video inference tasks is around 80-99%. After applying model acceleration techniques [8, 29], the condition commonly holds.

2.4 Design Space

Scope. We focus on the general ingest phase of video inference workloads, i.e., from receiving videos to obtaining inference results (see Fig. 1). After inference, downstream applications might use analytics results in various ways. Potential optimizations in downstream applications are out of the scope of this work.

Design goals. We have four main design goals:

(1) Reduce decoding. First of all, we must reduce the decoding overhead while preserving high inference accuracy.

(2) Support commodity cameras. Commodity cameras usually do not support secondary programming. Supporting legacy cameras and new commercial cameras is painstaking and even infeasible for new video compression and on-camera frame filtering techniques.

(3) Support offline videos. Offline stored videos have been encoded with a certain video codec. An ideal packet gating solution should be codec-agnostic and require no additional transcoding overhead.

Table 1: Feature comparison of our proposed packet gating and complementary methods designed for efficient video inference.

Methods	Reduce Decode	Commodity Cameras	Offline Videos	Cross-Stream
Video Compression	✓	✗	✗	✗
On-Camera FF	✓	✗	✗	✗
On-Server FF	✗	✓	✓	✗
Model Acceleration	✗	✓	✓	✗
PacketGame	✓	✓	✓	✓

(4) Cross-stream coordination. The packet gating strategy for large-scale concurrent streams should have a global optimization view and elastic scalability.

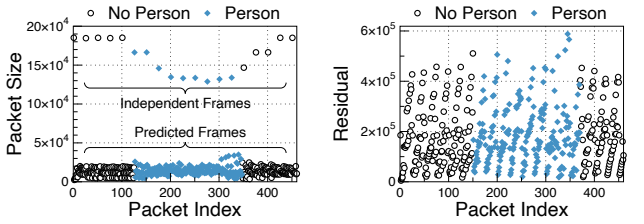
Core idea. To alleviate our identified bottleneck of concurrency level and meet all the design goals, we propose a new idea: to add a filter module for parsed packets, named *packet gating*, before the decoder, which selects a subset of necessary video packets to decode from all streams. From the perspective of the above four design goals, Tab. 1 illustrates the novelty of our proposed multi-stream packet gating approach PacketGame compared with existing methods. Note that, our proposed PacketGame has no conflict with the listed existing methods and can work as a complement to them (see Sec. 6.5).

3 CHALLENGES

Building an effective method based on our proposed idea of packet gating involves two non-trivial challenges, namely non-adaptive packet representation and inefficient cross-stream coordination.

3.1 Non-Adaptive Packet Representation

Since packet gating works just after the parser, only some metadata of the video packet is available, such as video codec, picture type, packet size, etc. So, in principle, we aim to build a mapping from packet metadata to whether this packet is necessary to decode or not. This work is the first step towards this objective. But from a high-level view, packet gating is a packet decision model. We can refer to ideas that have been explored in network management, including packet classification [17, 43] and network traffic classification [27, 41]. And to ensure video delivery quality under limited network resources, several approaches [24, 50] are proposed to selectively discard video frames that minimize the distortion quantified by metrics such as PSNR (Peak Signal-to-Noise Ratio) and MS-SIMM (Multiscale-Structural Similarity). Recent work [52] also proposes a residual-based feature, that can be estimated using video packet sizes, for the selective super-resolution task. However, experimental results show that these approaches either fail to discriminate between necessary and redundant packets or cannot adapt to various inference tasks. For example, setting the maximal false-positive rate as 10%, residual-based selection results in only 6.1% true-positive rate while PacketGame achieves 76.6%. Considering a person counting inference task, we plot the packet size and residual feature [52] of a video clip in Fig. 3. To discriminate



(a) Temporal and non-linear correlation of packet size. **(b) Residual feature has poor discriminability.**

Figure 3: Distribution of packet sizes and residual features for person detection inference task.

packets with and without detected people requires a temporal and non-linear representation of the packet size. And handcrafted residual features of necessary and redundant packets present highly indistinguishable patterns.

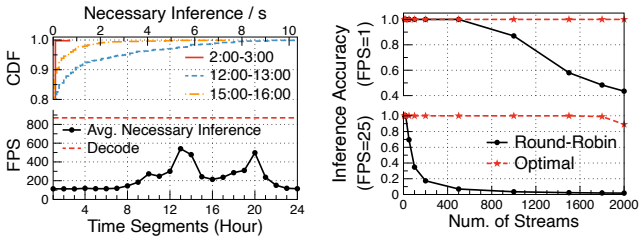
▹ ***Insight 1: Metadata-feedback hybrid representation of video packets.***

End-to-end learning can provide adaptability to data and task-dependent representation [14, 46, 53]. Therefore, we leverage a super-lightweight neural network to learn to predict with packet metadata for various video content and inference tasks. On the other hand, the downstream inference model can provide online feedback for packet gating performance. So we propose to combine metadata and feedback as a hybrid representation for video packets (§ 5.1, 5.2).

3.2 Inefficient Cross-Stream Coordination

To maximize the overall analytics concurrency on multiple video streams, we also need to carefully coordinate the packet decoding resource across streams. Using a stream-agnostic (e.g., round-robin) scheduler leads to significant performance degradation. We benchmark on our video analytics system using 1108 streams. Fig. 4a shows the distribution of necessary inference for the person counting task in one day. Given a video stream, we consider an inference as necessary if the counting result is different from the latest number. The necessary inference for the person counting task presents two (morning and evening) peaks that are consistent with common sense. And we can see that the decoding capability (870 FPS) is actually enough if we can perfectly identify necessary packets from all streams (540.8 FPS at most). As shown in Fig. 4b, compared with the optimal cross-stream strategy, the round-robin approach results in sub-optimal performance quickly with an increased number of streams, since it is agnostic to the necessity of decoding. For example, given 25 FPS analytics frequency and 90% target accuracy, the optimal strategy supports 2000 concurrent streams, while the round-robin approach can support only 30 streams.

▹ ***Insight 2: Gating packets with a global awareness of multi-stream states.***



(a) Distribution of necessary inference in one day. **(b) Inference accuracy with two analytics frequencies.**

Figure 4: Stream-agnostic scheduling results in significant performance degradation.

As an online decision process, we need to carefully trade-off between exploration and exploitation over all streams, using the constrained decoding resource. So we design a combinatorial algorithm that considers both gating *confidence* (i.e., selection probability) and heterogeneous decoding overheads as the stream state (§ 5.3).

4 FRAMEWORK

With these insights in mind, we define the multi-stream packet gating problem and analyze its algorithmic structure (§ 4.1). We propose a framework with a theoretical performance guarantee (§ 4.2) and identify the key gaps from theory to practice (§ 4.3).

