Tangram: High-resolution Video Analytics on Serverless Platform with SLO-aware Batching

Haosong Peng, Yufeng Zhan, Peng Li, Yuanqing Xia

Haosong Peng
livion_i@icloud.com

School of Automation
Beijing Institute of Technology

July 25, 2024

Outline

1. Introduction
2. Background and Motivation
3. Design of Tangram
4. Evaluation
5. Conclusion

Introduction

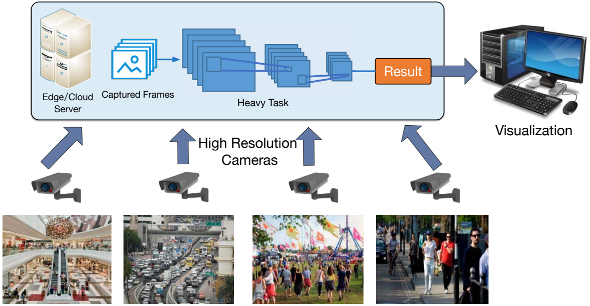


Figure: Organizations are deploying cameras at scale for video analytics.

- Due to **limited resources**, cameras cannot deploy AI models to perform video analysis, especially when dealing with high-resolution videos.
- Cameras need to **offload** computing tasks to the edge or cloud to complete complex, heavy tasks.

Figure: High-resolution cameras have a wide field of view.

Introduction

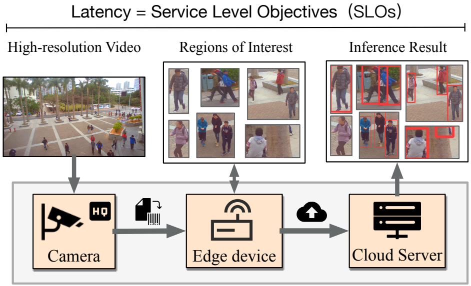


Figure: Content-aware approach.

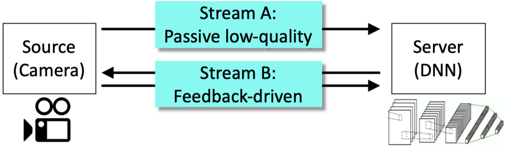
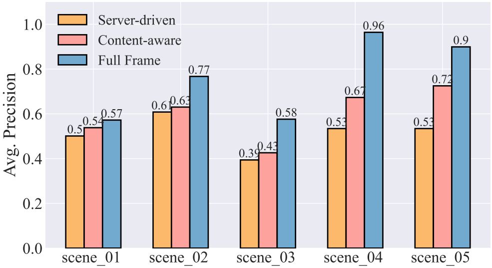


Figure: Server-driven approach.

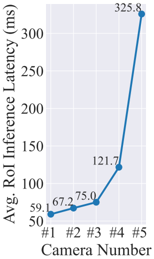
Transmitting complete high-resolution videos requires substantial network bandwidth resources.

- **Content-aware approach**: The data is sent to the cloud for inference after determining the regions of interest on the camera or edge device;
- **Server-driven approach**: It allows edge devices to send low-quality videos to the cloud. The cloud then identifies RoIs and provides feedback on their positions. Only these RoIs encoded in high quality are sent in the second transmission round.

Introduction



(a) Loss of inference accuracy in high-resolution videos.



(b) Latency v.s. #Camera

Figure: Previous methods are hard to adapt to high-resolution videos.

Weakness:

(1) Accuracy decline for server-driven and content-aware approaches in high-resolution object detection.
(2) Two rounds of transmission require additional communication overhead.
(3) System can not scale up when the camera number is increasing.

Our Work

How can we optimize both **bandwidth and computational** overhead in edge-based high-resolution video analysis scenarios, while ensuring system **scalability**?

Figure: The Design Preview of Tangram

Challenges and our solutions:

(1) *Identify RoIs* $\rightarrow$ Adaptive Frame Partitioning;

(2) *Unify the RoIs to accommodate parallel computing* $\rightarrow$ Patch Stitching;

(3) *Scheduling algorithm for dynamic requests* $\rightarrow$ Online SLO-aware Batching.

Outline

1. Introduction
2. **Background and Motivation**
3. Design of Tangram
4. Evaluation
5. Conclusion

Background

Serverless Function

Serverless Function is an event-driven computing service and has many advantages such as 1)Efficient and O&M-free; 2) Elastic and Highly Available; 3) Low Cost on Demand. It is highly suitable for large-scale video analysis scenarios, where each camera will request task inference in the cloud.

For example, An invocation of the Alibaba Cloud Function is charged based on the execution time and the allocated resource as

$$C_{Ali} = T_f \cdot (n_C \cdot P_C + m_M \cdot P_M + m_G \cdot P_G) + P_{req}, \tag{1}$$

where T_f is the function execution time, n_C, m_M, and m_G are the vCPU, GB of memory, and GB of GPU memory used by the function instance, respectively.

- Price of vCPU: $P_C = 2.138 \times 10^{-5}\$/vCPU \cdot s$;
- Price of memory: $P_M = 2.138 \times 10^{-5}\$/GB \cdot s$;
- Price of GPU: $P_G = 1.05 \times 10^{-4}\$/GB \cdot s$;
- Basic price: $P_{req} = 2 \times 10^{-7}\$$.

Motivation

Redundancy in Video Inference Data

Table: Redundancy in video inference data on PANDA4K dataset.

Index	Scene Name (# Frame)	# Person	RoIs Prop.△ (%)	Redundancy◇(%)
1	University Canteen (234)	123	5.4510	12.39
2	OCT Habour (234)	191	8.3141	11.28
3	Xili Crossroad (234)	393	5.9132	9.24
4	Primary School (148)	119	14.1561	15.43
5	Basketball Court (133)	54	5.0354	15.43
6	Xinzhongguan (222)	857	5.2316	10.93
7	University Campus (180)	123	2.5860	10.31
8	Xili Street 1 (234)	325	9.6297	10.65
9	Xili Street 2 (234)	152	8.7498	9.25
10	Huaqiangbei (234)	1730	9.6732	9.16

represents "The number of " ;
△ The ratio of the total area of RoIs to the whole frame;
◇ Non-RoIs inference time proportion.

Motivation

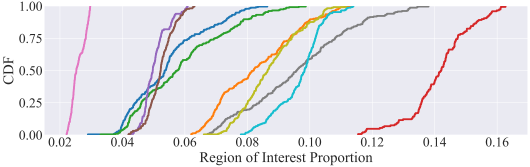
Fluctuation of Inference Workloads

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

AIZU THE UNIVERSITY OF AIZU

JULY 23-26 2024 ICDCS Jersey City, New Jersey, USA

(a) Temporal variation of the object area in ten scenes.

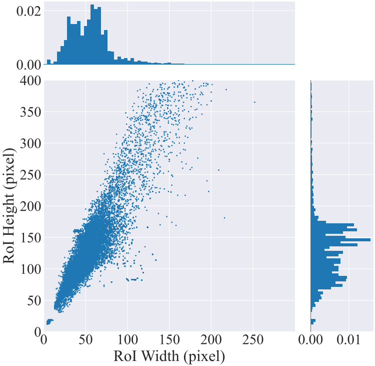


(b) The cumulative distribution function (CDF) of RoI proportion.

Figure: The variation of video inference workloads in the ten real-world scenes.

Motivation

Challenges of RoIs Batching



(a) Sizes of RoIs in scene_01.

(b) The inference accuracy of the PANDA dataset at different resolutions on Yolov8x.

Figure: Challenges of RoIs batching.

Outline

Tangram Overview

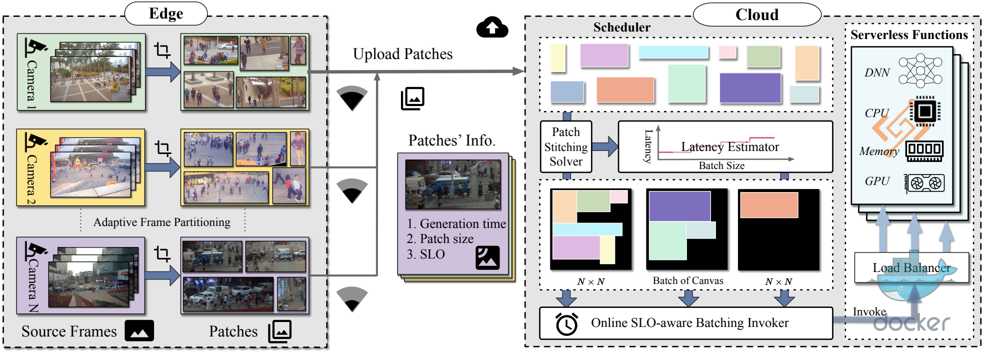


Figure: Overview of Tangram

(1) Captured high-resolution video frames are partitioned into **patches** by **Adaptive Frame Partitioning** algorithm;