4.1 Formalization

Given m concurrent streams of videos, at each round t, under the decoding resource budget B, we select a subset of packets from the arrived m packets to decode. We define $c_{t,i}$ as the cost of decoding the packet of stream i at round t. To illustrate the practical implication, we give a running example based on our experience on a video analytics system, where we have m=1000 concurrent RTSP streams (25 FPS) from IP cameras. We divide one second into 25 rounds, so we receive 1000 packets at each round. Common video codecs (e.g., H.264 and VP9) have two types of encoded frames, independent (I-frame) and predicted (P/B-frame) [5], and their costs are heterogeneous. In our example, the edge server's resource budget supports decoding 11 I-frame packets or 32 P/B-frame packets at each round. Let $x_{t,i}$ denote the feature vector summarizing the information of stream i at round t. In our case, it consists of the packet size and picture type. Naturally and same as previous work [19, 37, 53], we assume the online feedback of whether running an inference is redundant is available. We define a set of Bernoulli variables $r_{t,i}$ as the redundancy feedback of the packet from stream i at round t. For example, consider the backend inference model as a detector of abnormal behaviors. Then if a decoded frame returns as "normal", we set the feedback as 0; and if it returns as "abnormal", we set the feedback as 1. The objective is to maximize the number of decoded packets that are necessary:

$$\max \sum_{t=1}^{T} \sum_{i \in S_t^{\mathcal{A}}} r_{t,i} \quad \text{s.t.} \quad \forall t \in [T], \ \sum_{i \in S_t^{\mathcal{A}}} c_{i,t} \leq B, \tag{1}$$

where $S_t^{\mathcal{A}}$ denotes the set of packets selected by algorithm $\mathcal{A}$ at round t. Maximizing this objective function means that we select more necessary packets to decode, thus translating to less loss of inference accuracy. Note that, in real-time operation, we have no way of knowing false negatives (frames that should have been decoded but were not) unless decoding every frame. Setting up a parallel pipeline to periodically decode all frames and evaluate the recall (similar to the fast-slow path design in LiveNet [36]) is a promising way to complement the selective feedback in our formalization.

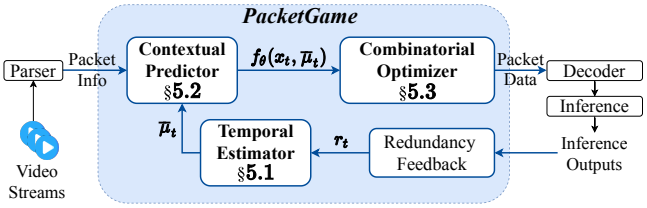


Figure 5: Overview of PacketGame framework. PacketGame serves as a plug-in between the packet parser and decoder in the video inference pipeline.

4.2 Overview

To solve such an optimization problem, we draw on theories of multi-armed bandits [21, 58] and propose a framework PacketGame . As shown in Fig. 5, PacketGame consists of three main modules. First, a temporal estimator estimates the feedback expectation $\bar{\mu}_t$ using the history of collected feedback. Second, a contextual predictor combines information from both packet metadata and the calculated feedback expectation. We propose to build a neural network-based predictor $f_\theta(x_t, \bar{\mu}_t)$ that predicts gating confidence. Third, given the confidence and decoding costs of all streams, we need to solve the constrained combinatorial optimization problem. The optimizer returns the final packets $S_t^{\mathcal{A}}$ to decode. The decoder feeds selected frames to the inference model and we use the inference outputs to calculate the redundancy feedback r_t. And temporal estimator updates once new feedback arrives.

4.3 Theory-Practice Gaps

Our PacketGame framework has a good theoretical performance guarantee (bounded online regret), see Sec. 5.4. To be practical in real systems, however, we summarize two gaps that have to be filled by careful design.

(1) Metadata embedding with inductive biases. The formalization assumes an effective feature vector x is given. In practice, we need to design how to embed metadata with inductive biases [25] of video packets. Take the packet size as an example, it depends on many configurations, including codec algorithms, bit rates, picture types, etc. Embedding prior knowledge (e.g., I and P/B frames have different size patterns in our case) as inductive biases is critical to neural network-based learning performance.

(2) Heterogeneous decoding overheads. The budget will be trivial if item costs are uniform since a greedy selection is optimal. In our case, decoding video packets in different GOPs (group of pictures) incur heterogeneous overheads. As illustrated in Fig. 6, the costs of current packets depend on the picture type, GOP size, decoding dependencies, and previous decisions. For the first stream, decoding

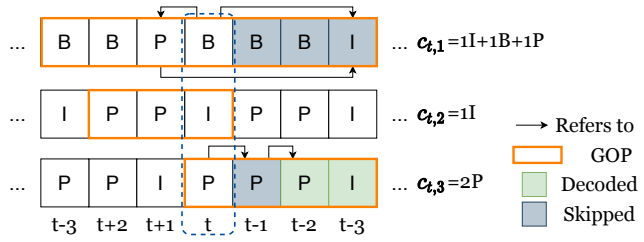


Figure 6: Non-uniform and dynamic decoding costs of packets in multiple video streams.

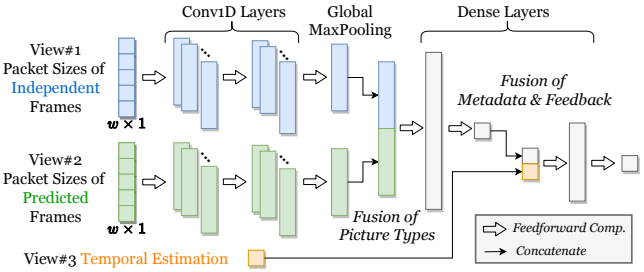


Figure 7: Architecture of contextual predictor. Colors represent different views of information.

the current B-frame (bidirectional predicted picture) packet depends on the first I-frame in GOP and the next P-frame. As we assume that the first I-frame is skipped (not decoded), the current cost of the first stream is $1I+1B+1P$. While for the second stream, the cost is $1I$ since the current I-frame has no dependency to decode. And for the third stream, we need to trace back to the first decoded P-frame, resulting in a $2P$ cost. We need to elaborate a combinatorial optimizer that targets our packet gating task. For example, our strategy needs to trade off this characteristic case: decoding the current P frame or waiting for the next I frame, especially when the GOP is large (typically in live streaming applications).

5 PACKETGAME DESIGN

This section introduces design of temporal estimator (§ 5.1), contextual predictor (§ 5.2), and combinatorial optimizer (§ 5.3) in PacketGame framework. And we present the performance guarantee of our overall algorithm (§ 5.4).

5.1 Temporal Estimator

The necessity of many inference tasks has temporal continuity. For example, an abnormal event will persist in the video frame for a period of time. And the live video will need resolution enhancement during periods of network issues. Therefore, the online redundancy feedback returned by the inference model can be useful for packet gating.

Redundancy feedback. Like previous work of frame filtering [19, 37, 53], we assume a redundancy measurement is available. It is natural and prevailing: For object counting tasks, if the inference result is the same as the latest one, we regard it as redundant; For detection tasks, if the IoU of bounding boxes is higher than a threshold, the inference is redundant; For classification tasks, we can set a subset of labels as redundant or check whether the label changes. Once we receive redundancy feedback on our selected packets, we record them for each stream.

Exploitation-exploration trade-off. We propose to set a temporal window of length w and calculate the probability of selection in next round by $\bar{\mu}_{t,i} = \frac{1}{w}\sum_{j=1}^{w} r_{t-j,i} + \sqrt{\frac{3lnT}{2T_{t,i}}}$, where $T_{t,i}$ denotes the times of selecting stream i in recent w rounds, i.e., $T_{t,i} \leftarrow \sum_{j=1}^{w} 1(i \in S_{t-j})$. The first item, the average reward in the temporal window, is for exploitation, i.e., the larger of recent reward, the higher probability to select for the next selection. The second item is for exploration. Intuitively, if we have only made a small number of attempts to a stream, then it should be selected with high priority. Related theoretical results [21] show that this form of exploitation-exploration trade-off has a good performance guarantee for online decision making. Our ablation experiments (see Sec. 6.3) also show the effectiveness of the proposed temporal estimator.