(2) Each patch is uploaded with a time stamp and DDL to the cloud as an inference request;

(3) The **Patch Stitching Solver** packs patches in the **Canvas**;

(4) The **Online SLO-aware Batching Invoker** decides when to trigger a batch for Serverless Function.

Adaptive Frame Partitioning

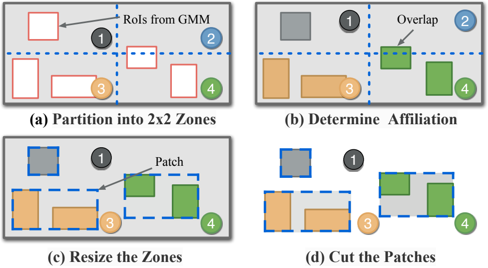


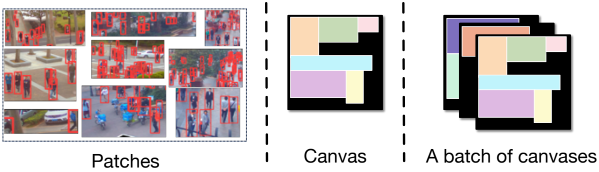
Figure: The process of adaptive frame partitioning algorithm.

(a) **Generate RoIs:** Each video frame is evenly divided into $X \times Y$ zones. We then use the Gaussian mixture model (GMM) to obtain the RoIs.

(b) **Determine affiliation:** Each RoI is associated with a specific zone. For every RoI, we calculate the overlap area with each zone. The RoI is assigned to the zone with the maximum overlap area, and it is added to the corresponding zone's list.

(c) **Resize the zones:** We resize each zone to the minimum enclosing rectangle that covers all RoIs associated with it.

(d) **Cut the patches**: Finally, each zone is cut out to form a patch.



Let $\mathbb{I} = \{1, \ldots, I\}$ denote the set of patches, $\mathbb{J} = \{1, \ldots, J\}$ denote the set of canvases, and $\mathbb{K} = \{1, \ldots, K\}$ denote the set of batches.

We define a binary variable x_i^j, where $x_i^j = 1$ if patch i is in canvas j, otherwise $x_i^j = 0$.
The $y_j^k = 1$ indicates that canvas j is placed in batch k, and is 0 otherwise.
And $z_i^k = 1$ denotes that patch i is in batch k, else it is 0.

Batching Problem Description

Our objective is to minimize the total computation cost of serverless functions, which is

$$\min \quad \sum_{k=1}^{K} T_f^k \left(n_C \cdot P_C + m_M \cdot P_M + m_G \cdot P_G\right) + P_{req} \qquad (2)$$

$$\text{s.t.} \quad \sum_{j=1}^{J} x_i^j = 1, \sum_{k=1}^{K} z_i^k = 1, \forall i \in \mathbb{I}, \qquad (3)$$

$$\sum_{i=1}^{I} s_i x_i^j \leq S, \forall j \in \mathbb{J}, \qquad (4)$$

$$w \sum_{j=1}^{J} y_j^k + \tau \leq m_G, \forall k \in \mathbb{K}, \qquad (5)$$

$$T_{i,wait} + T_f^k \leq SLO_i, i \in \{i | z_i^k = 1, \forall i \in \mathbb{I}\}, \qquad (6)$$

$$T_f^k = f(\sum_{j=1}^{J} y_j^k, n_C^k, m_m^k, m_G^k), \forall k \in \mathbb{K}, \qquad (7)$$

where τ is the model size, w represents the GPU memory occupied by a single canvas, s_i is the size of patch i, and S is the canvas size.