5.2 Contextual Predictor

Besides redundancy feedback from the past, metadata of present packets can also be helpful. For example, a sudden fire will cause relatively static frames to change significantly, causing the size of encoded packets to fluctuate. Unlike traditional packet classification tasks where we can manually design effective rules based on metadata such as the port number and size [41], the correlation between the metadata and inference redundancy label is non-linear and complex. So we design a neural network targeted to the metadata of video packets.

Video encoding-related inductive biases. Due to different encoding mechanisms [5], independent frames can be decoded by the packet itself while predicted frames need to refer to other packets. For example, a B-frame depends on the I-frame and the next P-frame in a GOP (see the first stream in Fig. 6). Therefore, the scale and distribution of packet sizes of these two types of packets are different. On the other hand, intuitively, the size of the two types of packets has different meanings. For independent frames, the packet size reflects the richness of the current frame. And for predicted frames, the packet size reflects the change compared to the reference frame. This difference and complementarity in input information inspire us to adopt multi-view learning [51], which has achieved success in many learning tasks. And we use separate embedding layers to learn features for two types of frames' packet sizes.

Sequential feature embedding. Like the temporal estimator, we set a temporal window of length w. So the input is a w-dimension vector that records the most recent w packet sizes. We utilize 1D convolution layers and a global max pooling layer as the feature embedding block, which is a common practice for time-series classification [57]. In fact, we also explored other types of neural network layers, including fully connected, recurrent, and LSTM layers. As a proof of concept, we select the 1D convolution layer due to its parameter efficiency and experimental performance. Then we concatenate two views of feature embedding and use a dense layer to predict the redundancy label.

Metadata-feedback fusion. Both the temporal estimator and the neural network can predict the probability of redundancy. We propose to fuse their predictions using dense layers after concatenating their outputs. Fig. 7 shows the architecture of our contextual predictor, which has three views of input information, i.e., packet sizes of independent and predicted frames and the output of temporal estimation. Our ablation experiments show that this metadata-feedback fusion brings considerable improvement (§ 6.3).

Multi-task extension. Running multiple inference models on the same video stream is a common demand for complex analytics systems, such as smart cities [18]. Our neural network design can be flexibly extended to support multi-task packet gating by setting the length of the last dense layer as the number of tasks. Benefiting from cross-task correlation, experimental results (see Fig. 11) show that a multi-task contextual predictor outperforms single-task ones. Similar results have also been reported by multi-task learning work [56].

Parameter optimization. The supervision label of our contextual predictor is redundancy feedback and we normalize it into 0-1 range, like previous work [19, 53]. And we adopt the binary cross-entropy loss, formally $L(r, y) = -(r\log(y) + (1-r)\log(1-y))$, where r, y denote true and predicted labels, respectively. In principle, our proposed neural network can be optimized by any gradient decent-based algorithms in an end-to-end manner. As a proof of concept and considering the implementation efficiency, in this work, we first train the contextual predictor (Python script) using offline inference records. Then we transform the trained weights into a binary runtime file and deploy it for real-time packet gating (no online parameter update). We will explore learning-related advances like online optimization and domain adaptation in future work.

5.3 Combinatorial Optimizer

Given the gating confidence of streams calculated by our contextual predictor, we need to select a subset of packets under the decoding budget. Recall that PacketGame is designed for concurrent analytics at scale, its computation efficiency and scalability must be very high. Therefore, we propose to first greedily select packets according to the ratio of confidence to cost, i.e., $f_{\theta_t}(x_{t,i}, \bar{\mu}_{t,i})/c_{t,i}$. Then using the remaining budget, we decode as many as possible packets that the current prioritized packet refers to. The decoding dependency in a GOP is in a form of a directed graph and can be efficiently parsed. This task-specific combinatorial algorithm has $O(m\log(m))$ computation complexity and linear scalability with respect to the number of concurrent streams m. Such a greedy-based optimizer can be arbitrarily bad for general combinatorial problems. Fortunately, the cost of decoding a video packet is approximately fractional. And based on this characteristic, we can prove that it has an approximation ratio of $1 - c/B$ where B is the decoding budget and c is the maximal cost.

Lemma 1 (Approximation Ratio). *For our approximately fractional knapsack problem, the greedy algorithm has an approximation ratio of* $1 - \frac{c}{B}$.

See Appendix A for the proof. In practice, c/B is typically lower than 0.05, which translates to a 95% higher approximation of the optimal results.

Algorithm 1: Multi-Stream Packet Gating Algorithm

input : Number of rounds T, window length w

for $t = 1, ..., T$ **do**
  Parse packet features $\{x_{t,i}\}_{i=1}^{m}$;
  for $i = 1, ..., m$ **do**
    $T_{t,i} \leftarrow \sum_{j=1}^{w} 1(i \in S_{t-j})$;
    $\bar{\mu}_{t,i} \leftarrow \frac{1}{w}\sum_{j=1}^{w} r_{t-j,i} + \sqrt{\frac{3lnT}{2T_{t,i}}}$;
    $p_{t,i} \leftarrow \frac{f_\theta(x_{t,i}, \bar{\mu}_{t,i})}{c_{t,i}}$;
  $P_t \leftarrow$ indices sorted by descending order of $p_{t,i}$;
  $b_t \leftarrow 0, k \leftarrow 0, S_t \leftarrow \emptyset$;
  while $b_t < B$ **do**
    $S_t \leftarrow S_t \cup \{P_t[k]\}$;
    $b_t \leftarrow b_t + c_{t,i}$;
    $k \leftarrow k + 1$;
  Decode all packets in S_t and maximal packets that $P_t[k]$ refers to by remaining budget;
  Receive redundancy feedback $r_{t,i}, \forall i \in S_t$;

5.4 Performance Guarantee

Based on the proposed three modules, Alg. 1 shows the overall algorithm, where $\bar{\mu}$ denotes the temporal estimator and f_θ denotes the contextual predictor. The length of the temporal window is set empirically and our experiments (Fig. 13) show its effect on performance. In each round, Alg. 1 first parses packet features (packet size and picture type) and predicts confidence $p_{t,i}$ for each stream. Next, PacketGame selects packets from m streams and sends them to the decoder. Then the inference model processes the decoded frames and returns redundancy feedback to PacketGame . Using Lemma 1 and existing results [21, 58], we can prove the regret bound of Alg. 1.

Theorem 1 (Regret Bound). *The regret of Alg. 1 in T rounds is at most* $\tilde{O}(\sqrt{T})$.

See Appendix. A for the proof. Such a theoretical guarantee is important for service-level objectives in real applications. Our experimental results also show the effectiveness of Alg. 1 in practice, e.g., 2.1-4.8× concurrency with over 90% inference accuracy under the same budget.

Other possible design choices. In principle, any online decision-making approach has the potential to work for our packet gating problem, e.g., deep reinforcement learning (DLR) [15]. DRL has achieved success in many networking and resource management tasks [39, 40]. However, due to its combinatorial nature, the action space is exponentially complex. Also, the requirement of fixed observation space and action space makes DRL approaches lack of scaling elasticity. For example, when the number of concurrent streams changes, we need to rebuild and retrain the deep neural network. On the other hand, in this work, we formalize the packet gating problem without considering query queueing and set a fixed decoding budget for each round. Scheduling packets with two orthogonal dimensions, time and streams, is more complex and is our future work.

Table 2: Summary of Datasets and Inference Tasks

Dataset	Video Source	Inference Task
Campus1K	IP Camera	Person Counting (PC) Anomaly Detection (AD)
YT-UGC	Offline Video	Super-resolution (SR)
FireNet	Mobile Camera	Fire Detection (FD)

Table 3: Overall efficiency improvement on four tasks with 90% target inference accuracy.

Method	Budget Saving / Concurrency Level			
	PC	AD	SR	FD
Temporal	52.6%/2.3x	71.8%/3.6x	75.8%/4.1x	50.5%/1.9x
Contextual	68.1%/2.9x	38.9%/1.7x	14.4%/1.1x	31.0%/1.5x
PacketGame	75.2%/3.6x	79.3%/4.8x	76.2%/4.3x	52.0%/2.1x

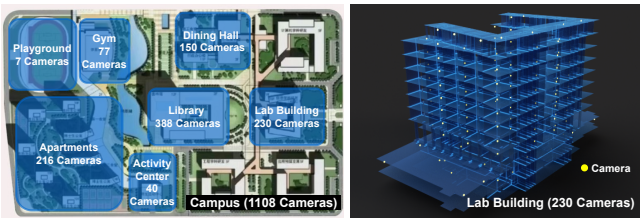


Figure 8: Distribution of 1108 cameras in a campus real-time video analytics system.

6 EVALUATION

We evaluate PacketGame prototype on various video inference tasks using both real analytics systems and public datasets. Our highlights are as follows:

- PacketGame saves 79.3% decoding budgets and achieves 4.8× concurrency level with over 90% inference accuracy, compared with original video inference workloads (§ 6.3).
- PacketGame shows robust effectiveness with respect to involved variables, including training size, window length, GOP size, and video codec (§ 6.4).
- PacketGame outperforms complementary state-of-the-art methods in improving end-to-end concurrency and has wider applicability (§ 6.5).

6.1 Implementation

We implemented and open-sourced [1] PacketGame based on the FFmpeg [5] and TensorFlow [13] libraries. To show that our design does not depend on specific frameworks, we also open-source the implementation based on MindSpore [6]. PacketGame parses binary videos using the *av_parser_parse2* API and obtains the packet size and picture type by accessing the *size* and *pict_type* attributes, respectively. PacketGame trains the contextual predictor with the RMSprop optimizer. If not mentioned, the same hyper-parameters are used: 5 window length, 2 convolutional layers with 32 units, 128 dense units, 2048 batch size, and 0.001 learning rate.

6.2 Experimental Setup

Datasets and inference tasks. To evaluate the performance of PacketGame , we selected three video datasets, as summarized in Tab. 2. (1) *Campus1K*. This dataset comprises videos in h265 format collected from 1108 IP cameras deployed across our university campus. The cameras captured footage at a frequency of 10 seconds per hour over a 24-hour period, resulting in a total of 4,432 ($1108 \times 10 \times 24/60$) hours of video. Fig. 8 shows the distribution of these cameras on campus. We deployed a person detection [28] model for mobility analysis (PC) and a pose-based action classification [2] model for anomaly detection (AD). (2) *YT-UGC* [10]. This large-scale dataset consists of 1179 videos in h264 format generated by YouTube users. To simulate fluctuations in video quality caused by bandwidth issues, we manually re-encoded video segments using lower bit rates. The dataset covers a diverse range of content and video qualities. For the inference task, we deployed a super-resolution model [1] on video clips to enhance video quality (SR). (3) *FireNet* [33]. The FireNet dataset contains 47 videos with fire and 17 videos without fire, captured by mobile phones. Since the original video clips only contain frames with or without fire, we randomly inserted fire clips into videos without fire. The dataset provides a challenging scenario for fire detection. For this inference task, we deployed a fire detection model [33] (FD).

Ethical considerations. When using the Campus1K dataset, all cameras were installed in public areas by the university, and we obtained proper authorization to conduct experiments. The processed packets and inference results, which include bounding boxes and object classes, do not raise privacy concerns, as it does not involve the collection or storage of sensitive or personally identifiable information.

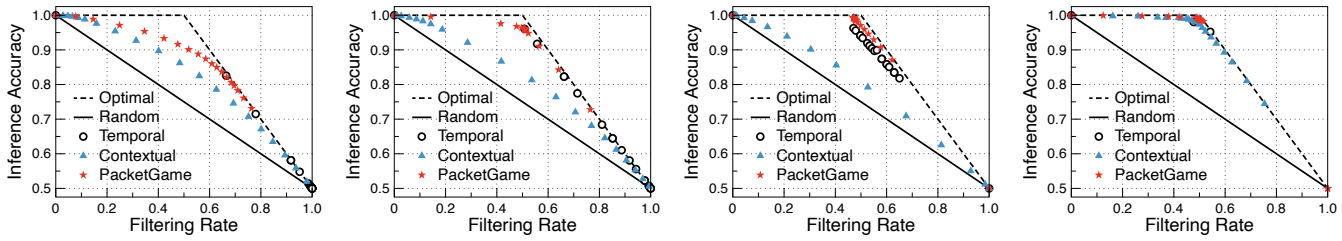
Baselines. To our best knowledge, PacketGame is the first method for video packet gating, thus we consider the following baselines and ablated versions of PacketGame for end-to-end comparisons: (1) Random. Randomly selects packets to decode under the budget. (2) Temporal. Use our proposed temporal estimator only. (3) Contextual. Use our proposed contextual predictor only (removing the temporal view). For complementary methods designed for video inference optimization, we consider four SOTA approaches: (4) Grace [49], A video compression approach. (5) Reducto [37], an on-camera frame filtering approach. (6) InFi [53], an on-server frame filtering approach. (7) TensorRT [8], a model acceleration approach. Sec. 2.2 has already given a more detailed introduction to these methods.

Devices. For the on-camera deployment experiment, we use a mobile phone (XIAOMI Mi 5). And for all the other experiments, we use an edge server that runs Ubuntu 20.04 with 12 Intel Core i7-5930K CPUs 3.50 GHz and 1 NVIDIA TITAN X GPU.

6.3 Overall Performance

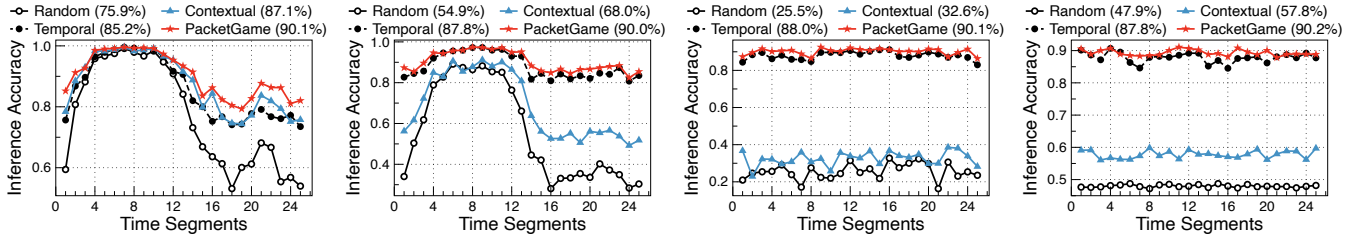
In evaluating the overall performance of PacketGame , we consider two perspectives: offline and online.

[1] https://github.com/yuanmu97/PacketGame



(a) Person Counting (b) Anomaly Detection (c) Super-resolution (d) Fire Detection

Figure 9: Offline filtering rate and inference accuracy on four tasks.



(a) Person Counting, B=248 (b) Anomaly Detection, B=207 (c) Super-resolution, B=238 (d) Fire Detection, B=480

Figure 10: Online inference accuracy over time with the same decoding budget (denoted by B, set to be the minimum value such that the average accuracy of PacketGame exceeds 90%).