- Constraint (3) states that each patch can only be placed on a particular canvas in a specific batch.
- Constraint (4) implies that the total area of all patches in a canvas should not exceed the canvas' area.
- Constraint (5) specifies that the GPU memory usage of each batch should not exceed the resource allocated to the function.
- Constraint (6) asserts that each patch should not violate the SLO, where $T_{i,wait}$ and SLO_i are the waiting time and SLO of patch i.
- Constraint (7) is the inference time of batch k, which is related to the size of the batch and the function configuration.

Algorithm Design

Online SLO-aware Batching Invoker.

Basic Idea: The scheduler receives patches one after another, and we only need to determine when to stop waiting and invoke the function. When the estimated time for the current batch is insufficient, then trigger the batch at once.

Latency Estimator: Estimate the inference time for different batch sizes offline.

Patch-stitching Solver: Use bin packing algorithms to stitch patches onto a canvas.

Algorithm 2: SLO-aware Batching Algorithm

Input: The information $\mathbb{P}_i = \{w_i, h_i, t_{ddl_i}\}$ of patch i, Canvas size $M \times N$

Initialize a queue $\mathbb{Q} = \{\emptyset\}$ to save the patches' info;
$\mathbb{C} \leftarrow \{\emptyset\}$, $\mathbb{C}_{old} \leftarrow \{\emptyset\}$;
while *True* **do**
    if *received patch i with $\mathbb{P}_i$* **then**
        $\mathbb{Q}$.append($\mathbb{P}_i$);
        $t_{DDL} \leftarrow \min\{t_{ddl_i}\}_{\mathbb{P}_i \in \mathbb{Q}}$;
        $\mathbb{C}_{old} \leftarrow \mathbb{C}$;
        $\mathbb{C} \leftarrow$ *Patch_stitching_solver*$(\mathbb{Q}, M, N)$;
        $T_{slack} \leftarrow$ *Latency_estimator*$(\mathbb{C})$;
        $t_{remain} \leftarrow t_{DDL} - T_{slack}$;
        if $t_{remain} > t$ *or memory*$(\mathbb{C}) > m_G - \tau$ **then**
            Invoke$(\mathbb{C}_{old})$;
            $\mathbb{Q} \leftarrow \{\mathbb{P}_i\}$, $\mathbb{C}_{old} \leftarrow \{\emptyset\}$;
            $\mathbb{C} \leftarrow$ *Patch_stitching_solver*$(\mathbb{Q}, M, N)$;
            $T_{slack} \leftarrow$ *Latency_estimator*$(\mathbb{C})$;
            $t_{remain} \leftarrow t_{DDL} - T_{slack}$;
        end
    end
    if $t = T_{remain}$ **then**
        Invoke$(\mathbb{C})$;
        $\mathbb{Q} \leftarrow \{\emptyset\}$, $\mathbb{C} \leftarrow \{\emptyset\}$, $\mathbb{C}_{old} \leftarrow \{\emptyset\}$;
    end
end

Outline

Experimental Setting

(1) **Testbed**: Alibaba Cloud Function Compute, Desktop with 2 NVIDIA GeForce RTX 4090 GPUs, NVIDIA Jetson Nano 4GB as the edge device.

(2) **Dataset**: PANDA dataset, resize the frames to 3840×2160 (4K) as the PANDA4K dataset.

(3) **Model**: Yolov8x with 68.2M parameters.

(4) **Serverless Function Configurations**: 2 vCPU, 4GB memory, and 6GB GPU memory.

(5) **Baselines**: Full Frame; Masked Frame; ELF; Clipper; MArk.

Performance of Tangram

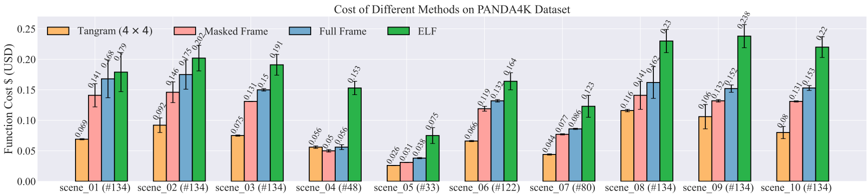


Figure: Cost of Tangram, ELF, Masked Frame, and Full Frame on ten scenes of PANDA4K (# the number of evaluation frames) on Alibaba Cloud Function Compute.