Offline. In the offline evaluation, we adopt a positive-to-negative sample ratio of 1:1 and measure the filtering rate and inference accuracy. To analyze these metrics, we adjust the threshold of prediction confidence from 0.0 to 1.0 and plot curves to illustrate the performance. The optimal curves are computed using ground-truth labels. Formally, let a, r denote inference accuracy and filtering rate and let TN denote the ratio of true-negative cases (redundant packets) in test sets. The formula of the optimal curve is $a = 1 - \max(r - TN, 0)$. As depicted in Fig. 9, experimental results demonstrate that both the temporal estimator and contextual predictor provide effective filtering performance. And by combining both modules in PacketGame , we achieve the best and nearly optimal performance. For instance, with a target accuracy of 90%, PacketGame achieves filtering rates of 51.8%, 56.5%, 57.7%, and 53.9% across different tasks. The optimal filtering rate is 60%, PacketGame comes very close to this optimal performance, showcasing its efficacy in accurately filtering redundant packets.

Online. In the online evaluation, we focus on processing concurrent streams while adjusting the decoding budget. The contextual predictor is trained using 80% randomly sampled data from each dataset, following prior research practices [53]. With a target inference accuracy of 90%, we report the saved decoding budget when processing 1000 streams concurrently. Tab. 3 illustrates the significant decoding budget savings achieved by PacketGame , ranging from 52.0% to 79.3%, while still achieving over 90% accuracy. Notably, the combination of the proposed temporal and contextual modules is necessary, resulting in additional savings of 7.1%, 7.5%, 0.4%, and 1.5%. Furthermore, we report the maximal concurrency level that can be achieved within the same decoding budget (875 FPS) while maintaining a 90% accuracy target. PacketGame effectively improves the concurrency levels across all four tasks, achieving 2.1× to 4.8× concurrency.

The contributions of the temporal estimator and contextual predictor differ for each task. While the temporal estimator is less effective for the PC task, it plays a dominant role in enhancing performance for the SR task. This difference can be attributed to the stable temporal pattern of SR videos, which is relatively unrelated to detecting people. Additionally, we analyze the inference accuracy over time. Fig. 10 shows the accuracy on different time segments, with the decoding budget denoted by B and the average accuracy indicated in the legend. The budget is set as the minimum value that ensures the average accuracy of PacketGame exceeds 90%. We observe intuitive curves for the PC and AD tasks, where selecting necessary packets is more challenging during the day (segments 16-20) compared to the night (segments 4-8), given the correlation between these tasks and human activities. In contrast, the temporal patterns of SR and FD videos are randomly simulated, resulting in relatively stable accuracy over time.

The offline and online evaluations provide a comprehensive assessment of PacketGame , demonstrating its effectiveness in accurately filtering redundant packets, significant decoding budget savings, and improved concurrency levels across various tasks and operational conditions.

Overheads. PacketGame serves as a plug-in in video inference pipelines and we report its computing overheads in Tab. 4. We consider three metrics: device-independent FLOPs (floating-point operations, profiled by TensorFlow profiler API *float_operation*), the latency per frame, and the energy per frame (on the mobile

Table 4: Overheads on an edge server and a mobile phone.

Model	FLOPs	Latency per Frame Edge/Mobile	Energy per Frame Mobile
MobileNetV1	1137M	4/116ms	410mJ
InFi (image)	351M	0.8/16ms	15mJ
Reducto (area)	N/A	0.9/20ms	22mJ
PacketGame	5K	7/154μs	<1mJ

phone). Experimental results show that computational costs of PacketGame are orders of magnitude less, compared with a light-weight model (MobileNetV1), the on-server frame filter (InFi [53]), and the on-camera frame filter (Reducto [37]). PacketGame has 5K FLOPs, only 0.004% of MobileNetV1 (1137M). And for latency, PacketGame costs 7 μs per frame, 570× faster than MobileNetV1 (4 ms per frame). Although not designed for on-camera deployment, running PacketGame on a mobile phone costs only 154μs and less than 1mJ energy. As a reference, InFi and Reducto cost 15 and 22 mJ per frame on the same mobile device. Therefore, in principle, performing on-camera packet gating is feasible and energy efficient.

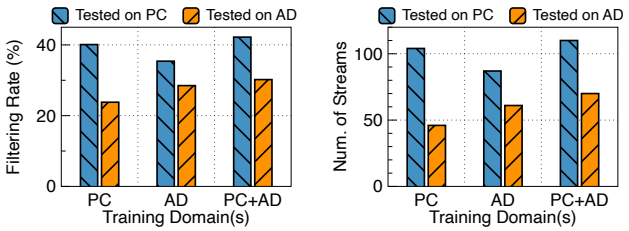
6.4 Micro-Benchmarks

Second, we explore the effects of the variables involved in PacketGame design.

Multi-task extension. To enhance the capabilities of the contextual predictor module, we have extended our design to support multi-task packet gating, as described in Sec. 5.2. For this extension, we consider two inference tasks, PC and AD, on the Campus1K dataset, treating them as separate domains. As depicted in Fig. 11, we observe that directly utilizing contextual predictors trained on the other domain leads to performance degradation. Specifically, the filtering rate is 16.3% lower for PC and 6.9% lower for AD, accompanied by a reduction of 58 concurrent streams for PC and 26 concurrent streams for AD. However, when employing the multi-task extended predictor, which leverages shared representations across tasks [56], we achieve improved performance. The multi-task extended predictor demonstrates a 2.1% higher filtering rate for PC and a 1.7% higher filtering rate for AD, resulting in an increase of 6 concurrent streams for PC and 9 concurrent streams for AD.

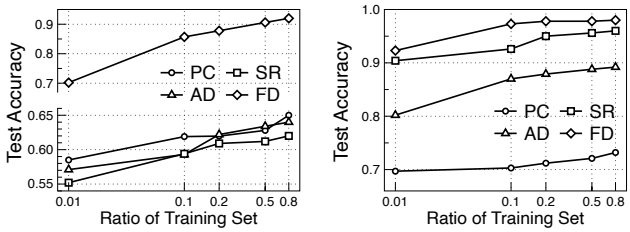
This improvement in performance can be attributed to the benefits of shared learning across multiple tasks. y training the contextual predictor simultaneously on multiple tasks, the model can leverage useful representations that are shared across domains. This shared representation learning enhances the overall learning capacity of the model and facilitates better performance for both PC and AD tasks.

Sensitivity to the training size. The size of the training sample has a significant impact on the efficiency of building PacketGame . To assess this sensitivity, we randomly sampled different ratios (0.01, 0.1, 0.2, 0.5, 0.8) of data and evaluated the classification accuracy on the same test set. As depicted in Fig. 12, the test accuracy consistently increases with an increasing training size, indicating the positive correlation between training sample size and the effectiveness of our proposed contextual predictor. Except for the



(a) Offline Filtering Rate (b) Online Concurrency

Figure 11: Multi-task extension of the contextual predictor. The target inference accuracy is 90%.



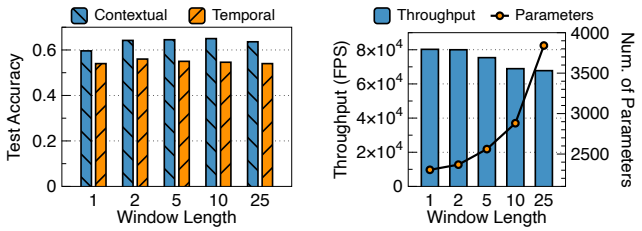
(a) Contextual Predictor (b) PacketGame

Figure 12: Test accuracy w.r.t. different sizes (1%, 10%, 20%, 50%, 80%) of training samples on four tasks.

extreme case where only 1% of the samples were utilized for training, both the contextual predictor (without the temporal view) and the complete PacketGame model demonstrate their ability to effectively learn from the available data. These findings underscore the importance of having a relatively sufficient amount of data for training PacketGame effectively, as it enables the model to capture diverse patterns and generalize well to unseen test data.

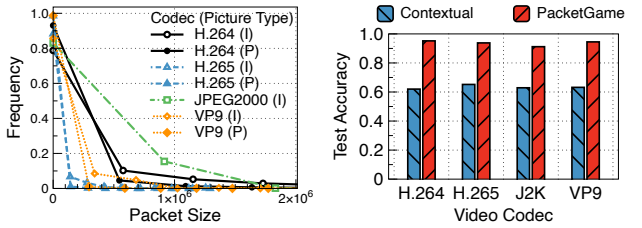
Effects of the window length parameter. The window length parameter in PacketGame plays a crucial role in determining both the packet gating performance and computational efficiency. To investigate its effects, we conducted experiments using different window lengths on the person counting task. The results, as illustrated in Fig. 13, reveal that the performance of both the contextual and temporal modules initially improves with an increasing window length and then starts to decline. Simultaneously, as the window length increases, the computational throughput decreases. Thus, a trade-off exists between accuracy and efficiency. We identified that a window length of 5 strikes a good balance between accuracy and efficiency, serving as the default choice. However, it is important to note that the optimal window length may vary depending on the specific application and requirements. Further exploration and fine-tuning of the window length parameter can be conducted to tailor PacketGame to different use cases and performance goals.

Video codec. The input streams processed by packet gating are generated by the video encoder. To examine the impact of different video codecs on PacketGame , we transcoded the original H.264 videos from the YT-UGC dataset using three different codecs [5]: H.265, JPEG2000, and VP9. Fig. 14a illustrates the clear differences in feature (picture type and packet size) distributions among the



(a) Learning Performance (b) Computational Efficiency

Figure 13: Effects of different window lengths (1, 2, 5, 10, 25) on the person counting task.



(a) Packet Size Histogram (b) Learning Performance

Figure 14: Effects of different video codecs (H.264, H.265, JPEG2000, VP9) on YT-UGC dataset.

different codecs. It is worth noting that, since the JPEG2000 codec produces streams with independent frames only, the contextual predictor of PacketGame removes the view of independent frames for this particular codec. Experimental results demonstrate that PacketGame exhibits robust performance (91.2-95.2% test accuracy) across all tested codecs, showcasing its versatility and adaptability to various video codecs commonly used in practice.

The findings regarding sensitivity to training size, the effects of the window length parameter, and the performance across different video codecs provide insights into the practical considerations and flexibility of PacketGame in real-world applications. By understanding and leveraging these factors, practitioners can optimize the performance and customization of PacketGame to suit their specific requirements and video analytics environments.

Extreme cases. To thoroughly investigate the capabilities and limitations of PacketGame , we conducted experiments considering two extreme cases that push the boundaries of the system.

(1) Extreme-low bit rate. In this scenario, we transcoded 1080p videos using an exceptionally low bit rate, such as 100K. We observed that, at such low bit rates, the packet size information becomes indistinguishable for most tasks. Consequently, the two views of packets in the contextual predictor of PacketGame no longer provide meaningful insights and tend to generate near-random guesses. However, this limitation has no effect on the temporal estimator, which relies on the temporal correlation of inference results. Therefore, even in extreme-low bit rate scenarios, PacketGame can still operate effectively by leveraging the temporal estimator to make accurate selections.

Table 5: Comparing PacketGame with complementary video inference methods on the person counting task. The target accuracy is 90%.

Method	Filtering Rate	Num. of Streams
Original	0%	1
TRT	0%	30
TRT+Grace	0%	30
TRT+Reducto	78.4%	162
TRT+InFi	85.1%	35
PacketGame	79.3%	5
TRT+PacketGame	79.3%	169

(2) Extreme-large GOP. In live streaming applications, it is common to encounter situations where the Group of Pictures (GOP) length is exceptionally large, such as 300. Experimental results indicate that the view of independent packets becomes less effective in these cases because the observations do not change frequently within such lengthy GOPs. However, we found that the other two views utilized by PacketGame , namely the contextual predictor and the temporal estimator, are not affected by extreme GOP lengths. Therefore, the overall performance of PacketGame remains robust and stable across different GOP settings.

These extreme cases shed light on the adaptability and resilience of PacketGame in challenging scenarios. While extremely low bit rates may limit the effectiveness of certain packet views, the temporal correlation of the packet sequence continues to provide valuable information. Similarly, in cases of large GOP lengths, although the view of independent packets may be less informative, the other views maintain their effectiveness. These findings highlight the robustness of PacketGame in diverse operational conditions and demonstrate that our hybrid design that combines metadata and feedback is quite necessary to handle extreme scenarios commonly encountered in real-world environments.

6.5 Comparisons with Complementary Video Inference Optimization Methods

To demonstrate the uniqueness and effectiveness of our proposed packet gating approach, we compare it with four video inference optimization methods: Grace [49], Reducto [37], InFi [53], and TensorRT (TRT) [8]. PacketGame is complementary to these methods: the optimization space of packet gating is overlapped with frame filtering [37, 53] and is orthogonal to video compression [49] and model acceleration [8]. While each of these methods contributes to improving different aspects of video inference, our focus is on enhancing end-to-end concurrency performance. TensorRT (TRT), for instance, improves the inference speed from 27.7 FPS to 753.9 FPS, which translates to a 30× concurrency improvement. On the other hand, Grace reduces the decoding cost but does not involve frame filtering. Consequently, its concurrency level remains limited by the inference speed, i.e., TRT+Grace also results in supporting 30 concurrent streams. Reducto, although it improves the number of concurrent streams from 30 (TRT only) to 162 (TRT+Reducto), requires modified cameras and does not support offline videos. InFi

reduces inference costs, but its concurrency bottleneck shifts to the decoding module, resulting in only 5 more concurrent streams. In contrast, our proposed packet gating approach is specifically designed to enhance the concurrency level by selecting packets for processing before decoding, thus reducing the costs associated with both the decoder and the inference model. As illustrated in Tab. 5, PacketGame outperforms these existing methods in terms of improving the number of concurrent video streams. Compared to the original method, PacketGame achieves a 5× concurrency. When combined with TRT, PacketGame supports an impressive 169 concurrent streams without requiring any modifications to the video sources. This highlights the effectiveness of PacketGame in enabling high levels of concurrency and scalability while maintaining compatibility with existing video inference methods.

7 DISCUSSION

Security auditing. Our approach demonstrates the feasibility of establishing a mapping from parsed packet metadata to whether an inference model should be executed. While this advancement offers significant benefits in terms of efficiency and resource utilization, it also introduces a potential security risk. Specifically, attackers who gain access to traffic metadata can potentially extract privacy-sensitive inference results, such as the detection of abnormal events occurring in certain locations. To mitigate this security risk, it is crucial to prioritize the protection of both RGB frames and packet-level metadata during transmission for secure video inference. By ensuring the confidentiality of both the visual content and the associated metadata, we can prevent unauthorized access and mitigate the potential exploitation of sensitive inference results by malicious entities.

Modality extension. In addition to supporting video packets, our design has the potential for extension to support other types of packet sequences. Recent work [53] has explored the generalization of frame filtering to input filtering and has presented a comprehensive framework that enables the filtering of various data modalities, including audio, motion sensor signals, and wireless signals. This modality extension represents an exciting avenue for future research and development of PacketGame . By broadening the scope of supported modalities, we can create a more versatile and adaptable system that caters to a wider range of multimedia applications and scenarios. The ability to filter and process diverse types of data streams in a unified manner opens up opportunities for enhanced inference efficiency.