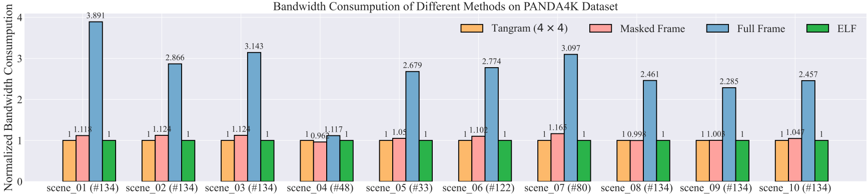


Figure: Bandwidth Consumption of Tangram, ELF, Masked Frame, and Full Frame on ten scenes of PANDA4K (# the number of evaluation frames) on Alibaba Cloud Function Compute.

Performance of Tangram

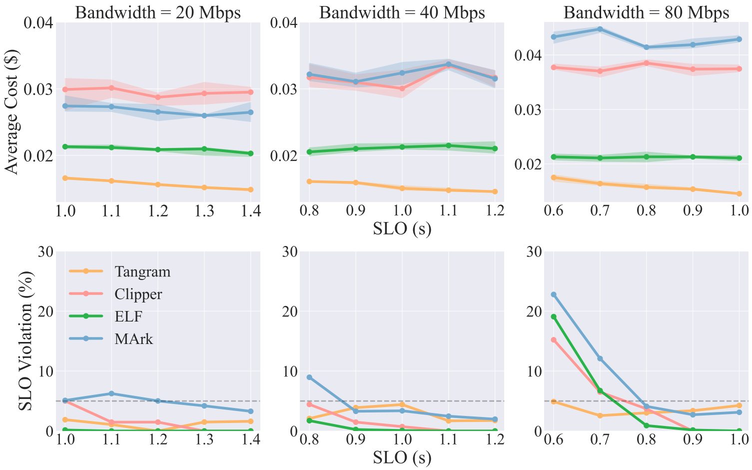


Figure: The end-to-end performance of Tangram under different bandwidths and SLO conditions.

Performance of Tangram

Table: Comparisons of Inference Accuracy (AP)

Scene	Accuracy (AP)				Scene	Accuracy (AP)			
	Full	Partitions (2x2)	Partitions (4x4)	Partitions (6x6)		Full	Partitions (2x2)	Partitions (4x4)	Partitions (6x6)
01	0.572	**0.583 (+0.011)**	0.573 (+0.001)	0.565 (-0.007)	**06**	0.686	**0.665 (-0.021)**	0.647 (-0.039)	0.644 (-0.042)
02	0.767	**0.756 (-0.011)**	0.747 (-0.020)	0.750 (-0.017)	**07**	0.698	0.663 (-0.035)	**0.692 (-0.006)**	0.672 (-0.026)
03	0.576	**0.570 (-0.006)**	0.549 (-0.027)	0.493 (-0.083)	**08**	0.638	**0.626 (-0.012)**	0.622 (-0.016)	0.549(-0.089)
04	0.964	0.962 (-0.002)	**0.964 (0)**	0.927 (-0.037)	**09**	0.598	0.587 (-0.011)	**0.598 (0)**	0.553 (-0.045)
05	0.899	0.893 (-0.006)	**0.894 (-0.005)**	0.830 (-0.069)	**10**	0.634	**0.615 (-0.019)**	**0.615 (-0.019)**	0.586 (-0.048)

Outline

1. Introduction
2. Background and Motivation
3. Design of Tangram
4. Evaluation
5. **Conclusion**

Conclusion

BEIJING INSTITUTE OF TECHNOLOGY
THE UNIVERSITY OF AIZU
JULY 23-26 2024 ICDCS Jersey City, New Jersey, USA

Conclusion

1. We design Tangram, a video analytics system that takes advantage of several techniques to optimize the cost of high-resolution video analytics in the cloud-edge scenario.
2. We develop and deploy a prototype on a testbed running real video analytics workloads and compare it with the state-of-the-art.
3. Experimental results demonstrate that Tangram can reduce bandwidth consumption by up to 74.30% and computation cost by up to 66.35%, respectively, while maintaining SLO violations within 5% and negligible accuracy loss.

Thank you for your attention!

Email: livion@bit.edu.cn