8 RELATED WORK

Video decoding acceleration. The industry has developed hardware and software solutions for video acceleration. Kiloview [4] developed various models of hardware decoders, e.g., DC230 and D260, which supports concurrently decoding multiple high-resolution videos. Intel oneVPL [12] provides OS-independent APIs for decoding videos across heterogeneous hardware (CPUs, integrated and discrete GPUs). NVDEC [7] is a hardware decoder contained in NVIDIA GPUs. The accelerated graphics engine of NVDEC supports faster video decoding. In academia, several work [48, 49] propose inference-aware compression for source videos by analyzing frame features and gradients of neural networks. Since the compression strategies are designed for inference (rather than human perception), these methods can achieve higher compression ratios and thus faster decoding. Recent work [32, 35] explore specialized deep neural networks on partially decoded frames which can also alleviate the decoding overhead. PacketGame is complementary to these efforts: the accelerated decoder increases the budget in our formalization.

Frame filtering. Efforts to improve efficiency in video analytics have led to the adoption of frame filter [19, 37, 53], a prevailing approach for removing redundant frames that do not contain relevant content for inference. Various strategies have been proposed in existing literature, leveraging low-level features [37] and learned deep features [19, 53]. All of them utilize RGB frame-level information as the input for filtering, so they must be applied after the decoder. PacketGame is the first attempt that inserts the filtering stage between the parser and decoder in the video inference pipeline. While there is some overlap between frame filtering and our proposed packet gating mechanism in terms of optimization space, they can still be applied complementarily.

Inference acceleration. Extensive work has been conducted on inference acceleration [23] within the field of deep learning. Various approaches have been explored, such as neural network pruning [42], low-rank factorization [22], quantization [16], and knowledge distillation [30]. TensorRT [8] implements many GPU-focused acceleration techniques, and they have gained significant popularity in production environments. While these advancements in model acceleration are valuable, it is important to note that they are orthogonal to PacketGame . In fact, these techniques reinforce the necessity of our proposed packet gating: with the increase of inference throughput, the bottleneck is further solidified in the decoding stage (§ 2.3). Therefore, the introduction of packet gating, as proposed in our research, becomes crucial.

9 CONCLUSION

In this paper, we identified that the end-to-end concurrency bottleneck is limited by video decoding and gave a quantitative condition. We proposed packet gating that selectively filters packets before running the video decoder. And we presented PacketGame , a general framework that is backed by both theoretical performance guarantees and practical system designs. Extensive evaluation using real systems and public videos shows that PacketGame can effectively improve the concurrency level of various video inference tasks.

ACKNOWLEDGMENTS

We thank the anonymous SIGCOMM reviewers for their constructive comments. The research is partially supported by National Key R&D Program of China under Grant No. 2021YFB2900103, No. 2021ZD0110400, Innovation Program for Quantum Science and Technology 2021ZD0302900 and China National Natural Science Foundation with No. 62132018, No. 61932016, "Pioneer" and "Leading Goose" R&D Program of Zhejiang, 2023C01029, "the Fundamental Research Funds for the Central Universities" WK2150110024 and CAAI-Huawei MindSpore Open Fund.

REFERENCES

[1] 2018. ISR. https://github.com/idealo/image-super-resolution
[2] 2019. PaddleDetection, Object detection and instance segmentation toolkit based on PaddlePaddle. https://github.com/PaddlePaddle/PaddleDetection
[3] 2020. AI that scans a construction site can spot when things are falling behind. https://buildots.com/
[4] 2022. DC230 Video/IP Camera Decoder. https://www.kiloview.com/en/decoder/h264-dc230
[5] 2022. FFmpeg. https://ffmpeg.org/ffmpeg.html
[6] 2022. MindSpore. https://github.com/mindspore-ai/mindspore
[7] 2022. NVIDIA Video Codec SDK. https://developer.nvidia.com/nvidia-video-codec-sdk
[8] 2022. TensorRT. https://developer.nvidia.com/tensorrt
[9] 2022. Twitch Statistics. https://backlinko.com/twitch-users
[10] 2022. UGC Dataset. https://media.withyoutube.com/
[11] 2022. YouTube Stats. https://www.wyzowl.com/youtube-stats/
[12] 2023. Intel oneAPI Video Processing Library. https://www.intel.com/content/www/us/en/developer/tools/oneapi/onevpl.html
[13] 2023. TensorFlow. https://www.tensorflow.org/
[14] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *Proceedings of the ICML 2016 Conference*. PMLR, 173–182.
[15] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.
[16] Mustafa Ayazoglu. 2021. Extremely lightweight quantization robust real-time single-image super resolution for mobile devices. In *Proceedings of the IEEE/CVF CVPR 2021 Conference*. 2472–2479.
[17] Florin Baboescu and George Varghese. 2001. Scalable packet classification. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 199–210.
[18] Johan Barthélemy, Nicolas Verstaevel, Hugh Forehead, and Pascal Perez. 2019. Edge-computing video analytics for real-time traffic monitoring in a smart city. *Sensors* 19, 9 (2019), 2048.
[19] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya Dulloor. 2019. Scaling video analytics on constrained edge nodes. *Proceedings of Machine Learning and Systems* 1 (2019), 406–417.
[20] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 155–168.
[21] Wei Chen, Liwei Wang, Haoyu Zhao, and Kai Zheng. 2021. Combinatorial semi-bandit in the non-stationary environment. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence*, Vol. 161. PMLR, 865–875.
[22] Yanjiao Chen, Baolin Zheng, Zihan Zhang, Qian Wang, Chao Shen, and Qian Zhang. 2020. Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. *ACM Computing Surveys (CSUR)* 53, 4 (2020), 1–37.
[23] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A Comprehensive Survey on Model Compression and Acceleration. *Artificial Intelligence Review* 53, 7 (2020), 5113–5155.
[24] Xavier Corbillon, Florian Boyrivent, Grégoire Asselin De Williencourt, Gwendal Simon, Géraldine Texier, and Jacob Chakareski. 2016. Efficient lightweight video packet filtering for large-scale video data delivery. In *Proceedings of the IEEE ICMEW 2016 Conference*. 1–6.
[25] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. 2020. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems* 33 (2020), 17429–17442.
[26] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *Proceedings of the ECCV 2016 Conference*. Springer, 391–407.
[27] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. 2006. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*. 281–286.
[28] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. 2021. YOLOX: Exceeding YOLO Series in 2021. *arXiv preprint* (2021).
[29] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* (2016).
[30] Zibin He, Tao Dai, Jian Lu, Yong Jiang, and Shu-Tao Xia. 2020. Fakd: Feature-affinity based knowledge distillation for efficient image super-resolution. In *Proceedings of the IEEE ICIP 2020 Conference*. IEEE, 518–522.
[31] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying large video datasets with low latency and low cost. In *Proceedings of the 13th USENIX OSDI Conference*. 269–286.
[32] Jinwoo Hwang, Minsu Kim, Daeun Kim, Seungho Nam, Yoonsung Kim, Dohee Kim, Hardik Sharma, and Jongse Park. 2022. CoVA: Exploiting Compressed-Domain Analysis to Accelerate Video Analytics. In *Proceedings of 2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 707–722. https://www.usenix.org/conference/atc22/presentation/hwang
[33] Arpit Jadon, Mohd Omama, Akshay Varshney, Mohammad Samar Ansari, and Rishabh Sharma. 2019. FireNet: a specialized lightweight fire & smoke detection model for real-time IoT applications. *arXiv preprint* (2019).
[34] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the ACM SIGCOMM 2018 Conference*. 253–266.
[35] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly Optimizing Preprocessing and Inference for DNN-Based Visual Analytics. *Proc. VLDB Endow.* 14, 2 (2020), 87–100. https://doi.org/10.14778/3425879.3425881
[36] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, Chen Sun, Gareth Tyson, and Hongqiang Harry Liu. 2022. LiveNet: A Low-Latency Video Transport Network for Large-Scale Live Streaming. In *Proceedings of the ACM SIGCOMM 2022 Conference*. ACM, 812–825.
[37] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the ACM SIGCOMM 2020 Conference*. 359–376.
[38] Peng Liu, Bozhao Qi, and Suman Banerjee. 2018. Edgeeye: An edge service framework for real-time intelligent video analytics. In *Proceedings of the 1st international workshop on edge systems, analytics and networking*. 1–6.
[39] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. 2019. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3133–3174.
[40] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*. 50–56.
[41] Thuy TT Nguyen and Grenville Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials* 10, 4 (2008), 56–76.
[42] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the 25th ACM ASPLOPS Conference*. 907–922.
[43] Yaxuan Qi, Lianghong Xu, Baohua Yang, Yibo Xue, and Jun Li. 2009. Packet classification algorithms: From theory to practice. In *Proceedings of the IEEE INFOCOM 2009 Conference*. IEEE, 648–656.
[44] Lijing Qin, Shouyuan Chen, and Xiaoyan Zhu. 2014. Contextual combinatorial bandit and its application on diversified online recommendation. In *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 461–469.
[45] Newton Spolaor, Huei Diana Lee, Weber Shoity Resende Takaki, Leandro Augusto Ensina, Claudio Saddy Rodrigues Coy, and Feng Chung Wu. 2020. A systematic review on content-based video retrieval. *Engineering Applications of Artificial Intelligence* 90 (2020), 103557.
[46] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al. 2021. Sparse r-cnn: End-to-end object detection with learnable proposals. In *Proceedings of the IEEE/CVF CVPR 2021 Conference*. 14454–14463.
[47] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. 2018. Bandwidth-Efficient Live Video Analytics for Drones Via Edge Computing. In *Proceedings of 2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 159–173.
[48] Xuedou Xiao, Juecheng Zhang, Wei Wang, Jianhua He, and Qian Zhang. 2022. Dnn-driven compressive offloading for edge-assisted semantic video segmentation. In *Proceedings of the IEEE INFOCOM 2022 Conference*. IEEE, 1888–1897.
[49] Xiufeng Xie and Kyu-Han Kim. 2019. Source compression with bounded dnn perception loss for iot edge computer vision. In *Proceedings of the ACM MobiCom 2019 Conference*. 1–16.
[50] Mariem Ben Yahia, Yannick Le Louedec, Gwendal Simon, Loutfi Nuaymi, and Xavier Corbillon. 2019. HTTP/2-Based Frame Discarding for Low-Latency Adaptive Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 15, 1, Article 18 (2019).
[51] Xiaoqiang Yan, Shizhe Hu, Yiqiao Mao, Yangdong Ye, and Hui Yu. 2021. Deep multi-view learning methods: a review. *Neurocomputing* 448 (2021), 106–129.
[52] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2022. NeuroScaler: Neural Video Enhancement at Scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*. ACM, 795–811.
[53] Mu Yuan, Lan Zhang, Fengxiang He, Xueting Tong, and Xiang-Yang Li. 2022. InFi: End-to-End Learnable Input Filter for Resource-Efficient Mobile-Centric Inference. In *Proceedings of the ACM MobiCom 2022 Conference*. ACM, 228–241.

[54] Mu Yuan, Lan Zhang, Zhengtao Wu, and Daren Zheng. 2020. High-quality Activity-Level Video Advertising. In *Proceedings of the IEEE/ACM IWQoS 2020 Conference*. 1–10.
[55] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and {Delay-Tolerance}. In *Proceedings of the 14th USENIX NSDI Conference*. 377–392.
[56] Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[57] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. 2017. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics* 28, 1 (2017), 162–169.
[58] Dongruo Zhou, Lihong Li, and Quanquan Gu. 2020. Neural Contextual Bandits with UCB-Based Exploration. In *Proceedings of the 37th ICML Conference*. JMLR, Article 1065, 11492–11502 pages.

Appendices are supporting material that has not been peer-reviewed.

A PROOFS

Proof of Lemma. 1.

Given predictions $f_\theta(x, \bar{\mu})$ as the item values and c_i as item costs, maximizing the accumulated value under the cost budget B is a knapsack problem. Greedily selecting items w.r.t. the ratio of value over cost can be arbitrarily bad for general cases. Fortunately, the cost of decoding is approximately fractional, e.g., $c_{t,1} = 1I + 1B + 1P$ in Fig. 6. Intuitively, when the remaining budget is lower than the next cost, we can still decode partial frames. And we assume the value of decoding reference frames follows the same fraction. Under this practical assumption, we can prove the approximation ratio as follows.

Proof. Let $V_{\mathcal{A}}$ denote the value returned by our algorithm. Consider two optimal solutions: opt for our approximately fractional knapsack problem and opt_F for the rigorous fractional knapsack problem. Then we have $V_{\mathcal{A}} \le \text{opt} \le \text{opt}_F$. Define b as the remaining budget, r as the value-cost ratio of the next item, and c as the maximal cost of items, so $b < c$. And since we select items greedily w.r.t. the value-cost ratio, $V_{\mathcal{A}} \ge (B - b)r$.

$$\frac{V_{\mathcal{A}}}{\text{opt}_F} = \frac{V_{\mathcal{A}}}{V_{\mathcal{A}} + br} \tag{2}$$

$$= \frac{1}{1 + \frac{br}{V_{\mathcal{A}}}} \tag{3}$$

$$\ge \frac{1}{1 + \frac{br}{(B-b)r}} \tag{4}$$

$$= \frac{1}{1 + \frac{b}{B-b}} \tag{5}$$

$$= 1 - \frac{b}{B} \tag{6}$$

$$\ge 1 - \frac{c}{B}. \tag{7}$$

So the approximation ratio: $\frac{V_{\mathcal{A}}}{\text{opt}} \ge \frac{V_{\mathcal{A}}}{\text{opt}_F} \ge 1 - \frac{c}{B}$. □

Proof of Theorem. 1.

We consider our formalized problem as a m-armed combinatorial contextual bandit problem, where the total number of rounds T is known. At the round $t \in [T]$, we observe the context consisting of both metadata and feedback estimation: $\{x_{t,i}, \bar{\mu}_{t,i} | i \in [m]\}$. Our algorithm selects a subset S_t of streams (arms) to decode and receives the feedback (reward) $r_{t,S_t} = \{r_{t,i} | i \in S_t\}$. We define regret as follows:

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} (r_t^* - r_{t,S_t})\right], \tag{8}$$

where $r_t^* = \max_{S \subseteq 2^{[m]}} \mathbb{E}\left[r_{t,S}\right]$ is the maximal expected reward at round t.

Proof. Using Lemma. 1, we have a (α, β)-approximation oracle, where $\alpha = 1 - c/B$ and $\beta = 1$. Based on this oracle and using existing results [21, 44, 58] we can derive a $\tilde{O}(\sqrt{T})$ regret bound under certain theoretical assumptions. □